FDEM: The Evolution and Application of the Finite Difference Element Method (FDEM) Program Package for the Solution of Partial Differential Equations

Willi Schönauer and Torsten Adolph
Rechenzentrum der Universität Karlsruhe
D-76128 Karlsruhe, Germany
E-mail schoenauer@rz.uni-karlsruhe.de
E-mail adolph@rz.uni-karlsruhe.de

Abschlussbericht des Verbundprojekts
FDEM: Weiterentwicklung und Anwendung des Finite Difference Element Method (FDEM) Programmpakets zur Lösung von partiellen Differentialgleichungen
BMBF Förderkennzeichen 01 IR A16 A

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01 IR A16 A gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Abstract

This report is the most detailed report about the Finite Difference Element Method (FDEM). FDEM has been implemented in the FDEM program package. The development of the method and of the code are closely interconnected.

FDEM is a black-box solver that solves by a finite difference method arbitrary non-linear systems of PDEs (partial differential equations) on an unstructured FEM grid. The 2-D or 3-D PDEs must be of elliptic or parabolic type. The FEM grid serves only for the structuring of the space, i.e. the determination of the neighboring nodes. In 2-D we use triangles, in 3-D tetrahedrons. For each node we generate with a sophisticated algorithm by means of neighboring nodes difference formulas of consistency order q, optionally $q = 2$ or $q = 4$ or $q = 6$. By the use of formulas of order $q + 2$ an estimate of the discretization error is obtained. For parabolic equations we use in time direction (for stability reasons) fully implicit difference formulas of consistency order $p < 6$, with error estimate by formulas of order $p + 1$.

The knowledge of the error permits a selfadaptation of the solution method. In time direction the order p and the time step are always automatically optimized. In space direction the solution can be adapted to a requested accuracy by grid refinement (bisection of triangle or tetrahedron edges).

For many technical applications the solution domain is composed from subdomains in which hold different PDEs, e.g. a fluid structure coupling. It is not possible to differentiate the solution across boundaries of the subdomains. Therefore we have introduced in FDEM "dividing lines" (which are in 3-D in effect dividing areas). These dividing lines are internal boundaries. The solutions on both sides of the dividing lines are coupled by coupling conditions (CCs). Thus one gets over the whole domain (composed of several subdomains) a global solution with global error estimate. The meshes on both sides of a dividing line have not to coincide, one may have non-matching grids.

Because FDEM must solve arbitrary non-linear systems of PDEs, the linearization is executed by the Newton-Raphson method. In order to make the method as robust as possible we check after each iteration step if the defect has decreased. If this does not hold we try with a selfadapted relaxation factor to reduce the defect. The Newton method is terminated if the Newton defect is smaller than a corresponding discretization error term, that no unnecessary digits are computed.

From the discretization of the PDEs result very large and sparse linear systems of equations. These are solved by the LINSOL program package that has also been developed at the Computer Center of the University of Karlsruhe. LINSOL comprises CG methods of quite different types for the iterative solution, and also contains a direct solver with optionally reduced fill-in that can be used as preconditioner for the iterative solvers.

FDEM and LINSOL have been developed from the beginning for efficient data structures on distributed memory parallel computers. Here the distribution of the data to the processors plays the decisive role. We use a 1-D domain decomposition that can be executed automatically and runs over dividing lines. For grid refinement a new distribution of the data is executed after each refinement step. The exchange of the data between the processors takes place by the quasi standard MPI. Thus FDEM is running efficiently on shared and distributed memory computers.

FDEM is a program package for the solution of PDEs that has **unique** properties. To us no other program package is known that unifies in a single code comparable properties concerning the flexibility of the solution method, of the solution domain, of the error estimate and of the parallelization.

As FDEM is a black-box solver, the user must enter the PDEs, BCs (boundary conditions) and CCs as Fortran code in given program frames. In Chapter 4 this is demonstrated for the PDEs of Section 3.4.

In Chapter 3 three examples for the application of FDEM to different technical problems are presented: for the numerical simulation of the manufacturing of metal bellows, of the lubrication gap in a Diesel High Pressure Injection Pump and of the oxygen diffusion in a PEM fuel cell.

Contents

1 Preface

On January 1, 2001 started the research project “FDEM: Weiterentwicklung und Anwendung des Finite Difference Element Method (FDEM) Programmpaketes zur Lösung von partiellen Differentialgleichungen” (the title of this report is the English translation). It was a cooperative project (Verbundprojekt) between the Computer Center of the University of Karlsruhe (Rechenzentrum der Universität Karlsruhe), the Institute for Metal Forming Technology of the University of Stuttgart (IFU=Institut für Umformtechnik) and three manufacturers: The IWKA-BKT (Stutensee near Karlsruhe), that were interested in the numerical simulation of the manufacturing process of metal bellows for which the IFU had to deliver the material equations, the High Pressure Diesel Injection Pump branch of Bosch (Stuttgart), that were interested in the numerical simulation of certain aspects of injection pumps at 2000 bar, and Freudenberg Forschungsdienste (Weinheim) that were interested in the numerical simulation of oxygen diffusion in the non-woven (Vliesstoff) layer of a fuel cell. The two universities were supported by the BMBF (German Ministry of Education and Research) under grant 01 IRA 16A and 16B, the manufacturers financed their part themselves.

There were two main goals of the research project: a) the evolution of the FDEM program package so that it could treat domains where different subdomains had non-matching grids and could slide relative to each other, and b) to demonstrate that FDEM could solve difficult industrial problems for which there is no standard software available on the commercial market. The evolution according to a) was needed to treat the problem of Bosch and will be needed for future problems of IWKA-BKT for multi-layered metal bellows.

We want to thank all institutions and people that cooperated to make this research project possible. We hope that the result will help to develop in the future better and more efficient technologies to advance the German economy in the worldwide technical competition.

This research report is organized as follows: In Chapter 2 the FDEM is presented in all details. Chapter 3 describes the application of FDEM to the three problem areas of the industrial partners. Chapter 4 is a type of User‘s guide for FDEM.

2 The FDEM Program Package

2.1 Introduction

The basic purpose of FDEM is to deliver a robust and efficient black box solver for the solution of arbitrary nonlinear systems of elliptic and parabolic PDEs under arbitrary nonlinear boundary conditions (BCs) on an arbitrary 2-D or 3-D domain. The domain may be composed from different subdomains with different PDEs and different non-matching grids. The subdomains may even slide relative to each other. The solution method is the FDM (Finite Difference Method) with arbitrary consistency order p in time direction (for parabolic PDEs) and arbitrary consistency order q in space direction. The difference formulas are generated on an unstructured FEM (Finite Element Method) mesh, in 2-D triangles, in 3-D tetrahedrons, hence the name Finite Difference Element Method (FDEM). It should be stressed that the FEM mesh (mostly generated by a commercial mesh generator, e.g. I-DEAS, PATRAN) is only used for the structure of the space, i.e. for the neighborhood relations of the nodes. The solution method is purely FDM.

The explicit character of the FDM (in contrast to the FEM) together with the fact that we can generate difference formulas of arbitrary consistency order allows a simple and easy access to the discretization error. The knowledge of the error in turn allows to adapt the solution by local mesh refinement to a prescribed relative tolerance, and it allows even to determine for each node the optimal consistency order.

It is evident that such a highly sophisticated algorithm cannot be invented from scratch. We needed more than a decade to develop all the building blocks that are finally used in FDEM. The basic ideas were developed in the SLDGL (Selbstadaptive Lösung von Differentialgleichungen) program package [1] that consisted of several subpackages for different types of elliptic and parabolic PDEs (1-D, 2-D, 3-D, fixed or variable grid etc.). This was for scalar computers. Then came the vector computers and we made a complete redesign, the FIDISOL program package [2] to vectorize the code efficiently, see Chapter 17 in [2]. Up to here the PDEs were solved on a rectangular domain with rectangular grid. So the geometrical flexibility was still missing. To improve the geometrical flexibility we developed the CADSOL (Cartesian Arbitrary Solver) program package [3]. Here we have arbitrary geometrical domain, but with a body-oriented grid. We developed the algorithm to generate difference and error formulas of arbitrary consistency order on an arbitrary set of 2-D or 3-D points and we also developed the concept of dividing lines to treat coupled domains with different PDEs. For a body-oriented grid each node knows its neighbors from the node indices. However, the usually hand-generated body-oriented grid is still a severe restriction. What we needed is such a method on an unstructured FEM mesh.

This final goal was attained in the FDEM program package: The structure of the 2-D or 3-D space is given by a corresponding FEM mesh that gives us full geometrical flexibility. The neighborhood relation between the nodes is given by the element list, we have dividing lines with matching grid and sliding dividing lines with non-matching grid to separate subdomains with different PDEs. The most challenging problem, however, was the efficient parallelization of such a complicated algorithm on distributed memory parallel computers. Here we make ample use of the basic principle of the separation of the selection and of the processing of the data to save communication [4]. A basic paper on FDEM is [5], a progress report is [6].

This chapter is organized as follows: In Section 2.2 the generation of difference and error formulas

is presented. Section 2.3 tells how we estimate the discretization error that is used in Section 2.4 to estimate the total error of the solution. In Section 2.5 the selfadaptation is discussed. Section 2.6 presents how different subdomains with different PDEs are treated by dividing lines. In Section 2.7 the extension to 3-D is discussed, Secion 2.8 presents the parallelization for distributed memory parallel computers. In Section 2.9 we make some remarks to our linear solver LINSOL. In Section 2.10 are presented some "academic" examples to demonstrate the properties of FDEM.

2.2 The generation of difference and error formulas of arbitrary consistency order for space and time

FDM means that the solution u is in each node locally approximated by a polynomial. Here we discuss at first the approximation in the space coordinates x, y, z. For the sake of simplicity we explain the procedure for 2-D, i.e. for x, y, and we discuss the extension to 3-D below. If we want a FDM of consistency order q we select for the approximation a polynomial of order q which means that we get the exact solution if the solution itself is a polynomial of order q. This is used for the test of the FDM, see later section. The 2-D polynomial of order q is

$$P_q(x,y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + \cdots + a_{m-1} y^q. \tag{2.2.1}$$

This polynomial has m coefficients a_i, where

$$m = (q+1)(q+2)/2. \tag{2.2.2}$$

For the determination of the m coefficients a_0 to a_{m-1} we need m nodes with coordinates (x_0, y_0) to (x_{m-1}, y_{m-1}) . For $q = 2$ we need $m = 6$ nodes, see Fig. 2.2.1.

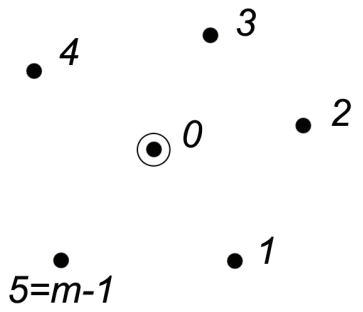


Figure 2.2.1: Example of $m = 6$ nodes for a polynomial of order $q = 2$.

If we would directly determine a polynomial that interpolates the 6 function values u_i of the 6 nodes, the function values would be "hidden" in the polynomial. However, for the FDM we need difference formulas where the function values u_i (variables) appear explicitly. Therefore we determine m "influence polynomials" $P_{q,i}$ of order q. The influence polynomial $P_{q,i}$ has function value 1 in node i and 0 in the other $m-1$ nodes:

$$P_{q,i} = \begin{cases} 1 & \text{in node } i, \\ 0 & \text{in other nodes.} \end{cases} \tag{2.2.3}$$

To determine the coefficients a_k of the influence polynomial $P_{q,i}$ we put into (2.2.1) the coordinates (x_j, y_j) of the m nodes and for $j = i$ the value of $P_{q,i}$ is 1 and 0 for $j \neq i$. This gives m systems of equations with the same matrix M and the m columns of the unit matrix I as r.h.s. These m systems can be expressed in the following form:

$$\begin{array}{rrcl} & & i = 0 \cdots & m-1 \\ equ:\ j = 0 \ : & 1 \cdot a_{0,i} + x_0 a_{1,i} + \cdots + y_0^q a_{m-1,i} = & 1 & 0 \\ & 1 & 0 & 0 \\ & . & . & . \\ & . & . & . \\ j = m-1: & 1 \cdot a_{0,i} + x_{m-1,i} a_{1,i} + \cdots + y_{m-1}^q a_{m-1,i} = & 0 & 1. \end{array} \tag{2.2.4}$$

The solution of each system are the coefficients $a_{0,i}$ to $a_{m-1,i}$ of the i^{th} influence polynomial $P_{q,i}$. If we denote by M the coefficient matrix of the systems (2.2.4)

$$\mathbf{M} = \begin{pmatrix} 1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 & \dots & y_0^q \\ 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \dots & y_1^q \\ \vdots & & & & & & & \vdots \\ 1 & x_{m-1} & & & & & & y_{m-1}^q \end{pmatrix} \tag{2.2.5}$$

and by A the matrix whose column i are the coefficients of the i^{th} influence polynomial, i.e. the solution of the i^{th} system (2.2.4), we can write (2.2.4) and its solution

$$M \cdot A = I,\ A = M^{-1} \tag{2.2.6}$$

which means that the coefficients of the i^{th} influence polynomial are the i^{th} column of the inverse M^{-1} of the coefficient matrix M (2.2.5), which is an interesting interpretation of the inverse in our case.

With the influence polynomials (2.2.3) we can define the interpolated function u which we denote by u_d (the index d means "discretized")

$$u_d(x, y) := \sum_{i=0}^{m-1} u_i P_{q,i}(x, y). \tag{2.2.7}$$

This interpolates with a polynomial of order q the u_i in the m nodes. If we need later for the sliding dividing lines the function value u in some arbitrary point in the space, we can interpolate it or as a "variable" express it by the neighboring function values using (2.2.7).
If we want difference formulas, we must differentiate (2.2.7) correspondingly. E.g. if we want a difference formula for u_x which we denote by $u_{x,d}$, we use

$$u_{x,d} := \frac{\partial u_d}{\partial x} = \sum_{i=0}^{m-1} u_i \frac{\partial P_{q,i}(x,y)}{\partial x}. \tag{2.2.8}$$

Similarly, if we want $u_{xx,d}$ we use $\partial^2 u_d/\partial x^2$ or for $u_{xy,d}$ we use $\partial^2 u_d/\partial x \partial y$.

Before we can use formulas of type (2.2.7) or (2.2.8) we must decide where we want to use them. In the case of Fig. 2.2.1 we want to use them for the node 0. Then we have to evaluate the $P_{q,i}(x,y)$ in (2.2.7) or the $\partial P_{q,i}(x,y)/\partial x$ in (2.2.8) for $x = x_0, y = y_0$. Then these expressions become mere numbers α_i or β_i and we have from (2.2.7) or (2.2.8)

$$u_d(x_0,y_0) := \sum_{i=0}^{m-1} \alpha_i u_i, \tag{2.2.9}$$

$$u_{x,d}(x_0,y_0) := \sum_{i=0}^{m-1} \beta_i u_i. \tag{2.2.10}$$

For each node we store the coefficients of the interpolation formula and of all derivative formulas with respect to x, y, xx, yy and xy, which are in 2-D 6 formulas with $m = (q+1)(q+2)/2$ coefficients, each. Below we will see that we also must store corresponding coefficients for the error formulas.

The next problem is: How to select m "good" nodes around the evaluation node where we want the formulas, e.g. node 0 in Fig. 2.1.1. The surrounding nodes should be as close as possible around the evaluation node because nodes that are far away increase the bandwidth of the resulting large and sparse matrix for the solution of our PDE, and they may introduce "false" (non-local) information if the function values change rapidly. This latter point is the reason why on a coarse mesh high order q is "overdrawn", i.e. gives larger errors than lower order.

The FEM mesh is usually generated by a mesh generator, e.g. PATRAN, I-DEAS. In 2-D we use linear triangles that are determined by 3 nodes, see Fig. 2.2.2. The element list which we call nek list

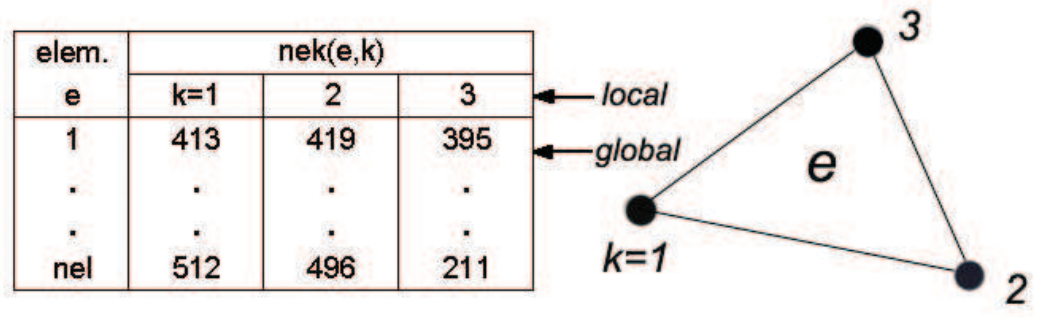


Figure 2.2.2: Illustration of triangular element and corresponding element list (nek list).

gives for each element the number of the global nodes for the 3 local nodes. Clearly there is also a node list where for each global node its coordinates x and y are stored. This is the basic information about the structure of the space. It should be recalled that we use a FDM on that FEM mesh. The FEM mesh is only used for the neighborhood relation between nodes.

It is easy to invert the nek list so that we get the nekinv list where for each node the information is stored in which element it occurs, see Fig. 2.2.3. An index counter vector with the length of the

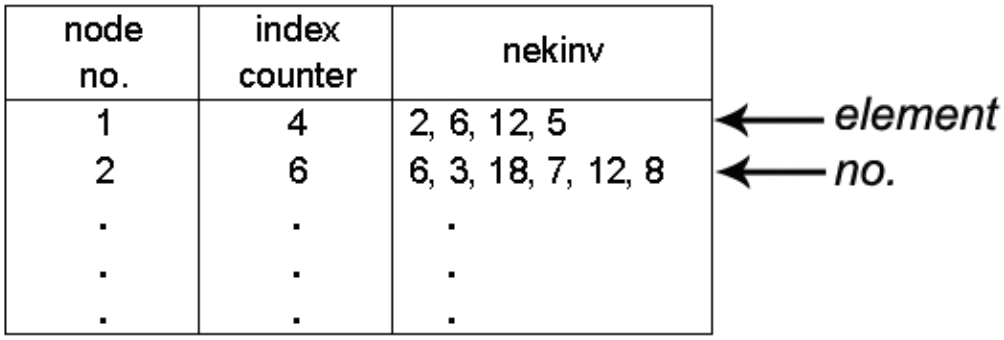


Figure 2.2.3: nekinv list that gives for each node the element numbers in which it occurs.

element list for each node is initialized by zeros. Then one goes through all node numbers of the nek list and for each node that is met in this processing the corresponding index counter of the node is increased by one. So one knows for each node in how many elements it occurs. Then an array nekinv is defined with length equal to the number of nodes and width equal to the maximal value in the index counter vector. Then one initializes again the index counter vector by zero and goes again through all nodes of the nek list. For each node that is met its index counter is increased by one and the element number is stored in the corresponding column position of the nekinv array.

The next problem is the determination of the nearest neighbor ring for each node, see Fig. 2.2.4. We want to create a list fstring (first ring) where for each node the node numbers of its direct neighbors on the mesh are stored, see Fig. 2.2.5. The nearest neighbor ring are all node numbers of all triangles in which the node occurs, see Fig. 2.2.4. The triangles (elements) are obtained from the nekinv list, Fig. 2.2.3. For each triangle we get the node numbers from the nek list, Fig. 2.2.2. For each node that occurs a “true” is entered in a logical list that extends over all nodes (initialized by “false”). Then for the central node also a “false” is entered. Now all nodes that have a “true” are entered in the fstring list, Fig. 2.2.5. Quite naturally in a first step the “width” of the fstring list is determined, then it is filled in a second step.

Now all the information about the structure of the space is stored in the fstring list for the nearest neighbor rings and the nek list and nekinv list could be deleted. If the “central” node is at or close to the boundary the nearest neighbor ring extends unsymmetrically into the interior of the domain. With the information of the fstring list it is now easy to search for nodes in further rings around the central node, see Fig. 2.2.4.

For the determination of the m coefficients of the m influence polynomials from the m linear

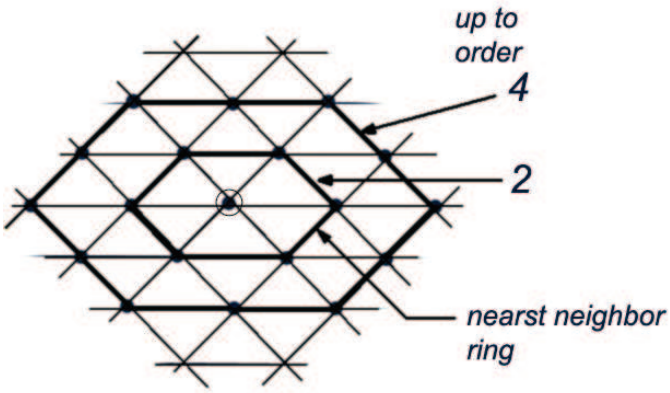


Figure 2.2.4: Illustration for nearest neighbor ring and ring search for nodes.

systems (2.2.4) we need for the computation of the $m \times m$ matrix M (2.2.5) m nodes. These nodes should be as close as possible to the central node because nodes far away introduce false (non-local) information. So we search for nodes in surrounding rings around the central node, see Fig. 2.2.4. Naively one would search only for m nodes. However, on straight lines the nodes are linearly dependent. Then the matrix M would be singular. Therefore we do not search for m nodes sufficient for the order q but for the order $q + \Delta q$. Usually we select $\Delta q = 4$ because, as we will see below, we also generate formulas of order $q + 2$ for the estimation of the error and we thus need additional nodes because of the linear dependencies. But there is still another request: If in Fig. 2.2.4 the "central" node would be at a boundary, we need for the order $q = 2$ at least 3 nodes in the

node no.	index counter nnr	fstring	
1	6	2, 5, 3, 8, 10, 12	← node
2	8	1, 3, 7, 6, 9, 5, 13, 11	← nr.
.			
.			
.			

Figure 2.2.5: fstring list that gives for each node the node numbers of the nearest neighbor ring on the mesh.

x-direction, i.e. 2 rings, and with the order $q + 2 = 4$ for the error estimate 4 rings, we therefore search for the nodes in up to $q + 2$ rings. Which limit is decisive, either the number of nodes m for $q + \Delta q$ or for the number of rings $q + 2$ depends on the situation of the central node in the mesh. We have now available $m + r$ nodes.

For the gathering of the nodes of the next ring (the first ring is the nearest neighbor ring) one creates a logical vector that extends over all nodes (for distributed memory parallelization over all local nodes, see later section). It is initialized by "false". Then one goes through all nodes of the previously selected ring and enters a "true" in the position of all nodes of the nearest neighbor rings of these nodes. So these nodes are "registered". Then one goes through all nodes of the two previous rings and enters a "false" in the corresponding position of the logical vector to exclude these nodes. The remaining positions with "true" give the numbers of the nodes of the next ring. This can be seen if one looks at Fig. 2.2.4.

Because we have selected more than the necessary number of m nodes there is the problem to select from the $m+r$ nodes the m best ones for the matrix M. The situation is depicted in Fig. 2.2.6. As mentioned above we want narrow formulas, i.e. with nodes close to the central node. However,

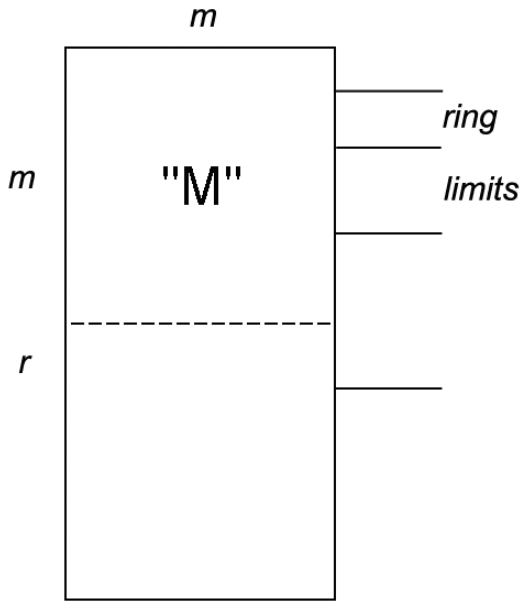


Figure 2.2.6: Illustration for the selection of m "good" equations from the m+r equations.

because of linear dependencies on straight lines there may not be information for the coefficient that should be computed by the actual row in the Gauss elimination process for the solution of (2.2.4). In such a situation the pivot (diagonal) element (by which must be divided) is zero or very small. Therefore we must use row pivoting: we search for the largest absolute value in the pivot column below the diagonal element and exchange the corresponding row and the old pivot row. However, the new pivot row may belong to a node that is far away from the central node. Therefore we arrange the nodes (equations) in the $(m + r) \times m$ matrix "M" (that contains the desired matrix M) in the

sequence of the rings, see Fig. 2.2.6. We prescribe a value ε_{pivot} and search for the pivot only in the equations of the actual ring. We allow a crossing of a ring limit only if

$$\mid pivot \mid < \varepsilon_{pivot}. \tag{2.2.11}$$

This procedure gives narrow difference and error formulas of high quality. The parameters Δq (usually $\Delta q = 4$) and ε_{pivot} are key parameters for the generation of the formulas. The value of ε_{pivot} may depend on the type of grid and of the order q. So by "playing" with the value of ε_{pivot} one may improve the formulas which can be seen directly by the better error estimate (see below). Rather robust values are $\varepsilon_{pivot} = 10^{-2}, 5 \cdot 10^{-3}, 10^{-3}$ for the orders $q = 2, 4, 6$. This algorithm has been developed and used for 2-D examples. However, it failed for certain 3-D examples. So we developed an alternative algorithm which will be described below. It should be mentioned that the inversion of M is executed by the Gauss-Jordan algorithm.

However, we want to mention at first two further points for the computation of the matrix "M" of Fig. 2.2.6 which is generated from $m + r$ nodes according to the prescription (2.2.5), i.e. a row i consists of $1, x_i, y_i, x_i^2, \cdots, y_i^q$. As we have a black-box solver we never know what are the values of the coordinates and thus x_i^q, y_i^q may have very large or very small values. Therefore we transform the set of nodes that has been selected, e.g. the nodes of Fig. 2.2.1 so that the central node is in the origin and the largest x- or y-coordinate is at $x = \pm 1$ and $y = \pm 1$, see Fig. 2.2.7. If a, b are the

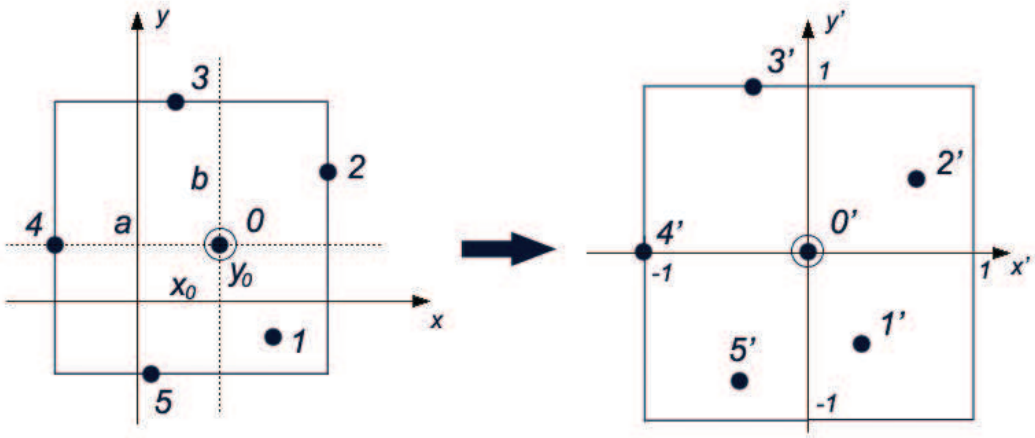


Figure 2.2.7: Illustration for the coordinate transformation $(x, y) \rightarrow (x', y')$.

maximal distances of a node in x, y-direction from the central node and x_0, y_0 are the coordinates of the central node, we have

$$\begin{aligned} x = ax' + x_0, \quad x' &= \frac{1}{a}(x - x_0), \\ y = by' + y_0, \quad y' &= \frac{1}{b}(y - y_0). \end{aligned} \tag{2.2.12}$$

We determine the influence polynomials in the transformed system, i.e. we form the matrix "M" with (2.2.5), but with x', y' from (2.2.12). The resulting coefficients are a_i, and the polynomial (2.2.1) now writes

$$P_q(x,y) = a'_0 + a'_1(\frac{1}{a}(x-x_0)) + a'_2(\frac{1}{b}(y-y_0)) + a'_3(\frac{1}{a}(x-x_0))^2 + \cdots \\ + a'_{m-1}(\frac{1}{b}(y-y_0))^q. \tag{2.2.13}$$

Because the central node in the transformed system is the origin, for the influence polynomial of the central node we have $a'_0 = 1$ and for the other influence polynomials we have $a'_0 = 0$.

The next problem is the normalization of the matrix "M". As we use a pivot threshold ε_{pivot} in (2.2.11) for the crossing of a ring limit, it is useful to normalize the linear system (2.2.4) that has now the matrix "M" of Fig. 2.2.6. Therefore we normalize to absolute row sum equal to 1, i.e. we divide each row by the sum of the absolute values of the elements of the row.

The extension to 3-D is straight forward. Here we use tetrahedrons for the structure of the space. A 3-D polynomial of order q is now

$$P_q(x,y,z) = a_0 + a_1x + a_2y + a_3z + a_4x^2 + a_5xy + a_6xz + a_7y^2 + \\ + a_8yz + a_9z^2 + a_{10}x^3 + \cdots + a_{m-1}z^q. \tag{2.2.14}$$

It has

$$m = (q+1)(q+2)(q+3)/6 \tag{2.2.15}$$

coefficients a_i that are basically determined as in 2-D. For the selection of the nodes for the matrix "M" , Fig. 2.2.6, we have now nearest neighbor balls, but for simplicity we will call them as in 2-D "rings". We select nodes that are sufficient for the order $q + \Delta q$ and we use at least $q + 2$ rings (balls), resulting in the $(m + r) \times m$ matrix "M".

Above we described the algorithm how to select from the $m + r$ rows of "M" m good ones with the criterion (2.2.11). This strategy that worked excellently in 2-D failed mostly in 3-D (in some examples it worked). So we generalized the condition (2.2.11) in the following way: during the solution process for the inversion of the matrix "M" of Fig. 2.2.6 we accept an element as pivot in the pivoting process, if

$$|\text{ element }| \geq \min(\varepsilon_{pivot}, \alpha \cdot \text{pivot}_{mean}, \text{pivot}_{max}). \tag{2.2.16}$$

Here ε_{pivot} is that of equ. (2.2.11), α is an appropriate parameter in the range 10^{-3} to 10^{-1}, $pivot_{mean}$ is the mean value of the absolute values of the pivot candidates of the pivot column, and $pivot_{max}$ is the maximal value.

However, if we go down in the pivot column to select the pivot element, it depends on the sequence of the nodes in the pivot column. Remember that the rows in the matrix "M" are created by corresponding nodes that have been gathered in the rings (2-D) or balls (3-D). For this sequence we have three types of strategies:
Arrangement:

1. according to single rings, so the nodes have been gathered,

2. combining two rings, e.g. ring 1 and 2 are treated like one ring, ring 3 and 4 as one ring etc.,

3. all gathered nodes are considered as one unit.

Sorting (in the corresponding arrangement):

a. for global node number,

b. for distance to central node,

c. at first sorting $a.$, then sorting $b.$

Note: the gathering on the parallel processors is made for the local node numbers, and for a regular grid there are several nodes with the same distance. If we do not apply $c.$, on a parallel computer the results may be different for different number of processors.

Search:

1. search linearly (independent of ring limits) until an element is found that fulfills (2.2.16),

2. search max. absol. element in the ring (for arrangement $1.$) or in double ring (for arrangement 2.), accept as pivot if (2.2.16) is fulfilled, else go to next ring/double ring.

In many "critical" examples the strategy "$2./c./1.$" for arrangement, sorting, search has been proved to be the best one. If we have a class of problem, e.g. the manufacturing of metal bellows, we can easily optimize the parameter α in (2.2.16) by looking at the error estimates. In this case $\alpha = 0.75 \cdot 10^{-1}$ was optimal. For the Bosch problem in axisymmetric cylindrical coordinates $\alpha = 0.01$ was the best value, but there ist a large good range.

We have learned in the solution process of the extreme technical problems with mesh ratios of $1 : 10000$ that the selection of the appropriate nodes, above all on curved grids, is a very critical and central point. The error estimate gives the insight in this selection process and shows the way to the best selection.

One should imagine what happens in 3-D: E.g. for a $100 \times 100 \times 100$ unstructured tetrahedral FEM mesh, generated by a mesh generator, for each of the one million nodes the nearest neighbor ring (ball) must be determined, and from that the necessary set of nodes for the $(m + r) \times m$ matrix "M". For each "M" the m appropriate nodes for the difference (and error) formula of arbitrary order q must be selected with the criterion (2.2.16). This must be done for interior nodes and boundary nodes. As FDEM is a black-box solver one never knows what mesh a user puts into the algorithm. Later we discuss the selfadaptation of the mesh and also of the order q. So the mesh and the difference formulas may be changing during the solution process. All these items necessitate an extremely robust algorithm. In such a situation it is mandatory to have an error estimate that tells us if our solution is reliable and how good it is.

Here are some remarks to "mesh-free" methods. We use in 2-D the triangular and in 3-D the tetrahedral FEM mesh only for the structure of the space, to determine the neighborhood relations between the nodes. If we have determined for each node its nearest neighbor ring (ball), we forget the FEM mesh. From this point on we have a "mesh-free" method that operates only on the nodes. So one could use instead of the FEM mesh an arbitrary set of points in the 2-D or 3-D space, with

the information which of the points are boundary points. Then one had to invent an algorithm to determine the nearest neighbor ring for each point. The simplest but most expensive algorithm is the search for the distance. But then there is the question how to distribute the nodes in the computational space. Such a distribution will be made efficiently by a triangular or tetrahedral grid that gives automatically the structure of the space. And thus we are back at our FDEM. Up to here we have

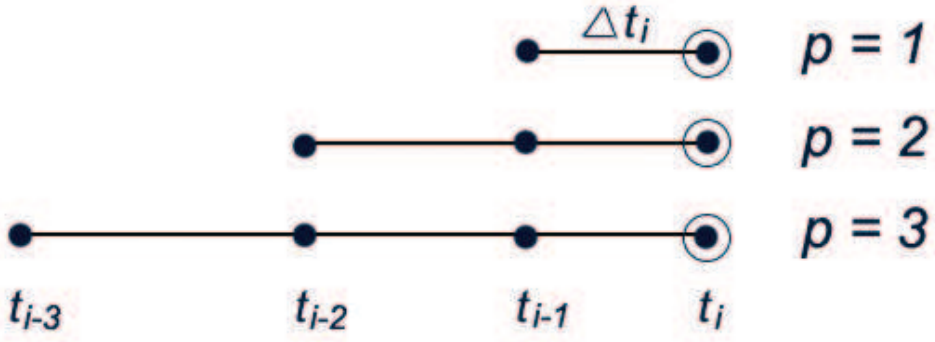


Figure 2.2.8: Illustration for the difference formulas of order p for u_t.

discussed the generation of 2-D or 3-D difference formulas for spatial direction. For parabolic PDEs we need 1-D formulas in time for the time derivative u_t. We use backward difference formulas of consistency order p that lead in FDEM to fully implicit methods for parabolic equations. Fig. 2.2.8 shows symbolically the formulas for the orders $p = 1$ to 3. For stability reasons we use the formulas only up to the order $p = 5$. The generation of such 1-D interpolation and difference formulas of type

$$P_p(t) = b_0 + b_1 t + b_2 t^2 + \cdots + b_p t^p \tag{2.2.17}$$

has been presented in detail in [1]. We use the Newton interpolation polynomial for the generation of the $p+1$ influence polynomials that are easily determined by Newton's scheme of divided differences. Basically they also could be determined by a 1-D version of our space method.

2.3 The estimate of the discretization error

In Section 2.2 we have explained how we can generate difference formulas of arbitrary consistency order q in space and order p in time. This can be used to estimate the discretization error. This is explained at first for the spatial formulas.

If we denote e.g. the difference formula for the derivative u_x by $u_{x,d}$ (index d means "discretized") or more precisely by $u_{x,d,q}$ which indicates that it is of consistency order q (exact for a polynomial of order q), we estimate the discretization error d_x by

$$d_x := u_{x,d,q+2} - u_{x,d,q}, \tag{2.3.1}$$

i.e. by the difference of the difference formulas of order $q+2$ and actual order q. The exact discretization error is

$$d_x := u_x - u_{x,d,q}. \tag{2.3.2}$$

So we have for the estimate replaced the unknown derivative u_x by a higher order formula $u_{x,d,q+2}$. Numerical investigations have shown that for the type of "central" formulas the odd orders are not better than the preceding even orders. Therefore we estimate the error of the actual order q by the difference to the order $q+2$.

But be careful: The estimate (2.3.1) assumes implicitly that the formula of order $q+2$ is a "better" formula, which means that it is "closer" to the derivative u_x. However, if we have a coarse grid relative to the changing of the solution and use a high order formula with many nodes (m nodes, for 2-D given by (2.2.2) and for 3-D by (2.2.15)), nodes that are far from the central node may introduce false information. This leads to the experience that on a coarse grid the higher order may give worse results than the lower order; the higher order is "overdrawn". Just this effect gives us a built-in self-control of the estimate (2.3.1). If we have large errors there is a large difference between order $q+2$ and q. If on the other hand there is a small error, we can trust our estimate. This was a very beneficial experience that we observed when we applied for the first time the estimate (2.3.1). It is trivial that the type of estimate holds for all other derivatives, e.g. u_{xx}, u_{xz} etc.

This effect of overdrawing an order leads us to the decision that we use for practical reasons only the orders $q = 2, 4, 6$. For the error estimate of the order $q = 8$ we needed the formula of order $q = 10$. Such a formula is usually (highly) overdrawn on practical meshes for technical problems. So we limited the order by $q = 6$ (error estimation by $q = 8$). The error estimate gives us also the possibility to check which order is the best one. We will see later that we use this property to select in a selfadaptive algorithm an own individual order for each node. Looking at the discretization error estimate allows us also to select optimal parameters for the selection of the nodes for the difference formulas, i.e. the parameter ε_{pivot} in (2.2.11) and α in (2.2.16). We still see below in the error equation how the mere discretization errors propagate into errors of the solution.

For the error estimate we do not use explicitly the derivatives of order $q+2$ and q as seen in (2.3.1) but we generate directly error formulas. If e.g. the formula for $u_{x,d,q+2}$ has coefficients a_i and for $u_{x,d,q}$ has coefficients b_i we have

$$\begin{aligned} d_x := a_0u_0 + a_1u_1 + \cdots + a_{m(q+2)-1}u_{m(q+2)-1} - \\ -(b_0u_0 + b_1u_1 + \cdots + b_{m(q)-1}u_{m(q)-1}). \end{aligned} \tag{2.3.3}$$

Here $m(q+2)$ (m of $q+2$) denotes m for 2-D from (2.2.2) and for 3-D from (2.2.15) where q has been replaced by $q+2$, and $m(q)$ is (2.2.2) or (2.2.15). Then we have

$$\begin{aligned} d_x := (a_0 - b_0)u_0 + (a_1 - b_1)u_1 + \cdots + (a_{m(q)-1} - b_{m(q)-1})u_{m(q)-1} + \\ +a_{m(q)}u_{m(q)} + \cdots + a_{m(q+2)-1}u_{m(q+2)-1}. \end{aligned} \tag{2.3.4}$$

The corresponding coefficients are directly stored as c_i so that we have for the evaluation

$$d_x := c_0u_0 + c_1u_1 + \cdots + c_{m(q+2)-1}u_{m(q+2)-1}. \tag{2.3.5}$$

As mentioned above we use for the time derivative u_t backward difference formulas of order $p = 1$ to 5, see Fig. 2.2.8. Here we estimate the discretization error by the difference to the next order $p+1$:

$$d_t := u_{t,d,p+1} - u_{t,d,p}. \tag{2.3.6}$$

For this estimate basically hold all the arguments that we have discussed for the spatial error estimate. We also store directly the error coefficients like in (2.3.5).

2.4 The black-box solver and the error equation

We want to solve arbitrary non-linear systems of 2-D or 3-D parabolic and elliptic PDEs under arbitrary non-linear boundary conditions (BCs) on an arbitrary unstructured mesh. The domain of solution may be composed of subdomains with non-matching grids and different PDEs and BCs. The user may prescribe a relative maximal error tol (tolerance) and in a selfadaptive process the mesh is refined and the order optimized to get the desired accuracy. Clearly, in 3-D such a process is always very expensive in computation and storage. It should be mentioned that in the equations (and quite naturally in the BCs) there may not be derivatives, so that we include algebro-differential systems. This is a very ambitious goal and we will show how we can solve this problem quite naturally and evidently by the FDEM.

The most general operator that we admit for the PDEs and BCs is in 3-D, with the unknown solution $u(t, x, y, z)$ has the form:

$$Pu \equiv P(t, x, y, z, u, u_t, u_x, u_y, u_z, u_{xx}, u_{yy}, u_{zz}, u_{xy}, u_{xz}, u_{yz}) = 0. \tag{2.4.1}$$

If we have a system of m PDEs the solution u and the operator Pu have m components:

$$u = \begin{pmatrix} u_1 \\ \dots \\ u_m \end{pmatrix}, \; Pu = \begin{pmatrix} P_1 u \\ \dots \\ P_m u \end{pmatrix}. \tag{2.4.2}$$

If t is included, the system must be parabolic, without t it must be elliptic. Basically it should also be possible to solve hyperbolic equations if they do not have discontinuities, but we do not have experiences in that area.

We explain the solution method for 2-D and discuss the extension to 3-D later. If we drop z in (2.4.1) we get the 2-D operator for PDEs and BCs:

$$Pu \equiv P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) \;\; = \;\; 0. \tag{2.4.3}$$

Because it is non-linear in u and its derivatives we linearize with the Newton-Raphson method by the approach

$$u \Leftarrow u^{(\nu+1)} \;\; = \;\; u^{(\nu)} + \Delta u^{(\nu)} \tag{2.4.4}$$

but we immediately drop the iteration index ν. We get a linear PDE or BC for the Newton correction function Δu (it is not yet discretized):

$$Q\Delta u \equiv -\frac{\partial Pu}{\partial u}\Delta u - \frac{\partial Pu}{\partial u_t}\Delta u_t - \frac{\partial Pu}{\partial u_x}\Delta u_x - \cdots - \frac{\partial Pu}{\partial u_{yy}}\Delta u_{yy} = \\ = P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{xy}, u_{yy}). \tag{2.4.5}$$

Here e.g. Δu_x is the x-derivative of the Newton correction function Δu. The r.h.s. $P(\cdots)$ is formally (2.4.3), but here we have dropped the Newton iteration index so that it is in reality $P(t, x, y, u^{(\nu)}, u_t^{(\nu)}, \cdots)$. This is the Newton residual or Newton defect. It is zero only if u is the solution, it is non-zero for $u^{(\nu)} \neq u$.

The $\partial Pu/\partial u$ are the Jacobi matrices. For a scalar PDE (one unknown variable) it is a scalar value. If we have a system of m PDEs, u and Pu have m components (2.4.2) and e.g.

$$\frac{\partial Pu}{\partial u_x} = \begin{pmatrix} \frac{\partial P_1 u}{\partial u_{1,x}} & \cdots & \frac{\partial P_1 u}{\partial u_{m,x}} \\ .. & \cdots & .. \\ \frac{\partial P_m u}{\partial u_{1,x}} & \cdots & \frac{\partial P_m u}{\partial u_{m,x}} \end{pmatrix} = \frac{\partial P_i u}{\partial u_{k,x}} \tag{2.4.6}$$

is a $m \times m$ matrix and likewise the other $\partial Pu/\partial u_{\cdots}$. They represent the dependencies of the operator Pu from the unknown function u and its derivatives and are introduced by the linearization process.

Now we discretize the linear Newton-PDE (2.4.5) using difference formulas of type (2.2.10). For the derivatives of Δu we do not use error estimates because these are errors of errors and thus small of second order. This discretization generates a large and sparse matrix Q_d. Fig. 2.4.1 illustrates how for a scalar PDE the term $\frac{\partial Pu}{\partial x}\Delta u_x$ contributes to row i of the matrix Q_d, where i denotes the central node of the formula for Δu_x. For a system of m PDEs there are $m \times m$ blocks instead of scalar elements.

The derivatives in the r.h.s. Pu of (2.4.5) are replaced by difference formula plus error estimate, e.g.

$$u_x \quad \Rightarrow \quad u_{x,d} + d_x \tag{2.4.7}$$

and we linearize in the error estimate terms which introduces again Jacobian matrices of type (2.4.6). These additional error terms on the "level of the equation", i.e. on the consistency level where we approximate a differential equation by a difference equation, create corresponding error terms on the "level of the solution". If we arrange all error terms on the l.h.s. we get the **error equation**

$$\begin{array}{c} level\ of\ solution \\ \Delta u_d = \Delta u_{Pu} + \Delta u_{D_t} + \Delta u_{D_x} + \Delta u_{D_y} + \Delta u_{D_{xy}} = \\ Q_d^{-1}\,[(Pu)_d + \ D_t + \{\underbrace{D_x \ + \ D_y \ + \ D_{xy}}_{space\ key\ error}\}]. \\ level\ of\ equation \end{array} \tag{2.4.8}$$

Q_d^{-1} is the inverse of the matrix Q_d of Fig. 2.4.1, but it is never computed explicitly (it would be a full matrix), $(Pu)_d$ is the "discretized" Newton residual, i.e. the r.h.s. Pu in (2.4.5), where discretized means that all derivatives have been replaced by difference formulas. The D_ν are discretization error terms, e.g.

$$D_t \;=\; \left(\frac{\partial Pu}{\partial u_t}\right)_d d_t, \quad D_x = \left(\frac{\partial Pu}{\partial u_x}\right)_d d_x + \left(\frac{\partial Pu}{\partial u_{xx}}\right)_d d_{xx}. \tag{2.4.9}$$

The terms in the brackets in (2.4.8) are error terms that can be computed on the level of the equation. The corresponding errors of the solution are arranged above these source terms. So on the level of the solution the total error Δu_d has been split up into its parts resulting from the terms in the brackets.

Δu_{Pu} is the Newton correction that results from the Newton residual $(Pu)_d$ and is computed from

$$Q_d \Delta u_{Pu} \;=\; (Pu)_d. \tag{2.4.10}$$

Here we see why we do not need Q_d^{-1} explicitly. The Newton correction Δu_{Pu} is the only error term that is in each Newton step applied to the solution according to (2.4.4). The other error terms in the first row of (2.4.8) are only used for the error control. If we would apply these terms we had (eventually) a "better" solution but no longer an error estimate.

D_x (2.4.9) shows transparently the contribution of x-discretization errors to the solution: The discretization error estimates d_x of $u_{x,d}$ and d_{xx} of $u_{xx,d}$ are multiplied with their Jacobian matrices, added and (formally) multiplied by Q_d^{-1} to transfer the error from the level of the equation to the

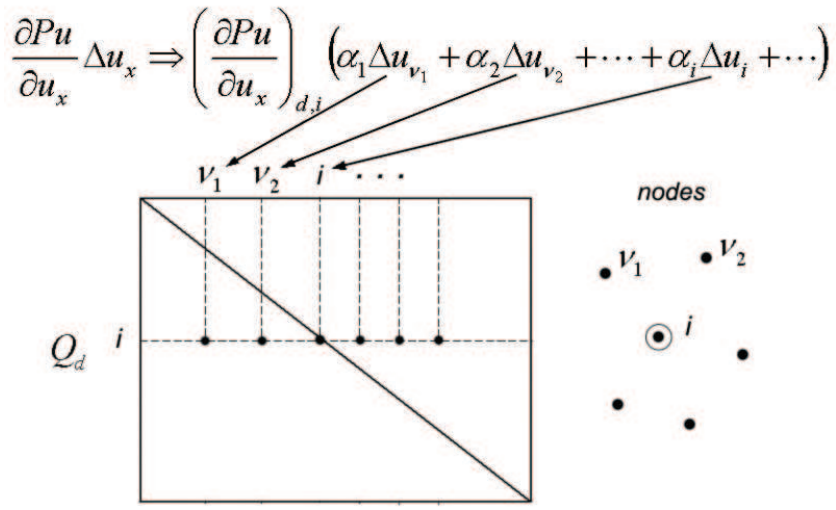


Figure 2.4.1: Illustration for the generation of row i of the matrix Q_d for a scalar PDE.

level of the solution. This possibility to follow explicitly the propagation of all errors is the essential advantage of FDEM.

In the error equation (2.4.8) the space errors in the braces $\{\}$ play the role of a key error. For given solution, grid and space order q this value is given in each node. In the sense of error balancing it does not make sense to have the other terms in the brackets, namely D_t (for parabolic equations) and $(Pu)_d$ larger or much smaller. If they are larger, they destroy the accuracy, if they are much smaller, they only waste computation time because the accuracy is then determined only by the space key error. So the error equation not only makes transparent all errors but tells us also how to balance the different errors for an efficient solution of the PDEs.

2.5 The selfadaptation for space and time

Let us assume we have a system of m PDEs and a grid with n_{nod} nodes. Then we have $n = m \cdot n_{nod}$ unknowns. If we want to discuss accuracy we must have a single number that characterizes the accuracy of the whole field of solution in the nodes. Therefore we introduce norms $\|\cdot\|$. Because for technical applications always the maximal values, e.g. maximal stress or velocity, are relevant, we introduce max norms. For the solution u_d (index d means "discretized") we use

$$\|u_d\| = \max_{\substack{i=1,m \\ k=1,n_{nod}}} |u_{d,i,k}|, \tag{2.5.1}$$

i.e. the maximum over all m components and all n_{nod} nodes.

For the errors we want to have relative error norms. However, for a black-box we never know if a solution component has a local zero value which makes a local relative error unfeasible. Therefore we use "global relative" errors. For a component i the global relative error is

$$\|\Delta u_d\|_{rel,i} = \frac{\max\limits_{k=1,n_{nod}} |\Delta u_{d,i,k}|}{\max\limits_{k=1,n_{nod}} |u_{d,i,k}|}, \tag{2.5.2}$$

i.e. we have the max error of the component i relative to the max of that component. If we have a zero or very small component in the whole domain, even this global relative error does not make sense. This allows to check individually the accuracy of the components. The global relative error is then

$$\|\Delta u_d\|_{rel} = \max_{i=1,m} \|\Delta u_d\|_{rel,i}, \tag{2.5.3}$$

i.e. the max of the global relative component errors.

The Newton correction Δu_{P_u} is computed from (2.4.10). In the sense of error balancing in (2.4.8) we stop the Newton iteration if

$$\|(P_u)_d\| < 0.1 \cdot max(0.5\ tolg, \|\{\}\|) \tag{2.5.4}$$

with $tolg$ from (2.5.13) (see explanation there) and where $\{\}$ denotes the space key error, see (2.4.8) and 0.1 is a "tuning factor". This means that we must compute the space key error $\{\}$ in each Newton step. However, as we need it nevertheless after the last Newton step for the computation of the error, this may not be too much additional computation time. The solution of (2.4.10) for the computation of a Newton correction is much more expensive.

Here we should mention that we have designed a very robust Newton algorithm: After each Newton step, before we accept a Newton correction, we check if

$$\|(P_u)_d^\nu\| < \|(P_u)_d^{\nu-1}\|, \tag{2.5.5}$$

i.e. we check if the Newton residual has decreased. If this does not hold, we use instead of (2.4.4) a damped Newton method by

$$u_d^{(\nu+1)} = u_d^\nu + \omega \Delta u_{P_u}^{(\nu)}, \tag{2.5.6}$$

where the relaxation factor ω initially is $\omega = 1$. If (2.5.5) does not hold, we repeatedly put $\omega \Leftarrow \omega/2$ and we try again until either (2.5.5) holds or $\omega < 0.01$. In the latter case we stop the Newton iteration and print out "Newton iteration does not converge". If $\omega < 1$ we have no longer the quadratic convergence of the Newton method. However, if we are close enough to the solution we must have $\omega = 1$ and quadratic convergence.

Another numerical engineering decision is to use the "simplified" Newton method, i.e. to iterate with the "old" matrix Q_d, if we have fast convergence, i.e. if

$$\|(P_u)_d^{(\nu+1)}\| < 0.1 \|(P_u)_d^{(\nu)}\| \tag{2.5.7}$$

holds. Then we do not compute a new matrix Q_d. If we use a direct solver for (2.4.10) we do not have to repeat the LU factorization in such a Newton step. However, then we cannot expect (fully) quadratic convergence.

Eventually the computation of the Newton correction Δu_{P_u} from (2.4.10) is executed by an iterative solution of the linear system. If ν denotes the index of the Newton iteration we want to solve

$$Q_d \Delta u_{P_u}^{(\nu)} = (P_u)_d^{(\nu-1)}. \tag{2.5.8}$$

If this system for the ν-th Newton correction $\Delta u_{P_u}^{(\nu)}$ is solved itself by an inner iteration with iteration index μ. we stop the inner iteration if the following condition holds:

$$\frac{\|Q_d \Delta u_{P_u}^{(\nu,\mu)} - (P_u)_d^{(\nu-1)}\|}{\|(P_u)_d^{\nu-1}\|} \leq \varepsilon^* = \tag{2.5.9}$$

$$= 0.1 max \left[\left(\frac{\|\Delta u_{P_u}^{(\nu-1)}\|}{\|u_d^{(\nu-1)}\|} \right)^2, \frac{0.8\|\{\}\|}{\|(P_u)_d^{(\nu-1)}\|}, \frac{0.4 tolg.}{\|(P_u)_d^{(\nu-1)}\|} \right].$$

We want to explain the three terms in the brackets. For the first term we assume that we want to solve iteratively a linear system $Ax = b$. In the k^{th} iteration step we have x_k and $r_k = Ax_k - b$. The error is $e_k = x - x_k$. Multiplication by A and adding/subtracting of b yields $Ae_k = Ax - Ax_k = Ax - b - Ax_k + b = -r_k$ because $Ax - b = 0$. In this notation the l.h.s. of (2.5.9) corresponds to

$$\frac{\|r_k\|}{\|b\|} = \frac{\| - Ae_k\|}{\|Ax\|} \approx \frac{\|A\|\|e_k\|}{\|A\|\|x\|} = \gamma \frac{\|e_k\|}{\|x\|}.$$

At the other hand for quadratic convergence of Newton's method we expect an error $\Delta x^{(\nu)} \approx (\Delta x^{(\nu-1)})^2$ or a relative error norm $(\|\Delta x^{(\nu-1)}\|/\|x^{(\nu-1)}\|)^2$. This means that the next Newton correction will change the digits in this region so that it does not make sense to compute the solution more accurately than to these digits. This explains the first term in the brackets. Term two in the brackets means in a similar way that it does not make sense to compute digits that are below the discretization error. Term three uses in the same way the tolerance $tolg$ on the level of the equation that is obtained from the user-prescribed relative tolerance tol as is explained below in equ. (2.5.13). The coefficients 0.1, 0.8, 0.4 are typical numerical engineering tuning factors.

So we can summarize the meaning of (2.5.9): It does not make sense to compute in the iterative solution of (2.5.7), i.e. in the iterative computation of a Newton correction $\Delta u_{p_u}^{(\nu)}$, more digits than are overwritten by the next Newton correction or are more accurate than the discretization error or the prescribed accuracy.

As we do not have a previous Newton correction for the first iteration we take

$$\varepsilon^* = 0.1 \quad \text{for} \quad \nu = 1 \tag{2.5.10}$$

and we restrict for practical reasons ε^* by

$$0.1 \geq \varepsilon^* \geq 10^{-4}. \tag{2.5.11}$$

Before we discuss the selfadaptation for time and space we need a scale of the accuracy on the level of the equation in the sense of the error equation (2.4.8). The user prescribes a global relative tolerance tol for the solution and requests

$$\|\Delta u_d\|_{rel} \leq tol \tag{2.5.12}$$

with $\|\Delta u_d\|_{rel}$ from (2.5.3). What is a corresponding value $tolg$ on the level of the equation that is needed for the control? The admissible error is $tol \cdot \|u_d\|$ because tol is a relative error. We know from the solution of (2.4.9) how the Newton residual $(P_u)_d$ is transformed from the level of the equation to the Newton correction Δu_{P_u} on the level of the solution. We assume that the same relation holds between the error $tol \cdot \|u_d\|$ (level of solution) and a corresponding value $tolg$ on the level of the equation and we make the numerical engineering approach

$$tolg := tol \cdot \|u_d\| \frac{\|(P_u)_d\|}{\|\Delta u_{P_d}\|}. \tag{2.5.13}$$

This value is used in the selfadaptation process.

Now we want to discuss the selection of the step size Δt and of the consistency order p in time direction for parabolic equations. Here we use backward difference formulas of the type of Fig. 2.2.8 and error estimates of type (2.2.3). At a time step t_k we want to compute a time increment Δt_{k+1} for the next time step that makes the size of the time discretization error term D_t (2.4.9) roughly 1/3 the size of the space key error term $\{\}$ in the error equation (2.4.8) in the sense of error balancing, or roughly 1/3 the size of $tolg$. If we have equidistant time step size Δt the time discretization error is $\sim (\Delta t)^p$. Therefore we request, if i denotes the i^{th} equation of a system of m equations,

$$\Delta t_{k+1} = \min_{i=1,m} \left[\frac{1}{3} max(\|\{\}_i\|, tolg) / \|D_{t,i}\| \right]_k^{1/p} \cdot \Delta t_k. \tag{2.5.14}$$

However, this is a prediction for the situation at time t_{k+1} that may fail. Therefore we check before we accept the new solution if

$$\|\Delta D_{t,i}\| < max(\|\{\}_i\|, tolg) \tag{2.5.15}$$

holds for all i and if not we drop the solution and compute from (2.5.14) a new Δt_{k+1}, now using in the r.h.s. the infomation of t_{k+1}. The user can prescribe limits Δt_{min}, Δt_{max} for Δt.

The control of the order p in the t-direction is made in the following way: The t-discretization error terms $\|D_{t,i}\|$ (2.4.9) for the m components i are computed for the actual order p and the neighboring orders $p\pm1$. If the error term of the actual order p is the smallest for all components i, the order p is optimal and is used for the next time step. If the error term for the order $p+1$ is the smallest one for <u>all</u> components i, i.e. the error decreases with increasing order, the order $p+1$ is used for the next time step. If on the other hand the error term for $p-1$ is the smallest one for only <u>one</u> component i, the order p is "overdrawn" and the order $p-1$ is used for the next step. As mentioned above the order p is limited to $1 \leq p \leq 5$ for stability reasons. If we took off the upper limit for tests, the method only occasionally selected $p = 6$ and returned quickly to $p \leq 5$.

Clearly the starting is with order $p = 1$. The user prescribes an initial value Δt_{init}. With this Δt_{init} 2 "blind" steps are executed. Then with the solution for t_0, t_1, t_2 the error for t_1 can be estimated. If the condition (2.5.15) does not hold, the solution is dropped and from (2.5.4) a new Δt is computed. This procedure is repeated until (2.5.15) is fulfilled. The algorithm for the optimization of the order p can start at t_3.

Here we want to discuss the optional possibility to compute a <u>*global*</u> error estimate in the time direction. At each time step t_k there is a time <u>*local*</u> error that can be computed from the error equation (2.4.8). However, be careful: How do we discretize Δu_t in the Newton differential equation (2.4.5)? If we look only at the errors at time t_k, the preceding values at time $t_{k-\nu}$ are considered to be "exact", i.e. we do not consider their errors. In this case we have in the formulas of types of Fig. 2.2.8 for the discretization of Δu_t in (2.4.5) only the term for t_k as nonzero. The Δu - values for preceding $t_{k-\nu}$ are considered to be zero. This gives from (2.4.8) the local error.

If we want to follow the <u>history</u> of the local errors in time, i.e. how they propagate in time, we must discretize Δu_t in (2.4.5) with all the previous error values Δu for $t_{k-\nu}$, i.e. in the formulas for Δu_t of type of Fig. 2.2.8 we have to consider the previous error profiles. These error profiles must be

stored like the solution for the preceding time steps. We always store 7 profiles because of the error estimate by the order $p = 6$ for the maximal actual order $p = 5$. This procedure then delivers the global error in time that gives the development of the discretization and linearization errors in time. The corresponding error equation is formally (2.4.8), but differs from that for the local error by the contribution of preceding errors in the Δu_t discretization.

These time discretization items are relevant only for parabolic equations. Now we return to the discussion of elliptic equations.

The next problem is the individual control of the space order separately for each node. This is a unique feature that becomes possible only by our simple and explicit estimate of the discretization error. Let us consider a node i and ask: What is its optimal (space) order q? Here we must at first mention that we consider for practical reasons only the orders $q = 2, 4, 6$. Numerical tests have shown that the odd orders do not give better results than the preceding even orders. So we consider only even orders. The error of the order $q = 6$ is estimated by the order $q = 8$. Practical experience has shown that for common mesh sizes the order $q = 8$ is mostly "overdrawn" so that the error estimate fails (built-in self-control). This has already been mentioned in Section 2.3. So we do not consider orders $q > 6$. Basically arbitrary orders could be used.

For the optimization of the order we compute at node i the space key error term $\{\}$ (see (2.4.8)) for the orders $q = 2, 4, 6$: $\|\{\}\|_{i,q=2,4,6}$. We take the higher order only if

$$\|\{\}\|_{i,higher\ order} \leq f \cdot \|\{\}\|_{i,lower\ order}, \tag{2.5.16}$$

where f is a numerical engineering tuning factor. Presently we take

$$f_{2\leftrightarrow 4} = 0.5,\ f_{4\leftrightarrow 6} = 0.01, \tag{2.5.17}$$

which means that we take the order 4 only if its error term is less than 0.5 that of the order 2 and we take order 6 only if its error term is less than 0.01 of that of the order 4. The reason is that higher order has more nodes in the difference formula which means that there are correspondingly more nonzeros in the large and sparse matrix Q_d, see Fig. 2.4.1, which makes the solution of the linear system (2.4.10) more costly. This holds more pronounced for the order 6 so that we accept this order only if its discretization error term is significantly smaller than that of the order 4.

This unique feature to have an individual optimal order for each node is rather expensive because we must store for each node the coefficients of the difference and error formulas for all 3 orders 2,4,6. However, it gives the most reliable error estimate because the method checks in each node if the order would be overdrawn which is visible by a larger error term for the higher order.

The next point is mesh refinement. The user prescribes a global relative tolerance tol and wants for the solution the accuracy requirement (2.5.12). If the initial grid does not fulfil (2.5.12) the only possibility is to refine the mesh where it is necessary. Because the control is made on the level of the equation (consistency level) we again need the value $tolg$ that is computed by (2.5.13) from tol. We check in each node i

$$\text{if } \|\{\}\|_i > s_{grid} \cdot tolg - \text{ then node } i \text{ is refinement node.} \tag{2.5.18}$$

We have again a tuning factor s_{grid} which depends largely on the type of problem to be solved so that no special value can be recommended. Eventually $s_{grid} = 10$ may be a starting guess. If the global relative error (2.5.3) $\|\Delta u_d\|_{rel} > tol$ and for the given value of s_{grid} no refinement node is found, s_{grid} is too large and we put $s_{grid} \Leftarrow 0.1 s_{grid}$, i.e. we reduce s_{grid} by a factor of 10. This procedure is eventually repeated until at least one refinement node is found. So we have ultimately a selfadaptation also for s_{grid} itself.

If in a triangle at least one node is a refinement node, the edges of the triangle are halved by 3 new nodes so that 4 similar triangles result, see Fig. 2.5.1. The value u_d at a new node is computed by a 2-D interpolation formula of order q (2.2.9), the order of the new node is the min of the order of its two neighbors. If in Fig. 2.5.1 the left neighbor triangle of the original triangle is not refined,

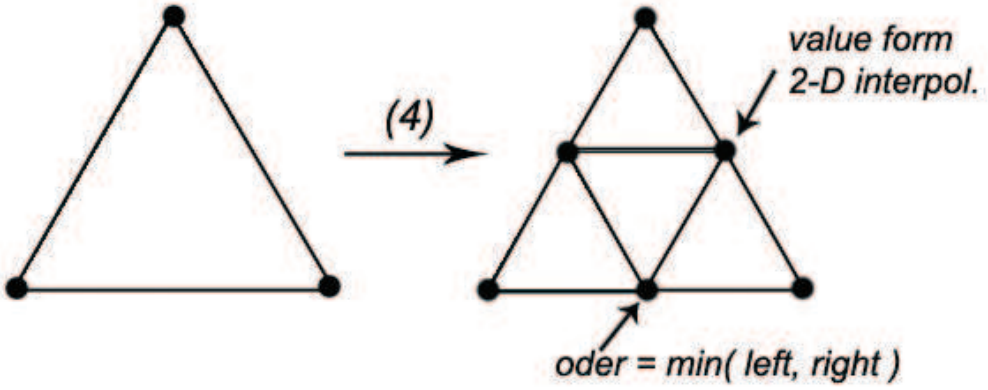


Figure 2.5.1: Illustration for the refinement of a triangle.

this triangle has 2 edges with 2 nodes and one edge with 3 nodes after the refinement step. If in a following refinement step an adjacent small triangle is again refined, there would be more than 3 nodes on an edge of the large triangle. Therefore we have limited for reasons of data organization the number of nodes on an edge to three. If more than three nodes on an edge would be created, the larger triangle must also be refined, but now not for reason of accuracy but for reason of data storage scheme. This induces a refinement cascade.

A sophisticated algorithm has been developed for the refinement cascade. For each triangle its refinement stage is stored: 0 means not refined, 1 means refined once etc. A logical list is created where for each triangle is noted in which refinement stage it must be refined. This list has the following shape:

element number	refinement stage	logical list for refinement stages			
		stage 0	1	2	3
1	1	false	true	false	false
2	0	true	false	false	false
3	3	false	false	false	true
4	2	false	false	(true)	false
⋮					

This means e.g. that element 1 is refined in stage 0 because of accuracy. Element 4 is refined because of the cascade. Before the refinement is made, it is checked if the triangles that must be refined have neighbors that are of one refinement stage lower, i.e. that are larger. Those triangles must also be refined that there are not more than three nodes on one edge.

The search for such neighbors is as follows: Investigate the three edges of a triangle. If there are 3 nodes on an edge no further search at this edge is needed because there is no larger neighbor. At other edges with the inverted nek list and logical lists for the triangles of the edge nodes the neighbor triangle is found and thus its refinement stage is known. This search must be executed after each stage of the refinement that starts with the largest elements and ends with the smallest ones. At the time of writing this report a doctoral thesis is in preparation that describes in all details the mesh refinement on a distributed memory parallel computer, see Remark 3 at the end of the References.

2.6 Dividing lines

In many technical applications the whole (global) solution results from the solution of coupled subdomains with (eventually) different PDEs. In Fig. 2.6.1 we have e.g. a block composed of 4 different

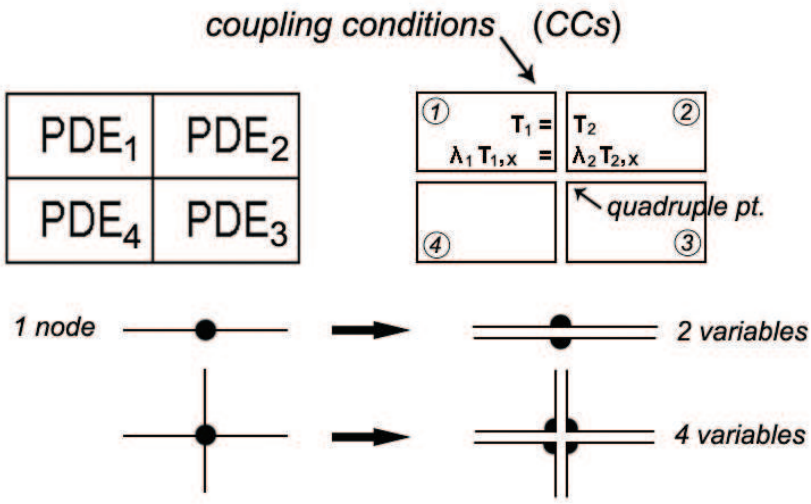


Figure 2.6.1: Illutration for dividing lines (DLs) and coupling conditions (CCs).

materials with different heat conduction coefficients. If we want to compute the heat flux in the whole block subjected to some boundary conditions, we can discretize the whole solution domain, but we cannot differentiate across a material boundary. Therefore we must compute the solution separately in the subdomains and couple the different solutions across the interfaces.

So from the geometrical configuration on the left side of Fig. 2.6.1 results the logical configuration

on the right side. For the separation of the subdomains we introduce “dividing lines” (DLs). From one geometrical node on an interface there result two logical nodes on the dividing line. Therefore at such a node we have two variables that each belong uniquely to one of the domains. For the two variables we need two coupling conditions (CCs), e.g. for a heat conduction problem we have equal temperature and heat flux which means $T_1 = T_2$, $\lambda_1 T_{1,x} = \lambda_2 T_{2,x}$ at the indicated DL in Fig. 2.6.1. At the crossing of two DLs we have a quadruple point with 4 variables and 4 CCs, see Fig. 2.6.1, at the intersection of 3 DLs we have 6 etc. If we have a system of m PDEs we have $2m$ or $4m$ or $6m$ CCs for the above mentioned cases.

As we cannot differentiate across a DL we use one-sided difference stars at the DLs that use function values of the corresponding subdomain. Thus the DLs are treated as “interior” boundaries. As a mesh generator delivers a configuration as shown on the left of Fig. 2.6.1, at the beginning of the solution process at first the new variables must be generated so that the logical configuration on the right of Fig. 2.6.1 results. Here we have a grid that goes straight through the whole domain, i.e. we have a matching grid on both sides of the DLs.

The solution algorithm then creates a global matrix Q_d for the whole domain that is composed from the PDEs in the interior, from the CCs at the interfaces of the subdomains and from the BCs at the exterior boundaries. Special care must be taken where a DL hits a boundary. There results a global solution with a global error estimate for the whole domain.

However, the situation in practical applications may be still more complicated: the different subdomains may have different grids and they may even slide relatively to each other, see Fig. 2.6.2. Here we can recognize that the lower boundary of the upper domain changes the property between

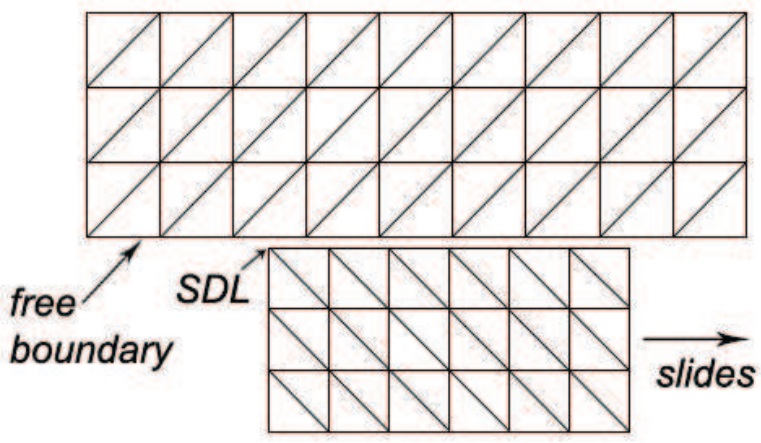


Figure 2.6.2: Illustration for sliding dividing line (SDL).

“free boundary” and “coupling boundary”. We call such an interface a sliding dividing line (SDL). It should be mentioned that the lower domain may not slide. Then we have a static non-matching grid that is also included in this algorithm.

The problem is now, how to couple the solutions of the subdomains across the SDL. For the DL

with matching grid we had always two (or more) coupled nodes that result from one geometrical node. The solution of the problem for the SDL is illustrated in Fig. 2.6.3. A geometrical node of

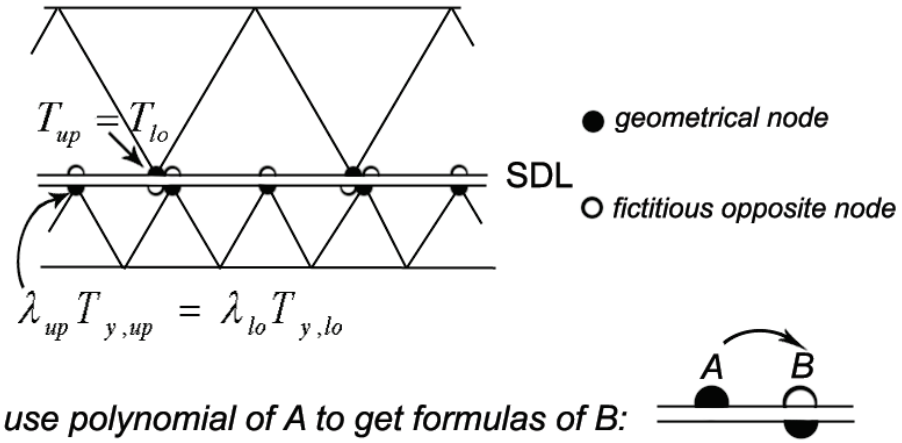


Figure 2.6.3: Illustration to the coupling across a sliding dividing line (SDL).

one grid generates automatically a fictitious opposite node on the other grid. However, for a FDM function values and derivatives are directly known only at geometrical nodes. What are the values at fictitious nodes, e.g. at node B in Fig. 2.6.3.? We search for a fictitious node B the nearest geometrical node of its grid, this is node A in Fig. 2.6.3. For node A we know the coefficients of the influence polynomials and these polynomials have been evaluated for x_A, y_A to get interpolation, difference and error formulas at A. We store for SDL nodes not only the coefficients of these formulas but also the cofficients of the influence polynomials, i.e. the inverse $A = M^{-1}$ (2.2.6). The influence polynomials are now evaluated at node $B = (x_B, y_B)$ and we get the coefficients of an interpolation formula (2.2.7) or a difference formula of type (2.2.8) for the fictitious nodes B. Thus we have difference and error formulas for the fictitious node B that are used in the coupling conditions between the geometrical node and its fictitious twin node like for the matching DL nodes.

For the DL we had e.g. in Fig. 2.6.1 for a heat condition problem the coupling conditions (CCs) equal temperature and equal heat flux that delivered the two equations for the two logical variables at the DL. However, for SDLs in Fig. 2.6.3 we have at a coupling node only one variable for the geometrical node. The formulas of the opposite fictitious node contain the variables of the formulas of the nearest opposite geometrical node, but no new variable. So we can impose at a geometrical node of a SDL only $\underline{one}$ CC. Therefore we prescribe in the example of a heat conduction problem in Fig. 2.6.3 for the geometrical nodes of the upper domain equal temperature $T_{up} = T_{l_0}$ and for the geometrical nodes of the lower domain equal heat flux $\lambda_{up} T_{y,up} = \lambda_{l_0} T_{y,l_0}$.

If we look at Fig. 2.6.2 we recognize that the lower boundary of the upper domain is partially free boundary and partially SDL that couples to the lower domain. A sophisticated algorithm has been developed to determine which node has which property. The details of the algorithm are not reported here.

It should be mentioned that for domains that are separated by SDLs the mesh refinement is made

independently for the different domains. This is possible because here we have non-matching grids.

2.7 Extension to 3-D

Up to now we have explained FDEM for 2-D. In the following we go again through sections 2.4 to 2.6 and discuss the extension to 3-D.

The general PDE and BC operator for 3-D is given in (2.4.1) and has been specialized to 2-D in (2.4.3). Corresponding to (2.4.5) the 3-D Newton PDE for the 3-D Newton correction function Δu now reads:

$$\begin{aligned} Q\Delta u \equiv & -\frac{\partial Pu}{\partial u}\Delta u - \frac{\partial Pu}{\partial u_t}\Delta u_t - \frac{\partial Pu}{\partial u_x}\Delta u_x - \cdots - \frac{\partial Pu}{\partial u_{yz}}\Delta u_{yz} \\ & = P(t, x, y, z, u, u_t, u_x, u_y, u_z, u_{xx}, u_{yy}, u_{zz}, u_{xy}, u_{xz}, u_{yz}). \end{aligned} \tag{2.7.1}$$

The discretization results like (2.4.8) now in the 3-D error equation

$$\begin{aligned} \Delta u_d = \Delta u_{Pu} + \Delta u_{D_t} + \Delta u_{D_x} + \Delta u_{D_y} + \Delta u_{D_z} + \Delta u_{D_{xy}} + \Delta u_{D_{xz}} + \Delta u_{D_{yz}} = \\ Q_d^{-1}\,[(Pu)_d + \;\; D_t + \{\; D_x \;+\; D_y \;+\; D_z \;+\; D_{xy} \;+\; D_{xz} \;+\; D_{yz} \;\}]. \\ \{\} = \text{space key error} \end{aligned} \tag{2.7.2}$$

The overall error Δu_d is now split into its 8 contributions on the level of the solution that result from the 8 contributions in the brackets on the level of the equation. The space key error $\{\ \}$ has now 6 terms.

In Section 2.5 the selfadaptation has been formulated that it holds basically also for 3-D. For the mesh refinement, Fig. 2.5.1, we halve the edges of a triangle if at least one of its nodes was a refinement node according to (2.5.18). Similarly in 3-D we halve the edges of a tetrahedron from which then result 8 "half tetrahedrons". Again the request that on an edge are at most 3 nodes leads to a refinement cascade where larger tetrahedrons must eventually be refined not because of a refinement node but for the sake of the request that comes from the rules of data organization. The sophisticated algorithm for the cascade that has been sketched for 2-D extends similarly to 3-D.

In Section 2.6 we introduced matching dividing lines (DLs) and non-matching sliding dividing lines (SDLs). In 3-D these are now dividing $\underline{surfaces}$. Nevertheless we will call them DLs or SDLs. For each geometrical node of a DL (that is now a surface) now result two logical nodes, each one belongs uniquely to one of the domains that are separated by the DL. At the intersections of several DLs similarly to Fig. 2.6.1 there may result more than two logical nodes from one geometrical node. It is difficult to show such a situation in a picture.

Still more complicated is the situation for SDLs similar to Fig. 2.6.2 and 2.6.3. Because the grids are non-matching, a node of one grid creates a fictitious node at the other side of the SDL and now one must search on the grid of that surface for the nearest geometrical node. If the two surfaces slide relatively to each other like depicted for 2-D in Fig. 2.6.2 one must determine which part of the surfaces is free boundary and which part is SDL. The corresponding algorithm is still far more complicated than in 2-D and is not explained here.

2.8 Parallelization

The numerical solution of PDEs, above all for 3-D problems, needs much computation and memory, often up to the available limits of the computer, or by other words: the user finally wanted a much larger computer than the available one. This need is the motor to drive the ever faster development of ever larger computers. Large computers are always parallel computers that combine the computation and memory of hundreds or thousands of processors. So there are the two main reasons for parallel computing: more computation and more memory.

The essential drawback of a parallel computer is the communication. The local data on a processor are accessible with the bandwidth and latency of the cache hierarchy or of the much slower memory. However, if remote data, i.e. data that is stored on other processors, must be accessed, the bandwidth and the latency of the communication network are essential parameters that determine the "usability" of a parallel computer. A cluster of workstations is also a parallel computer, but it suffers from bad communication parameters.

So the essential goal of parallelization is to minimize communication. In a certain amount communication and storage can be exchanged: If data are requested from another processor at the moment when they are needed, the processor has to wait until the data have arrived. If the data, e.g. information about the nodes of a mesh, is stored in p parts on p processors, communication must be executed if for the generation of difference and error formulas data of other processors are needed. If on the other hand in a preparatory step data that will be needed is stored also on the processors that will use it, i.e. the data are stored several times, this creates storage overhead, but avoids communication.

For an efficient parallelization one should always follow the priciple of the "separation of the selection and of the processing of the data", which means in this context: at first store the needed remote data on the own processor and then process the data, see [4], p. 136. The FDEM program package has been designed with this priciple in mind.

If FDEM should run on all types of parallel computers with shared and distributed memory, the only possibility is message passing. Therefore we use the quasi-standard MPI. Many examples have been published that demostrate that MPI is more efficient than the shared memory quasi-standard OpenMP, even if there is a shared memory or global address space.

In FDEM it is often necessary to rearrange the data over the processors. Here we make use of the "basket principle", Fig. 2.8.1: Each processor sends its (old) data in a "basket" in a ring shift through all processors and each processor takes out the data that it needs for the "new" data distribution. As the sends and receives of the MPI messages must fit together and it is often not known how many data must be exchanged between which processes, the processors exchange in a first step the information about the data to be exchanged and then in a second step exchange the data.

When the mesh data are read (usually from a file) they are distributed in p equal parts to the p processors in the order of the (global) node numbers. However, we want to have "neighboring" nodes, that are needed for the generation and evaluation of difference formulas, on the own processor to avoid communication. Therefore we sort the nodes for their x-coordinate. This is made by presorting of the nodes on each processor and then sorting over the processors where up to $p/2$ processors are active. This sorted sequence of the nodes is distributed in p equal parts to the processors which means a one-dimensional domain decomposition of the domain, see left part of Fig. 2.8.2 for $p = 4$ processors.

To avoid communication we store the needed data of the left and right processor or processors also

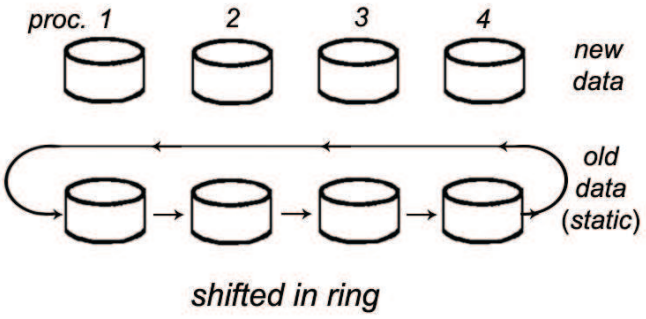


Figure 2.8.1: Illustration of the basket principle for $p = 4$ processors.

on the own processor. We call this overlap data. For the order q we need for the error formulas $q + 2$ rings, i.e. grid lines (2-D) or grid planes (3-D) to the left and right. We determine a "mean" edge length h of the elements and we store the nodes of the left and right processors

$$\begin{array}{lllll} \text{from} & x_{left} & - & a_{overlap} \cdot (q+2)h \\ \text{to} & x_{right} & + & a_{overlap} \cdot (q+2)h, \end{array} \tag{2.8.1}$$

where $a_{overlap} \geq 1$ is a safety factor. There is a list where for each processor is stored the information about the coordinates of its own leftmost and rightmost coordinate x_{left} and x_{right}. This list is stored on each processor so that it knows which data are stored on the other processors. The transfer of the overlap data is made in such a way that the whole data of the concerned left and right processor(s) is stored on the own processor and then superfluous data is eliminated.

For the elements we have the rule that an element belongs to the processor that holds its leftmost node on the x-axis, i.e. the node with smallest x-coordinate. After the distribution of the nodes the elements are distributed according to this rule. In Fig. 2.8.3 the triangle belongs to the processor that owns node 1. A similar rule holds for the edges which is important if an edge must be halved in the mesh refinement process. In Fig. 2.8.3 edges 1 and 3 belong to the processor that holds node 1, edge 2 belongs to the processor that holds node 2.

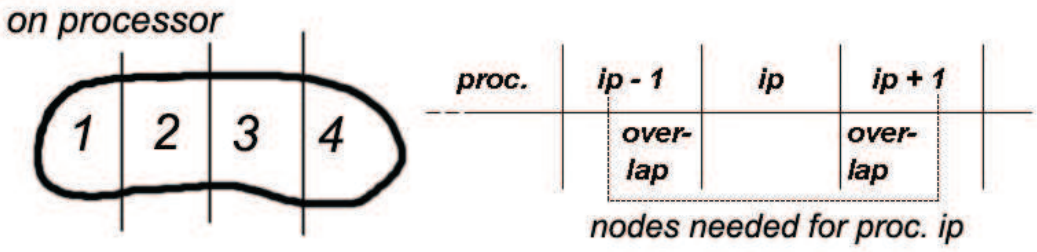


Figure 2.8.2: Illustration for the distribution of the data to the processors.

If there is mesh refinement, after each refinement step the nodes and elements are redistributed. this is exactly the same procedure as for the first distribution because new nodes are always added at the end of the old node list.

Now we must differ between global node numbers that include all nodes and local node numbers that include all nodes stored on a processor, including the overlap nodes. After the distribution of the nodes and elements to the processors these get a local node number and local element number on the processor. We have then on each processor a local nek list that gives for each element its local nodes and a local inverted nek list that gives for each node the local numbers of the elements to which it belongs. So we have the following node information:

- local node number
- global node number
- home processor
- domain number (all domains, separated by DLs or SDLs, but also all boundaries have a domain number because a boundary is treated as “domain”)
- number of coupling nodes to which the node couples (=1 for regular nodes, > 1 for DLs or SDLs, so for =1 it couples only to itself, for =2 it couples to one other node etc.)
- coefficients of difference and error formulas
- coordinates

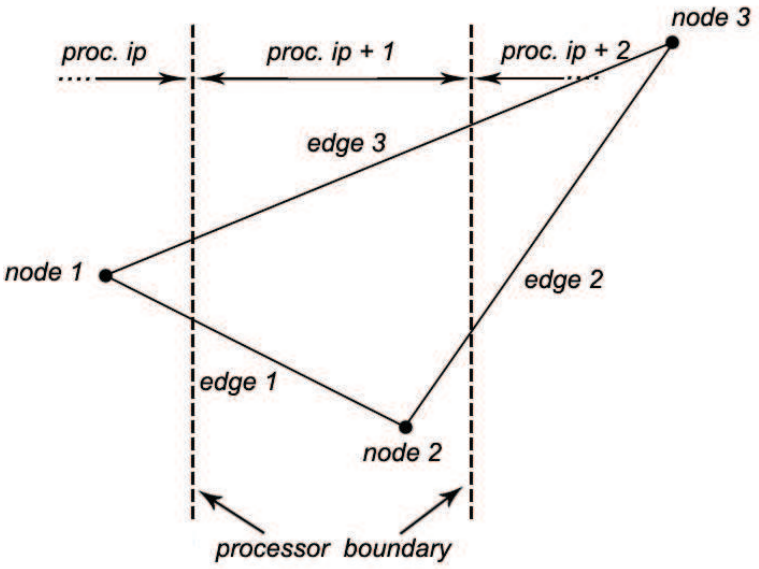


Figure 2.8.3: Illustration for the owning of triangles and edges.

- consistency order q
- solution (m values for a system of m PDEs)
- local element numbers of the elements to which the node belongs

element information

- local element number
- global element number
- home processor
- domain number
- local node numbers of the nodes that belong to the element
- element of a DL or SDL: yes/no
- refinement stage

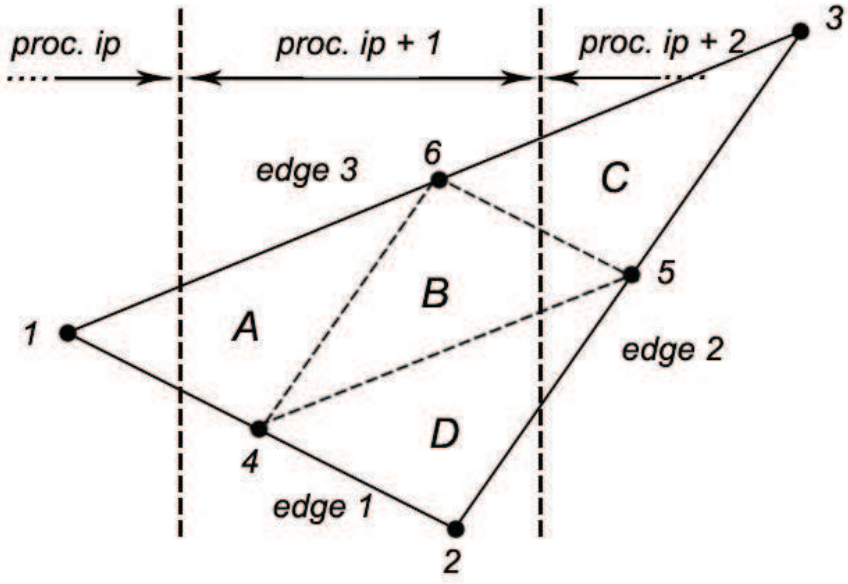


Figure 2.8.4: Illustration for the refinement of a triangle whose nodes are distributed to 3 processors.

An element may have in 2-D 3 to 6 nodes, depending on the refinement stage of its neighboring elements. If one of the nodes of the element of Fig. 2.8.3 is a refinement node, the element must be refined, see Fig. 2.8.4. The home of the original triangle is processor ip, there is its leftmost node. However, this triangle is surely also in the overlap of processor $ip + 1$ and $ip + 2$. The rule is

that only the home processor ip refines the triangle. Processor ip owns also the edges 1 and 3, but edge 2 belongs to processor $ip + 1$. The resulting smaller triangle A belongs to processor ip, but the triangles B, C, D belong to processor $ip + 1$. Similar ownings hold for the new edges. Besides the refinement by the accuracy request (2.5.18) there is also the refinement cascade (explained in Section 2.5) so that on an edge there are no more than 3 nodes because of the data organization.

Newly created nodes and triangles must get new global numbers that are behind the old numbers. However, that the processors can work in parallel they must know how many new nodes/triangles are created by the other processors so that they know what is the number range of their new numbers. Therefore in an initial step this information is created and exchanged between the processors. Then the refinement process is started from the largest to the smallest elements with continuous exchange of information between the processors, always with distinguishing between home data and overlap data. The illustrations 2.8.3 and 2.8.4 have been shown for 2-D. It is practically impossible to illustrate graphically the corresponding situation in 3-D, but the rules are the same.

It is quite obvious that the whole algorithm for the mesh refinement for distributed memory parallel computers is extremely complicated and we needed much more time than expected to develop this algorithm. It will be presented in a separate paper, see Remark 3 at the end of the References. After the refinement process the data (nodes, triangles) are again distributed newly onto the processors as shown in Fig. 2.8.2.

The solution process of the PDEs starts with the data distributed onto the processors where on each processor a local numbering over all own and overlap data is used. So each processor can compute its part of the matrix Q_d, Fig. 2.4.1 and the r.h.s. $(Pu)_d$ of equation (2.4.10) completely independent of the other processors without communication, see Fig. 2.8.5.

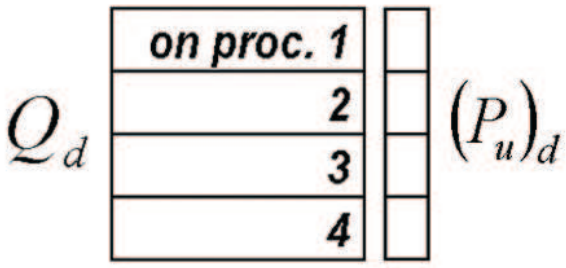


Figure 2.8.5: Illustration for the distribution of the data to the processors.

As each processor has all the necessary data in its memory with local numbering, it can compute for its own nodes (not for overlap nodes) the difference and error formulas, its part of the matrix Q_d and r.h.s. $(Pu)_d$ as if it would be a single processor and not a processor in a parallel computer. The key for this seemingly quite simple procedure is the overlap and the local numbering. It is clear that between Newton steps the values of u_d for the overlap data must be exchanged between the processors.

2.9 Remarks to the linear solver LINSOL

When a processor has computed its part of the matrix Q_d and r.h.s. $(Pu)_d$ of (2.4.10), see Fig. 2.8.5, it calls the parallelized linear solver LINSOL. LINSOL has been developed together with the previous PDE solvers FIDISOL [2], Chapter 17 and CADSOL [3] and has been enhanced continuously since that time. It is now publicly available, see [7] and the references given there. As the p processors call LINSOL independently they are automatically synchronized by the data.

Originally LINSOL was a purely iterative linear solver with different types of CG (conjugate gradient) methods. As there are CG methods with quite different properties, a polyalgorithm has been developed that switches over methods, see [2], Chapter 17. The polyalgorithm starts with a fast but less robust method (stagnates if the matrix is not sufficient diagonally dominant), switches to a medium method if the convergence is not satisfactory, and to a very robust but very slow method as emergency exit if the convergence is very slow.

Because many of our technical problems could not be solved efficiently by pure CG we developed a very sophisticated parallelized LU or ILU preconditioning [8] (this reference can be accessed via [7], documentation). For full LU preconditioning one has a direct solver with automatic post-correction. LINSOL has also several bandwidth optimizers [9]. For 3-D problems an efficient bandwidth optimizer is essential.

LINSOL has 8 basic data structures: diagonals full and packed, rows full and packed, columns full and packed, main diagonal and starry sky (double index). A matrix is composed from these basic "elements", an information vector gives the necessary information. These data structures are split up into row and column blocks to support an efficient parallelization of the matrix-vector multiplication [4] which is the kernel operation of all iterative solvers. For the (I)LU factorization we use a single row wrap-around over the processors with an active buffer window for efficient load balancing. The factorized parts L and U then are reorganized in packed columns and rows for an efficient forward elimination and backward substitution, following the principle of the separation of the selection and processing of the data.

2.10 Academic test examples

In this section we present "academic" test examples (in contrast to "real" hard technical examples in later chapters). We explain our test method for 2-D. The problem we want to solve, i.e. the PDEs and BCs, is abbreviated by

$$P_u \equiv P(x, y, u, u_x, u_y, u_{xx}, u_{yy}, u_{xy}) \quad = \quad 0. \tag{2.10.1}$$

For the test of our program, i.e. the test of the FDEM solver and of the PDEs and BCs that are programmed by the user, we use a PDE whose exact solution is known. This PDE should have as far as possible the properties of the original problem (2.10.1). For this purpose we prescribe the *test solution* $\overline{u}(x, y)$ and generate from (2.10.1) a problem that has the solution $\overline{u}$. This problem is the test PDE

$$Pu - P\overline{u} = 0. \tag{2.10.2}$$

$P\overline{u}$ is our problem with the known function $\overline{u}(x,y)$ instead of the unknown function u. $P\overline{u}$ is then a given function of x, y which is in (2.10.2) an absolute term (that contains no variables). Let us take for illustration the PDE

$$Pu \equiv u_{xx} + u_{yy} = 0. \tag{2.10.3}$$

We want to have the test solution

$$\overline{u}\,(x,y) = x^4 + x^2y^2 + y^4. \tag{2.10.4}$$

So we have

$$\begin{aligned} \overline{u}_x = 4x^3 + 2xy^2,\ \overline{u}_{xx} = 12x^2 + 2y^2, \\ \overline{u}_y = 2x^2y + 4y^3,\ \overline{u}_{yy} = 2x^2 + 12y^2. \end{aligned} \tag{2.10.5}$$

We get

$$\begin{aligned} P\overline{u} = \overline{u}_{xx} + \overline{u}_{yy} &= 12x^2 + 2y^2 + 2x^2 + 12y^2 \\ &= 14x^2 + 14y^2. \end{aligned} \tag{2.10.6}$$

Our test PDE is then

$$Pu - P\overline{u} \equiv u_{xx} + u_{yy} - (14x^2 + 14y^2) = 0. \tag{2.10.7}$$

So we see that $P\overline{u}$ is a pure forcing term that does not influence the part with the variables.
Quite naturally we must proceed similarly with the BCs. Let us assume we have Dirichlet BCs on the boundary

$$u - f(x,y) = 0, \tag{2.10.8}$$

then the test BCs are

$$\underbrace{u - f(x,y)}_{BC(u)} - \underbrace{(\overline{u}(x,y) - f(x,y))}_{BC(\overline{u})} = 0, \tag{2.10.9}$$

which formally gives

$$u - \overline{u}(x,y) = 0 \tag{2.10.10}$$

with $\overline{u}$ from (2.10.4).

The test problem (2.10.6), (2.10.10) has the desired solution $\overline{u}$ (2.10.4). The test solution $\overline{u}$ is a polynomial of order 4. If we use a solution method of consistency order $q = 4$ we must get the exact solution $\overline{u}$ and an error estimate in the range of the rounding error. If we use a solution method of consistency order $q = 2$ we get an error that should be well estimated. Theoretically we should

get an exact error estimate because the error is estimated by a polynomial of order 4. However, the polynomial of order 4 (which would be $\overline{u}$) cannot be reproduced exactly by the solution method of order $q = 2$. So far the illustration of the test PDE (2.10.2) by a simple example.

If we want to check if we have correctly programmed our PDE, we must program $P\overline{u}$ completely independently of Pu. Then we start our solution process for the test problem (2.10.2) with initial (starting) solution $u = \overline{u}$. Then the defect $Pu - P\overline{u}$ must be small, it is usually in the range of 10^{-11}. We then compare the estimated relative error $\|\Delta u_d\|_{rel}$ (2.5.3) to the exact error

$$\frac{\|\overline{u} - u_d\|}{\|u_d\|}. \tag{2.10.11}$$

Usually we choose as test funtion $\overline{u}$ a polynomial of order 2, 4, 6 and solve with consistency order $q = 2, 4, 6$. Then the relative error $\|\Delta u_d\|_{rel}$ and the exact error (2.10.11) are usually in the range of rounding errors. We will see examples below.

In the generation of the Newton correction Δu_{Pu} the Jacobian matrices enter as can be seen in Fig. 2.4.1 for $\partial Pu/\partial x$. They also enter into the computation of the error as can be seen in (2.4.9). As the generation of the formulas for the Jacobians and the programming of the formulas is a dangerous source of errors, we have developed a "Jacobi tester". E.g. for $\partial Pu/\partial x$ we compare the value of $\partial Pu/\partial x$ computed from the corresponding subroutine to a difference quotient

$$\frac{\Delta Pu}{\Delta u_x} = \frac{P(\cdots, u_x + \varepsilon, \cdots) - Pu}{\varepsilon} \tag{2.10.12}$$

and print out where there are unadmissible differences. This is a severe test for the Jacobians. Above this has been explained for a scalar PDE. If we have a system of m PDEs we test the $m \times m$ Jacobian matrices, e.g. (2.4.6) in the same way by components. It should be mentioned that the term $P\overline{u}$ in the test PDE (2.10.4) does not change the Jacobians because it is an explicit function of x, y (and z in 3-D). The Jacobi test gives the exact value of the derivative only if the variable occurs linearly in Pu. If it occurs in the form of $f(u)$ the derivative is accurate up to $\varepsilon \cdot f'(u)$. Nevertheless also in such a case the Jacobi tester reveals immediately an error, either caused by a false derivative or by a typing error in the code. According to our experience the Jacobian matrices are the main source of errors in the implementation of the PDEs.

Another important question for non-linear PDEs is: Does the Newton-Raphson method converge? Newton's method converges quadratically, but only if we are close to the solution. If we are far away with the starting solution, everything may happen. For this reason we have above introduced a damped Newton with a relaxation factor that controls if the Newton defect $(Pu)_d$ decreases in the Newton step. According to our experience the test PDE gives a good impression how the real problem will behave. Above all we have seen that for a bad selection of the nodes for the difference formulas Newton may diverge while it converges for a better selection. Therefore we start for the test of the Newton convergence by a disturbed initial solution. We solve the test PDE (2.10.2) with starting solution $1.01\ \overline{u}$ (1% disturbance) or with $1.1\ \overline{u}$ (10% disturbance) and observe the convergence. So we can see if we can solve our PDEs at least for the test solution. For some plasticity models Newton did not converge for more than 0.1% disturbance of the test function $\overline{u}$. Consequently we could not find a solution of the original problem because Newton diverged. So we know from our test PDE that the PDE system is correctly programmed and that there are solutions of this type of problem, but

Table 2.10.1: Results for the solution of (2.10.13), (2.10.14) on a circle with 751 nodes for different consistency orders q and test functions $\overline{u}$.

	order $q=2$		order $q=4$		order $q=6$	
type $\bar{u}$	error exact error estim.	CPU sec	error exact error estim.	CPU sec.	error exact error estim.	CPU sec.
pol. order 6	0.177 0.159	0.55	$0.904E-2$ $0.138E-1$	0.63	$0.300E-10$ $0.260E-7$	5.81
sugar- loaf	$0.439E-1$ $0.554E-1$	0.44	$0.144E-1$ $0.954E-2$	0.62	$0.229E-1$ 3.448 *)	4.66

*) here the order 8 for the error estimate is overdrawn (too coarse grid)

that the system is so sensitive to minimal disturbances that practical physical solutions are unusable, see the corresponding discussion in a later chapter.

If we have successfully solved our test PDE (2.10.2) we can take out of the code the terms of $P\overline{u}$ and we have regained our original problem that we wanted to solve. Practically we write the code for $P\overline{u}$ in separate lines that then are declared by a "!" in Fortran as comment and thus are ineffectual.

In the following we present some 2-D "academic" test examples. We want to solve the following system of 3 PDEs for velocity components u and v and vorticity ω:

$$\begin{aligned} u_{xx}+u_{yy}+\omega_y-f_1&=0,\\ v_{xx}+v_{yy}-\omega_x-f_2&=0,\\ u\omega_x+v\omega_y-(\omega_{xx}+\omega_{yy})/Re-f_3&=0. \end{aligned} \tag{2.10.13}$$

These are the Navier-Stokes equations in velocity/vorticity form for a 2-D viscous fluid. We set the Reynolds number $Re=1$. The f_i are forcing functions that are selected so that we get a prescribed solution $\overline{u}(x,y)$. So we ultimately solve a test PDE. The BCs are

$$u-g_1=0, \qquad v-g_2=0, \qquad \omega+u_y-v_x-g_3=0. \tag{2.10.14}$$

Here the g_i are again forcing functions that are determined so that we get the prescribed solution $\overline{u}(x,y)$. We prescribe

$$\overline{u}=\overline{v}=\overline{\omega}=\text{polynomial in } x,y \text{ of order 6} \tag{2.10.15}$$

or

$$\overline{u}=\overline{v}=\overline{\omega}=e^{-32(x^2+y^2)} \tag{2.10.16}$$

which is a sugar loaf type function, see Fig. 2.10.1.

We compute on an IBM SP WinterHawk2 with Power3-2 processors, 375 MHz. We use parallel computing with 8 processors. The CPU time in sec is for the master processor 1.

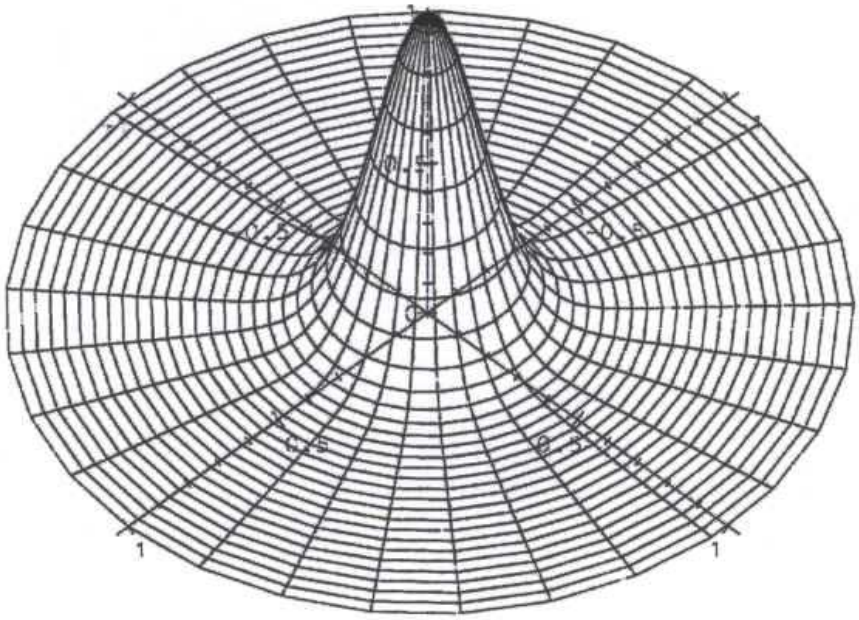


Figure 2.10.1: Sugar loaf type function equ. (2.10.16).

We solve the equations (2.10.13), (2.10.14) on a circle with radius=1 on a grid with 751 nodes, 1410 elements that has been generated by the commercial mesh generator I-DEAS. The results are shown in Table 2.10.1. If we use for $\overline{u}$, $\overline{v}$, $\overline{w}$ a polynomial of order 6 and use for the solution consistency order $q = 6$ we must reproduce the exact solution up to rounding errors. The exact global relative error (2.10.11) is $0.3 \cdot 10^{-10}$, the estimated global relative error (2.5.3) is $0.26 \cdot 10^{-7}$ which means that both are in the rounding error range. If we solve with orders $q = 4$ or 2 the errors increase and the CPU time decreases. If we take for $\overline{u}$, $\overline{v}$, $\overline{w}$ the sugar loaf test solution (2.10.16) the exact and estimated errors go down from the order $q = 2$ to the order $q = 4$, but they increase for the order $q = 6$: This is the built-in self-control of the error estimate. That the exact error goes up means that the order 6 is already "overdrawn": the used grid is too coarse for this order and change of function values on the grid, above all near the top of the sugar loaf Fig. 2.10.1. The error is estimated by the order $q = 8$ which is still more overdrawn and results in an error estimate of 3.45, i.e. 345%. This shows that and how the error estimate fails in this case. The most important property is that it shows that it fails. The user sees only the 3.45 error estimate and thus knows that he cannot trust the error estimate. He does not know if the solution is usable. The exact error is $0.23 \cdot 10^{-1}$, i.e. 2.3%, but he does not know it in the general case. Here we know it only because we know for test purposes the exact solution.

Table 2.10.2: Results of the selfadaptation of mesh and order for sugar-loaf test function for $tol = 0.25 \cdot 10^{-2}$.

cycle	no. of nodes	no. of elem.	no. of nodes ref.	no. of nodes with order			global relat. error		sec. for cycle
				2	4	6	exact	estimated	
1	751	1410	230	443	304	4	0.161E-1	0.108E-1	2.53
2	1623	3075	104	281	1336	6	0.613E-2	0.622E-2	6.03
3	2408	4398	—	237	2152	19	0.166E-2	0.169E-2	13.43

For the demonstration of selfadaptation we solve the same problem with the sugar-loaf test function on the same computer, but we now switch on the mesh refinement and order control and we request a relative tolerance $tol = 0.25 \cdot 10^{-2}$, i.e. 0.25% in the check (2.5.12). The result is shown in Table 2.10.2.

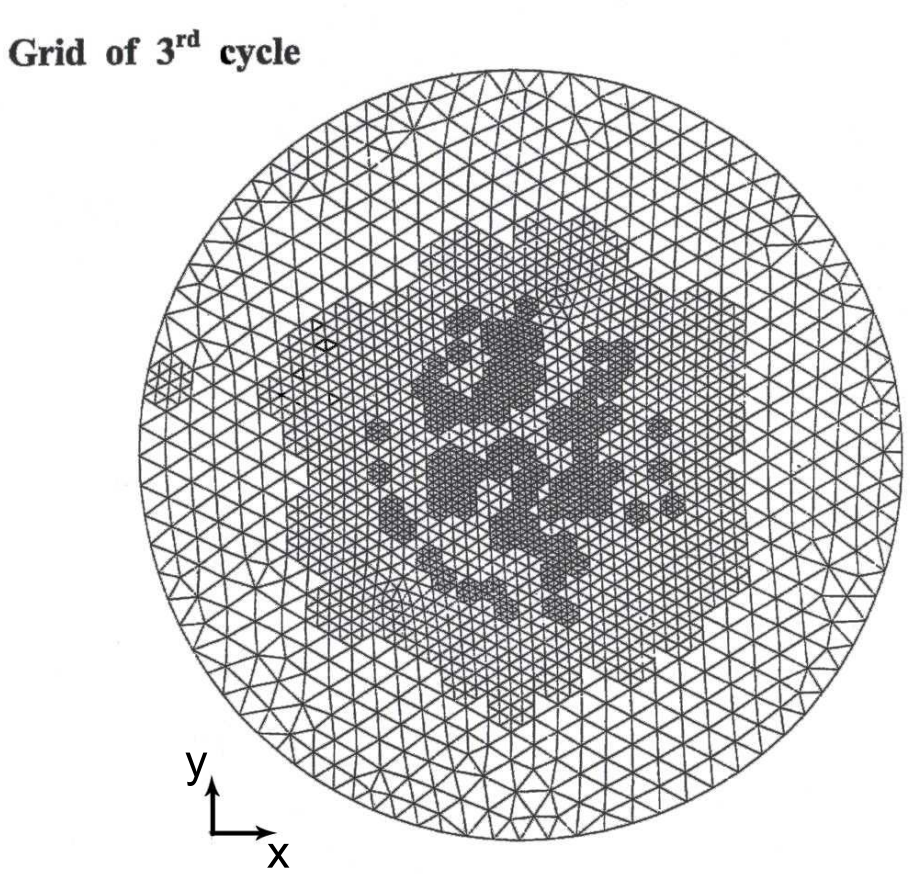


Figure 2.10.2: Refined grid after the 3^{rd} cycle of Table 2.10.2.

In cycle 1 we start with the 751 nodes, 1410 elements of the original grid that has been generated by the I-DEAS mesh generator. The condition (2.5.18) and the refinement cascade condition find 230 refinement nodes. From the 751 nodes 443 are solved with order $q = 2$, 304 with $q = 4$ and only 4 with $q = 6$. The exact and estimated errors are $0.16 \cdot 10^{-1}$ and $0.11 \cdot 10^{-1}$. Then between cycle 1 and cycle 2 the triangles that contain the 230 refinement nodes are refined by which there result totally 1623 nodes and 3075 elements that are then redistributed on the 8 processors according to the x-coordinate with the overlap as described above. In cycle 2 now 104 refinement nodes are found that lead for cycle 3 to 2048 nodes and 4398 elements. Then there are found no refinement nodes and the exact and estimated errors are below tol. Observe the accurate estimate of the error which results from the fact that the optimal order is selected by the order control. The CPU time for a cycle clearly increases with the increasing number of nodes.

Fig. 2.10.2 shows the grid after the 3^{rd} cycle. The coarsest grid is the 751 node grid, the medium sized grid is that of cycle 2 and the fine grid is that of cycle 3. The refined grid on the left boundary of the circle eventually results from a bad choice of the nodes for the difference formulas that resulted in a larger error estimate. In this sense the mesh refinement is also a control for the quality of the difference formulas. In order to see the influence of the order q and of the grid we made a series

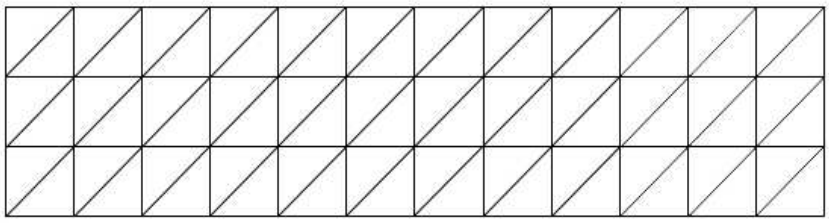


Figure 2.10.3: Type of grid for 4×1 domain.

computation with 5 grids and different orders for the solution of (2.10.13), (2.10.14) for the test function (2.10.16) (sugar loaf with top in the middle of the domain) on a 4×1 domain with the grid type of Fig. 2.10.3. The characteristics of the 5 grids are shown in Table 2.10.3. The number of grid points in x- and y-direction is doubled from one grid to the other which results in the 4-fold number of nodes and unknowns. We compute on the same IBM SP as above, but now for grids 1 to 4 with 16 processors and for grid 5 with 64 processors for storage reasons. The results are shown in Table 2.10.4. We are interested only in the exact and estimated errors and in the CPU time (of processor 1).

Table 2.10.3: Characteristics of the 5 grids.

grid♯	dimension	nodes	elements	unknowns
1	80×20	1600	3002	4800
2	160×40	6400	12402	19200
3	320×80	25600	50402	76800
4	640×160	102400	203202	307200
5	1280×320	409600	816002	1228800

Table 2.10.4: Results for solution of (2.10.13), (2.10.14) on the 4×1 domain with test function (2.10.16) for the 5 grids of Table 2.10.3.

grid #	no. proc.	order $q = 2$ error exact / error estim.	CPU sec	$q = 4$ error exact / error estim.	CPU sec	$q = 6$ error exact / error estim.	CPU sec
1	16	0.1024 0.291E-1	0.97	0.1429 0.4048	1.32	0.3383 0.5702	2.80
2	16	0.152E-1 0.533E-2	2.20	0.993E-2 0.122E-1	3.42	0.987E-3 0.195E-2	6.94
3	16	0.297E-2 0.126E-2	9.15	0.196E-2 0.183E-2	7.02	0.159E-5 0.187E-5	19.40
4	16	0.307E-3 0.302E-3	18.37	0.989E-5 0.101E-4	26.02	0.346E-7 0.432E-7	104.24
5	64	0.758E-4 0.749E-4	43.10	0.392E-6 0.388E-6	65.03	0.939E-9 0.991E-9	154.5

For grid 1 the smallest errors are for the order $q = 2$, i.e. the orders $q = 4$, 6 are overdrawn. For the other grids higher order gives smaller errors. If we go for one grid from order $q = 2$ to order $q = 4$ we see how the errors go down with increasing order which is the more pronounced the finer the grid. If we go down in a column for fixed order q we find approximately the error law that the error goes down like $(1/2)^q$. For the very small errors we are already in the rounding error range so that the error law does no longer hold.

Theoretically we should get for grid 4 and grid 5 the same CPU time because we have 4-fold number of processors. However, if the number of unknowns increases from 307 200 to 1 228 800 the condition number of the matrix Q_d increases which results in much more iterations for the iterative solver BiCGSTAB2 [10], see also [11], p. 139. Thus a problem with 4-fold number of unknowns is for an iterative solver a problem that needs much more than the 4-fold computation because with the number of unknowns the properties of the linear system become significantly worse.

The following is a 3-D example. We want to solve the system of 3 PDEs for the variables u, v, w which is again of Navier-Stokes type

$$\begin{aligned} u_{xx} + u_{yy} + u_{zz} + u + uu_x + vu_y + wu_z - f_1 = 0, \\ v_{xx} + v_{yy} + v_{zz} + v + uv_x + vv_y + wv_z - f_2 = 0, \\ w_{xx} + w_{yy} + w_{zz} + w + uw_x + vw_y + ww_z - f_3 = 0, \end{aligned} \tag{2.10.17}$$

with the BCs

$$u - g_1 = 0, \qquad v - g_2 = 0, \qquad w - g_3 = 0. \tag{2.10.18}$$

The f_i and g_i are forcing functions that are determined so that the exact solution is

$$\overline{u} = \overline{v} = \overline{w} = e^{-32(x^2+y^2+z^2)}. \tag{2.10.19}$$

We solve on a unit cube with 8 processors of the IBM SP as above. We use the iterative CG solver ATPRES that is very robust but very slow, see [7]. We start with a $10 \times 10 \times 10$ grid, compute with fixed order $q = 4$, switch on the mesh refinement. For the refinement we prescribe 3 cycles with total refinement, i.e. each node is a refinement node. In 3-D with this coarse initial grid local refinement caused difficulties. It resulted in a quite irregular grid so that the error increased instead of the expected decreasing. However, the total refinement preserved the regular (now finer) grid and decreased the error.

Table 2.10.5: Results for complete mesh refinement for the solution of (2.10.17), (2.10.18).

cycle	no. of nodes	no. of elements	no. of unknows	global relat. error		sec. for cycle
				exact	estimated	
1	1000	3645	3000	0.315E-2	0.417E-2	12.5
2	6130	29160	18390	0.401E-3	0.828E-3	66.2
3	43361	233280	130089	0.264E-4	0.951E-4	548.0

Table 2.10.5 shows the results for the 3 cycles. In 3-D we have tetrahedron elements. For total refinement from each tetrahedron result 8 tetrahedrons which gives the sequence $3645 \rightarrow 29160 \rightarrow 233280$ tetrahedrons which shows how fast the number of elements grows in 3-D for total refinement. We also can see that the error estimate in 3-D is worse than in 2-D. Here we estimate the error of the 3-D polynomials of order $q = 4$ by the error polynomials of order 6. Under these circumstances we find it really astonishing that the maximal difference between order 4 and 6 for 43361 local approximations is so small. This shows the robustness of our error estimate.

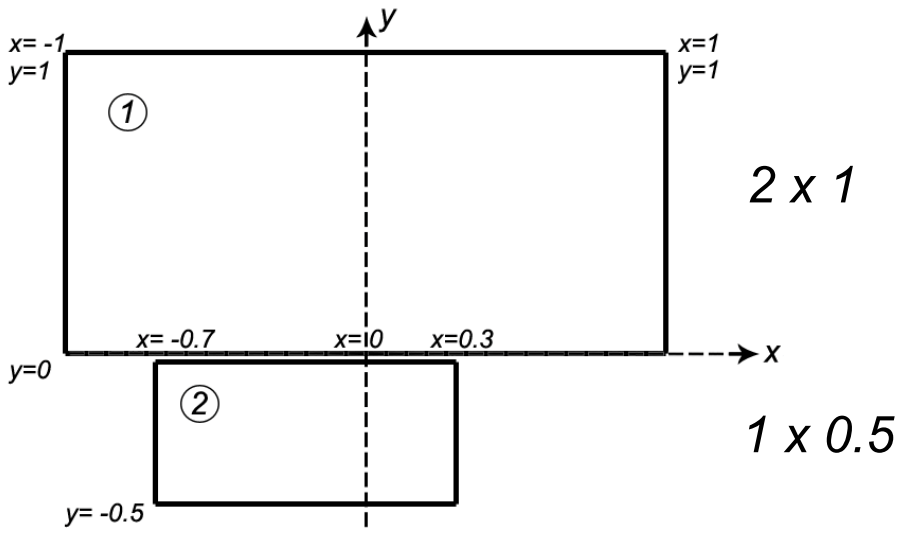


Figure 2.10.4: Domains for the example with SDL (non matching grids).

As mentioned above we used for the solution of the linear system for the computation of the Newton correction and error the generalized CG method ATPRES which is robust and slow. The kernel operation of CG-type methods is the matrix-vector multiplication (MVM). ATPRES needs 2 MVMs per iteration step. In cycle 3 of Table 2.10.5 we needed for the 3 Newton iterations 28+2091+1622 MVMs and for the computation of the error 1247 MVMs. This is the essential computation of the 548 sec total time (of processor 1).

The next of these "academic" examples is for a 2-D Sliding Dividing Line (SDL). Fig. 2.10.4 shows a 2×1 domain and a 1×0.5 domain and their (fixed) relative position to each other. We use on both domains a 40×20 grid or an 80×40 grid. So the smaller domain has half the mesh lengths of the larger domain, we have thus non-matching grids. We solve the problem (2.10.13) (2.10.14) from above. The test function is either (2.10.15) or (2.10.16) with top at $x = 0$, $y = 0$, but we add $+1$ in the larger domain ① and $+2$ in the smaller domain ②. The coupling conditions (CCs) at the SDL are the jump in the function values and equal derivatives. We compute on 8 processors of the IBM SP. The results are shown in Table 2.10.6.

Table 2.10.6: Results for the example with Sliding Dividing Line (SDL) of Fig. 2.10.4.

type $\bar{u}$	order $q = 2$		$q = 4$		$q = 6$	
	error exact error estim.	CPU sec (Newt.)	error exact error estim.	CPU sec (Newt.)	error exact error estim.	CPU sec (Newt.)
grid 40×20 for each domain (4800 unknowns)						
pol. ord.6	0.2164 0.1801	2.61 (6)	0.405E-2 0.503E-2	2.88 (4)	0.167E-11 0.714E-10	2.75 (1)
sugar- loaf	0.3121 0.1163	1.62 (4)	0.3044 0.1926	3.12 (6)	0.552E-1 0.612E-1	6.16 (5)
grid 80×40 for each domain (19200 unknowns)						
pol. ord.6	0.729E-1 0.631E-1	8.91 (5)	0.451E-3 0.732E-3	4.29 (1)	0.135E-11 0.917E-10	9.58 (1)
sugar- loaf	0.551E-1 0.159E-1	5.05 (3)	0.211E-1 0.109E-1	8.69 (4)	0.664E-2 0.437E-2	18.41 (4)

Because the coupling of the global solution for the two domains is made by the CCs at the SDL we need more Newton iterations than for a single domain. Therefore we give in Table 2.10.6 the number of Newton steps in parentheses below the CPU time. The starting solution is always the exact solution. If we took another starting solution, e.g. $u = 1$, we would need one or two more Newton steps. This starting solution explains why we need only 1 Newton step for test function as polynomial of order 6 and consistency order $q = 6$ because here the starting solution is the exact solution.

3 Applications

3.1 Introduction to Applications

Besides the evolution of the FDEM program package to moving subdomains with non-matching grid the application of FDEM to some industrial problems should be demonstrated as result of the research project. When we addressed some firms if they would be interested to participate in a cooperation, it became immediately clear that they were interested only if they had a problem for which there was no standard software on the commercial market available. This means that the selected problems are really difficult problems. The agreement was that the firms delivered the PDEs and BCs and we tried to solve them with FDEM.

The first problem was the simulation of the manufacturing process of metal bellows (Metallbälgen) from stainless steel sheet. However, the industrial partner IWKA could only deliver the problem description and not the PDEs. They were no specialists for the plasticity equations of stainless steel sheet. So we looked for a partner that could deliver the necessary PDEs and we addressed (by the advice of the metallurgists of the University of Karlsruhe) the Institute for Metal Forming Technology (IFU) of the University of Stuttgart. They promissed to deliver the necessary PDEs and to determine the corresponding empirical coefficients in the equations by tensile tests in their laboratory. For this purpose IWKA had to furnish the tensile test pieces. Unfortunately, the IFU used itself for their calculations commercial codes for metal forming processes and had themselves no practical experience with the PDEs that were used in these codes. All these codes were FEM codes where the PDEs are not explicitly used as we do it in the FDM. So a for us terrible learning process started: The IFU delivered a plasticity model, we programmed it and tested the code with the test methods described above so that we were sure that the PDEs were implemented correctly. Then we tried to solve the physical problem for the simulation of the tensile test to reproduce the measurements. However, either the numerical solution of the physical problem failed because the Newton iteration failed to converge, even for severe linearizations of the equations, or the solution was physically unrealistic. Then we got another plasticity model from the IFU and the play started anew. This for us extremely frustrating process lasted two and a half years until we finally, after the official end of the project funding, had the desired plasticity model that could simulate the measurements. Only then the programming of the manufacturing process could start. The details of this part of the project are presented in Section 3.2.

The second industrial project was delivered by the High Pressure Diesel Injection Pump branch of Bosch. It is a fluid-structure interaction problem. The injection pressure for Diesel is now up to $2000\ bar$. This means that under this force the piston and the housing of the pump are considerably deformed so that the lubrication and caulking gap that has a width of only a few micrometers is affected correspondingly. The difficulty comes from the different scales: The housing and length of the gap is in the range of centimeters, the diameter of the piston in the range of millimeters, but the width of the gap is in the range of micrometers. If we want to solve the PDEs correctly, which are for the piston and the housing the elasticity equations and for the fluid in the gap the incompressible Navier-Stokes equations, we must resolve the equations over the width of the gap. So we will have quite different grids for housing, piston and gap. For the present case the piston is fixed. In later applications the piston will be moving and also the heat conduction will be computed. These problems need the coupling of the solutions for housing, piston and gap and thus initiated the

evolution of FDEM for Sliding Dividing Lines (SDLs) that allow a global solution with global error estimate for the coupled domains of housing, piston and gap. The details of this part of the project are discussed in Section 3.3.

The third industrial project was from the area of fuel cells. The manufacturer Freudenberg produces non-woven materials that are used in PEM (proton exchange membrane) fuel cells. Originally, the intention was to simulate a whole fuel cell and to find out how the properties of the non-woven material that is used as a gas diffusion layer (GDL) influences the performance of the fuel cell. However, when Freudenberg Forschungsdienste, our partner, investigated the problem of compiling the PDEs for the whole cell they recognized that they were not able within the intended effort to formulate the equations at the catalytic membrane and to determine the necessary coefficients for the chemical reactions. Therefore they simplified the problem to GDL with catalytic production of water vapor at the membrane. The details of this part of the project are presented in Section 3.4.

Originally there was still a fourth industrial partner from the area of electric machinery. However, because of financial difficulties they closed the research lab with which we intended to cooperate soon after the start of the FDEM project.

3.2 Simulation of the manufacturing of metal bellows

As mentioned above we went together with the scientists of the IFU through an extremely frustrating learning process until we finally had a system of PDEs that could describe the plastic deformation of stainless steel sheet. We got plasticity models at 13.7.01, 15.12.01, 25.1.02, 3.4.02, 17.7.02, 14.8.02, 25.10.02 and finally 18.12.02 (this is the German writing of date: day, month, year). The last model was—as we believed—a correct complete description of the desired plasticity model. It is extremely non-linear. We programmed the model and tested the code with the test function. However, due to this non-linearity Newton's method did not converge if we started with a 0.1% disturbed solution. This was a bad warning for the solution of the physical problem that had as starting solution the solution of the last elastic step. As expected Newton's method did not converge. We then linearized the equations in time by taking appropriate function values from the previous time step which helped for the first plastic step but resulted in non-physical results for further time steps. Our conclusion is: We cannot decide from the point of view of the numerical solution if this plasticity model is a valid model. We only can state that it is an unusable model. Nevertheless we will describe in Section 3.2.2 the numerical solution method of this model. Before we can do this we must at first discuss in Section 3.2.1 the solution of the elastic equations that deliver the starting solution for the plastic equations and finally describe the spring-back solution at the end of the metal forming process.

Because the full "theoretical" plasticity model could not be used we tried, like FEM-users do, to solve also for the plastic region the elastic equations and to project the stresses that are then much too large down to the yield curve. This resulted in far too large stresses because the material was too hard. If the material yields it is soft. Then we got at 27.5.03 a proposal of the IFU for a "soft" elasticity module. We played with the empirical coefficients in this approach and could for the first time simulate the tensile test approximately. By the investigation between the simulation and the experiment we could recognize how the functional approach had to be modified to simulate exactly the physical experiment of the tensile test machine. So we had finally the desired plasticity model for the stainless steel sheet. This part of the research is described in Section 3.2.3.

Now we could attack with this plasticity model the simulation of the manufacturing of a single

wave of a metal bellow. A whole bellow is composed of a sequence of single waves, but the computation of a single wave determines the properties of the whole bellow. The manufacturing process consists of different phases: initial blowing with pure hydroforming, then combined forming by pressure and moving tool, and finally the spring-back if pressure and form are taken off. However, the numerical simulation revealed that the steel sheet bursts under the internal pressure and "explosion" into the form. Because of the inherent instability of the problem this resulted in (nearly) unsurmountable difficulties. This part of the research is presented in Section 3.2.4.

The equations that we want to solve are presented in the report of the IFU on their part of the compound project [12]. Quite naturally we have continuously to refer to equations given in that report. If we e.g. refer to equation (x.y) in [12] we denote it by [12](x.y).

3.2.1 The numerical solution of the elasticity equations for the tensile test

We want to simulate the tensile test in order to adapt the plasticity model to the measurements. However, the tensile test starts with elastic deformation. So we must at first solve the elasticity equations.

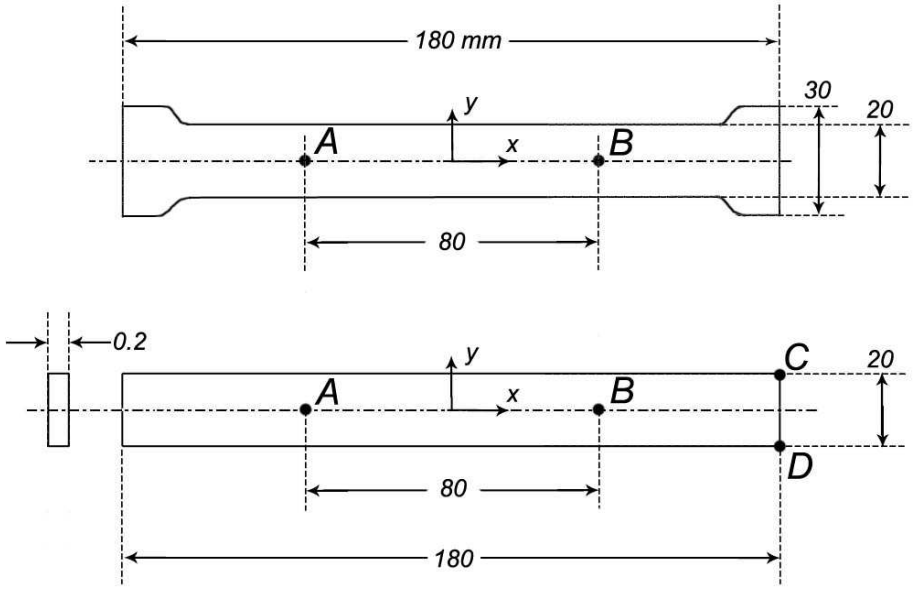


Figure 3.2.1.1: Tensile test piece (above) and simplification (below).

Fig. 3.2.1.1 shows in the upper part the test piece. It is clamped in the end regions. As we do not know how the force is entered into the test piece we simplify the configuration, see Fig. 3.2.1.1 lower part.

Because the thickness of the metal sheet is only $0.2\ mm$ we use a 2-D model in x, y. We get the elasticity equations for 2-D by equating the geometrical definition of strain and the definition of strain from Hooke's law. From [12](2.11) and [12](2.26) we get for ε_{xx}

$$\frac{1}{E_x}[\sigma_{xx} - \nu_{xy}\sigma_{yy}] = \frac{\partial u_x}{\partial x}. \tag{3.2.1.1}$$

In [12] is used an orthotropic Hooke material law (in contrast to the usual isotropic law) because by the rolling process of the thin steel sheet the properties of the material are different in the rolling direction and orthogonal to it. Here we assume that the rolling direction is the x-direction in Fig. 3.2.1.1. Thus E_x is Young's module in x-direction, similarly E_y in y-direction, and there is the shear module G_{xy}. ν_{xy} is Poisson's ratio. Values for the steel of the test piece are given in [12](2.81) to [12](2.83). The stresses are denoted by $\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$. The strains $\varepsilon_{..}$ are defined in [12](2.11) to [12](2.14) by the derivatives of the displacements u_x, u_y in x- and y-direction, and by the stresses in [12](2.26) to [12](2.31).

As we will see later, for plastic deformation not the displacements u_x, u_y but the displacement velocities v_x, v_y are decisive:

$$v_x = \frac{\partial u_x}{\partial t}, \qquad v_y = \frac{\partial u_y}{\partial t}. \tag{3.2.1.2}$$

With the displacement velocities we can define strain velocities. For small deformations they are given as the linear part of equs. [12](2.5) to [12](2.10), e.g.

$$\dot{\varepsilon}_{xx} = \frac{\partial v_x}{\partial x}, \qquad \dot{\varepsilon}_{yy} = \frac{\partial v_y}{\partial y}, \qquad \dot{\varepsilon}_{xy} = \frac{1}{2}(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}). \tag{3.2.1.3}$$

They are used in the formulation of the plasticity equations, see Section 3.2.2.

If we want to simulate the tensile test until the test piece breaks, we must start with the elastic PDEs and then, if the yield limit is crossed, continue with the plastic PDEs. So, at this limit, we had to change from the variables u_x, u_y that are used in the elastic PDEs to the variables v_x, v_y that are used in the plastic PDEs. Because this is unsuitable we use in both cases the variables v_x, v_y. For this purpose we solve the elastic PDEs in incremental form. We proceed in the elastic part by a time increment Δt. We have

$$v_x = \frac{\partial u_x}{\partial t} \approx \frac{\Delta u_x}{\Delta t}. \tag{3.2.1.4}$$

Therefore we use

$$\begin{aligned} v_x &= \frac{\Delta u_x}{\Delta t}, \qquad \Delta u_x = \Delta t \cdot v_x, \\ v_y &= \frac{\Delta u_y}{\Delta t}, \qquad \Delta u_y = \Delta t \cdot v_y. \end{aligned} \tag{3.2.1.5}$$

This gives us the displacements in the time increment Δt if v_x, v_y are known.

There is still another reason to proceed incrementally in time: If we solve the plastic equations we have large deformations, i.e. the original configuration is strongly disturbed. Therefore, after each time step, we update the coordinates of the nodes by the displacements:

$$\begin{aligned} x_{new} &= x_{old} + \Delta u_x, \\ y_{new} &= y_{old} + \Delta u_y, \end{aligned} \tag{3.2.1.6}$$

where the index "old" denotes the values after the preceding time step.

For the stresses the incremental advancing means that we compute stress increments $\Delta\sigma_{xx}$, $\Delta\sigma_{yy}$, $\Delta\sigma_{xy}$ that are then added to the "old" value of the previous time step. This means a stretching of the material in increments instead of a continuous stretching.
For the incremental stretching equ. (3.2.1.1) becomes

$$\frac{1}{E_x}[\Delta\sigma_{xx} - \nu_{xy}\Delta\sigma_{yy}] = \frac{\partial\Delta u_x}{\partial x}. \tag{3.2.1.7}$$

We have

$$\begin{aligned} \sigma_{xx} &= \sigma_{xx,old} + \Delta\sigma_{xx}, & \Delta\sigma_{xx} &= \sigma_{xx} - \sigma_{xx,old}, \\ \sigma_{yy} &= \sigma_{yy,old} + \Delta\sigma_{yy}, & \Delta\sigma_{yy} &= \sigma_{yy} - \sigma_{yy,old}, \\ \sigma_{xy} &= \sigma_{xy,old} + \Delta\sigma_{xy}, & \Delta\sigma_{xy} &= \sigma_{xy} - \sigma_{xy,old}. \end{aligned} \tag{3.2.1.8}$$

From (3.2.1.5) we get

$$\begin{aligned} \frac{\partial\Delta u_x}{\partial x} &= \frac{\partial(\Delta t \cdot v_x)}{\partial x} = \Delta t\frac{\partial v_x}{\partial x}, \\ \frac{\partial\Delta u_y}{\partial y} &= \frac{\partial(\Delta t \cdot v_y)}{\partial y} = \Delta t\frac{\partial v_y}{\partial y}, \\ \frac{\partial\Delta u_x}{\partial y} &= \frac{\partial(\Delta t \cdot v_x)}{\partial y} = \Delta t\frac{\partial v_x}{\partial y}, \\ \frac{\partial\Delta u_y}{\partial x} &= \frac{\partial(\Delta t \cdot v_y)}{\partial x} = \Delta t\frac{\partial v_y}{\partial x}. \end{aligned} \tag{3.2.1.9}$$

If we replace in (3.2.1.7) $\Delta\sigma_{..}$ and Δu_x by (3.2.1.8) and (3.2.1.9) we get

$$\frac{1}{E_x}[\sigma_{xx} - \sigma_{xx,old} - \nu_{xy}(\sigma_{yy} - \sigma_{yy,old})] - \Delta t\frac{\partial v_x}{\partial x} = 0. \tag{3.2.1.10}$$

By a similar procedure we get for ε_{yy} from [12](2.11) and [12](2.27)

$$-\frac{\nu_{xy}}{E_x}(\sigma_{xx} - \sigma_{xx,old}) + \frac{1}{E_y}(\sigma_{yy} - \sigma_{yy,old}) - \Delta t\frac{\partial v_y}{\partial y} = 0. \tag{3.2.1.11}$$

and for ε_{xy} we get from [12](2.12) and [12](2.31)

$$\frac{1}{G_{xy}}(\sigma_{xy} - \sigma_{xy,old}) - \Delta t(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}) = 0. \tag{3.2.1.12}$$

These are 3 equations for the 5 unknowns v_x, v_y, σ_{xx}, σ_{yy}, σ_{xy}. The missing 2 PDEs come from the balancing of the forces which are for 2-D from [12](2.2)

$$\frac{\partial\sigma_{xx}}{\partial x} + \frac{\partial\sigma_{xy}}{\partial y} = 0, \tag{3.2.1.13}$$

$$\frac{\partial\sigma_{xy}}{\partial x} + \frac{\partial\sigma_{yy}}{\partial y} = 0. \tag{3.2.1.14}$$

Here we do not need to formulate the balancing in incremental form because it holds as consequence of the linearity of the equations for the σ's and the $\Delta\sigma$'s.

Table 3.2.1.1: Sequence of variables and equations for the solution of the elastic equations.

no.	variable	equation
1	v_x	(3.2.1.10)
2	v_y	(3.2.1.12)
3	σ_{xx}	(3.2.1.13)
4	σ_{yy}	(3.2.1.11)
5	σ_{xy}	(3.2.1.14)

The numerical solution of this system of 5 PDEs turned out to be rather critical because we use in FDEM "centralized" difference stars and in the PDEs there are only first derivatives which lead to odd/even uncoupling of the solution (the BCs are given below). We see the discretization error from the error estimate und we found that the sequence of variables and PDEs that is given in Table 3.2.1.1 was optimal. Because we found in the simulation of the tensile test with the simplified configuration of Fig. 3.2.1.1, lower part, that $\sigma_{yy} = 0,\ \sigma_{xy} = 0$ theoretically and numerically, we used also a "3-equation model" where the PDEs for σ_{yy} and σ_{xy} were replaced by explicit value zero. Then we have the sequence of variables and PDEs given in Table 3.2.1.2.

Table 3.2.1.2: Sequence of variables and PDEs for the "3-equation model" for elastic equations.

no.	variable	equation
1	v_x	(3.2.1.10)
2	v_y	(3.2.1.11)
3	σ_{xx}	(3.2.1.13)
4	σ_{yy}	$\sigma_{yy} = 0$
5	σ_{xy}	$\sigma_{xy} = 0$

Before we discuss the BCs we want to give the expressions for the stress components of a surface and the normal and tangential stresses. Fig. 3.2.1.2 shows a surface element dS with components dS^x, dS^y and the normal n (length 1) with components n^x, n^y. In σ_{xx} the first index denotes the normal direction of the surface on which σ acts and the second index denotes the direction of the stress. From that definition the force dF at a surface element is, with $\sigma_{xy} = \sigma_{yx}$:

$$dF = \begin{pmatrix} dS^x\sigma_{xx} & +dS^y\sigma_{xy} \\ dS^x\sigma_{xy} & +\,dS^y\sigma_{yy} \end{pmatrix} \qquad \begin{matrix} x\text{-component} \\ y\text{-component.} \end{matrix} \tag{3.2.1.15}$$

Force on dS^x, on dS^y

From Fig. 3.2.1.2 we can see that

$$\frac{dS^x}{dS} = \frac{n^x}{1}, \qquad \frac{dS^y}{dS} = \frac{n^y}{1},$$

from which we get

$$dS^x = n^x\, dS, \qquad dS^y = n^y\, dS. \tag{3.2.1.16}$$

So we get from (3.1.2.15)

$$dF = \begin{pmatrix} \sigma_{xx}\, n^x + \sigma_{xy}\, n^y \\ \sigma_{xy}\, n^x + \sigma_{yy}\, n^y \end{pmatrix} dS. \tag{3.2.1.17}$$

Because stress is $\sigma = \frac{dF}{dS}$ we get for the stress vector at the surface

$$\sigma \qquad = \begin{pmatrix} \sigma_{xx}n^x & +\sigma_{xy}n^y \\ \sigma_{xy}n^x & +\sigma_{yy}n^y \end{pmatrix} \qquad \begin{matrix} x\text{-component} \\ y\text{-component.} \end{matrix} \tag{3.2.1.18}$$

surface element with normal in: x-dir. y-direction

At a surface whose normal is in the x-direction we get from (3.2.1.18)

$$n = \begin{pmatrix} n^x \\ n^y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} : \sigma = \begin{pmatrix} \sigma_{xx} \\ \sigma_{xy} \end{pmatrix}. \tag{3.2.1.19}$$

and at a surface whose normal is in the y-direction

$$n = \begin{pmatrix} n^x \\ n^y \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} : \sigma = \begin{pmatrix} \sigma_{xy} \\ \sigma_{yy} \end{pmatrix}. \tag{3.2.1.20}$$

In Fig. 3.2.1.3 we see the tangent vector t at the surface element dS. We can directly see from the

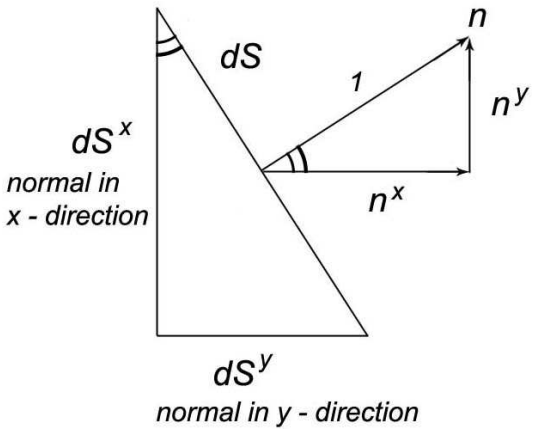


Figure 3.2.1.2: Illustration for surface element dS.

geometry that $t^x = -n^y$ (negative direction), $t^y = n^x$. Thus we get

$$t = \begin{pmatrix} t^x \\ t^y \end{pmatrix} = \begin{pmatrix} -n^y \\ n^x \end{pmatrix}. \tag{3.2.1.21}$$

If we denote by σ^T the transpose of the stress vector (3.2.1.18) we get the normal stress as the component of the stress vector in the normal direction as the scalar product of σ and n:

$$\sigma_n = \sigma^T \cdot n = \sigma_{xx}(n^x)^2 + 2\sigma_{xy}n^x n^y + \sigma_{yy}(n^y)^2, \tag{3.2.1.22}$$

and similarly with (3.2.1.21) the tangential component

$$\sigma_t = \sigma^T \cdot t = -\sigma_{xx}n^x n^y + \sigma_{xy}((n^x)^2 - (n^y)^2) + \sigma_{yy}n^x\, n^y. \tag{3.2.1.23}$$

We will need these expressions later in this report.

Now we want to discuss the BCs for the solution of the equations of Table 3.2.1.1 for the elastic part of the tensile test with the simplified test piece of Fig. 3.2.1.1. Fig. 3.2.1.4 shows the BCs. We want to explain them a little. At the upper and lower edge, without the corners (end points of the edge), we use for v_x, v_y, σ_{xx} the indicated PDEs. For the BCs, like for the PDEs, it is important to have the right PDE for the right variable, else there are numerical problems (divergence of LINSOL, divergence of Newton, large errors). The upper and lower edge are force-free edges. The stress components there are zero. The normal is in the y-direction, $n^y = 1$ (upper) or $n^y = -1$ (lower), $n^x = 0$. thus from (3.2.1.18) we get $\sigma_{xy} = 0$ and $\sigma_{yy} = 0$ from x- and y-component to be zero.

The left end of the simplified test piece is fixed: $v_x = 0$. The right end moves with the speed $v_{x,test}$ of the tensile test maschine. For the other variables the BCs of the left and right end are the same. For v_y we use PDE (3.2.1.12), but in order to fix the test piece in the y-direction we use $v_y = 0$

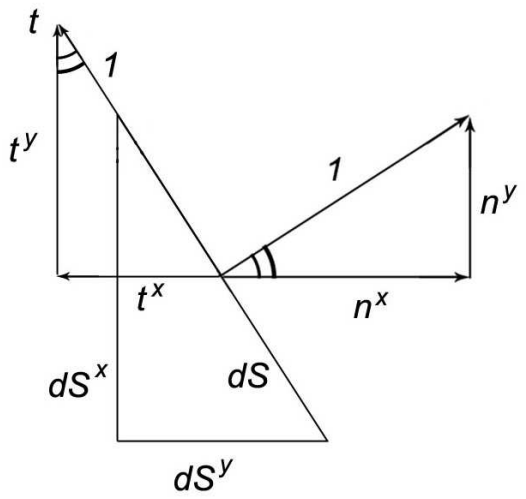


Figure 3.2.1.3: Illustration for tangent t.

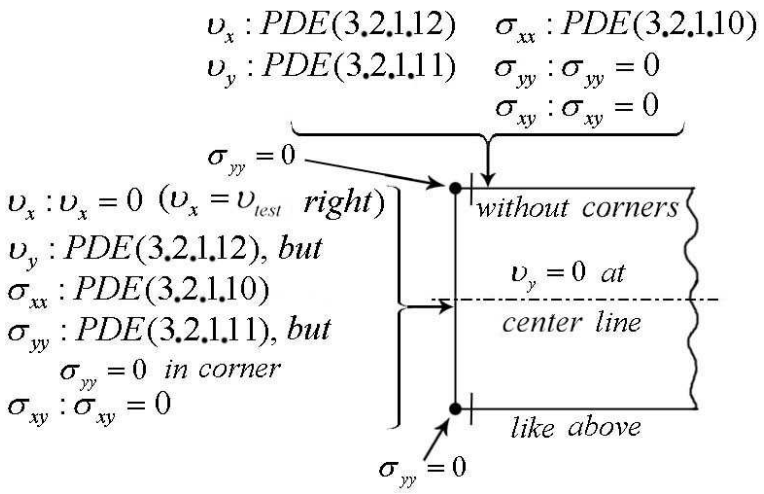


Figure 3.2.1.4: BCs for the simplified tensile test piece for the equations of Table 3.2.1.1.

at the nodes of the center line of the test piece. σ_{xx} and σ_{yy} result from the movement of the right end, we use PDEs (3.2.1.10) and (3.2.1.11). Because there is no y-component of the stress and we have the normal with $n^x = \pm 1$, $n^y = 0$ we get from (3.2.1.18) $\sigma_{xy} = 0$. Thus we have used at all edges the basic elasticity equations (3.2.1.10)–(3.2.1.12).

For the "3-equation model" of Table 3.2.1.2 we merely replace the condition for σ_{yy} at the left and right end by $\sigma_{yy} = 0$.

Because the elastic equations are only used to produce starting conditions for the plastic equations we do not present results here. They are trivial because the problem is linear and the solution could be obtained by the theory because of the simple geometry. However, if we look carefully the problem is severely non-linear because we move the coordinates of the configuration when the test piece extends in the x-direction and shrinks consequently in the y-direction. The results of the full model of Table 3.2.1.1 showed that $\sigma_{yy} \approx 0$ and $\sigma_{xy} \approx 0$ so that we later used the "3-equation model" of Table 3.2.1.2. The spatial maximal relative discretization errors were estimated in the range 10^{-8} down to 10^{-10} because of the linearity of a single time step.

3.2.2 The attempt to solve numerically a "full" plasticity model for the tensile test

After many vain trials the IFU finally presented a "full" plasticity model, the theoretical background is given in [12]. Here we restrict to the discussion of the numerical solution. We could solve numerically the system of PDEs—for the test function. However, all attempts to get solutions for the physical model failed because of the extreme non-linearity of the system. These attempts, for the model discussed below and for all the preceding models, consumed most of the project resources. The only result was: the model is "unusable" for practical applications. Nevertheless we present here the numerical solution method to show how we attacked the problem.

For the plasticity we have now, besides the variables $v_x, v_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy}$ that had been introduced in the elastic part, an additional variable: the plasticity parameter λ. With the discussion in [12] about

the von Mises stress and the equations [12](2.32), [12](2.39), [12](2.56) we get that the material gets plastic, i.e. is above the yield stress, if we have

$$\overline{\sigma} \geqq Y_0 + K(\varepsilon_0 + \lambda)^n, \tag{3.2.2.1}$$

with the equivalence stress

$$\overline{\sigma} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 - \sigma_{xx}\sigma_{yy} + 3\sigma_{xy}^2} \tag{3.2.2.2}$$

for the 2-D case. The values for Y_0, K, ε_0, n are given for the stainless steel sheet in [12](2.72) to [12](2.77) where we take the xx-values und have $Y_0 = A_{xx}$. As λ is a variable whose value may be different for each node, and as $\overline{\sigma}$ depends also on the values of the σ's of the node, we must check in each node individually the condition (3.2.2.1) to decide if the material at the node is elastic or plastic.

The question is now: with which equations the next time step (in a time incremental procedure) should be solved, with the elastic or plastic PDEs? For the simulation of the manufacturing process of the metal bellows we know for the previous time step if a surface node of the steel sheet is a free node or if it is forced by the tool, i.e. moves with the tool. As we want to develop for stability reasons a totally implicit method we must know which conditions hold at the end of the actual time step because they decide. Therefore we always make two time steps: a test step with the conditions of the previous step to check if the steel is elastic or plastic at the end of the step (and for the manufacturing: is a boundary node free or forced) and then execute the computation step with the conditions found at the end of the test step.

The explicit plasticity equations are obtained by equating the geometrical expression for the strain velocity, e.g. $\dot{\varepsilon}_{xx}$ [12](2.5) with that of the plastic expression [12](2.68). In the geometrical expression is now the linear Cauchy term and the non-linear Green term. With the notation

$$\dot{\lambda} = \frac{d\lambda}{dt} = \frac{\partial\lambda}{\partial t} + \frac{\partial\lambda}{\partial x}v_x + \frac{\partial\lambda}{\partial y}v_y \tag{3.2.2.3}$$

from [12](2.51) where we have used $dx/dt = v_x,\ dy/dt = v_y,\ \partial\lambda/\partial z = 0$ we get

$$\begin{aligned}(\frac{\partial\lambda}{\partial t} + v_x\frac{\partial\lambda}{\partial x} + v_y\frac{\partial\lambda}{\partial y})\frac{1}{2\overline{\sigma}}(2\sigma_{xx} - \sigma_{yy}) - \\ -(\frac{\partial v_x}{\partial x} + \frac{\partial u_x}{\partial x}\frac{\partial v_x}{\partial x} + \frac{\partial u_y}{\partial x}\frac{\partial v_y}{\partial x}) = 0.\end{aligned} \tag{3.2.2.4}$$

Similarly we get by $\dot{\varepsilon}_{yy}$

$$\begin{aligned}(\frac{\partial\lambda}{\partial t} + v_x\frac{\partial\lambda}{\partial x} + v_y\frac{\partial\lambda}{\partial y})\frac{1}{2\overline{\sigma}}(2\sigma_{yy} - \sigma_{xx}) - \\ -(\frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y}\frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial y}\frac{\partial v_y}{\partial y}) = 0\end{aligned} \tag{3.2.2.5}$$

and by $\dot{\varepsilon}_{xy}$

$$\begin{aligned}(\frac{\partial\lambda}{\partial t} + v_x\frac{\partial\lambda}{\partial x} + v_y\frac{\partial\lambda}{\partial y})\frac{3\sigma_{xy}}{2\overline{\sigma}} - \\ -[\frac{1}{2}(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}) + \frac{\partial u_x}{\partial x}\frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x}\frac{\partial v_y}{\partial y}] = 0.\end{aligned} \tag{3.2.2.6}$$

These are 3 equations for the 6 variables $v_x, v_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy}, \lambda$. Two further equations are the equilibrium equations (3.2.1.13) and (3.2.1.14) that are the same as for elastic deformation. The 6^{th} equation is the equation for λ that is in reality an equation for $\dot{\lambda}$, equation [12](2.60) that we rearrange in the form

$$\begin{aligned}
&\frac{\partial\lambda}{\partial t}+v_x\frac{\partial\lambda}{\partial x}+v_y\frac{\partial\lambda}{\partial y}-\frac{1}{Kn(\varepsilon_0+\lambda)^{n-1}}\cdot\frac{1}{\bar{\sigma}}\cdot\\
&\quad\cdot\{\frac{1}{2}[2\sigma_{xx}(\frac{\partial\sigma_{xx}}{\partial t}+v_x\frac{\partial\sigma_{xx}}{\partial x}+v_y\frac{\partial\sigma_{xx}}{\partial y})+\\
&\quad+2\sigma_{yy}(\frac{\partial\sigma_{yy}}{\partial t}+v_x\frac{\partial\sigma_{yy}}{\partial x}+v_y\frac{\partial\sigma_{yy}}{\partial y})+\\
&\quad+6\sigma_{xy}(\frac{\partial\sigma_{xy}}{\partial t}+v_x\frac{\partial\sigma_{xy}}{\partial x}+v_y\frac{\partial\sigma_{xy}}{\partial y})-\\
&\quad-\sigma_{xx}(\frac{\partial\sigma_{yy}}{\partial t}+v_x\frac{\partial\sigma_{yy}}{\partial x}+v_y\frac{\partial\sigma_{yy}}{\partial y})-\\
&\quad-\sigma_{yy}(\frac{\partial\sigma_{xx}}{\partial t}+v_x\frac{\partial\sigma_{xx}}{\partial x}+v_y\frac{\partial\sigma_{xx}}{\partial y})]\}=0.
\end{aligned}\tag{3.2.2.7}$$

Here we recognize that the first 3 terms (that are $\dot{\lambda}$) and the terms in the parentheses are total time derivatives d/dt. The corresponding justification for these equations is given in [12].

We have now the problem to solve these equations. The sequence of the variables and equations is given in Table 3.2.2.1 similarly to Table 3.2.1.1. The BCs are that of Fig. 3.2.1.4 where we replace the elastic equations by the corresponding plastic equations: (3.2.1.10) → (3.2.2.3), (3.2.1.11) → (3.2.2.4), (3.2.1.12) → (3.2.2.5).

There are two problems. The first problem are the time derivatives. The PDEs are time evolution PDEs, i.e. of parabolic type. FDEM can solve directly parabolic equations. However, in this case we cannot use this possibility but we must proceed by time increments as explained above for the elastic equations. Therefore we use in (3.2.2.4)–(3.2.2.7) e.g.

$$\frac{\partial\lambda}{\partial t}\approx\frac{\Delta\lambda}{\Delta t}=\frac{\lambda-\lambda_{old}}{\Delta t},\quad \frac{\partial\sigma_{xx}}{\partial t}\approx\frac{\Delta\sigma_{xx}}{\Delta t}=\frac{\sigma_{xx}-\sigma_{xx,old}}{\Delta t}\tag{3.2.2.8}$$

and similarly for σ_{yy}, σ_{xy}. λ_{old}, $\sigma_{xx,old}$ etc. are the profiles of the previous time step. The second problem are the derivatives $\partial u_x/\partial x$, $\partial u_x/\partial y$, $\partial u_y/\partial x$, $\partial u_y/\partial y$, i.e. the space derivatives of the displacements u_x, u_y in the non-linear Green terms of the strain velocities in equations (3.2.2.4)–(3.2.2.6). We have for a node i: $u_{x,i}=x_i(t)-x_i(t=0)$. However, we know $x_i(t)$ only after the moving of the coordinates at the end of a time step. Therefore we use

$$u_{x,i}=x_i(t_{old})-x_i(t=0),\quad u_{y,i}=y_i(t_{old})-y_i(t=0),\tag{3.2.2.9}$$

where t_{old} is the time of the previous step. We store the profiles of u_x and u_y for all nodes and compute with the difference formulas the required derivatives with respect to x and y.

Table 3.2.2.1: Sequence of variables and equations for the solution of the "full" plastic equations.

no	variable	equation
1	v_x	(3.2.2.4)
2	v_y	(3.2.2.6)
3	σ_{xx}	(3.2.1.13)
4	σ_{yy}	(3.2.2.5)
5	σ_{xy}	(3.2.1.14)
6	λ	(3.2.2.7)

This system of equations is, with the exception of the two linear equilibrium equations, extremely non-linear, above all the λ-equation (3.2.2.7). Observe that in $\overline{\sigma}$ (3.2.2.2) the stresses appear non-linearly. We compute with the elastic equations until in the test step the condition (3.2.2.1) holds, then for this node the plastic equations are used for the computation step. As long as the equations are elastic, we have $\lambda = 0$. The starting values for the other variables are those of the last elastic step. So we solve, by the time discretization (3.2.2.8), in each time step an elliptic problem. This means a fully implicit procedure because all 6 variables are computed at the new time step. So we expect unconditional stability in time.

We have implemented the PDEs and used a test polynomial $\overline{u}$ of second order and solved with consistency order $q = 2$. So we should reproduce the test solution in the range of the rounding errors. This could indeed be observed which proves that the equations were implemented correctly. We switched on the Jacobi tester that finally did no longer report errors so that we are sure that the Jacobian matrices were correct. It should be mentioned that for these extremely non-linear equations the Jacobian matrices are correspondingly complicated expressions that could be made treatable only by the definition of smaller intermediate expressions.

Then we used, as usual, a starting solution with 1% disturbance and expected that Newton's method finds back with a 1% correction to the exact solution. However, Newton diverged. It tried to reduce the Newton defect by a relaxation factor ω and finally gave up if ω was below 0.01. Then we made a series of numerical experiments that showed: below a disturbance of 0.2% Newton found back the exact solution, for larger disturbances it diverged. So we see that we had correctly implemented the PDEs, but the extreme non-linearity made the numerical solution very critical.

When we then took off the terms of the test solution and tried to solve the physical problem as continuation of the elastic solution, Newton did not converge (as we had expected). We then made numerous experiments with linearizations of the system. If the index "old" denotes the values at the previous time step, we replaced $\overline{\sigma}$ by $\overline{\sigma}_{old}$, or in the λ-terms $v_x(\partial\lambda/\partial x) \to v_{x,old}(\partial\lambda/\partial x)$, or in (3.2.2.7) $(\varepsilon_0 + \lambda) \to (\varepsilon_0 + \lambda_{old})$, or the whole braces $\{\} \to \{\}_{old}$. We could get solutions of the linearized systems, but only for one plastic time step. The second plastic time step then gave always physically unrealistic solutions.

In order to simplify the system we intended to go to a "3-equation model" like in Table 3.2.1.2 with $\sigma_{yy} = 0, \sigma_{xy} = 0$. However, this is not possible and shows the strange character of the system: In equation (3.2.2.4) we have for $\sigma_{yy} = 0, \sigma_{xy} = 0$

$$\frac{1}{2\overline{\sigma}}(2\sigma_{xx} - \sigma_{yy}) = \frac{1}{2\sqrt{\sigma_{xx}^2}} \cdot 2\sigma_{xx} = 1. \tag{3.2.2.10}$$

So the variable σ_{xx} for which this equation should be used, drops out. We leave it to the metallurgists to discuss this property.

The consequence of these very frustrating numerical experiments is: the PDEs of the "full" plasticity model can be solved as we have seen for the test solution, but they are "unusable" for the physical problem because Newton converges only very close to the solution as a consequence of the extreme non-linearity. We then gave up to try to find physical solutions of these equations as continuation of the elastic solution.

3.2.3 The approach with a "plastic" E-module for the tensile test

Because we could not get physical solutions of the "full" plastic model equations we proceeded as follows: We solved the elastic equations of Table 3.2.1.1 for a time incremental step and checked if the equivalent stress was above the yield stress with the condition (3.2.2.1). If we are above the yield surface we have the situation of Fig. 3.2.3.1: The value of $\overline{\sigma}$ is too large.

Therefore we project all stresses down so that we come down at or below the yield surface. We use the approach

$$\sigma_{xx} = a\,\sigma_{xx,el}, \quad \sigma_{yy} = a\,\sigma_{yy,el}, \quad \sigma_{xy} = a\,\sigma_{xy,el}, \tag{3.2.3.1}$$

where the index "el" means computed with the elastic equations. The projected stresses fulfil still the equilibrium equations (3.2.1.13) and (3.2.1.14) because the factor a drops out. We have with (3.2.3.1), (3.2.2.2)

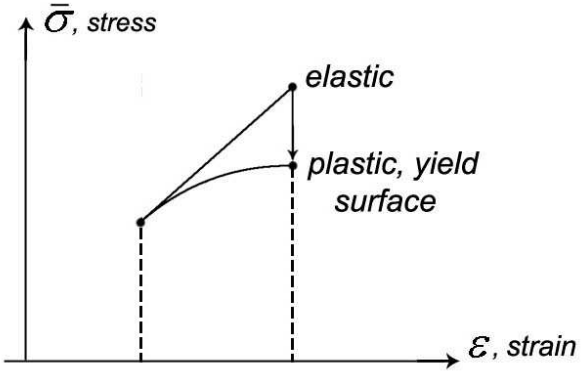


Figure 3.2.3.1: Illustration for $\overline{\sigma}$ too large.

$$\overline{\sigma} = \sqrt{a^2\sigma_{xx,el}^2 + \cdots} = a\sqrt{\sigma_{xx,el}^2 + \cdots} = a\,\overline{\sigma}_{el}. \tag{3.2.3.2}$$

With the yield stress Y [12](2.39)

$$Y = Y_0 - K(\varepsilon_0 + \lambda)^n, \tag{3.2.3.3}$$

we want the coefficient a such that $\overline{\sigma} \leqq Y$. For more flexibility we use

$$a = 1 - \alpha(1 - \frac{Y_0 + K(\varepsilon_0 + \lambda)^n}{\sqrt{\sigma^2_{xx,el} - \sigma^2_{yy,el} - \sigma_{xx,el}\sigma_{yy,el} + 3\sigma^2_{xy,el}}}). \tag{3.2.3.4}$$

For $\alpha = 1$ we have $\overline{\sigma} = Y$, i.e. we project onto the yield surface, for $\alpha = 2$ we have the situation of Fig. 3.2.3.2: we go down twice the difference to the yield surface so that the yield surface runs between the elastic and the projected values. With these projected stresses and PDE (3.2.2.7) the value λ for the next time step is computed.

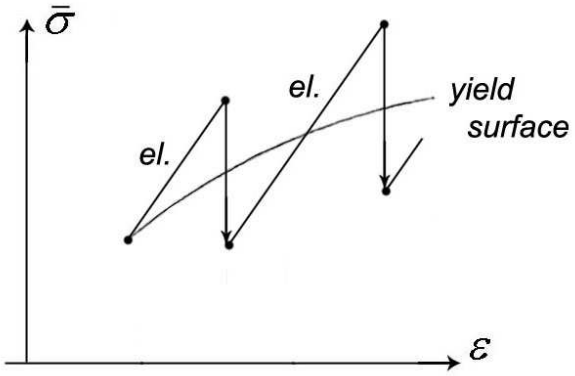


Figure 3.2.3.2: Illustration of (3.2.3.4) for $\alpha = 2$.

We made numerical experiments with the coefficients in the elastic equations given in [12](2.81) to [12](2.83) and for Y given in [12](2.75) to [12](2.77). However, the results were such that the stresses were far too large for a certain strain: The material was too hard. If the steel yields it is "dough" and not a spring. We tried with lower values of the E-module to meet the measurements, but the attempts failed. This was *not* the way to solve the problem.

From the numerical experiments it became obvious that the E-module must be very "soft" and depend on the strain ε. Therefore the IFU made the following proposition, for Y_x (in our notation)

$$Y_x = Y_{0,x} + K_x(\varepsilon_{0,x} + \varepsilon_{xx})^{n_x}, \tag{3.2.3.5}$$

from which follows with $E_x = \partial Y_x / \partial \varepsilon_{xx}$ [12](2.78)

$$E_x = K_x\, n_x(\varepsilon_{0,x} + \varepsilon_{xx})^{n_x - 1}. \tag{3.2.3.6}$$

The proposed coefficients are [12](2.75) to [12](2.77):

$$Y_{0,x} = 60\frac{N}{mm^2}, \quad K_x = 175\frac{N}{mm^2}, \quad \varepsilon_{0,x} = 1.02, \quad n_x = 0.293. \tag{3.2.3.7}$$

As we have for plastic deformation large strain, we must take for ε the non-linear Cauchy-Green form [12](2.3). For 2-D we have in tensor notation

$$\varepsilon_{ik} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right) + \frac{1}{2}\sum_{j=1}^{2}\frac{\partial u_j}{\partial x_i}\,\frac{\partial u_j}{\partial x_k} \tag{3.2.3.8}$$

from which we get in our notation

$$\varepsilon_{xx} = \frac{\partial u_x}{\partial x} + \frac{1}{2}\left((\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2\right), \tag{3.2.3.9}$$

$$\varepsilon_{yy} = \frac{\partial u_y}{\partial y} + \frac{1}{2}\left((\frac{\partial u_x}{\partial y})^2 + (\frac{\partial u_y}{\partial y})^2\right), \tag{3.2.3.10}$$

$$\varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) + \frac{1}{2}\left(\frac{\partial u_x}{\partial x}\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\frac{\partial u_y}{\partial y}\right). \tag{3.2.3.11}$$

If we equate ε_{xx} from Hooke's law [12](2.26) and from (3.2.3.9) we get

$$\frac{1}{E_x}[\sigma_{xx} - \nu_{xy}\sigma_{yy}] = \frac{\partial u_x}{\partial x} + \frac{1}{2}\left((\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2\right). \tag{3.2.3.12}$$

If we want to proceed incrementally as explained in the context of equs. (3.2.1.7)–(3.2.1.10) we get from (3.2.3.12) for an incremental stress and displacement

$$\frac{1}{E_x}[\Delta\sigma_{xx} - \nu_{xy}\Delta\sigma_{yy}] = \frac{\partial\Delta u_x}{\partial x} + \frac{1}{2}\left((\frac{\partial\Delta u_x}{\partial x})^2 + (\frac{\partial\Delta u_y}{\partial x})^2\right). \tag{3.2.3.13}$$

For a time increment Δt we get with (3.2.1.5) and (3.2.1.8)

$$\begin{aligned}&\frac{1}{E_x}[\sigma_{xx} - \sigma_{xx,old} - \nu_{xy}(\sigma_{yy} - \sigma_{yy,old})] - \\ &-\Delta t\frac{\partial v_x}{\partial x} - \frac{1}{2}\Delta t^2\left((\frac{\partial v_x}{\partial x})^2 + (\frac{\partial v_y}{\partial x})^2\right) = 0.\end{aligned} \tag{3.2.3.14}$$

Similarly we get for ε_{yy} and ε_{xy}

$$\begin{aligned}&-\frac{\nu_{xy}}{E_x}(\sigma_{xx} - \sigma_{xx,old}) + \frac{1}{E_y}(\sigma_{yy} - \sigma_{yy,old}) - \\ &-\Delta t\frac{\partial v_y}{\partial y} - \frac{1}{2}\Delta t^2\left((\frac{\partial v_x}{\partial y})^2 + (\frac{\partial v_y}{\partial y})^2\right) = 0,\end{aligned} \tag{3.2.3.15}$$

$$\frac{2(1+\nu_{xy})}{E_x+E_y}\sigma_{xy} - \frac{1}{2}\Delta t\left(\frac{\partial \upsilon_x}{\partial y}+\frac{\partial \upsilon_y}{\partial x}\right) - \\ -\frac{1}{2}\Delta t^2\left(\frac{\partial \upsilon_x}{\partial x}\,\frac{\partial \upsilon_x}{\partial y}-\frac{\partial \upsilon_y}{\partial x}\,\frac{\partial \upsilon_y}{\partial y}\right)=0. \tag{3.2.3.16}$$

Here we have used [12](2.83)

$$G_{xy}=\frac{E_x+E_y}{4(1+\nu_{xy})}. \tag{3.2.3.17}$$

The equilibrium equations (3.2.1.13), (3.2.1.14) are the same as in the elastic case.

Like explained for Table 3.2.1.2 we reduce with $\sigma_{yy}=0$, $\sigma_{xy}=0$ these 5 equations to the "1-D" 3-equation model for the simplified tensile test piece of Fig. 3.2.1.1. Here we write down explicitly these equations:

$$\frac{1}{E_x}(\sigma_{xx}-\sigma_{xx,old})-\Delta t\frac{\partial \upsilon_x}{\partial x}-\frac{1}{2}\Delta t^2\left((\frac{\partial \upsilon_x}{\partial x})^2+(\frac{\partial \upsilon_y}{\partial x})^2\right)=0, \tag{3.2.3.18}$$

$$-\frac{\nu_{xy}}{E_x}(\sigma_{xx}-\sigma_{xx,old})-\Delta t\frac{\partial \upsilon_y}{\partial y}-\frac{1}{2}\Delta t^2\left((\frac{\partial \upsilon_x}{\partial y})^2+(\frac{\partial \upsilon_y}{\partial y})^2\right)=0, \tag{3.2.3.19}$$

$$\frac{\partial \sigma_{xx}}{\partial x}=0. \tag{3.2.3.20}$$

These are 3 equations for the 3 variables υ_x, υ_y, σ_{xx}. The BCs are those of Fig. 3.2.1.4, but now we have at the left/right boundary υ_y: PDE (3.2.1.19), σ_{xx}: PDE (3.2.3.18), $\sigma_{yy}=0$, $\sigma_{xy}=0$ and at the upper/lower boundary υ_x: PDE (3.2.1.18), υ_y: PDE (3.2.3.19), σ_{xx}: $\partial\sigma_{xx}/\partial x=0$, the non-mentioned conditions are the same.

Now we use for the simulation of the tensile test this 3-equation model, i.e. the "elastic" equations also in the plastic region, but the constant elastic E-module E_x is now replaced by E_x from (3.2.3.6). Thus we go automatically along the yield limit. Here ε_{xx} is computed from the non-linear Cauchy-Green form (3.2.3.9)

$$\varepsilon_{xx}=\left\{\frac{\partial u_x}{\partial x}+\frac{1}{2}\left[(\frac{\partial u_x}{\partial x})^2+(\frac{\partial u_y}{\partial x})^2\right]\right\}_{old}. \tag{3.2.3.21}$$

The index "old" means that the values are computed from the result of the previous time step. For the computation of $\partial u_x/\partial x$, $\partial u_y/\partial x$ see the remarks to equation (3.2.2.9). Here u is the global displacement. The check if the steel is elastic or plastic is made with Y from (3.2.3.3), i.e. by the condition (3.2.2.1).

Here we should explain more precisely how we proceed: For a new time step we make at first a test step with the conditions of the previous step, i.e. nodes that were elastic at the end of the previous computation step are solved with the elastic equations. At the end of the test step we check

if the nodes are elastic or plastic and then the computation step is executed with the corresponding equations. At the end of the computation step the nodes are again checked for elastic or plastic and with these properties the next test step is executed.

After the computation step a new value for the plasticity parameter λ is computed for plastic nodes. As long as a node is elastic we have $\lambda = 0$ until it becomes plastic for the first time. When a node is plastic we compute λ from the PDE (3.2.27). For the 3-equation model ($\sigma_{yy} = 0$, $\sigma_{xy} = 0$) (3.2.2.7) reduces to

$$\frac{\partial \lambda}{\partial t} + \upsilon_x \frac{\partial \lambda}{\partial x} + \upsilon_y \frac{\partial \lambda}{\partial y} - \frac{1}{Kn(\varepsilon_0 + \lambda)^{n-1}} \frac{1}{\overline{\sigma}} \cdot \\ \cdot \sigma_{xx} \left(\frac{\partial \sigma_{xx}}{\partial t} + \upsilon_x \frac{\partial \sigma_{xx}}{\partial x} + \upsilon_y \frac{\partial \sigma_{xx}}{\partial y} \right) = 0. \tag{3.2.3.22}$$

Time derivatives $\partial\lambda/\partial t$, $\partial\sigma_{xx}/\partial t$ are discretized with (3.2.2.8). In this scalar PDE only λ is the unknown function, all other values are taken from the actual computation step. The so computed value of λ is then used in the next time step.

Then we made numerous numerial experiments with E_x from (3.2.3.6) to simulate the measurements of the tensile test. We started with coefficients that were proposed by the IFU and failed completely to meet with the simulation the physical measurements. Then we varied systematically the coefficients and observed how we had to change the values that we came with the numerical result closer to the measurements. We finally ended up with a result that was not satisfactory, but the possibilities of the coefficients were exhausted.

Therefore we looked how we could adapt E_x by an additional functional term. We ended up with

$$E_x = (1 + \delta_x(\varepsilon_{xx}))\ K_{x,0}\ n_x(\varepsilon_{x,0} + \varepsilon_{xx})^{n_x - 1} \tag{3.2.3.23}$$

which means that in (3.2.3.6) the constant value K_x is replaced by

$$K_x = (1 + \delta_x(\varepsilon_{xx}))K_{x,0}, \tag{3.2.3.24}$$

where $\delta_x(\varepsilon_{xx})$ is itself a function of ε_{xx}. By an appropriate choice of this function we could nearly perfectly simulate the measurement. For the solution of the PDEs we take $\varepsilon_{xx} = \varepsilon_{xx,old}$, i.e. the value of the previous time step to have an explicit formula.

However, there was a new problem: obviously the volume of the simulated test piece was not maintained, it was too small. This means that the transverse contraction ν was too large. We had used the constant value of [12](2.82) $\nu_{xy} = 0.5055$. So the problem was much more complicated: We had to adapt the parameters of our approach so that we reproduced the measurements and at the same time to determine a value of ν that maintained the volume.

Fig. 3.2.3.3 shows the simplified test piece with the notations for the cross section F, the volume V and the dimensions l, b, h in x-, y-, z-direction. The actual cross section F and initial value F_0 are

$$F = b \cdot h, \quad F_0 = 20 \cdot 0.2 = 4\ mm^2, \tag{3.2.3.25}$$

see Fig. 3.2.1.1. The actual volume V and initial value V_0 are

$$V = F \cdot l, \quad V_0 = 4 \cdot 180 = 720\ mm^3. \tag{3.2.3.26}$$

The values of l and b can be determined by the difference of the x- and y-coordinates of correspond-

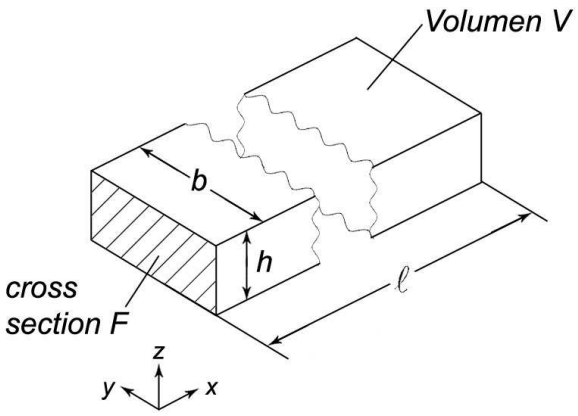


Figure 3.2.3.3: Notations for the determination of the cross section F and Volume V.

ing corner nodes.

Now we want to discuss the determination of h, i.e. the actual thickness of the metal sheet. The original value is $h_0 = 0.2\ mm$. By the expansion of the test piece and the corresponding transverse contraction also in the z-direction the thickness shrinks from h_0 to the actual value h. In the same way as we have derived from [12](2.26) for ε_{xx} the incremental PDE (3.2.1.10) we compose from [12](2.11) and [12](2.28) for ε_{zz} with $\sigma_{yy} = 0$, $\sigma_{xy} = 0$ (3-equation model) the PDE

$$-\frac{\nu_{xz}}{E_x}(\sigma_{xx} - \sigma_{xx,old}) - \Delta t\, \frac{\partial v_z}{\partial z} = 0. \tag{3.2.3.27}$$

Note that σ_{xx} is constant for the simplified test piece. With $\Delta t \cdot v_z = \Delta u_z$ we get

$$\frac{\partial \Delta u_z}{\partial z} = -\frac{\nu_{xz}}{E_x}(\sigma_{xx} - \sigma_{xx,old}). \tag{3.2.3.28}$$

This is a linear differential equation for Δu_z, the incremental displacement in z-direction. We have the situation of Fig. 3.2.3.4: Δu_z has negative slope. It goes through the origin and at the "surface" with distance h it has the value $\Delta u_z = h \cdot slope$. The new thickness is $h = h_{old} - slope \cdot h_{old} = (1 - slope)h_{old}$. So we get with the slope $\partial \Delta u_z / \partial z$ (3.2.3.28)

$$h = \left[1 - \frac{\nu_{xz}}{E_x}(\sigma_{xx} - \sigma_{xx,old})\right] h_{old}. \tag{3.2.3.29}$$

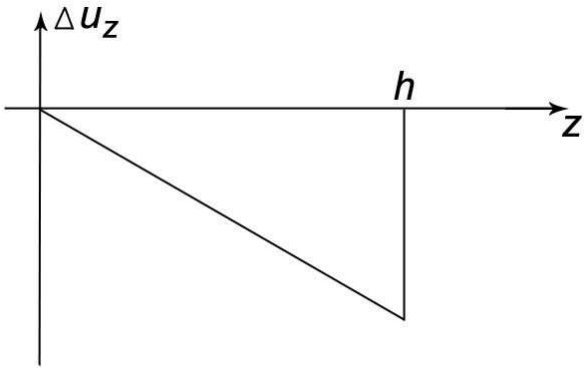


Figure 3.2.3.4: Illustration for equ. (3.2.3.28).

We have formally derived this relation for elastic deformation. However, with our approach of "soft" E-module and plastic value of ν we use this relation with $\nu_{xz} = \nu_{plastic}$ and E_x from (3.2.2.23) also for plastic deformation. Here we do not need the non-linear Green strain tensor because the corresponding terms of type $(\partial u_x/\partial z)^2$ (compare (3.2.3.9)) are zero for our 3-equation model (2-D). The transverse contraction is here the only effect that must be considered in z-direction. With h from (3.2.3.29), F from (3.2.3.25) we can compute the actual volume V from (3.2.3.26).

As mentioned above we observed a too small value of the volume V because by the elastic value ν_{xy} in (3.2.3.19) that determines the width of the test piece and ν_{xz} in (3.2.3.29) that determines the thickness,the contraction was too large. Therefore we had to look for a value of ν that maintained the volume V at the initial value V_0. We found by numerical experiments that $\nu_{xy} = \nu_{xz} = \nu_{plastic}$ must be a function of ε_{xx} (not a constant) of the form

$$\nu_{plastic}(\varepsilon_{xx}) = -0.09 \cdot \varepsilon_{xx}^3 + 0.066 \cdot \varepsilon_{xx}^2 - 0.0107 \cdot \varepsilon_{xx} + 0.492. \tag{3.2.3.30}$$

For this ν the volume V of the test piece was nearly equal to $V_0 = 720\ mm^3$, see Fig. 3.2.3.5. For the solution of the PDEs we take $\varepsilon_{xx} = \varepsilon_{xx,old}$, i.e. the value of the previous time step to have an explicit formula.

Now we must discuss how we can compare the measurements and the numerical simulation. From the tensile test machine we get a stress $\sigma_{machine}$ that is defined by

$$\sigma_{machine} = \frac{\text{force}_{machine}}{F_0}, \tag{3.2.3.31}$$

where $\text{force}_{machine}$ is the measured actual force to extend the test piece and F_0 is the original cross section (3.2.3.25). Note that this is a purely fictitious stress because it does not consider the contraction of the cross section when the test piece extends. The value $\sigma_{machine}$ is given as function of the strain ε_{80} with

$$\varepsilon_{80} = \frac{d_{80} - 80}{80}, \tag{3.2.3.32}$$

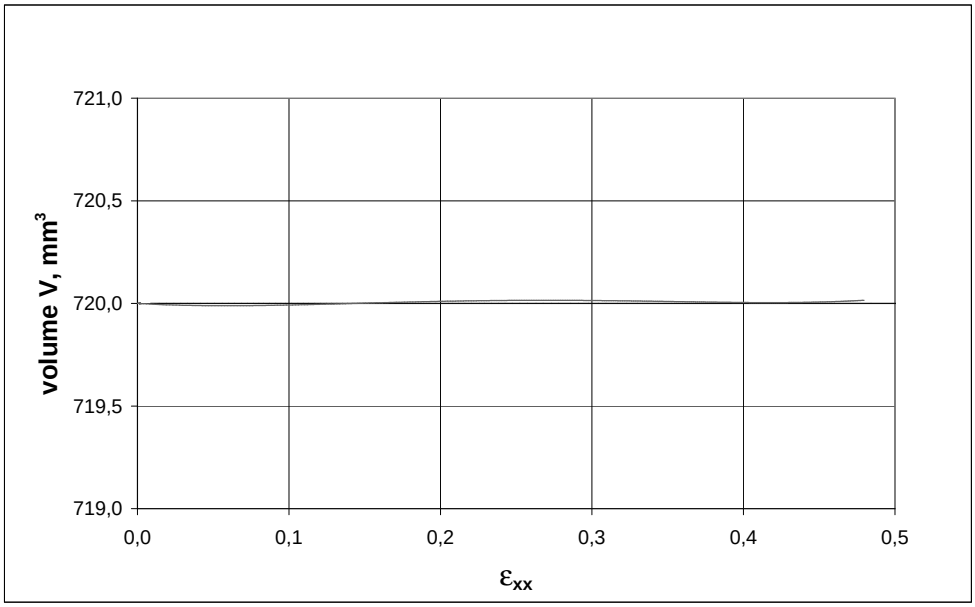


Figure 3.2.3.5: Volume V of the test piece for the simulation of the tensile test as function of ε_{xx} (3.2.3.21).

where d_{80} is the distance of the two control points A and B of Fig. 3.2.1.1 that have the original distance $80\ mm$. The value of d_{80} is measured by two claws that are fixed on the test piece.

Both values $\sigma_{machine}$ and ε_{80} are not directly available in the solution of the elastic/plastic PDEs. Therefore we compute

$$\sigma_{comp} = \frac{\sigma_{xx} \cdot F}{F_0}, \tag{3.2.3.33}$$

where σ_{xx} is the solution of the PDEs from the 3-equation model (const. on test piece) and F and F_0 are the actual and original cross section (3.2.3.25) with h from (3.2.3.29). A value $\varepsilon_{80,comp}$ is determined from

$$\varepsilon_{80,comp} = \frac{\overline{A'B'} - \overline{A'_0B'_0}}{\overline{A'_0B'_0}}. \tag{3.2.3.34}$$

A'_0, B'_0 are two grid points closest to A, B of Fig. 3.2.1.1, $\overline{A'_0B'_0}$ is the original distance of these grid points and $\overline{A'B'}$ the actual distance. For the comparison of measurement and simulation we put in the figures

$$\sigma_{machine} \Longleftrightarrow \sigma_{comp}, \qquad \varepsilon_{80} \Longleftrightarrow \varepsilon_{80,comp}. \tag{3.2.3.35}$$

We got early in the project from the IFU the measured values of a tensile test as a table of values of d_{80} and $\sigma_{machine}$. The corresponding curve $\sigma_{machine}$ over ε_{80} can be seen in different scales in Figs. 3.2.3.6 to 3.2.3.10. Unfortunately this measurement on which are based all our following conclusions and adaptutions cannot be found in the IFU report [12].

Now we want to discuss the results of the simulation. For the simplified test piece, lower part of Fig. 3.2.1.1, we used a triangular grid with 182×23 nodes in x-, y-direction. The maximal relative errors were for $v_x : 0.4 \cdot 10^{-12}$, $v_y : 0.7 \cdot 10^{-9}$, $\sigma_{xx} : 0.2 \cdot 10^{-13}$ so that we can say the results for a time step have spatial error close to zero. This is the result of the simple test piece for the 3-equation model where σ_{xx} is constant on the whole test piece. The maximal relative error for λ was $0.2 \cdot 10^{-13}$.

The speed of the tensile test machine $v_x = v_{x,test}$, see the BCs of Fig. 3.2.1.4, is different for the elastic and plastic part. Therefore we use different time steps: $\Delta t = \Delta t_{elastic}$ in the elastic region until a first node becomes plastic in the test steps. Then we do not execute the corresponding computation step but change Δt to a smaller value $\Delta t = \Delta t_{trans}$ to meet better the transition point and continue the computation until again a first node becomes plastic. then we set $\Delta t = \Delta t_{plastic}$ and increase the speed of the tensile test machine in $nincr$ steps from the elastic speed ($0.4 \cdot 10^{-2}\ mm/sec$) to the plastic speed $(0.4\ mm/sec)$ because a sudden change is not realistic. We continue the computation until the test piece "breaks" at $\sigma_{machine} = 995\ N/mm^2$ (stop the computation). We did numerical experiments to see how the result changes with the Δt's and selected the values so that the errors are far below 1%. We used $\Delta t_{elastic} = 5.0\,sec$, $\Delta t_{trans} = 0.5\,sec$, $\Delta t_{plastic} = 0.5\,sec$, $nincr = 5$. In Figs. 3.2.3.6–3.2.3.10 we present the result of the measurement and of the computation in different scales. In Fig. 3.2.3.11 is presented the relation between $\varepsilon_{80,comp}$ (3.2.3.34) and ε_{xx} (3.2.3.21) for the actual computation. Variable coefficients like $\delta_x(\varepsilon_{xx})$ in (3.2.3.23) or $\nu_{plastic}(\varepsilon_{xx})$ (3.2.3.30) must be taken as functions of ε_{xx} (which is used as $\varepsilon_{xx,old}$ for the solution of the PDEs) that we can generalize these relations.

The results shown above have been computed with the following set of coefficients and functions that appear in Y (3.2.3.3) and E_x (3.2.3.23):

$$
\begin{aligned}
Y_0 = 60\frac{N}{mm^2}, K = 215\frac{N}{mm^2}, \qquad & (3.2.3.36)\\
\varepsilon_0 = 1.02, \qquad n = 0.293,&\\
\delta_x(\varepsilon_{xx}) = \begin{cases} 0 \le \varepsilon_{xx} \le 0.08: & \delta = 0 \\ 0.08 \le \varepsilon_{xx} \le 0.36: & \delta = \frac{100}{70}\varepsilon_{xx} - \frac{8}{70} \\ \varepsilon_{xx} > 0.36: & \delta = \frac{325}{70}\varepsilon_{xx} - \frac{89}{70}, \end{cases}&\\
K_{x,0} = 26000\frac{N}{mm^2},&\\
\varepsilon_{x,0} = 1.02, n_x = 0.1.&
\end{aligned}
$$

The main result of this Section 3.2.3 is the "soft" E-module (3.2.3.23) that permits together with the PDEs (3.2.3.18)–(3.2.3.20) the perfect simulation of the tensile test for stainless steel sheet. The appropriate set of coefficients and functions is given by (3.2.3.30) and (3.2.3.36). This is not a mere playing with coefficients but it is the macroscopic expression for microscopic crystal perturbation

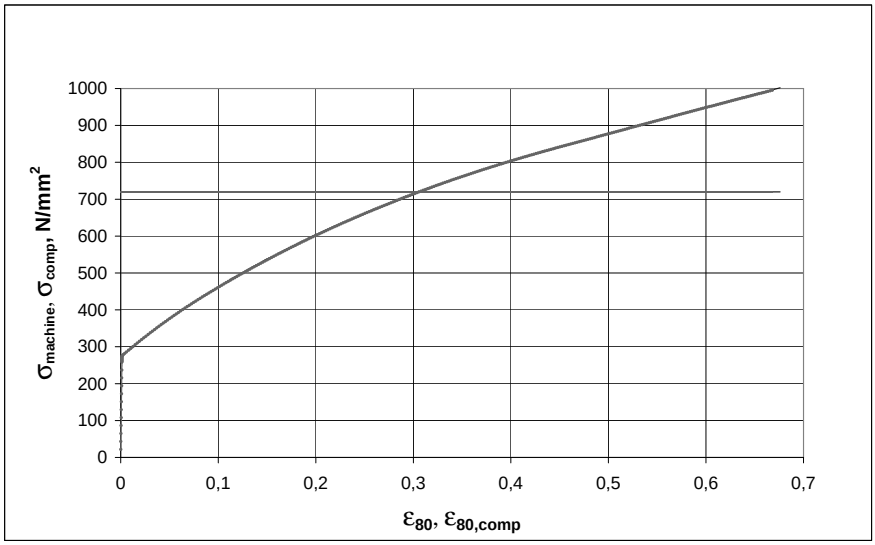


Figure 3.2.3.6: Stress/strain relation measured (solid line) and computed (dashed line), overview. Here both coincide.

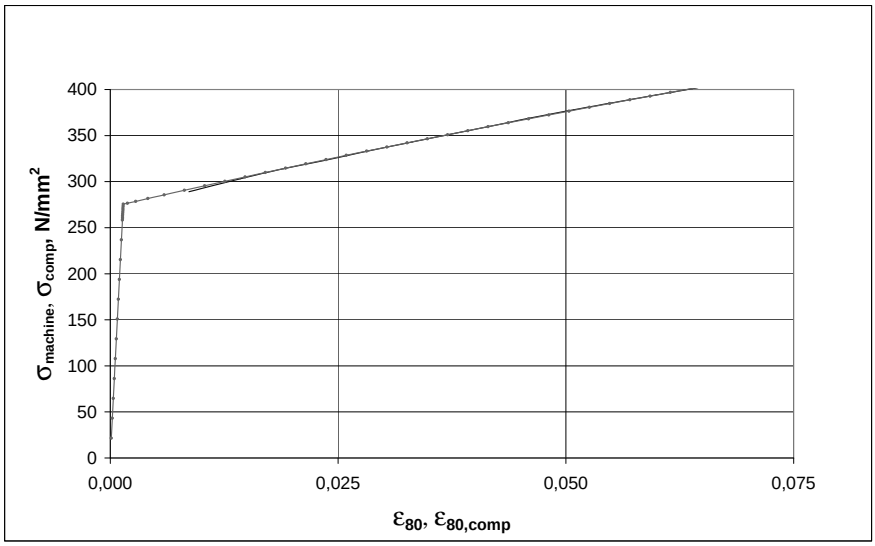


Figure 3.2.3.7: Results for small ε.

processes in the metal sheet when it is deformed plastically. It is the task of the metallurgist to analyze these results. Our task is to use this approach for the simulation of the manufacturing process of

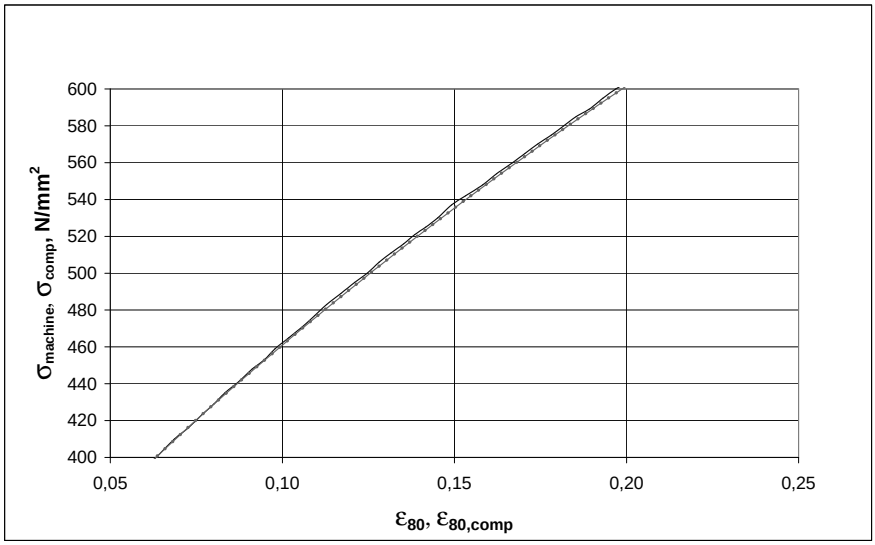


Figure 3.2.3.8: Results for small to medium ε.

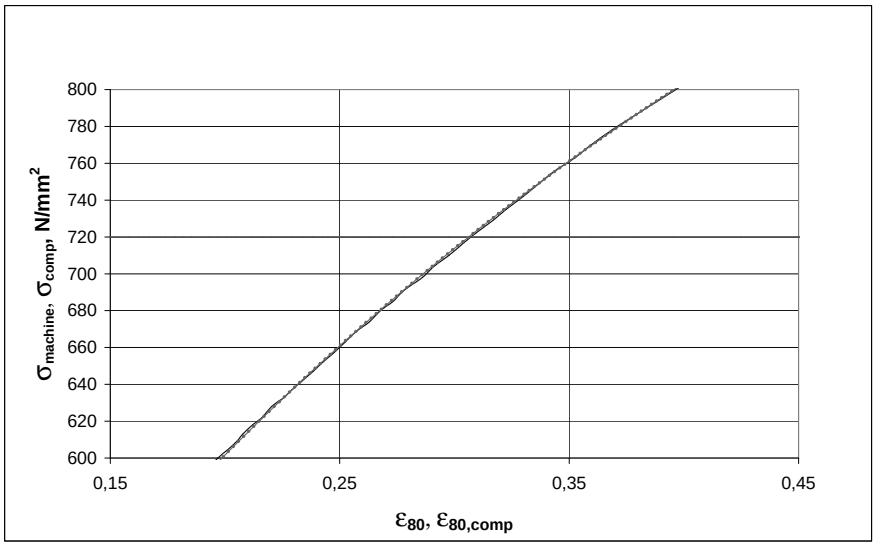


Figure 3.2.3.9: Results for medium ε.

metal bellows. This is discussed in the next section.
Remark: Unfortunately we had later to recognize that there was a false information or a misunderstanding: The values of the stresses for the measurements were based not on the original cross section

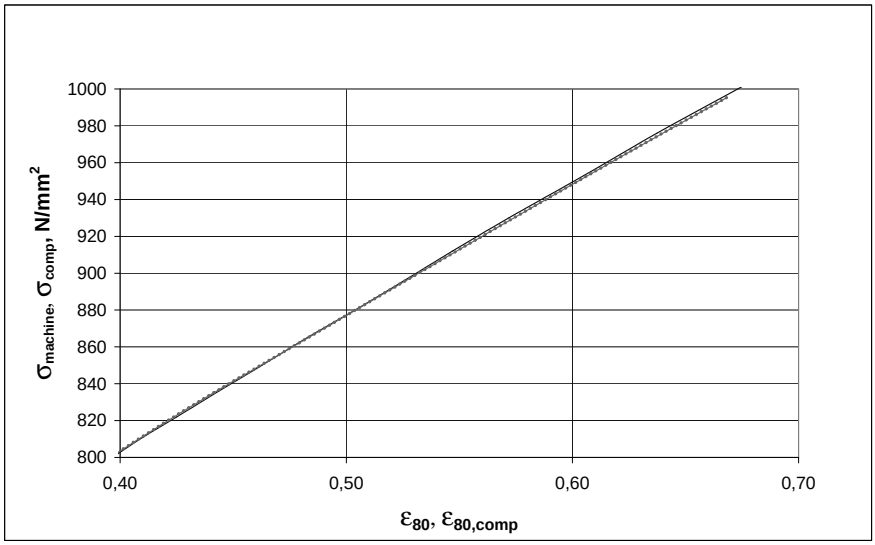


Figure 3.2.3.10: Results for large ε.

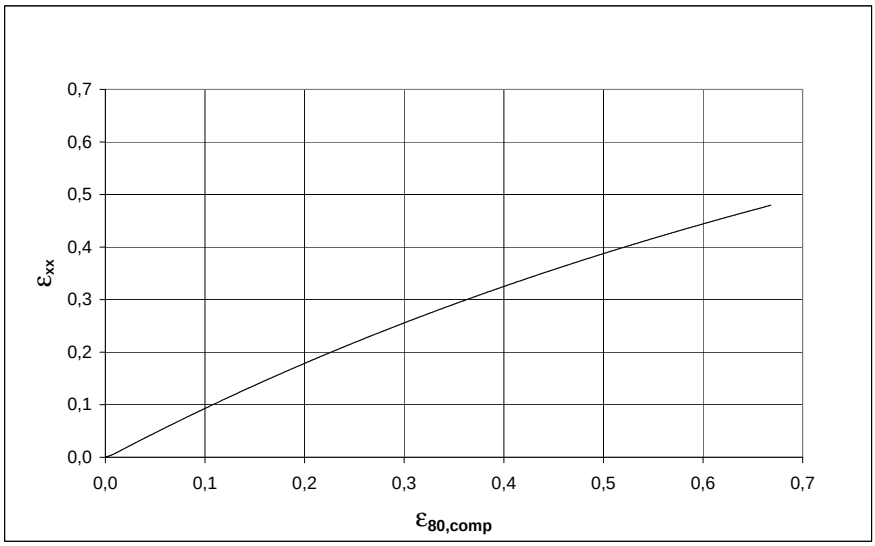


Figure 3.2.3.11: Relation between $\varepsilon_{80,comp}$ und ε_{xx}.

of the test piece (as we assumed above) but on the actual cross section, see the context of equation (3.2.4.46) and the new values there.

3.2.4 Simulation of the manufacturing of a single wave of the metal bellow

The manufacturing process of a wave of a metal bellow is explained in [12], Chapter 4. There the geometry of the tool, its position at the beginning and at the end are shown in [12] Figs. 4.1 and 4.5. In the experiment at IWKA that we want to simulate several waves have been formed. However, here we want to simulate the forming process of a single wave. Therefore we must introduce "artificial" BCs that cut the single wave out of a group of waves. Thus the initial configuration is that of Fig. 3.2.4.1. The manufacturing then is executed in the following way: In an initial blowing step a pressure is built up in the interior of the metal sheet pipe. Then the right tool and the right end of the pipe section move with a prescribed speed to the left until the right tool hits the left tool so that we have the situation of Fig. 3.2.4.2. Then, after a "resting" phase in which other waves are formed the pressure is taken off and then the tools, that are axially separated, are removed. Now in the spring-back phase the remaining inherent stresses are relieved until an equilibrium configuration is reached, see Fig. 3.2.4.3. The comparison of Figs. 3.2.4.2 and 3.2.4.3 shows the importance of the spring-back phase.

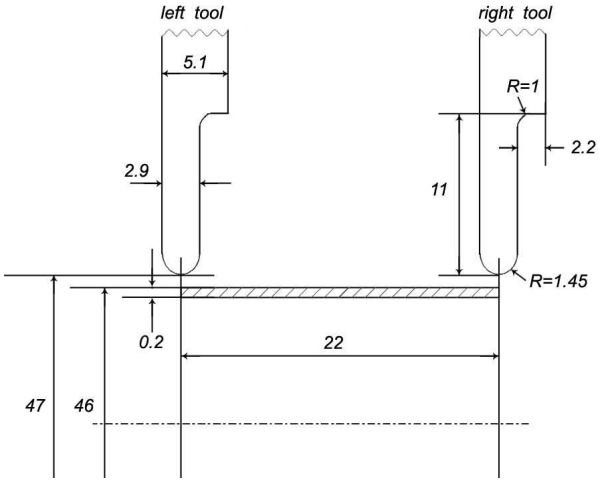


Figure 3.2.4.1: Initial position of tool. Scale is in mm.

Now we want to present the rotationally symmetric PDEs for elastic deformation in cylindrical coordinates, see Fig. 3.2.4.4. The variables are the displacement velocities v_z, v_r and the stresses σ_{zz}, σ_{rr}, $\sigma_{\varphi\varphi}$, σ_{rz}. Although we have for cylindrical symmetry no dependence on φ and there is $v_\varphi \equiv 0$, there is the stress $\sigma_{\varphi\varphi}$ in circumferential direction. The basic equations are given in [12], Chapter 3. The geometric definition of the orthotropic elastic deformation is given in [12](3.38) to (3.41), the deformation by the stresses (Hooke's law) is given in [12](3.44) to (3.48). We proceed like in Section 3.2.1 for the tensile test incrementally in time steps. We have with displacement u and increment Δu

$$v_z = \frac{\partial u_z}{\partial t} \approx \frac{\Delta u_z}{\Delta t}, \tag{3.2.4.1}$$

and we define

$$\Delta u_z = \Delta t \cdot v_z, \qquad \Delta u_r = \Delta t \cdot v_r. \tag{3.2.4.2}$$

After each time step we update the coordinates of the nodes with

$$z_{new} = z_{old} + \Delta u_z, \qquad r_{new} = r_{old} + \Delta u_r. \tag{3.2.4.3}$$

The index "old" denotes the result of the previous time step.

If we equate the expresions for ε_{zz} of [12](3.38) and [12](3.44) and go like in Section 3.2.1 to the

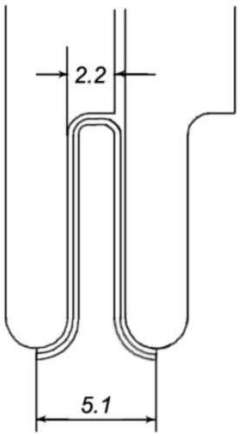


Figure 3.2.4.2: Closed tools.

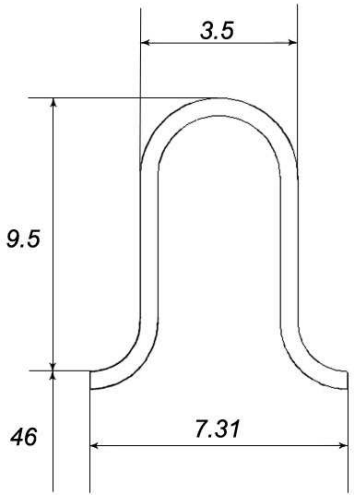


Figure 3.2.4.3: Final configuration of a wave.

incremental form we get

$$\frac{1}{E_z}[\sigma_{zz} - \sigma_{zz,old} - \nu_{\varphi z}(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}) - \nu_{zr}(\sigma_{rr} - \sigma_{rr,old})] - \Delta t\frac{\partial v_z}{\partial z} = 0 \tag{3.2.4.4}$$

Similarly we get from ε_{rr}, $\varepsilon_{\varphi\varphi}$, $\varepsilon_{rz} = \varepsilon_{zr}$

$$-\frac{\nu_{\varphi z}}{E_\varphi}(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}) - \frac{\nu_{rz}}{E_z}(\sigma_{zz} - \sigma_{zz,old}) + \frac{1}{E_r}(\sigma_{rr} - \sigma_{rr,old}) - \Delta t\frac{\partial v_r}{\partial r} = 0, \tag{3.2.4.5}$$

$$-\frac{\nu_{z\varphi}}{E_z}(\sigma_{zz} - \sigma_{zz,old}) + \frac{1}{E_\varphi}[\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old} - \nu_{\varphi r}(\sigma_{rr} - \sigma_{rr,old})] - \Delta t\frac{v_r}{r} = 0, \tag{3.2.4.6}$$

$$\frac{1}{2G_{rz}}(\sigma_{rz} - \sigma_{rz,old}) - \frac{1}{2}\Delta t(\frac{\partial v_r}{\partial z} + \frac{\partial v_z}{\partial r}) = 0. \tag{3.2.4.7}$$

We have $\nu_{\varphi z} = \nu_{z\varphi}$, $\nu_{rz} = \nu_{zr}$. From [12](3.4) we get the equilibrium equations with $\partial/\partial\varphi = 0$, $\sigma_{r\varphi} = 0$, $\sigma_{\varphi z} = 0$

$$\frac{\partial\sigma_{rr}}{\partial r} + \frac{\partial\sigma_{rz}}{\partial z} + \frac{1}{r}(\sigma_{rr} - \sigma_{\varphi\varphi}) = 0, \tag{3.2.4.8}$$

$$\frac{\partial\sigma_{rz}}{\partial r} + \frac{\partial\sigma_{zz}}{\partial z} + \frac{\sigma_{rz}}{r} = 0 \tag{3.2.4.9}$$

which need not be written in incremental form as explained after equ. (3.2.1.14). In Table 3.2.4.1 the sequence of the variables and equations is shown. The values for the coefficients in the equations are given in [12](3.79) to (3.81).

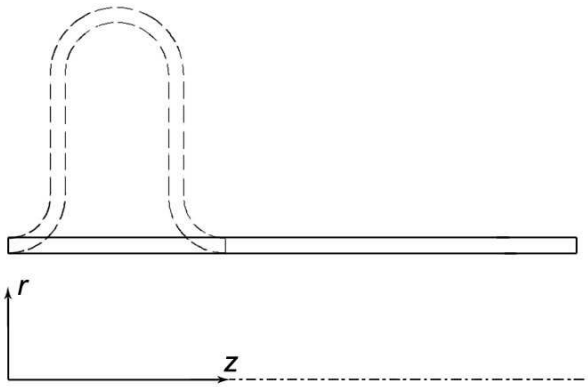


Figure 3.2.4.4: Coordinate system with initial and final configuration. φ is circumferential direction.

Table 3.2.4.1: Sequence of variables and equations for the manufacturing of the metal bellow.

no	variable	equation
1	v_z	(3.2.4.4)
2	v_r	(3.2.4.5)
3	σ_{zz}	(3.2.4.9)
4	σ_{rr}	(3.2.4.8)
5	$\sigma_{\varphi\varphi}$	(3.2.4.6)
6	σ_{rz}	(3.2.4.7)

For the plastic deformation we will use the same basic equations but now with variable "plastic" E-module, "plastic" Poisson ratio ν and non-linear Green strain term. This will be explained below. Here we want at first discuss the BCs that we use. These are the "same" in both cases. As we check in each node if it is elastic or plastic we may have adjacent nodes with different properties. This may occur above all in a bending of the metal sheet where in the inner neutral zone the stresses may be lower.

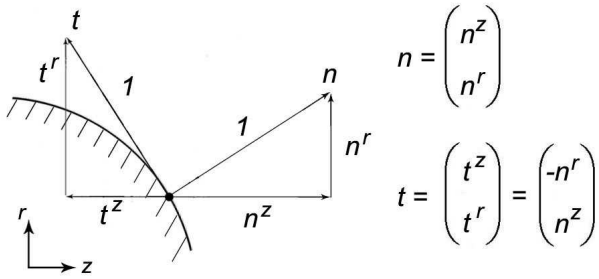


Figure 3.2.4.5: Illustration to normal n and tangent t.

In Fig. 3.2.4.5 we illustrate the normal n and tangent t with their components in cylindrical coordinates. We have

$$n = \begin{pmatrix} n^z \\ n^r \end{pmatrix}, \quad t = \begin{pmatrix} t^z \\ t^r \end{pmatrix} = \begin{pmatrix} -n^r \\ n^z \end{pmatrix}. \tag{3.2.4.10}$$

Similarly to (3.2.1.18) we have the stress at the surface of the metal sheet now in cylindrical coordinates

$$\sigma \quad = \begin{pmatrix} \sigma_{zz}n^z & +\sigma_{rz}n^r \\ \sigma_{rz}n^z & +\sigma_{rr}n^r \end{pmatrix} \quad \begin{matrix} z\text{-component} \\ r\text{-component} \end{matrix} \tag{3.2.4.11}$$

surface element with normal in $\quad z$-dir. $\quad r$-direction

The normal and tangential components of σ are

$$\sigma_n = \sigma^T \cdot n = \sigma_{zz}(n^z)^2 + 2\sigma_{rz}n^z n^r + \sigma_{rr}(n^r)^2 \tag{3.2.4.12}$$

$$\sigma_t = \sigma^T \cdot t = -\sigma_{zz}n^z n^r + \sigma_{rz}((n^z)^2 - (n^r)^2) + \sigma_{rr}n^z n^r \tag{3.2.4.13}$$

Fig. 3.2.4.6 shows symbolically the 4 boundaries. For the boundary (c) we must distinguish if a

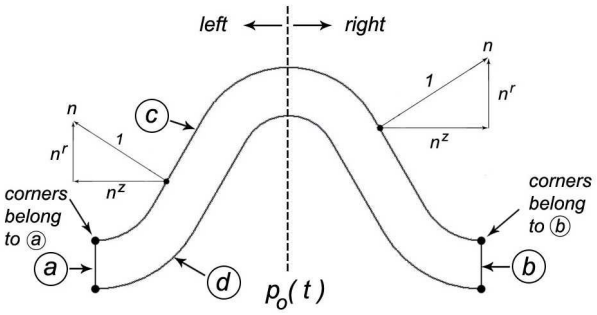


Figure 3.2.4.6: Symbolic illustration of the 4 boundaries a, b, c, d of the metal sheet.

node is "free", i.e. it is not forced by the tool, or if it is "forced", i.e. its movement is dictated by the tool. As the left half of the original metal sheet tube can contact only the left tool (see Fig. 3.2.4.1) we check only if its nodes "touch" the left tool, similarly we check the right half with the right tool. We proceed as follows: In a test step that is executed with the conditions of the previous computation step, we check if a node is elastic or plastic and for the nodes of boundary (c) also, if a node is free or forced. How we do it is explained below.

Boundary conditions at the boundaries (a) and (b) of Fig. 3.2.4.6:

$$\begin{aligned} v_z: \quad & v_z = 0 \text{ at (a)}, \\ & v_z = v_{tool} \text{ at (b)}, \\ v_r: \quad & PDE\,(3.2.4.7). \end{aligned}$$

A special case are the upper left and right corners. If in the initial blowing phase an upper corner touches the tool (observe in Fig. 3.2.4.1 that there is initially a gap of $0.5\ mm$ between the metal sheet and the tool), i.e. if in the test step

$$r_{corner} = r_{corner,old} + \Delta t \cdot v_{r,corner} \geqq 23.5 \tag{3.2.4.14}$$

we have $v_{r,corner} = 0$. This condition holds until in the phase where the pressure is reduced we have

$$\sigma_{n,corner} > 0. \tag{3.2.4.15}$$

Then the corner does no longer touch the tool, the explanation for >0 is given at the discussion of boundary (c).

$$\begin{aligned} \sigma_{zz}: &\quad PDE\ (3.2.4.4),\\ \sigma_{rr}: &\quad PDE\ (3.2.4.5),\\ \sigma_{\varphi\varphi}: &\quad PDE\ (3.2.4.6),\\ \sigma_{rz}: &\quad PDE\ (3.2.4.8). \end{aligned}$$

Before we present the BCs for boundary (c) of Fig. 3.2.4.6 we must explain how we check if a node is free or forced by the tool. The surface nodes of the boundary (c) of Fig. 3.2.4.6 are denoted by S_i. We assume that the left and right tool of Fig. 3.2.4.1 are also given pointwise by points T_j, see Fig. 3.2.4.7. Each node S_i of the left half of the surface (c) of the metal sheet searches (with a sophisticated algorithm) for the two nearest nodes T_{left} and T_{right} of the left tool, those of the right half of the metal sheet search at the right tool. Then we have the situation of Fig. 3.2.4.8. The normal n of node S_i right tool is determined orthogonal to the line $S_{i-1}S_{i+1}$ (we count nodes as shown in the figure). We determine the intersection A of the normal n with the line $T_{left}T_{right}$. The vector from S_i to A is $a \cdot n$. We have

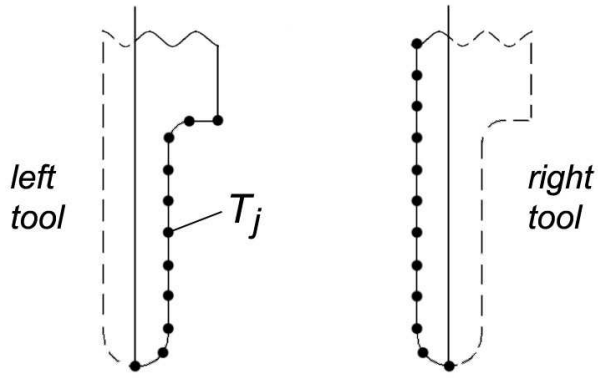


Figure 3.2.4.7: Pointwise representation of the tools.

$$\overline{AS_i} = a \cdot n = \begin{pmatrix} z_A - z_{S_i} \\ r_A - r_{S_i} \end{pmatrix} = \begin{pmatrix} a \cdot n^z \\ a \cdot n^r \end{pmatrix}, \tag{3.2.4.16}$$

and we determine a from

if $\quad |n^z| > |n^r| \quad$ then
$\quad a = (z_A - z_{S_i})/n^z$
else
$\quad a = (r_A - r_{S_i})/n^r$
endif

to avoid division by zero. If d denotes the thickness of the metal sheet (e.g. $0.2\ mm$) we denote

if $a > \alpha \cdot d$ then
  node S_i is free
else
  node S_i is forced
endif.

Above all if $a < 0$ S_i would be "in" the tool. We use e.g. $\alpha = 0.1$.

If at the other hand a node is forced, we must check if the node wants to lift off from the surface of the tool and become a free node. Our idea was: if the normal stress σ_n (3.2.4.12) becomes negative the node wants to lift off. However, when we investigated the sign of σ_n at a node that could not lift off because it is pressed against the tool, we saw that there σ_n is negative. From that we conclude that a node wants to lift off if $\sigma_n > 0$. To have some "security" we define:

$$\text{if} \qquad \sigma_n > 1 \qquad \text{the node becomes free.} \tag{3.2.4.17}$$

After each test step we determine for each node of the metal sheet if it is elastic or plastic (see below) and for the nodes of boundary (c) if the node is free or forced. The computation step then is executed with these properties. Likewise we determine after each computation step again these properties and execute the test step of the next time step with these new properties. Now we can present the BCs of boundary (c) of Fig. 3.2.4.6.

Boundary conditions at the boundary (c) of Fig. 3.2.4.6:

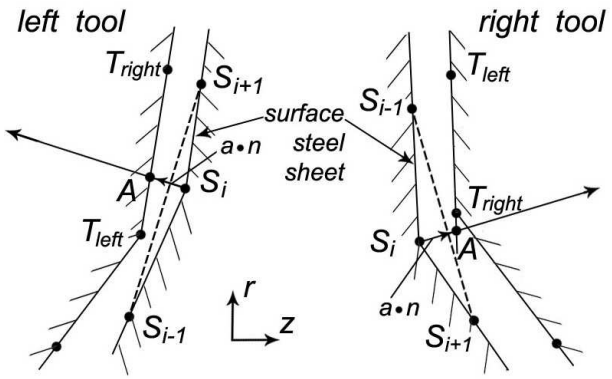


Figure 3.2.4.8: Illustration for the check for free or forced node.

node is free:

for	$\lvert n^z\rvert \geqq \lvert n^r\rvert$	$\lvert n^z\rvert < \lvert n^r\rvert$
$v_z:$	$PDE\ (3.2.4.4),$	$PDE\ (3.2.4.7),$
$v_r:$	$PDE\ (3.2.4.7),$	$PDE\ (3.2.4.5),$
$\sigma_{zz}:$	$\sigma_{zz}n^z + \sigma_{rz}n^r = 0,$	$PDE\ (3.2.4.4),$
$\sigma_{rr}:$	$PDE\ (3.2.4.5),$	$\sigma_{rz}n^z + \sigma_{rr}n^r = 0,$
$\sigma_{\varphi\varphi}:$	$PDE\ (3.2.4.6),$	$PDE\ (3.2.4.6),$
$\sigma_{rz}:$	$\sigma_{rz}n^z + \sigma_{rr}n^r = 0,$	$\sigma_{zz}n^z + \sigma_{rz}n^r = 0.$

The condition with "=0" set one of the components of the surface stress equal to zero, see (3.2.4.11).
node is forced:

for	$\lvert n^z\rvert \geqq \lvert n^r\rvert$	$\lvert n^z\rvert < \lvert n^r\rvert$
$v_z:$	$equ.\ (3.2.4.20),$	$PDE\ (3.2.4.7),$
$v_r:$	$PDE\ (3.2.4.7),$	$equ.\ (3.2.4.21),$
$\sigma_{zz}:$	$PDE\ (3.2.4.4),$	$PDE\ (3.2.4.4),$
$\sigma_{rr}:$	$PDE\ (3.2.4.5),$	$PDE\ (3.2.4.5),$
$\sigma_{\varphi\varphi}:$	$PDE\ (3.2.4.6),$	$PDE\ (3.2.4.6),$
$\sigma_{rz}:$	$equ.\ (3.2.4.24),$	$equ.\ (3.2.4.24).$

The condition for v_z in the left column and for v_r in the right column are the conditions that a forced node of the metal sheet surface (c) can move only along the tool surface. We want to derive this condition. The situation is illustrated in Fig. 3.2.4.9. We assume that the metal creeps "upwards" along the tool with velocity v with components v_z, v_r in z, r-direction at the left tool and v' at the right tool. At the left tool we have

$$\frac{v_r}{v_z} = \frac{n^z_{tool}}{-n^r_{tool}}$$

from which we get

$$v_z = -v_r \frac{n^r_{tool}}{n^z_{tool}} \quad \text{for } |n^z_{tool}| \geq |n^r_{tool}|, \tag{3.2.4.18}$$

$$v_r = -v_z \frac{n^z_{tool}}{n^z_{tool}} \quad \text{for } |n^z_{tool}| < |n^r_{tool}|. \tag{3.2.4.19}$$

At the right tool we have the creeping v' along the tool, but the tool moves with $v_{z,tool}$ to the left. We have

$$\frac{v'_r}{-v'_z} = \frac{-n^z_{tool}}{-n^r_{tool}}$$

from which we get

$$v_z^{'} = -v_r^{'}\frac{n_{tool}^r}{n_{tool}^z} \quad \text{for } |n_{tool}^z| \geqq |n_{tool}^r|,$$

$$v_r^{'} = -v_z^{'}\frac{n_{tool}^z}{n_{tool}^z} \quad \text{for } |n_{tool}^r| < |n_{tool}^r|.$$

We have $v_z = v_{z,tool} + v_z^{'}$ or $v_z^{'} = v_z - v_{z,tool}$ and we have $v_r = v_r^{'}$. So we get

$$v_z = v_{z,tool} - v_r\frac{n_{tool}^r}{n_{tool}^z} \quad \text{for } |n_{tool}^z| \geqq |n_{tool}^r|, \tag{3.2.4.20}$$

$$v_r = -(v_z - v_{z,tool})\frac{n_{tool}^z}{n_{tool}^r} \quad \text{for } |n_{tool}^r| < |n_{tool}^r|. \tag{3.2.4.21}$$

If we put $v_{z,tool} = 0$ we get the equations (3.2.4.19) and (3.2.4.20). So the equations (3.2.4.21) and (3.2.4.22) hold for the left tool $(v_{z,tool} = 0)$ and for the right tool.

The condition for σ_{rz} for the forced node of boundary (c) is the condition for the tangential component σ_t (3.2.4.13). If we neglect friction we have $\sigma_t = 0$. However, there is eventually considerable frictional stress between tool and metal sheet for forced nodes. From [12], Section 3.6.2, we have the frictional stress

$$\sigma_F = m_F\sigma_n \tag{3.2.4.22}$$

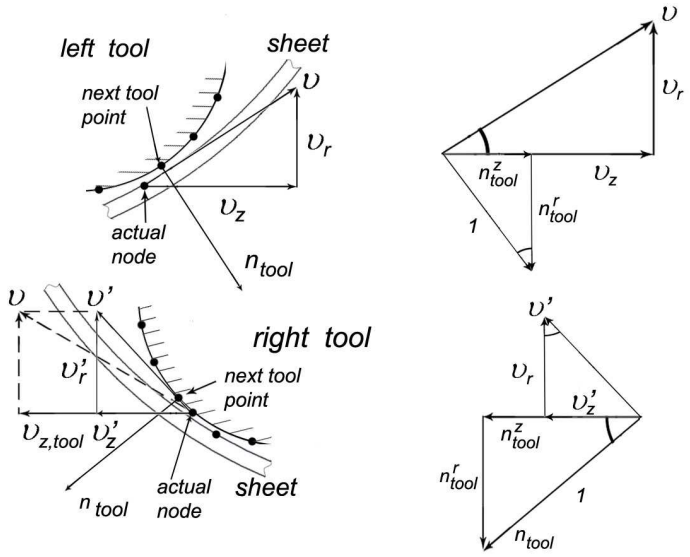


Figure 3.2.4.9: Illustration for the creeping of the metal sheet along the left tool (above) and right tool (below).

with the friction coefficient $m_F = m_{static}$ (=0.08 for metal sheet with oiled surface) for static friction (this is in reality not a "friction" but only a frictional stress because there is no relative movement) and $m_F = m_{kin}$(=0.06) for kinetic (or sliding) friction. The boundary condition then is

$$\sigma_t = \pm m_F \sigma_{n,old} \tag{3.2.4.23}$$

with σ_t (3.2.4.13) and $\sigma_{n,old}$ (3.2.4.12) where "old" means that σ_n is computed from the data of the previous time step. Thus the friction acts like an external force.
There are now two problems: The first one is if we have static or kinetic friction. For kinetic friction the metal sheet creeps along the tool, i.e. has a relative movement to the tool. As the left tool does not move and the right tool moves only in the z-direction we have a relative movement if a forced node of the surface (c) has $v_r \neq 0$. So we take

$$\begin{array}{lrcl} \text{for } v_r \neq 0: & m_F & = & m_{kin}, \\ \text{else}: & m_F & = & m_{static}. \end{array}$$

The second problem is the sign "+" or "-" in (3.2.4.24) which depends on the sign of σ_t and of $\sigma_{n,old}$. As mentioned in the context of equ. (3.2.4.18) at a forced node $\sigma_n < 0$. Therefore we use

$$\sigma_t = -m_F \sigma_{n,old} \tag{3.2.4.24}$$

with σ_t from (3.2.4.13) and $\sigma_{n,old}$ from (3.2.4.12), but for the values of the previous time step. For m_F see the context of equ. (3.2.4.23).

The numerical behaviour was much more critical than we expected—and our error estimate told us what we had to do to get an accurate solution—so we feared that the metal sheet that creeps along the surface of the tool according to Fig. 3.2.4.9, would leave the tool. Remember that the sheet and tool surface polygons are composed from straight lines between the nodes. Therefore we used the following simplified method: In the test step we check if a node that was free attaches to the tool according to Fig. 3.2.4.8. Then we prescribe in the computation step the values for v_z, v_r so that the node just meets the tool. Then this node is a forced node for the following time steps, i.e. we have $v_z = 0$ left, $v_z = v_{tool}$ right, and we have in both cases $v_r = 0$. So the node adheres to the tool. Therefore we have now for attaching and attached nodes the following BCs at boundary (c):

node attaching	**node attached**
$v_z : v_z = v_{z,test}$	$v_z = 0\ (left),\ v_z = v_{tool}\ (right)$
$v_r : v_r = v_{r,test}$	$v_r = 0$
$\sigma_{zz} : PDE\ (3.2.4.4)$	$PDE\ (3.2.4.4)$
$\sigma_{rr} : PDE\ (3.2.4.5)$	$PDE\ (3.2.4.5)$
$\sigma_{\varphi\varphi} : PDE\ (3.2.4.6)$	$PDE\ (3.2.4.6)$
$\sigma_{rz} : PDE\ (3.2.4.7)$	$PDE\ (3.2.4.7)$.

As the metal sheet nodes are now attached to the tool there is no relative movement and thus no friction.

Boundary conditions at the boundary (d) of Fig. 3.2.4.6: Here we have normal stress $\sigma_n = -p$ and tangential stress $\sigma_t = 0$, with σ_n, σ_t from (3.2.4.12), (3.2.4.13). If $|n^z| > |n^r|$ (3.2.4.12) is a

good equation for σ_{zz}, else for σ_{rr}, in both cases (3.2.4.13) can be used for σ_{rz}, except if $|n^z| = |n^r|$, where σ_{rz} drops out of the equation. Therefore we subdivide the application of (3.2.4.12), (3.2.4.13) into 3 regions, depending on $|n^z|$ and $|n^r|$. We have introduced the same procedure in Section 3.3.1 in the context of Fig. 3.3.1.6 (here n^x, n^y). There the 3 sectors I, II, III are defined (this was at an earlier time, therefore it is explained there). Thus we have the following BCs at boundary (d):

	sector I	sector II	sector III
$v_z:$	$PDE\ (3.2.4.4),$	$PDE\ (3.2.4.4),$	$PDE\ (3.2.4.7),$
$v_r:$	$PDE\ (3.2.4.7),$	$PDE\ (3.2.4.7),$	$PDE\ (3.2.4.5),$
$\sigma_{zz}:$	$\sigma_n = -p_0,$	$\sigma_t = 0,$	$PDE\ (3.2.4.4),$
$\sigma_{rr}:$	$PDE\ (3.2.4.5),$	$PDE\ (3.2.4.5),$	$\sigma_n = -p_0,$
$\sigma_{\varphi\varphi}:$	$PDE\ (3.2.4.6),$	$PDE\ (3.2.4.6),$	$PDE\ (3.2.4.6),$
$\sigma_{rz}:$	$\sigma_t = 0,$	$\sigma_n = -p_0,$	$\sigma_t = 0.$

Here σ_n, σ_t are from (3.2.4.12), (3.2.4.13). The interior pressure $p_0(t)$ is approximated by [12](4.1)–(4.3), the approximated function is shown in Fig. [12](4.3).

Up to here we have discussed the BCs that hold for the initial blowing phase, the phase where the right tool moves, the rest phase where the pressure $p_0(t)$ changes only slightly and finally up to the discharging of the pressure down to $p_0 = 0$. Until then the tools are still closed and the formed metal sheet is still fixed by the tools. Now the axially separated tools are opened and the bellow wave expands by the remaining inherent residual stresses until an equilibrium state is reached. This is the essential spring-back phase. We stop the computation if with fixed left end the right end of the wave does no longer move, i.e. if

$$|v_{z,max}| < 10^{-3} \tag{3.2.4.25}$$

where the max is taken over the nodes of the right end (boundary (b) of Fig. 3.2.4.6).
However, the stainless steel has after the forming process locally different properties E and ν because the internal crystal structrure has been strongly disturbed by the deformation. The disturbance depends on the total strain that the metal has passed through. Upon our request the IFU did a series of experiments with tensile test pieces that had undergone different strains ε. In the tensile test machine the strain is measured as

$$\varepsilon_{80} = \frac{l - 80}{80} \tag{3.2.4.26}$$

where l is the length between two test points A und B of the test piece whose original distance is $80\ mm$, see Fig. 3.2.1.1. However, this value ε_{80} is not available in the simulation process. Here we measure the non-linear strain ε_{xx} by equ. (3.2.3.8). From the simulation of the tensile test we have the relation of ε_{xx} as function of ε_{80} given in Fig. 3.2.3.11. In [12], Section 5.1.1, the results of the measurements of the different material parameters for different values of the initial strain ε_{80} are given.

For the spring-back expansion of the bellow wave we use the elastic equations of Table 3.2.4.1, but now with reduced material parameters. We assume the bellow wave fixed in z-direction at the

left end and the other 3 surfaces are free surfaces. The BCs at the 4 boundaries (a)-(d) of Fig. 3.2.4.6 for the spring-back computation are:

Boundary conditions at the boundaries (a) and (b):

$$
\begin{array}{ll}
v_z = 0 \quad \text{at (a)}, & v_z : PDE\ (3.2.4.4) \quad \text{at (b)}, \\
v_r : PDE\ (3.2.4.7), & \\
\sigma_{zz} = 0, & \\
\sigma_{rr} : PDE\ (3.2.4.5), & \\
\sigma_{\varphi\varphi} : PDE\ (3.2.4.6), & \\
\sigma_{rz} = 0. &
\end{array}
$$

The conditions $\sigma_{zz} = 0$ and $\sigma_{rz} = 0$ mean that the z-component and r-component of the stress are zero, see (3.2.4.11) for $n^z = \pm 1$, $n^r = 0$.

Boundary conditions at the boundaries (c) and (d):

$$
\begin{array}{rll}
\text{for} & |n^z| \geqq |n^r| & |n^z| < |n^r| \\
v_z : & PDE\ (3.2.4.4), & PDE\ (3.2.4.7), \\
v_r : & PDE\ (3.2.4.7), & PDE\ (3.2.4.5), \\
\sigma_{zz} : & \sigma_{zz} n^z + \sigma_{rz} n^r = 0, & PDE\ (3.2.4.4), \\
\sigma_{rr} : & PDE\ (3.2.4.5), & \sigma_{rz} n^z + \sigma_{rr} n^r = 0, \\
\sigma_{\varphi\varphi} : & PDE\ (3.2.4.6), & PDE\ (3.2.4.6), \\
\sigma_{rz} : & \sigma_{rz} n^z + \sigma_{rr} n^r = 0, & \sigma_{zz} n^z + \sigma_{rz} n^r = 0.
\end{array}
$$

Here the relations with n^z, n^r mean again that the corresponding components of σ are zero, see (3.2.4.11).

Here we want to summarize again the pre-requisites for a successful computation of the spring-back. At first we must have the correct values of the stresses σ_{zz}, σ_{rr}, $\sigma_{\varphi\varphi}$, σ_{rz} at the end of the forming process. These are calculated with the "soft" E-module and ν determined from the simulation of the tensile test. These parameters depend on the non-linear strains $\varepsilon_{\alpha,\beta}$ that are individual for each node. Then, for the spring-back expansion, we have again individual parameters E and ν that depend also on the local $\varepsilon_{\alpha,\beta}$, i.e. the previous deformation history. So we recognize that the result of the spring-back calculation is an extremely hard and sensitive test for the quality of the material parmeters.

Up to here we have discussed the problem with elastic PDEs in mind. Now we want to discuss the $\underline{plastic}$ PDEs with the "soft" E-module and corresponding function ν. In Section 3.2.3 we have seen that in the 1-D case E and ν are functions of ε_{xx}, i.e. of the deformation history. For the simulation of the manufacturing of the metal bellow we extend this approach to cylindrical coordinates. With the displacements u_z, u_r in z, r-direction we get from [12](3.8)–(3.12) the non-linear Cauchy-Green

strains for rotational symmetry ($u_\varphi = 0$, $\partial/\partial\varphi = 0$):

$$\varepsilon_{zz} = \frac{\partial u_z}{\partial z} + \frac{1}{2}(\frac{\partial u_z}{\partial r})^2 + \frac{1}{2}(\frac{\partial u_z}{\partial z})^2, \tag{3.2.4.27}$$

$$\varepsilon_{rr} = \frac{\partial u_r}{\partial r} + \frac{1}{2}(\frac{\partial u_r}{\partial r})^2 + \frac{1}{2}(\frac{\partial u_r}{\partial z})^2, \tag{3.2.4.28}$$

$$\varepsilon_{\varphi\varphi} = \frac{u_r}{r} + \frac{1}{2}(\frac{u_r}{r})^2, \tag{3.2.4.29}$$

$$\varepsilon_{rz} = \frac{1}{2}(\frac{\partial u_z}{\partial r} + \frac{\partial u_r}{\partial z}) + \frac{1}{2}(\frac{\partial u_r}{\partial r}\cdot\frac{\partial u_z}{\partial r} + \frac{\partial u_r}{\partial z}\cdot\frac{\partial u_z}{\partial z}). \tag{3.2.4.30}$$

For the definition of the "soft" E-module for rotationally symmetric cylindrical coordinates we generalize relation (3.2.3.23) in the following way:

$$E_z = (1-\delta_x(\varepsilon_{zz}))K_{x,0}\, n_x(\varepsilon_{x,0}+\varepsilon_{zz})^{n_x-1}, \tag{3.2.4.31}$$

$$E_r = (1-\delta_x(\varepsilon_{rr}))K_{x,0}\, n_x(\varepsilon_{x,0}+\varepsilon_{rr})^{n_x-1}, \tag{3.2.4.32}$$

$$E_\varphi = (1-\delta_x(\varepsilon_{\varphi\varphi}))K_{x,0}\, n_x(\varepsilon_{x,0}+\varepsilon_{\varphi\varphi})^{n_x-1}. \tag{3.2.4.33}$$

Here $K_{x,0}$, n_x, $\varepsilon_{x,0}$ are the values of (3.2.3.34), $\delta_x(\varepsilon_{zz})$ is the function given in (3.2.3.34) where ε_{xx} is replaced by ε_{zz} and similarly for $\delta_x(\varepsilon_{rr})$, $\delta_x(\varepsilon_{\varphi\varphi})$. For ε we always use the value ε_{old}, i.e. the value of the previous time step to get an explicit expression for the E's.
For G_{rz} we use the relation [12](3.81)

$$G_{rz} = \frac{E_r + E_z}{4(1+\nu_{rz})}. \tag{3.2.4.34}$$

Here we must at first clarify what means ν_{rz}, see (3.2.4.36) below. In Section 3.2.3 we have seen that Poisson's ratio $\nu_{plastic}$ (here we do not differ between ν_{xy} and ν_{xz}) for plastic deformation depends on ε_{xx} as given in (3.2.3.29). We call this ν_{xx} because it depends on ε_{xx}. We generalize this relation and define

$$\nu_{zz} = \nu(\varepsilon_{zz}), \quad \nu_{rr} = \nu(\varepsilon_{rr}), \quad \nu_{\varphi\varphi} = \nu(\varepsilon_{\varphi\varphi}). \tag{3.2.4.35}$$

This means that we take e.g. for ν_{zz} the value of $\nu_{plastic}$ of (3.2.3.29) and replace ε_{xx} by ε_{zz} etc. Here we also use for ε the value ε_{old} of the previous time step. In G_{rz} (3.2.1.36) we use:

$$\text{for} \quad G_{rz} \quad \text{take} \qquad \nu_{rz} = min(\nu_{rr}, \nu_{zz}) \tag{3.2.4.36}$$

which means that we take the value of ν for minimal transverse contraction.

Like for the tensile test we have for plastic deformation large strains so that we must take the non-linear Cauchy-Green strain. If we take for ε_{zz} equ. (3.2.4.29) and go with (3.2.4.2) to the incremental form, then in (3.2.4.4) the negative last term (linear strain) is replaced and we get (with e.g.

$\nu_{\varphi z}/E_z = \nu_{z\varphi}/E_\varphi$, see [12](2.18))

$$\begin{aligned} \frac{1}{E_z}(\sigma_{zz} - \sigma_{zz,old}) - \frac{\nu_{\varphi\varphi}}{E_\varphi}(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}) - \\ -\frac{\nu_{rr}}{E_r}(\sigma_{rr} - \sigma_{rr,old}) - \\ -\Delta t\frac{\partial \nu_z}{\partial z} - \frac{1}{2}\Delta t^2((\frac{\partial \nu_z}{\partial r})^2 + (\frac{\partial \nu_z}{\partial z})^2) &= 0. \end{aligned} \tag{3.2.4.37}$$

In the same way we get for ε_{rr}, $\varepsilon_{\varphi\varphi}$, ε_{rz} from (3.2.4.30)–(3.2.4.32)

$$\begin{aligned} \frac{\nu_{\varphi\varphi}}{E_\varphi}(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}) - \frac{\nu_{zz}}{E_z}(\sigma_{zz} - \sigma_{zz,old}) + \frac{1}{E_r}(\sigma_{rr} - \sigma_{rr,old}) - \\ -\Delta t\frac{\partial \nu_r}{\partial r} - \frac{1}{2}\Delta t^2((\frac{\partial \nu_r}{\partial r})^2 + (\frac{\partial \nu_r}{\partial z})^2) = 0, \end{aligned} \tag{3.2.4.38}$$

$$\begin{aligned} -\frac{\nu_{zz}}{E_z}(\sigma_{zz} - \sigma_{zz,old}) + \frac{1}{E_\varphi}(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}) - \\ -\frac{\nu_{rr}}{E_r}(\sigma_{rr} - \sigma_{rr,old}) - \Delta t\frac{\nu_r}{r} - \frac{1}{2}\Delta t^2(\frac{\nu_r}{r})^2 = 0, \end{aligned} \tag{3.2.4.39}$$

$$\begin{aligned} \frac{2(1 + min(\nu_{rr}, \nu_{zz}))}{E_r + E_z}(\sigma_{rr} - \sigma_{rr,old}) - -\frac{1}{2}\Delta t(\frac{\partial \nu_z}{\partial r} + \frac{\partial \nu_r}{\partial z}) - \\ -\frac{1}{2}\Delta t^2(\frac{\partial \nu_r}{\partial r} \cdot \frac{\partial \nu_z}{\partial r} + \frac{\partial \nu_r}{\partial z} \cdot \frac{\partial \nu_z}{\partial z}) = 0. \end{aligned} \tag{3.2.4.40}$$

Here the E's are from (3.2.4.33)–(3.2.4.35) and the ν's are from (3.2.4.37). The equilibrium equations (3.2.4.8), (3.2.4.9) are the same.

For the boundary conditions of the 4 boundaries (a)-(d) of Fig. 3.2.4.6 that have been presented above we must now replace the elastic PDEs (3.2.4.4)–(3.2.4.7) by the plastic PDEs (3.2.4.39)–(3.2.4.42).

The solution of the PDEs is executed in the following way, similar to the simulation of the tensile test: For each time step at first a test step is executed with the conditions that were given after the previous computation step, i.e. a node that was recognized there as elastic, is solved with the elastic PDEs, else with the plastic PDEs and a node of boundary (c) of Fig. 3.2.4.6 that was forced is solved with the BCs for a forced node, else with the BCs for a free node. After the test step we check if a node is elastic or plastic and if a node of boundary (c) is forced or free and then the computation step is executed with the corresponding properties. After the computation step for those nodes that have been solved with the plastic equations a new value of the plasticity parameter is computed (see below). Then again the check is executed if a node is elastic or plastic and if a node of boundary (c) is forced or free. Then with these properties the test step for the next time step is executed.

To check if a node is elastic or plastic we need the equivalent stress [12](3.50) for cylindrical coordinates:

$$\overline{\sigma} = \sqrt{\sigma_{rr}^2 + \sigma_{\varphi\varphi}^2 + \sigma_{zz}^2 - \sigma_{rr}\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi}\sigma_{zz} - \sigma_{zz}\sigma_{rr} + 3\sigma_{rz}^2}. \tag{3.2.4.41}$$

The yield stress Y is given by (3.2.3.3)

$$Y = Y_0 + K(\varepsilon_0 + \lambda)^n, \tag{3.2.4.42}$$

and its coefficients by (3.2.3.34). We define:

$$\begin{array}{ll} \text{if} & \text{for a node} \quad \overline{\sigma} \geqq Y \qquad \text{it is plastic,} \\ \text{else} & \text{it is elastic.} \end{array}$$

We check as mentioned above after each test step and again after each computation step, if a node is elastic or plastic.

As long as all nodes are elastic, the plasticity parameter λ is zero. If a node is plastic we compute a new value of λ from the PDE [12](3.57) in cylindrical coordinates

$$\begin{aligned} \frac{\partial\lambda}{\partial t} + \frac{\partial\lambda}{\partial r}\nu_r + \frac{\partial\lambda}{\partial z}\nu_z = \frac{1}{Kn(\varepsilon_0+\lambda)^{n-1}}\frac{1}{2\overline{\sigma}} \cdot \\ \cdot \Big\{ 2\sigma_{rr}\frac{\partial\sigma_{rr}}{\partial t} + 2\sigma_{\varphi\varphi}\frac{\partial\sigma_{\varphi\varphi}}{\partial t} + 2\sigma_{zz}\frac{\partial\sigma_{zz}}{\partial t} + 6\sigma_{rz}\frac{\partial\sigma_{rz}}{\partial t} - \\ -\sigma_{rr}\frac{\partial\sigma_{\varphi\varphi}}{\partial t} - \sigma_{\varphi\varphi}\frac{\partial\sigma_{rr}}{\partial t} - \sigma_{rr}\frac{\partial\sigma_{zz}}{\partial t} - \sigma_{zz}\frac{\partial\sigma_{rr}}{\partial t} - \sigma_{\varphi\varphi}\frac{\partial\sigma_{zz}}{\partial t} - \sigma_{zz}\frac{\partial\sigma_{\varphi\varphi}}{\partial t} + \\ + \Big[2\sigma_{rr}\frac{\partial\sigma_{rr}}{\partial r} + 2\sigma_{\varphi\varphi}\frac{\partial\sigma_{\varphi\varphi}}{\partial r} + 2\sigma_{zz}\frac{\partial\sigma_{zz}}{\partial r} + 6\sigma_{rz}\frac{\partial\sigma_{rz}}{\partial r} - \\ -\sigma_{rr}\frac{\partial\sigma_{\varphi\varphi}}{\partial r} - \sigma_{\varphi\varphi}\frac{\partial\sigma_{rr}}{\partial r} - \sigma_{rr}\frac{\partial\sigma_{zz}}{\partial r} - \sigma_{zz}\frac{\partial\sigma_{rr}}{\partial r} - \\ -\sigma_{\varphi\varphi}\frac{\partial\sigma_{zz}}{\partial r} - \sigma_{zz}\frac{\partial\sigma_{\varphi\varphi}}{\partial r} \Big]\nu_r + \\ + \Big[2\sigma_{rr}\frac{\partial\sigma_{rr}}{\partial z} + 2\sigma_{\varphi\varphi}\frac{\partial\sigma_{\varphi\varphi}}{\partial z} + 2\sigma_{zz}\frac{\partial\sigma_{zz}}{\partial z} + 6\sigma_{rz}\frac{\partial\sigma_{rz}}{\partial z} - \\ -\sigma_{rr}\frac{\partial\sigma_{\varphi\varphi}}{\partial z} - \sigma_{\varphi\varphi}\frac{\partial\sigma_{rr}}{\partial z} - \sigma_{rr}\frac{\partial\sigma_{zz}}{\partial z} - \sigma_{zz}\frac{\partial\sigma_{rr}}{\partial z} - \\ -\sigma_{\varphi\varphi}\frac{\partial\sigma_{zz}}{\partial z} - \sigma_{zz}\frac{\partial\sigma_{\varphi\varphi}}{\partial z} \Big]\nu_z \Big\}. \end{aligned} \tag{3.2.4.43}$$

Here time derivatives are discretized in the following way:

$$\frac{\partial\lambda}{\partial t} = \frac{\lambda - \lambda_{old}}{\Delta t}, \qquad \frac{\partial\sigma_{rr}}{\partial t} = \frac{\sigma_{rr} - \sigma_{rr,old}}{\Delta t} \tag{3.2.4.44}$$

and similarly the other time derivatives. The index "old" denotes the value of the previous time step. In (3.2.4.46) the only unknown function is λ, the ν's and σ's are the result of the actual computation step. This means that the new value of λ is then used in the next time step. This is exactly the procedure that has been applied in the simulation of the tensile test where the parameters for the determination of E und ν have been determined as function of ε.

So far everything was prepared for the computation and we implemented the corresponding equations into FDEM. For a certain number of time steps everything went well, the errors were small, e.g. below 1% for a certain grid, and they became smaller for finer grids. However, after a certain time the errors started to grow continuously, the steel tube started to oscillate: bubble and hole alternating in the middle of the tube, and the values of the stresses and displacement velocities got unreasonable. We changed the BCs, changed the grid, changed the time step Δt: the situation became worse with smaller time step. We limited the E-module to a lower limit of e.g. $40000\,N/mm^2$, i.e. we made the steel harder, and then we could compute until the right tool started to move and the errors were small at that time. The steel tube had extended in the middle by roughly $0.4\,mm$. We knew from the error estimate that the solution was okay. However, when the tool started to move the metal tube buckled, and it buckled inwards and not as we expected from the manufacturing process outwards to form a wave of the bellow. Buckling means bifurcation of the solution, there is no longer a unique solution. Buckling inwards simply occured because it is energetically cheaper than buckling outwards. So the harder steel is not the solution of the problem. As mentioned above we shift the grid by the displacement after each time step. So the computed solution is the solution for the old grid and not for the new one. This is a type of explicit procedure and we assumed that this is the reason for the oscillations. Therefore we introduced the grid iteration:

If k is the iteration index, we check if

$$\frac{\left|v_r^k - v_r^{k-1}\right|}{v_r^k} < \varepsilon_{grid} \tag{3.2.4.45}$$

and if not, we make a next iteration, i.e. compute a new solution,
on the new grid k.

We also can prescribe a min. and max. number of iterations. So finally we have a solution on the new grid, which means also with respect to the grid we have a fully implicit procedure.

If we then repeated the computation everything was as before: same solution, same small error—until a certain time step. Then at the next time step the grid iteration diverged: the tube started to oscillate during the iteration, bubble, hole, bubble etc. until the solution became nonsense. Again with a larger E-module we could compute with small error until the tool started, then occured buckling as mentioned above.

After a long and painful time we had the enlightenment: When the grid iteration diverges, the steel tube bursts. Until now everybody (IFU and IWKA) assumed that the forming of a wave of the bellow is hydroforming under the internal pressure. However, it is explosion forming. The metal tube does not bend into the form, it explodes or flies into the form. If we take harder steel, i.e. larger E-module, the steel tube withstands to the internal pressure, and buckles if the tool starts to move, but this does not simulate the bellow forming process.

Before we discuss further the simulation of the manufacturing we want to mention some points that came up during the numerical attempts mentioned above. Besides the large oscillations there developed small oscillations with high "frequencies". We found similar oscillations at the Bosch problem and there could cure them by a "parabola smoothing", see the context of Fig. 3.3.5.1. For the bellow forming process we at first smoothed the inner and outer surface and then all grid lines parallel to these surfaces. By an appropriate choice of the smoothing parameters n_{smooth} (3.3.5.5) and β, see (3.3.5.4), we could cure these high frequency oscillations, but this had practically no influence on the large oscillations that destroyed the solution when the tube bursted.

Another, for us very frustrating point was the following: During the investigations we wanted measured data of the tensile test that is very dense in the transition region from elastic to plastic deformation. In the discussions with the IFU we got the information that the measurements of the stress for the tensile test, e.g. of Fig. 3.2.3.6, were based on the actual (reduced) cross section of the probe and not, as we assumed, on the constant original cross section. The IFU could not clearly explain, how the actual cross section was determined. It could not be cleared if this was originally false information from the part of the IFU or a misunderstanding on our part.

So we started again to determine from the tensile test a function for $E(\varepsilon_{xx})$ of type (3.2.3.23) and $\nu(\varepsilon_{xx})$ of type (3.2.3.30), now based on the actual (reduced) cross section of the test piece. We again adapted the set of coefficients and obtained similarly to (3.2.3.36), the coefficients not mentioned here are those of (3.2.3.36):

$$\delta_x(\varepsilon_{xx}) = \begin{cases} 0 \le \varepsilon_{xx} \le 0.08: & \delta = 0, \\ 0.08 \le \varepsilon_{xx} \le 0.30: & \delta = -\frac{100}{70}\varepsilon_{xx} + \frac{8}{70}, \\ \varepsilon_{xx} > 0.30: & \delta = \frac{30}{70}\varepsilon_{xx} - \frac{31}{70}, \end{cases} \tag{3.2.4.46}$$

$$K_{x,0} = 21000\ N/mm^2,$$
$$\varepsilon_{x,0} = 1.02, \eta_x = 0.1,$$

and for $\nu_{plastic}(\varepsilon_{xx})$ instead of (3.2.4.30)

$$\nu_{plastic}(\varepsilon_{xx}) = -0.0175\ \varepsilon_{xx}^3 + 0.007\ \varepsilon_{xx}^2 + 0.001475\ \varepsilon_{xx} + 0.4916. \tag{3.2.4.47}$$

Fig. 3.2.4.10 shows the comparison of the measurement and of the simulation with the above given coefficients for the tensile test. Now the stress is based on the actual cross section, see context of Fig. 3.2.3.3.

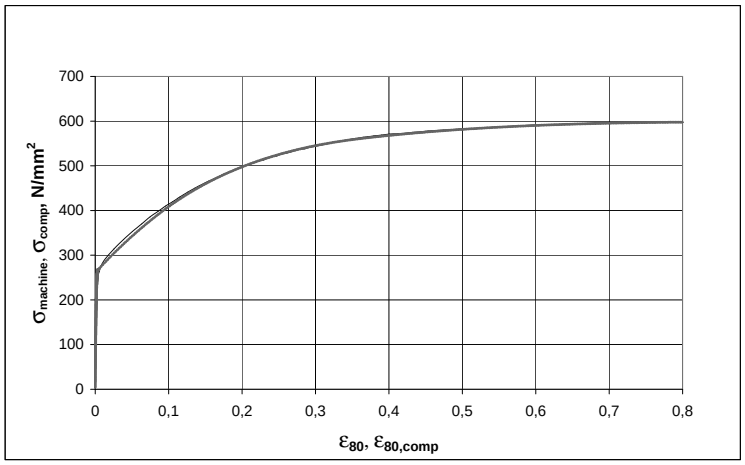


Figure 3.2.4.10: Measured and simulated tensile test based on the actual cross section. The line for the measurements is composed from very dense dots.

In the test step we determine if a node is free or forced, but also if it is elastic or plastic. The criterion is the comparison of the equivalent stress $\bar{\sigma}$ (3.2.4.41) and the plasticity parameter λ that

is computed from the PDE (3.2.4.43). The criterion is given in the context of equation (3.2.4.42). This is a pointwise switching from elastic to plastic, individually for each node. This means that for two neighboring nodes one node may be elastic and the other node plastic. However, in the steel there is no "switching", there is always a continuous transition. As we attributed at first the problems with the oscillations to the switching, we chose to describe the transition from elastic to plastic by a "transition function" that goes continuously from elastic to plastic by the choice of an appropriate functional approach. Our plastic E-module just describes the moving along the yield surface of the metal sheet. There is also another open question in this context that the IFU could not answer to us: The tensile test runs in several minutes, so the crystalline structure of the steel has time to adapt. However, the forming of a wave of the bellow runs in some milliseconds so that the crystalline structure has less or no time to adapt which results presumably in another value for the E-module: $E = E(\varepsilon, \dot{\varepsilon})$, i.e. E may depend not only on the strain but also on the strain velocity. As we do not have information on the dependence on $\dot{\varepsilon}$ we take for E the above mentioned function $E(\varepsilon)$ determined from the tensile test.

Now back to the transition function for E. We made at first an approach with a parabola, but got no satisfactory results. From the graph of E we concluded that an exponential approach would be better. For ν we use a 3^{rd} order parabola. For $\varepsilon < \varepsilon_{trans1}$ we use $E_{elastic}$ and $\nu_{elastic}$, for $\varepsilon > \varepsilon_{trans2}$ we use $E_{plastic}$ and $\nu_{plastic}$. Between ε_{trans1} and ε_{trans2} we use the transition functions

$$\begin{aligned} E_{trans}(\varepsilon) &= 10^{a_0 + a_1 \varepsilon}, \\ \nu_{trans}(\varepsilon) &= b_0 + b_1 \varepsilon + b_2 \varepsilon^2 + b_3 \varepsilon^3 \end{aligned} \tag{3.2.4.48}$$

with

$$\begin{aligned} &\varepsilon_{trans1} = 0, \quad \varepsilon_{trans2} = 0.00405, \\ &a_0 = 5.288, \quad a_1 = -208.214, \\ &b_0 = 0.4916, \quad b_1 = 0.1475 \cdot 10^{-2}, \quad b_2 = 0.7 \cdot 10^{-2}, \quad b_3 = -0.175 \cdot 10^{-1}. \end{aligned}$$

In a time stepping procedure these values are determined for the value of ε of the previous time step. Fig. 3.2.4.11 shows $E_x(\varepsilon_{xx})$ for the whole ε_{xx} range of the tensile test, Fig. 3.2.4.12 shows the value for very small ε_{xx}. In the figures we can see how rapidly E_x drops, observe the logarithmic scale for E_x. As we have now a transition function with continuous transition from elastic to plastic we do no longer need to compute $\bar{\sigma}$ and λ and "switch" from elastic to plastic. We use the same "elastic" equations, only with an appropriate variable E-module. We now could call these equations elastic/plastic. We now must extend this approach from the cartesian coordinate system to the rotationally symmetric cylindrical coordinate system. In equations (3.2.4.31)–(3.2.4.33) we extended E and in (3.2.4.35) ν with the approach $E(\varepsilon_{xx})$, $\nu(\varepsilon_{xx})$ from the tensile test to the cylindrical coordinates. However, if the metal sheet is bent up into the form, see Fig. 3.2.4.2, what is then the meaning of ε_{zz}, ε_{rr}, $\varepsilon_{\varphi\varphi}$? At the beginning of the metal forming process we have the straight tube, the main direction is the z-direction, see Fig. 3.2.4.4. However, if the sheet is bent up the main direction on the side walls of a wave is the r-direction. For this reason we decided to quit for the elastic/plastic equations in cylindrical coordinates the orthotropic model and to go back to an isotropic model with

$$\begin{aligned} E &= E(\varepsilon_{max}),\ \varepsilon_{max} = max(|\varepsilon_{zz}|, |\varepsilon_{rr}|, |\varepsilon_{\varphi\varphi}|), \\ \nu &= \nu(\varepsilon_{max}). \end{aligned} \tag{3.2.4.49}$$

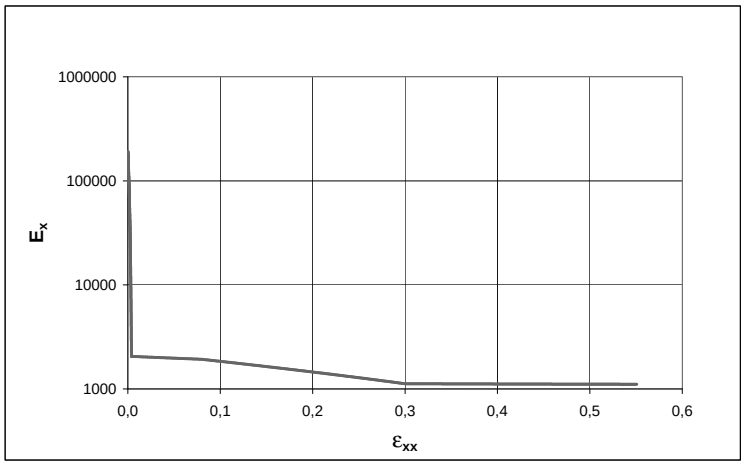


Figure 3.2.4.11: $E_x(\varepsilon_{xx})$ with the transition function for the whole range of ε_{xx}.

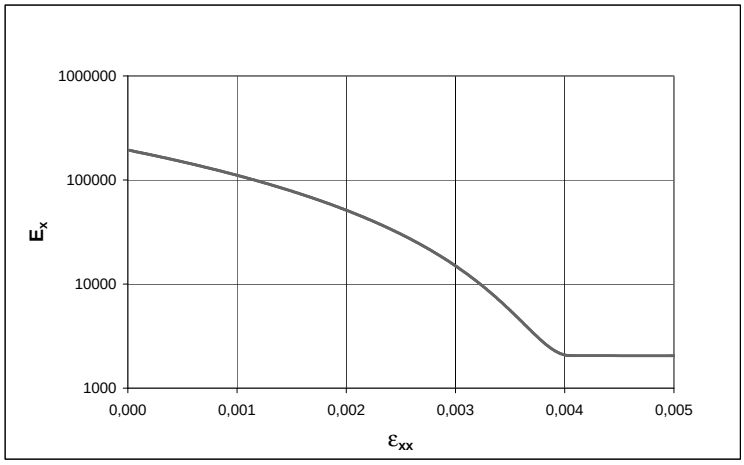


Figure 3.2.4.12: $E_x(\varepsilon_{xx})$ with the transition function for small ε_{xx}.

Again ε_{max} is determined from the previous time step. G_{rz} is replaced by E with the relation [12](3.81).

Now we want to continue our considerations what to do when the grid iteration diverges, i.e. the steel tube bursts. The expansion of the steel sheet into the form, see Fig. 3.2.4.2, is far beyond the elastic/plastic approach. As long as we use the elastic/plastic equations the metal has still a "memory" of its past. However, if it expands far into the form it behaves like dough. For the elastic/plastic approach we have stress proportional to strain, for dough we have stress proportional to strain velocity. In [12](3.15)–(3.19) we have the expressions for the strain velocities in cylindrical coordinates. For rotational symmetry, $\partial/\partial\varphi = 0$, $u_\varphi = 0$, $v_\varphi = 0$ and taking only the linear terms we get

$$\begin{aligned} &\dot{\varepsilon}_{rr} = \tfrac{\partial v_r}{\partial r},\ \dot{\varepsilon}_{\varphi\varphi} = \tfrac{v_r}{r}\left\{-\tfrac{u_r}{r^2}\tfrac{\partial r}{\partial t}\right\},\ \dot{\varepsilon}_{zz} = \tfrac{\partial v_z}{\partial z}, \\ &\dot{\varepsilon}_{rz} = \tfrac{1}{2}\left(\tfrac{\partial v_r}{\partial z} + \tfrac{\partial v_z}{\partial r}\right). \end{aligned} \tag{3.2.4.50}$$

In $\dot{\varepsilon}_{\varphi\varphi}$ we have u_r in the braces which is the displacement from the original position. However, dough has no memory and forgets the original position. Therefore we drop this term in $\dot{\varepsilon}_{\varphi\varphi}$. In equation [12](3.44) we have the stress/strain relation (now isotropic):

$$\varepsilon_{zz} = \frac{1}{E}\left(\sigma_{zz} - \nu\sigma_{\varphi\varphi} - \nu\sigma_{rr}\right).$$

For the metal dough we now make a quite similar approach by

$$\dot{\varepsilon}_{zz} = \frac{1}{F}\left(\sigma_{zz} - \nu\sigma_{\varphi\varphi} - \nu\sigma_{rr}\right), \tag{3.2.4.51}$$

with a proportionality factor between stress and strain velocity which we call F, whose dimension is $[F] = Ns/mm^2$. Taking $\dot{\varepsilon}_{zz}$ from (3.2.4.50) we get the PDE, writing it with all terms to the l.h.s.:

$$\frac{1}{F}\left(\sigma_{zz} - \nu\sigma_{\varphi\varphi} - \nu\sigma_{rr}\right) - \frac{\partial v_z}{\partial z} = 0. \tag{3.2.4.52}$$

This is again a quasi-steady equation. We proceed in time steps in which the grid moves in displacement increments $\Delta u = \Delta t \cdot v$. Therefore we use again the incremental form which is now for (3.2.4.52), with the index "old" for the values of the previous time step:

$$\frac{1}{F}\left[\sigma_{zz} - \sigma_{zz,old} - \nu\left(\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}\right) - \nu\left(\sigma_{rr} - \sigma_{rr,old}\right)\right] - \frac{\partial v_z}{\partial z} + \frac{\partial v_{z,old}}{\partial z} = 0. \tag{3.2.4.53}$$

Here the derivatives of v_z and $v_{z,old}$ are taken on the same (actual) grid.

This is the stress-strain velocity relation obtained for $\dot{\varepsilon}_{zz}$ (3.2.4.50). In the same way we formulate the corresponding equations for $\dot{\varepsilon}_{rr}$, $\dot{\varepsilon}_{\varphi\varphi}$ and $\dot{\varepsilon}_{rz}$. We do not write down these equations here. Additionally hold the equilibrium equations (3.2.4.8) and (3.2.4.9). The BCs are the same as before if we replace in the BCs the equations (3.2.4.4) to (3.2.4.7) by the corresponding equations for the "dough", e.g. equation (3.2.4.4) by (3.2.4.53).

Then we proceeded as follows: We computed with the elastic/plastic equations until the grid iteration diverged. The last converged solution was stored and used as starting solution with the equations for the "dough". As we have no value for F, we did numerical experiments with different values of F: the solution showed the expected behaviour, the steel sheet was more soft for small F and harder for large F. However, the basic behaviour was the same as for the elastic/plastic approach. The steel tube bent a bit into the form, but the errors grew and after some time steps the solution was meaningless. So this approach failed for the computation of the bursting tube. What to do now?

The idea is, to use instead of the quasi-steady equations now unsteady equations, but which ones? If we form the partial derivative of the dough equations with respect to time, e.g. of equation (3.2.4.52) and then discretize the time derivatives by e.g. $\frac{\partial\sigma}{\partial t} = \left(\sigma - \sigma_{old}\right)/\Delta t$ we get just equation (3.2.4.53), thus no new information. So, what to do now?

The next idea is to form the <u>total</u> derivative of the equations, because the grid moves, and the grid moves rather strongly if the sheet bends into the form. For moving coordinates we have

$$\sigma = \sigma(t, z, r) = \sigma(t, z(t), r(t)).$$

Thus the <u>total</u> derivative with respect to time is

$$\frac{d\sigma(t, z(t), r(t))}{dt} = \frac{\partial\sigma}{\partial t} + \frac{\partial\sigma}{\partial z}\frac{\partial z}{\partial t} + \frac{\partial\sigma}{\partial r}\frac{\partial r}{\partial t} = \frac{\partial\sigma}{\partial t} + \frac{\partial\sigma}{\partial z}v_z + \frac{\partial\sigma}{\partial r}v_r, \tag{3.2.4.54}$$

as we have $\partial z/\partial t = v_z$, $\partial r/\partial t = v_r$. This relation holds for σ_{zz}, σ_{rr}, $\sigma_{\varphi\varphi}$ and σ_{rz}. In (3.2.4.52) also appears $\partial v_z/\partial z$. Thus we also need the total derivative of derivatives of displacement velocities, e.g.

$$\frac{d}{dt}\left(\frac{\partial v_z(t,z(t),r(t))}{\partial z}\right) = \frac{\partial^2 v_z}{\partial z \partial t} + \frac{\partial^2 v_z}{\partial z^2} v_z + \frac{\partial^2 v_z}{\partial z \partial r} v_r. \tag{3.2.4.55}$$

Similarly we can form $d(\partial v_z/\partial r)/dt$, $d(\partial v_r/\partial z)/dt$ and $d(\partial v_r/\partial r)/dt$. For $\dot{\varepsilon}_{\varphi\varphi}$ (3.2.4.50) there also will be needed

$$\begin{aligned} \tfrac{d}{dt}\left(\tfrac{v_r}{r}\right) &= \tfrac{\partial}{\partial t}\left(\tfrac{v_r}{r}\right) + \tfrac{\partial}{\partial z}\left(\tfrac{v_r}{r}\right) v_z + \tfrac{\partial}{\partial r}\left(\tfrac{v_r}{r}\right) v_r \\ &= \tfrac{r\frac{\partial v_r}{\partial t} - v_r^2}{r^2} + \tfrac{1}{r}\tfrac{\partial v_r}{\partial z} v_z + \tfrac{r\frac{\partial v_r}{\partial r} - v_r}{r^2} v_r \\ &= \tfrac{1}{r}\tfrac{\partial v_r}{\partial t} - 2\tfrac{v_r^2}{r^2} + \tfrac{1}{r}\tfrac{\partial v_r}{\partial z} v_z + \tfrac{1}{r}\tfrac{\partial v_r}{\partial r} v_r. \end{aligned} \tag{3.2.4.56}$$

We now can form the total derivatives of all 4 stress/strain velocity equations of which we have presented the one for $\dot{\varepsilon}_{zz}$ as equation (3.2.4.52). If we do this, we immediately discretize the time derivatives, e.g.

$$\begin{aligned} \tfrac{\partial \sigma_{zz}}{\partial t} &= \tfrac{\sigma_{zz} - \sigma_{zz,old}}{\Delta t}, \\ \tfrac{\partial^2 v_z}{\partial z \partial t} &= \tfrac{\partial}{\partial t}\left(\tfrac{\partial v_z}{\partial z}\right) = \tfrac{\frac{\partial v_z}{\partial z} - \left(\frac{\partial v_z}{\partial z}\right)_{old}}{\Delta t}. \end{aligned} \tag{3.2.4.57}$$

Here the index "old" means the solution in the same node at the previous time step and $(\partial v_z/\partial z)_{old}$ is computed on the actual (new) grid from the old solution. If we use all these relations we get from the 4 stress/strain velocity equations of type (3.3.4.52) the time-discretized total derivative equations:

$$\begin{aligned} &\tfrac{1}{F}\Big[\tfrac{\sigma_{zz} - \sigma_{zz,old}}{\Delta t} + \tfrac{\partial \sigma_{zz}}{\partial z} v_z + \tfrac{\partial \sigma_{zz}}{\partial r} v_r - \\ &\nu\left(\tfrac{\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}}{\Delta t} + \tfrac{\partial \sigma_{\varphi\varphi}}{\partial z} v_z + \tfrac{\partial \sigma_{\varphi\varphi}}{\partial r} v_r\right) - \\ &\nu\left(\tfrac{\sigma_{rr} - \sigma_{rr,old}}{\Delta t} + \tfrac{\partial \sigma_{rr}}{\partial z} v_z + \tfrac{\partial \sigma_{rr}}{\partial r} v_r\right)\Big] - \\ &\left(\tfrac{\frac{\partial v_z}{\partial z} - \left(\frac{\partial v_z}{\partial z}\right)_{old}}{\Delta t} + \tfrac{\partial^2 v_z}{\partial z^2} v_z + \tfrac{\partial^2 v_z}{\partial z \partial r} v_r\right) = 0, \end{aligned} \tag{3.2.4.58}$$

$$\begin{aligned} &\tfrac{1}{F}\Big[-\nu\left(\tfrac{\sigma_{\varphi\varphi} - \sigma_{\varphi\varphi,old}}{\Delta t} + \tfrac{\partial \sigma_{\varphi\varphi}}{\partial z} v_z + \tfrac{\partial \sigma_{\varphi\varphi}}{\partial r} v_r\right) - \\ &\nu\left(\tfrac{\sigma_{zz} - \sigma_{zz,old}}{\Delta t} + \tfrac{\partial \sigma_{zz}}{\partial z} v_z + \tfrac{\partial \sigma_{zz}}{\partial r} v_r\right) + \\ &\tfrac{\sigma_{rr} - \sigma_{rr,old}}{\Delta t} + \tfrac{\partial \sigma_{rr}}{\partial z} v_z + \tfrac{\partial \sigma_{rr}}{\partial r} v_r\Big] - \\ &\left(\tfrac{\frac{\partial v_r}{\partial r} - \left(\frac{\partial v_r}{\partial r}\right)_{old}}{\Delta t} + \tfrac{\partial^2 v_r}{\partial z \partial r} v_z + \tfrac{\partial^2 v_r}{\partial r^2} v_r\right) = 0, \end{aligned} \tag{3.2.4.59}$$

$$\begin{aligned}
&\tfrac{1}{F}\Big[-\nu\left(\tfrac{\sigma_{zz}-\sigma_{zz,old}}{\Delta t}+\tfrac{\partial\sigma_{zz}}{\partial z}v_z+\tfrac{\partial\sigma_{zz}}{\partial r}v_r\right)+\\
&\tfrac{\sigma_{\varphi\varphi}-\sigma_{\varphi\varphi,old}}{\Delta t}+\tfrac{\partial\sigma_{\varphi\varphi}}{\partial z}v_z+\tfrac{\partial\sigma_{\varphi\varphi}}{\partial r}v_r-\\
&\nu\left(\tfrac{\sigma_{rr}-\sigma_{rr,old}}{\Delta t}+\tfrac{\partial\sigma_{rr}}{\partial z}v_z+\tfrac{\partial\sigma_{rr}}{\partial r}v_r\right)\Big]-\\
&\left(\tfrac{1}{r}\tfrac{v_r-v_{r,old}}{\Delta t}-2\tfrac{v_r^2}{r^2}+\tfrac{1}{r}\tfrac{\partial v_r}{\partial z}v_z+\tfrac{1}{r}\tfrac{\partial v_r}{\partial r}v_r\right)=0,
\end{aligned}\tag{3.2.4.60}$$

$$\begin{aligned}
&\tfrac{1+\nu}{F}\left(\tfrac{\sigma_{rz}-\sigma_{rz,old}}{\Delta t}+\tfrac{\partial\sigma_{rz}}{\partial z}v_z+\tfrac{\partial\sigma_{rz}}{\partial r}v_r\right)-\\
&\tfrac{1}{2}\left(\tfrac{\frac{\partial v_r}{\partial z}-\left(\frac{\partial v_r}{\partial z}\right)_{old}}{\Delta t}+\tfrac{\partial^2 v_r}{\partial z^2}v_z+\tfrac{\partial^2 v_r}{\partial z\partial r}v_r\right)-\\
&\tfrac{1}{2}\left(\tfrac{\frac{\partial v_z}{\partial r}-\left(\frac{\partial v_z}{\partial r}\right)_{old}}{\Delta t}+\tfrac{\partial^2 v_z}{\partial z\partial r}v_z+\tfrac{\partial^2 v_z}{\partial r^2}v_r\right)=0.
\end{aligned}\tag{3.2.4.61}$$

These 4 equations are supplemented by the 2 equilibrium equations (3.2.4.8) and (3.2.4.9) that hold also for the unsteady solution.

Now the first 4 of these 6 equations for the 6 variables are non-linear, e.g. by terms like $\frac{\partial\sigma_{zz}}{\partial z}v_z$, and they contain second order derivatives of the displacement velocities, e.g. $\frac{\partial^2 v_z}{\partial z^2}$ or $\frac{\partial^2 v_z}{\partial z\partial r}$. This changes completely the character of the PDEs compared to the linear elastic/plastic equations.

In these equations is the unknown F-module. We had made the speculative approach (3.2.4.51) for the stress/strain velocity relation, but we had no value for F. Our intention is to try to simulate the manufacturing process by these equations with different values of F until we find a value that "fits" to the observed behaviour. The BCs for the elastic equations had been discussed in the context of Fig. 3.2.4.6. Now hold the same BCs, but we must replace the PDEs (3.2.4.4)–(3.2.4.7) by the PDEs (3.2.4.58)–(3.2.4.61).

The execution of the computation is as follows: we compute in time incremental form with time step size Δt with the elastic/plastic equations until the quasi-steady grid iteration diverges. The last converged solution is stored and is the starting solution of the unsteady equations. We solve the unsteady equations in time steps with step size Δt.

The simple explicit marching in time direction with the shifting of the grid by the equations (3.2.4.2), (3.2.4.3) after the time step leads to an explicit time marching procedure where all derivatives have been formed on the old grid, i.e. the grid of the previous time step. However, if the steel sheet "explodes" into the form, we have large displacements and the new grid is quite different from the old grid. Therefore we introduced here also a grid iteration: After the computation of a solution we recompute the solution repeatedly on the new grid until the grid comes to rest. If k is the iteration index of the grid iteration, we check if

$$\frac{\left\|v_z^k-v_z^{k-1}\right\|}{\|v_z^k\|}+\frac{\left\|v_r^k-v_r^{k-1}\right\|}{\|v_r^k\|}<\varepsilon_{grid}\tag{3.2.4.62}$$

and continue the iteration if (3.2.4.62) is not fulfilled. We also can prescribe a min. and max. number of iterations. If we use a smoothing of the grid, we apply the smoothing after each grid iteration. So finally we have at the end of the time step a solution on the new grid. As the equations are non-linear and the moving of the grid is also a non-linear process, we have in reality now an extremely non-linear problem. The converged solution of a time step then is the starting solution for the next time step.

The experience has shown that such iteration processes like the grid iteration at each time step can be accelerated by a relaxation factor ω or even can be made convergent at all by a small relaxation factor. Therefore we introduce the possibility of a relaxation factor into the grid iteration by

$$displacement = \omega \cdot displacement_{method}, \tag{3.2.4.63}$$

where $displacement_{method}$ is the displacement that the method delivers. For a small value of ω the grid shifts slowly during the grid iteration. As the grid iteration is stopped by the condition (3.2.4.62) we should now use $\omega \cdot \varepsilon_{grid}$ as stopping criterion because for small ω the change in v is correspondingly smaller than for $\omega = 1$. The effect of a small ω is that the actual grid can follow better the development of the solution.

When we implemented the equations (3.2.4.58)–(3.2.4.61) as usual (and as described in the first part of this report) we at first tested them by a test polynomial solution of second order, i.e. we added absolute terms that the exact solution is the prescribed polynomial. If we start with the exact solution, the Newton residual should be in the range of $10^{-10} - 10^{-12}$. However, for these equations the residual was 10^{-7} which indicates a problem. When we started with a disturbed solution, e.g. $1.01 \times$ exact solution (1% disturbance), the Newton iteration stopped at a residual of 10^{-5} because this was small in relation to the discretization error, and we did not get the prescribed solution but another one. As such an extremely non-linear system may have many solutions, obviously the Newton iteration drifted to another solution although we had started close to the exact solution. From this behaviour of the test problem we expected similar difficulties of the physical problem.

For the solution of the physical problem we started from the last grid-converged quasi-steady solution, but we could not get a physically reasonable solution. We experimented with different values of the F-module and of the time step size Δt, but no reasonable solution could be obtained. So we concluded that the equations (3.2.4.58)–(3.2.4.61) are not suited for the computation of the exploding steel sheet. But what to do now?

If we look at equation (3.2.4.54), we have replaced e.g. $\partial z/\partial t$ by v_z. This is correct if we are in a completely continuous environment, but it may be inadequate if we proceed in incremental steps. For an incremental procedure where the grid is shifted after each time step, the total time discretization should better be formulated thus:

$$\frac{d\sigma(t,z(t),r(t))}{dt} = \frac{\partial\sigma}{\partial t} + \frac{\partial\sigma}{\partial z}\frac{\partial z}{\partial t} + \frac{\partial\sigma}{\partial r}\frac{\partial r}{\partial t} = \frac{\sigma - \sigma_{old}}{\Delta t} + \frac{\partial\sigma}{\partial z}\frac{z - z_{old}}{\Delta t} + \frac{\partial\sigma}{\partial r}\frac{r - r_{old}}{\Delta t}, \tag{3.2.4.64}$$

where values with the index "old" are of the previous time step. This relation holds for all σ's: σ_{zz}, σ_{rr}, $\sigma_{\varphi\varphi}$, σ_{rz}. Similarly we discretize instead of (3.2.4.55) now e.g.

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial v_z(t,z(t),r(t))}{\partial z}\right) &= \frac{\partial}{\partial t}\left(\frac{\partial v_z}{\partial z}\right) + \frac{\partial^2 v_z}{\partial z^2}\frac{\partial z}{\partial t} + \frac{\partial^2 v_z}{\partial z \partial r}\frac{\partial r}{\partial t} \\ &= \frac{\frac{\partial v_z}{\partial z} - \left(\frac{\partial v_z}{\partial z}\right)_{old}}{\Delta t} + \frac{\partial^2 v_z}{\partial z^2}\frac{z - z_{old}}{\Delta t} + \frac{\partial^2 v_z}{\partial z \partial r}\frac{r - r_{old}}{\Delta t}. \end{aligned} \tag{3.2.4.65}$$

Similar relations hold for $d(\partial v_z/\partial r)/dt$, $d(\partial v_r/\partial z)/dt$ and $d(\partial v_r/\partial r)/dt$. Instead of (3.2.4.56) we have now

$$\begin{aligned}\frac{d}{dt}\left(\frac{v_r}{r}\right) &= \frac{\partial}{\partial t}\left(\frac{v_r}{r}\right)+\frac{\partial}{\partial z}\left(\frac{v_r}{r}\right)\frac{dz}{dt}+\frac{\partial}{\partial r}\left(\frac{v_r}{r}\right)\frac{dr}{dt}\\ &= \frac{\partial}{\partial t}\left(\frac{v_r}{r}\right)+\frac{1}{r}\frac{\partial v_r}{\partial z}\frac{dz}{dt}+\frac{r\frac{\partial v_r}{\partial r}-v_r}{r^2}\frac{dr}{dt} \qquad (3.2.4.66)\\ &= \frac{\frac{v_r}{r}-\left(\frac{v_r}{r}\right)_{old}}{\Delta t}+\frac{1}{r}\frac{\partial v_r}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{1}{r}\frac{\partial v_r}{\partial r}\frac{r-r_{old}}{\Delta t}-\frac{v_r}{r^2}\frac{r-r_{old}}{\Delta t}.\end{aligned}$$

If we use all these relations we get from the 4 stress/strain velocity equations of type (3.3.4.52) now the time discretized new total derivative equations (instead of (3.2.4.58)–(3.2.4.61)):

$$\begin{aligned}&\frac{1}{F}\left[\frac{\sigma_{zz}-\sigma_{zz,old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial r}\frac{r-r_{old}}{\Delta t}-\right.\\ &\nu\left(\frac{\sigma_{\varphi\varphi}-\sigma_{\varphi\varphi,old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)-\\ &\left.\nu\left(\frac{\sigma_{rr}-\sigma_{rr,old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)\right]-\\ &\left(\frac{\frac{\partial v_z}{\partial z}-\left(\frac{\partial v_z}{\partial z}\right)_{old}}{\Delta t}+\frac{\partial^2 v_z}{\partial z^2}\frac{z-z_{old}}{\Delta t}+\frac{\partial^2 v_z}{\partial z\partial r}\frac{r-r_{old}}{\Delta t}\right)=0,\end{aligned} \qquad (3.2.4.67)$$

$$\begin{aligned}&\frac{1}{F}\left[-\nu\left(\frac{\sigma_{\varphi\varphi}-\sigma_{\varphi\varphi,old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)-\right.\\ &\nu\left(\frac{\sigma_{zz}-\sigma_{zz,old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)+\\ &\left.\frac{\sigma_{rr}-\sigma_{rr,old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial r}\frac{r-r_{old}}{\Delta t}\right]-\\ &\left(\frac{\frac{\partial v_r}{\partial r}-\left(\frac{\partial v_r}{\partial r}\right)_{old}}{\Delta t}+\frac{\partial^2 v_r}{\partial z\partial r}\frac{z-z_{old}}{\Delta t}+\frac{\partial^2 v_r}{\partial r^2}\frac{r-r_{old}}{\Delta t}\right)=0,\end{aligned} \qquad (3.2.4.68)$$

$$\begin{aligned}&\frac{1}{F}\left[-\nu\left(\frac{\sigma_{zz}-\sigma_{zz,old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{zz}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)+\right.\\ &\frac{\sigma_{\varphi\varphi}-\sigma_{\varphi\varphi,old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{\varphi\varphi}}{\partial r}\frac{r-r_{old}}{\Delta t}-\\ &\left.\nu\left(\frac{\sigma_{rr}-\sigma_{rr,old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{\partial\sigma_{rr}}{\partial r}\frac{r-r_{old}}{\Delta t}\right)\right]-\\ &\left(\frac{\frac{v_r}{r}-\left(\frac{v_r}{r}\right)_{old}}{\Delta t}-\frac{v_r}{r^2}\frac{r-r_{old}}{\Delta t}+\frac{1}{r}\frac{\partial v_r}{\partial z}\frac{z-z_{old}}{\Delta t}+\frac{1}{r}\frac{\partial v_r}{\partial r}\frac{r-r_{old}}{\Delta t}\right)=0,\end{aligned} \qquad (3.2.4.69)$$

$$\begin{aligned}
&\tfrac{1+\nu}{F}\left(\tfrac{\sigma_{rz}-\sigma_{rz,old}}{\Delta t}+\tfrac{\partial\sigma_{rz}}{\partial z}\tfrac{z-z_{old}}{\Delta t}+\tfrac{\partial\sigma_{rz}}{\partial r}\tfrac{r-r_{old}}{\Delta t}\right)-\\
&\tfrac{1}{2}\left(\tfrac{\frac{\partial v_r}{\partial z}-\left(\frac{\partial v_r}{\partial z}\right)_{old}}{\Delta t}+\tfrac{\partial^2 v_r}{\partial z^2}\tfrac{z-z_{old}}{\Delta t}+\tfrac{\partial^2 v_r}{\partial z\partial r}\tfrac{r-r_{old}}{\Delta t}\right)- \qquad (3.2.4.70)\\
&-\tfrac{1}{2}\left(\tfrac{\frac{\partial v_z}{\partial r}-\left(\frac{\partial v_z}{\partial r}\right)_{old}}{\Delta t}+\tfrac{\partial^2 v_z}{\partial z\partial r}\tfrac{z-z_{old}}{\Delta t}+\tfrac{\partial^2 v_z}{\partial r^2}\tfrac{r-r_{old}}{\Delta t}\right)=0.
\end{aligned}$$

These 4 equations are supplemented by the equilibrium equations (3.2.4.8) and (3.2.4.9).

In contrast to the extremely non-linear equations (3.2.4.58)–(3.2.4.61) these equations now are linear. The linearity comes from the fact that time derivatives of the coordinates are expressed explicitly by difference quotients of the coordinates and the coordinates are not variables. If we expressed these time derivatives by displacement velocities as we did in (3.2.4.58)–(3.2.4.61) the equations would be non-linear. All the previous remarks hold concerning the BCs, if we replace the non-linear equations by the above linear equations, concerning the algorithmic procedure with starting from the last converged quasi-steady solution and concerning the grid iteration.

When we implemented the equations (3.2.4.67)–(3.2.4.70) the numerical behaviour for the test polynomial was quite different from that of the non-linear equations: now the Newton residual for the exact polynomial was 10^{-12} and for a disturbed solution the disturbance was corrected in one Newton step. This is the natural consequence of the linearity of the equations.

These equations now have the obvious property that we can continuously compute out of the starting profile. If we use, as we do it, as initial guess for the Newton iteration at a certain time step the old solution and we use for test purposes the old grid and apply the old BCs, the equations are fulfilled, the Newton residual is zero and the old solution is reproduced. This did not hold for the non-linear equations (3.2.4.58)–(3.2.4.61). For the new unsteady equations with time stepping the pressure changes and we get other solutions.

Now we have a system of PDEs and BCs that should describe the explosion forming, i.e. how the metal tube explodes into the tool. However, what is the value of the F-module, that replaces the E-module of the elastic/plastic equations? The computational procedure is as follows: We compute with the elastic/plastic equations with the quasi-steady equations in time steps Δt (moving grid) and with full grid iteration, until the grid iteration shows "explosion", i.e. instead to converge to a solution the displacement velocity increases in the grid iteration until the solution becomes "nonsense". The last converged quasi-steady solution is the starting solution for the unsteady "dough" equations.

Before we discuss the choice of the value of the F-module we want to show in detail the last converged quasi-steady solution. We computed with the grid 201×39, with time stepping of $\Delta t = 10\ ms$, stopping criterion (3.2.4.45) $\varepsilon_{grid} = 10^{-4}$. The last converged time step was step 25, it needed 235 grid iterations which shows that we are close to the explosion limit. Here we must mention, that we changed for these investigation the geometry. In Fig. 3.2.4.1 the inner diameter of the tool is $47\ mm$. We computed instead with $46\ mm$ so that the steel tube immediately touches the tool, i.e. the upper left and right corner of the tube are fixed. The starting time of the computation is $40\ ms$ because only at that moment the pressure starts to build up (before that time nothing happens). So the time for step 25 is $40 + 25 \cdot 10 = 290\ ms$. We discuss the results of this last converging step of the elastic/plastic equations before the steel tube "explodes". Fig. 3.2.4.13 shows the form of the

steel sheet. In the middle it has been bent up by $0.1\ mm$, half the thickness of the steel sheet.

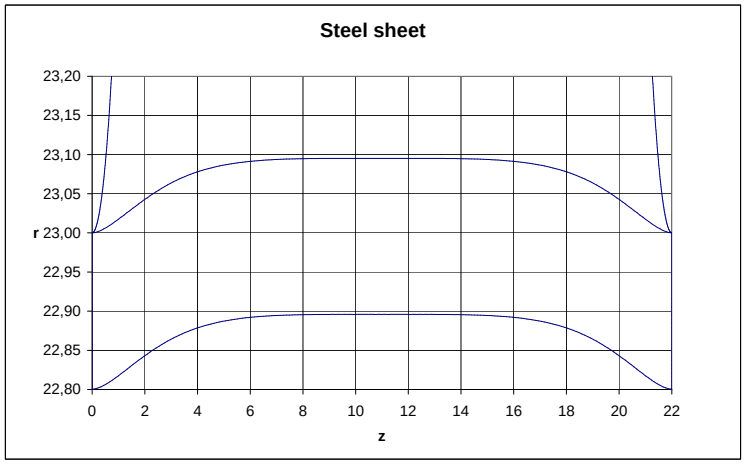


Figure 3.2.4.13: Form of the steel tube in time step 25. Observe the strongly increased scale for r. In the upper left and right corners the tool can be seen.

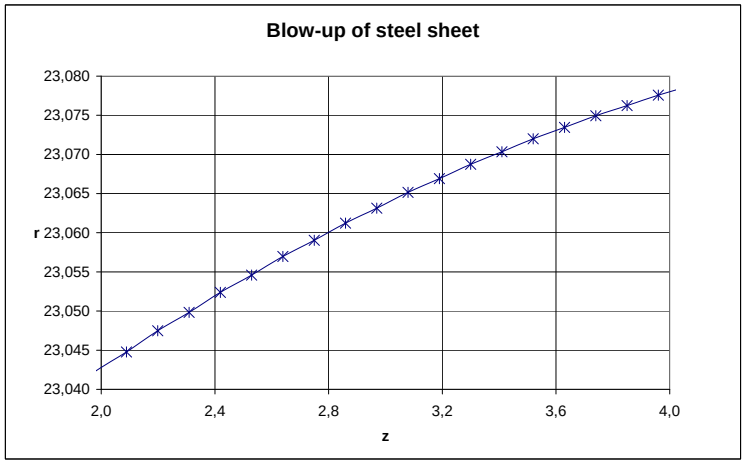


Figure 3.2.4.14: Blow-up of the upper surface of the steel sheet of step 25 to show that it is smooth.

Fig. 3.2.4.14 shows a blow-up of the upper surface of the steel sheet between $z = 2$ and $z = 4$ that shows that the surface is smooth. Fig. 3.2.4.15 shows the displacement velocity v_r at upper and lower surface, both curves nearly coincide for this scale. Fig. 3.2.4.16 shows a blow-up of v_r between $z = 2$ and $z = 4$. We can recognize in this scale that v_r is "wavy". We can also clearly recognize that v_r of the lower surface (squares) is a bit larger than v_r at the upper surface (stars) which means that the steel sheet becomes thinner. Because v_r is "wavy" we applied also for test purposes the smoothing of Section 3.3.5 for the grid of the steel sheet, see Fig. 3.3.5.1, with the smoothing parameters $n_{smooth} = 2$ (3.3.5.5), i.e. 2 smoothing steps, and the smoothing parameter $\beta = 0.5$ (3.3.5.4), i.e. reduction of the curvature by a factor 0.5. We smoothed on all grid lines parallel to the surface that were originally horizontal lines. The result was that the "waves" of v_r

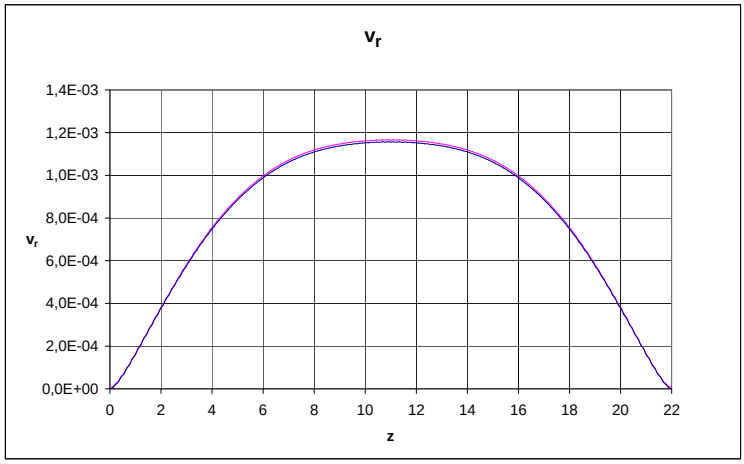


Figure 3.2.4.15: Displacement velocity v_r $[mm/s]$ at upper and lower surface for step 25, both curves nearly coincide.

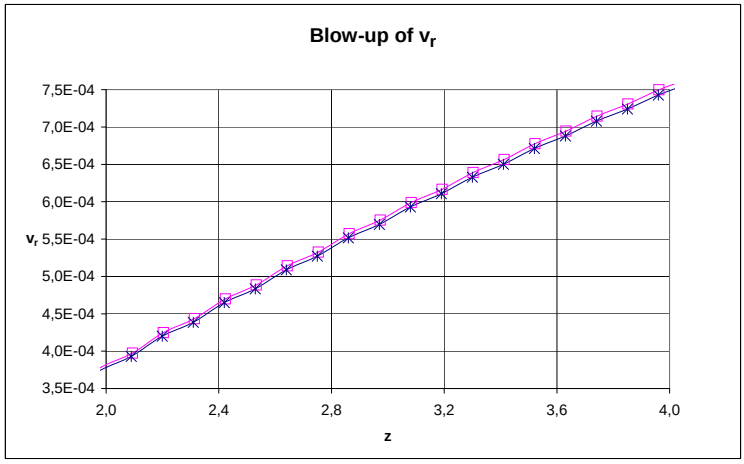


Figure 3.2.4.16: Blow-up of v_r between $z = 2$ and $z = 4$. The stars are at the upper, the squares at the lower surface.

became larger and the error estimates increased by nearly one order of magnitude. So smoothing of the surface does not cure the waves of v_r.

The following contour plots of the 6 variables and their errors for time step 25 are gray scale in a black-and-white printout, but they are colored in the online version of the paper which gives much more information. Therefore it is recommended to look at these figures at the screen.

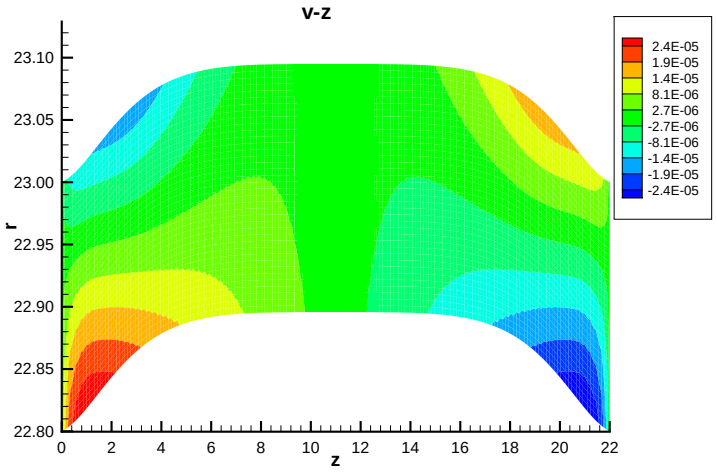


Figure 3.2.4.17: Displacement velocity v_z $[mm/s]$.

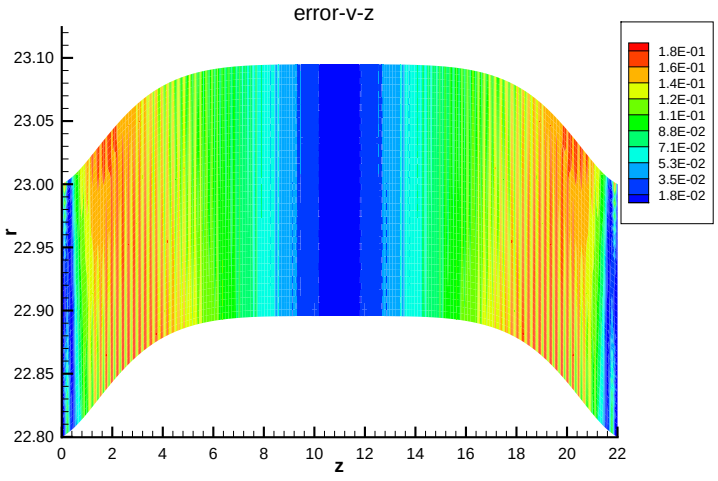


Figure 3.2.4.18: Error estimate for v_z.

Fig. 3.2.4.17 shows v_z, i.e. the movement of the steel sheet in z-direction. The largest values are in the lower corner region, left with positive and right with negative v_z, so we can see the creeping direction of the sheet material. Fig. 3.2.4.18 shows the global relative error of v_z. The largest errors are at the shoulders, the smallest in the middle and in the end regions. Fig. 3.2.4.19 shows v_r, i.e. the movement in the r-direction. The largest values are in the middle region, the smallest values are at the fixed end points.

The next figures show the stresses and their errors. σ_{zz}, Fig. 3.2.4.21, has large positive values

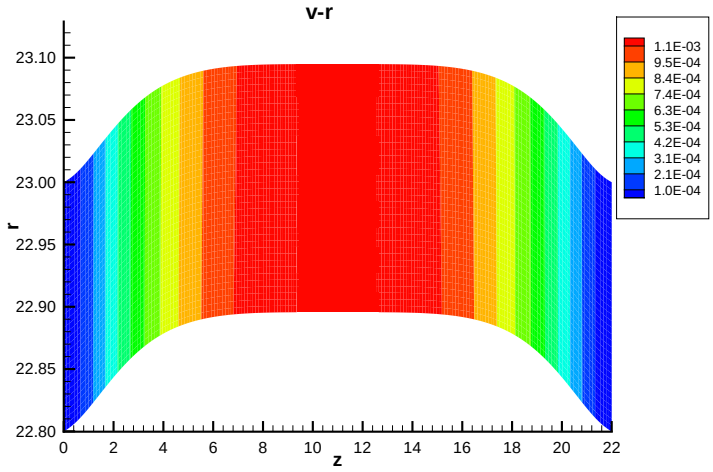


Figure 3.2.4.19: Displacement velocity v_r $[mm/s]$.

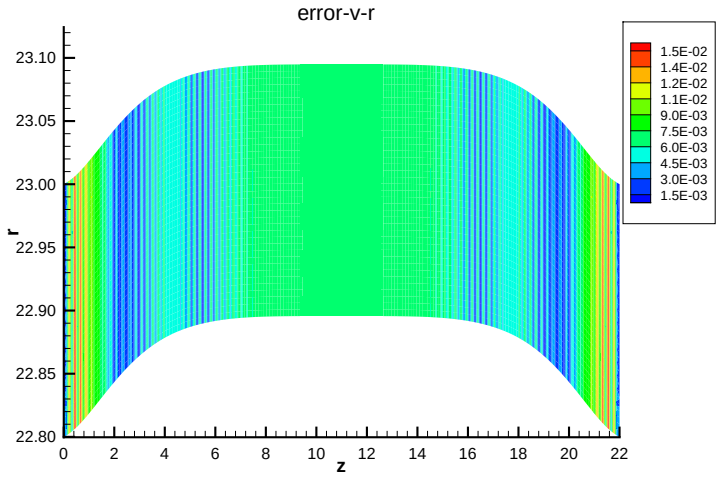


Figure 3.2.4.20: Error estimate for v_r.

at the lower left and right corner and large negative values at the upper left and right corners. The errors, Fig. 3.2.4.22, are small nearly everywhere. σ_{rr}, Fig. 3.2.4.23, is small compared to σ_{zz}. The global relative errors, Fig. 3.2.4.24, are seemingly large, but as the value of σ_{rr} is much smaller than that of σ_{zz}, the absolute values of the errors are like those of σ_{zz}. $\sigma_{\varphi\varphi}$, Fig. 3.2.4.25, is positive in the whole middle region and has negative values at the upper corners. The errors, Fig. 3.2.4.26, are small except at the left and right edges, which is not visible at the scale of the plots. σ_{rz}, Fig. 3.2.4.27, has the largest positive value in the middle of the left edge and the largest negative value in the middle

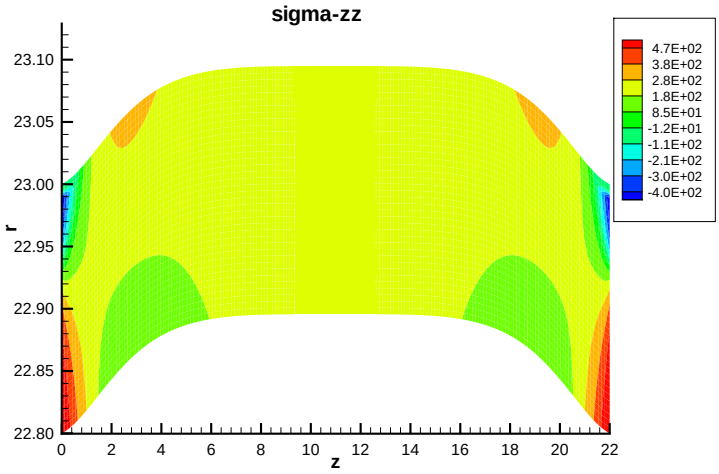


Figure 3.2.4.21: Stress component σ_{zz} $[N/mm^2]$.

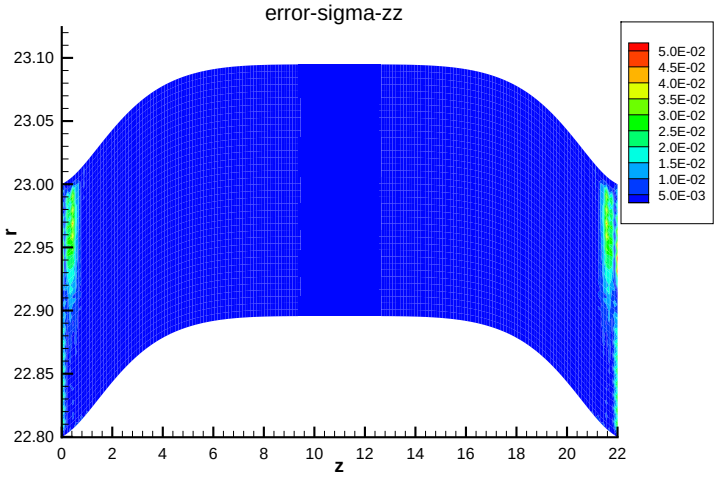


Figure 3.2.4.22: Error estimate for σ_{zz}.

of the right edge. There are also the largest relative errors, Fig. 3.2.4.28, which seem to be large, but are in absolute value like those for σ_{zz} and $\sigma_{\varphi\varphi}$.

Table 3.2.4.2 shows the maximal values of the variables, of their maximal global relative error estimates and of the mean error estimates for step 25. This table tells us that there where the mean errors are much smaller than the max. errors these max. errors are confined to a narrow region. It tells us also that the error seems to be large where the value of the variable is small relative to that of the other variables of the same type, e.g. for v_z which is two orders of magnitude smaller than v_r

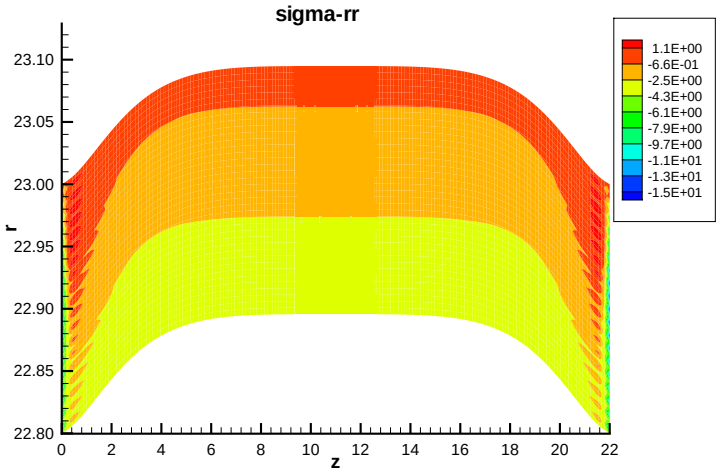


Figure 3.2.4.23: Stress component σ_{rr} $[N/mm^2]$.

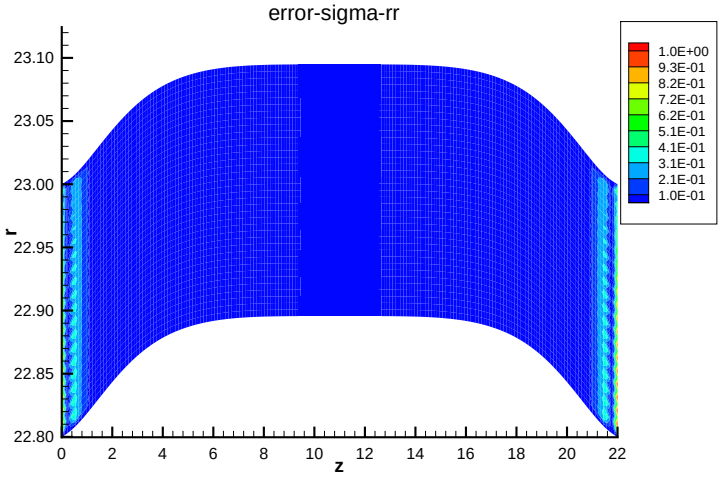


Figure 3.2.4.24: Error estimate for σ_{rr}.

or for σ_{rr} that is much smaller than σ_{zz}. This comes from the fact that we show relative errors. The absolute values of the errors, that are decisive, are of the same size.

Fig. 3.2.4.29 shows the E-module for time step 25. From the program output we get the values $E_{max} = 188781\ N/mm^2$ and $E_{min} = 3373\ N/mm^2$. The distribution of the values of E in Fig. 3.2.4.29 shows that the low values are in the middle region and in the lower left and right corners. From Fig. 3.2.4.11 and 3.2.4.12 we see that E is small for large strain ε. The large values of E occur in a relatively small region close to the left and right end, there the strain must be small.

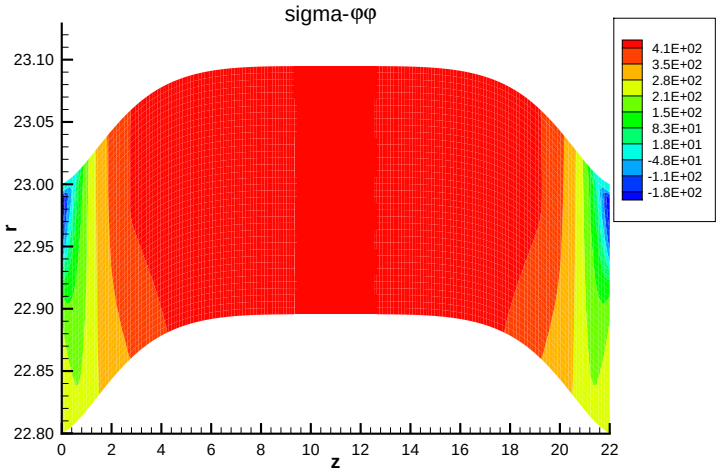


Figure 3.2.4.25: Stress component $\sigma_{\varphi\varphi}$ $[N/mm^2]$.

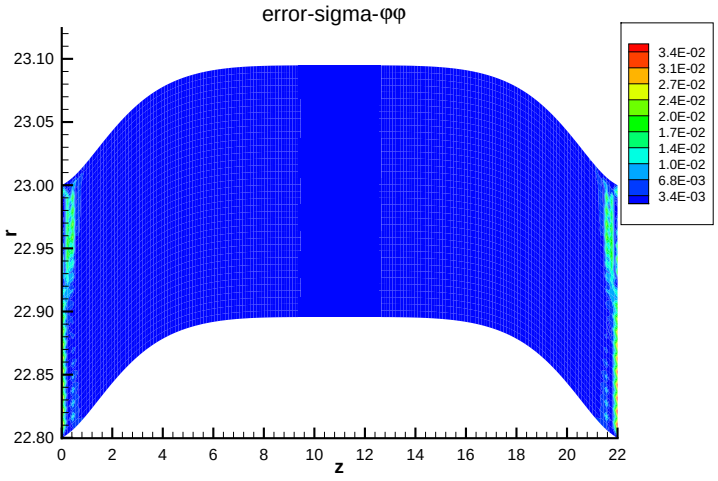


Figure 3.2.4.26: Error estimate for $\sigma_{\varphi\varphi}$.

So the elastic/plastic equations comprise a wide region of the E-module, i.e. of hard and soft steel.

So far we have discussed the last converging step of the elastic/plastic equations before the tube "explodes". This solution is the starting solution for the unsteady "dough" equations (3.2.4.67)–(3.2.4.70) and (3.2.4.8) and (3.2.4.9). In this system of PDEs the E-module $[N/mm^2]$ is replaced by the F-module $[Ns/mm^2]$. As our approach to simulate the bursting steel tube is a purely "academic" attempt, we do not have values for F. Note that we are "numerical engineers" and not metallurgists and we do not know if in the literature this approach is published. However, even if it has been

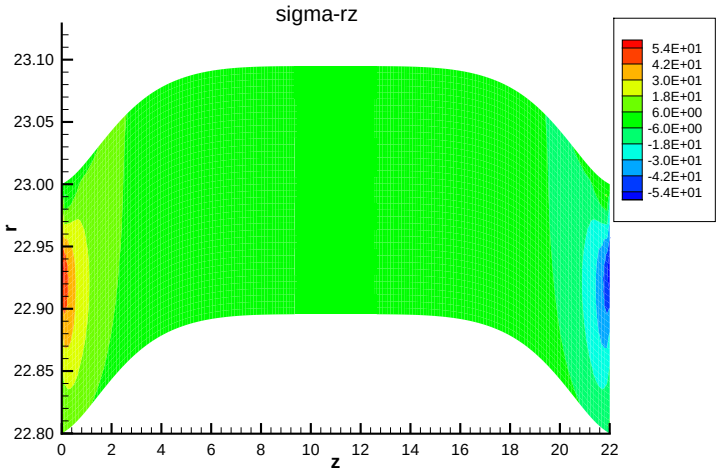


Figure 3.2.4.27: Stress component σ_{rz} $[N/mm^2]$.

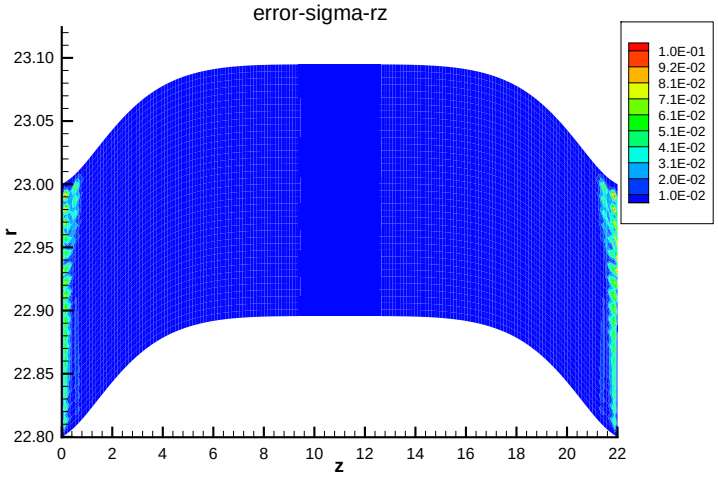


Figure 3.2.4.28: Error estimate for σ_{rz}.

published we could not find there the correct value of F for our stainless steel. So our idea is to "play" with F until we have found a value that reproduces the manufacturing process of the IWKA. Thus the numerical simulation replaces the measurements.

Our goal is now to compute in time direction, starting from the values of the elastic/plastic step 25, see Table 3.2.4.2. As we want to have a fully implicit solution method that includes the displacement of the computational grid, we also make a grid iteration at each time step, i.e. we recompute the solution on the shifted grid until the stopping criterion (3.2.4.62) is fulfilled with $\varepsilon_{grid} = 10^{-3}$.

Table 3.2.4.2: Maximal values of the variables, of the max. relative error estimates and of the mean relative error estimates for time step 25.

no.	variable	max. value	max. relat. error	mean error
1	v_z	0.3006E-4	0.12	0.20E-1
2	v_r	0.1167E-2	0.14E-1	0.42E-2
3	σ_{zz}	569.3	0.14E-1	0.46E-3
4	σ_{rr}	8.31	0.21	0.23E-2
5	$\sigma_{\varphi\varphi}$	476.7	0.93E-2	0.43E-3
6	σ_{rz}	63.45	0.16E-1	0.28E-3

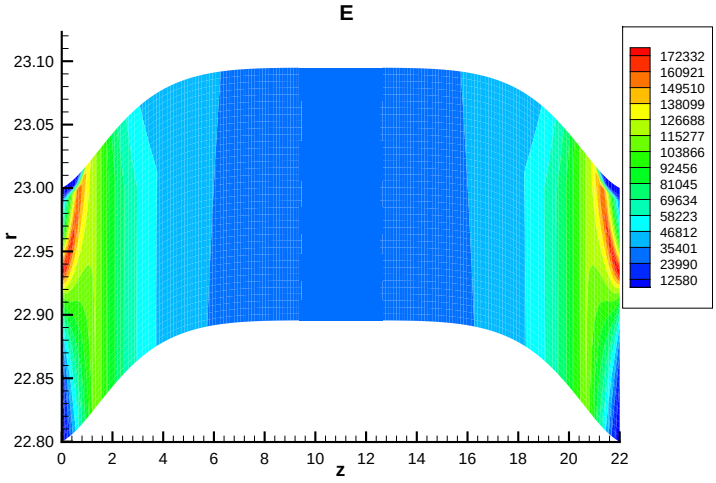


Figure 3.2.4.29: E-module $[N/mm^2]$ for time step 25.

Thus we have a solution on the new grid within the prescribed stopping criterion. However, the grid iteration diverges if we do not prescribe a sufficiently small time step size Δt. As a too large time step Δt needs many grid iterations or even causes the grid iteration to diverge and a too small time step proceeds very slowly in time direction, we programmed a

> time step size control (Δt control): If the number of grid iterations exceeds $n_{grid,max} = 20$, we set $\Delta t \Leftarrow \Delta t/2$ and we restart this time step. After 20 time steps we set $\Delta t \Leftarrow 2 \cdot \Delta t$. That Δt does not become too large we limit $\Delta t \leq \Delta t_{max} = 0.05\ ms$.

These values resulted from many numerical experiments.

In Table 3.2.4.3 we can see the values and error estimates of $\sigma_{\varphi\varphi}$ and v_r for step 26, computed with $\Delta t = 0.01$, i.e. for the first time step for the unsteady "dough" equations, for three different values of F. Clearly v_r is smaller for larger F (harder steel). Compare the results to Table 3.2.4.2. From these results we decided to use for the unsteady explosion computation the value $F = 10^5$.

Table 3.2.4.3: F-test: variables and error estimates for different values of F for step 26 with $\Delta t = 0.01\ ms$.

F	$\sigma_{\varphi\varphi}$	error $\sigma_{\varphi\varphi}$	v_r	error v_r
10^3	476.8	0.12E-1	0.1574E-2	4.32
10^4	476.8	0.12E-1	0.1207E-2	0.56E-1
10^5	476.8	0.12E-1	0.1171E-2	0.56E-1

For the following unsteady computation we selected the following parameters:

$\Delta t_{start} = 0.01\ ms,\ \Delta t_{max} = 0.05\ ms,$
$n_{grid,max} = 20,$
F= $10^5\ Ns/mm^2$.

Table 3.2.4.4: Values of the 6 variables, of their maximal and of their mean global relative error estimates for time step 26.

no.	variable	max. value	max. relat. error	mean error
1	v_z	0.3021E-4	0.31	0.48E-1
2	v_r	0.1171E-2	0.56E-1	0.12E-1
3	σ_{zz}	569.3	0.18E-1	0.25E-3
4	σ_{rr}	8.31	0.18	0.24E-2
5	$\sigma_{\varphi\varphi}$	476.8	0.12E-1	0.22E-3
6	σ_{rz}	63.45	0.22E-1	0.22E-3

Table 3.2.4.4 shows the values of the 6 variables, of their maximal and of their mean global relative error estimates for time step 26, i.e. the first time step with the "dough" equations. If we compare the values of the variables to those of Table 3.2.4.2, i.e. to the last step of the elastic/plastic equations, we see only minor changes that come from the increased time. This shows that the "dough" equations give the expected results. However, the max. errors have increased by factors between 4 (for v_r) and 1.25 (for σ_{zz} and $\sigma_{\varphi\varphi}$). So the transition from the quasisteady elastic/plastic equations to the unsteady "dough" equations is satisfactory. Observe that the max. error of v_r is 5.6% and the mean error is 1.2%. This is explained by the "waves" of Fig. 3.2.4.16 that result already from the elastic/plastic equations and thus are transferred to the "dough" equations.

We then started the computation in time direction for the "dough" equations, with $\Delta t_{start} = 0.01\ ms$, $F = 10^5\ Ns/mm^2$. We hoped that the metal sheet is pressed into the form by the internal pressure and then the tool closes to form the wave of the bellow. We needed one and a half year of painful time to recognize that this is not possible with this system of equations. The reason is: The system of "dough" PDEs is unstable in time. Unstable means that small disturbances increase with time and destroy the solution. We attributed the failure at first to our program code, then to the computational parameters and/or to the solution algorithm. Neither finer step sizes in space or time

could change the basic behaviour, nor smaller tolerances for the grid iteration, nor other values of the F-module. The algorithm is fully implicit in time, including the shifting of the grid, i.e. the solution is that on the new grid. Thus our numerical method proves the instability of the "dough" equations empirically in time. A theoretician could make a linearized Fourier analysis in time, he would get the same conclusion. From our initial experiments with the corresponding elastic/plastic equations we can conclude that the total derivative in time of these equations leads also to unstable equations in time.

Table 3.2.4.5: Function values and max. global relative errors of $\sigma_{\varphi\varphi}$ and v_r for time steps $45 - 465$ (every 20^{th} step) and Δt of that step. Some of the maximal values have negative sign.

time step	$\sigma_{\varphi\varphi}$	max. error $\sigma_{\varphi\varphi}$	v_r	max. error v_r	Δt
45	477.2	0.12E-1	0.1246E-2	0.53E-1	0.01
65	478.0	0.12E-1	0.1407E-2	0.47E-1	0.02
85	479.7	0.12E-1	0.1731E-2	0.38E-1	0.04
105	481.7	0.12E-1	0.2142E-2	0.31E-1	0.05
125	483.7	0.11E-1	0.2558E-2	0.28E-1	0.05
145	485.7	0.12E-1	0.2979E-2	0.52E-1	0.05
165	487.9	0.18E-1	0.3406E-2	0.10	0.05
185	489.5	0.63E-1	0.3794E-2	0.58	0.05
205	503.1	0.12	0.4166E-2	0.63	0.025
225	524.0	0.34	0.4415E-2	1.72	0.0125
245	546.5	0.49	0.7054E-2	2.47	0.16E-2
265	556.6	0.52	0.5603E-2	1.83	0.16E-2
285	599.6	0.90	0.7062E-2	2.52	0.31E-2
305	689.2	1.27	0.8418E-2	2.36	0.16E-2
325	661.1	1.29	0.8450E-2	2.42	0.78E-3
345	677.4	1.31	0.9966E-2	2.63	0.78E-3
365	716.3	3.33	0.1359E-1	19.7	0.39E-3
385	718.5	6.69	0.1599E-1	37.6	0.39E-3
405	736.6	11.8	-0.1611E-1	62.2	0.78E-3
425	799.8	16.1	-0.2005E-1	73.2	0.20E-3
445	837.4	19.0	-0.2143E-1	84.3	0.98E-4
465	871.1	22.3	-0.2227E-1	100.7	0.98E-4

Table 3.2.4.5 shows the max. function values and max. global relative error estimates of $\sigma_{\varphi\varphi}$ and v_r for time steps $45 - 465$ every 20^{th} time step. There is also shown the actual time step size that is controlled by the grid iteration as explained above and has an upper limit of $0.05\ ms$. Up to step 125 the max. relative error of v_r is below 3%, this means that the mean error is below 1%, compare to Table 3.2.4.4. So we have still a "good" solution. However, beyond step 125 the max. global relative error estimate above for v_r starts growing and has at step 245 a value of 247% which means the solution is nonsense. In Figs. 3.2.4.30 to 3.2.4.36 are shown the values of v_r and a blow-up between $z = 2$ and $z = 4$ for time steps $125 - 245$ every 20^{th} step. These figures show drastically how small

disturbances of v_r that are already visible in Fig. 3.2.4.16 at first grow slowly and then grow very fast and destroy the solution. Even the most sophisticated solution method like ours cannot suppress the growing of the disturbances because the system of PDEs is inherently unstable in time direction. Only PDEs with damping terms could cure the situation.

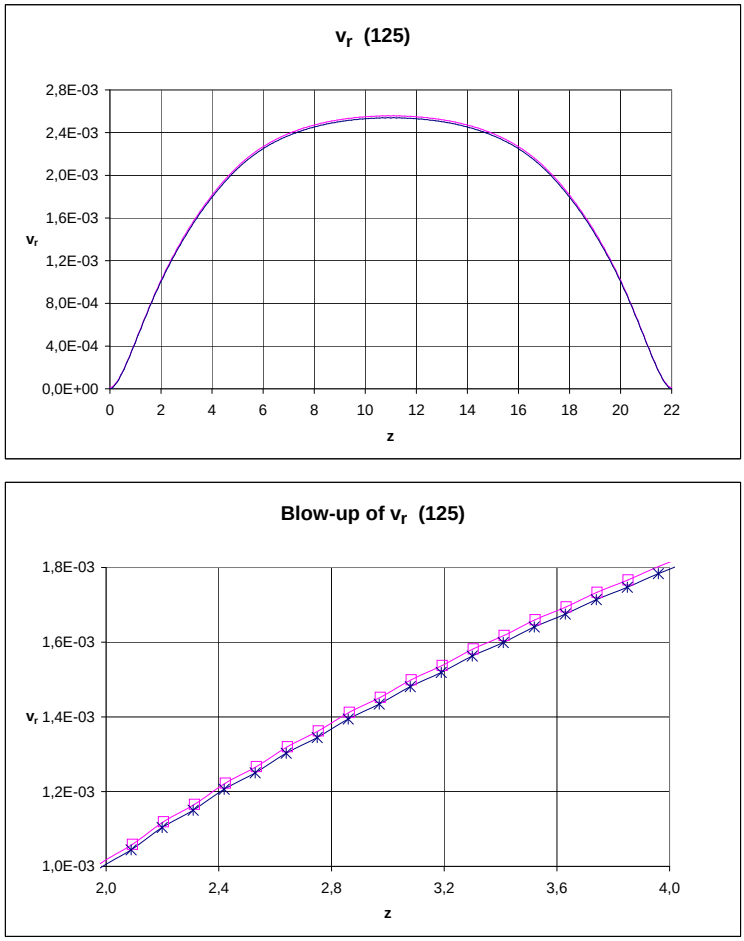


Figure 3.2.4.30: v_r and a blow-up for time step 125, stars: upper, squares: lower surface.

All attempts failed to damp the oscillations by smoothing the solution: We used the nodes of the nearest neighbor ring, attributed to the value of the central node a weight α and to the values of the remaining nodes a value $(1-\alpha)$, with different values of α. However, this smoothing is an additional disturbance of the solution, the errors increased and the solution became nonsense quite earlier than without smoothing. Also repeated smoothing steps did not help. This demonstrates that one cannot do better than to solve the equations without additional intervention.

What does this result mean? The real physical steel tube “explodes” into the form. This process is surely described by model equations of “dough” type. We are “numerical engineers”, no metallurgists. According to our assumption dissipative terms are missing in the equations. If we get from whomsoever the correct unsteady PDEs that describe the explosion forming, we will solve them with

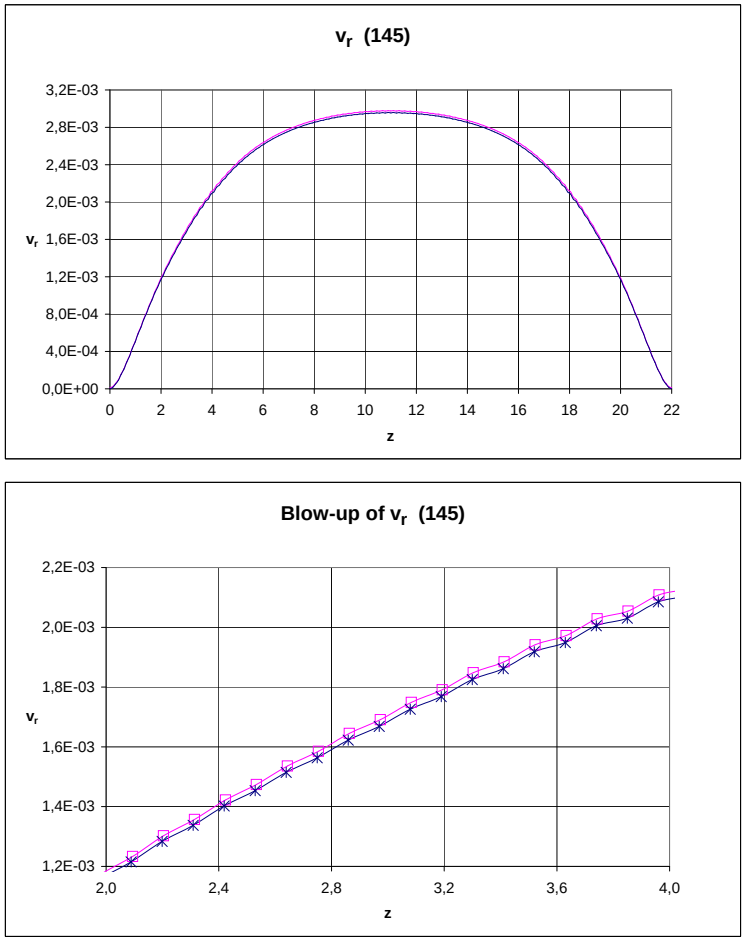


Figure 3.2.4.31: v_r and a blow-up for time step 145, stars: upper, squares: lower surface.

error estimate. There is the physical phenomenon that the steel tube explodes into the form, thus there must be a system of model equations that describe this process. Their numerical solution with error estimate then gives the simulation of the manufacturing process in the computer and allows the optimization of the process surely better than by trial and error.

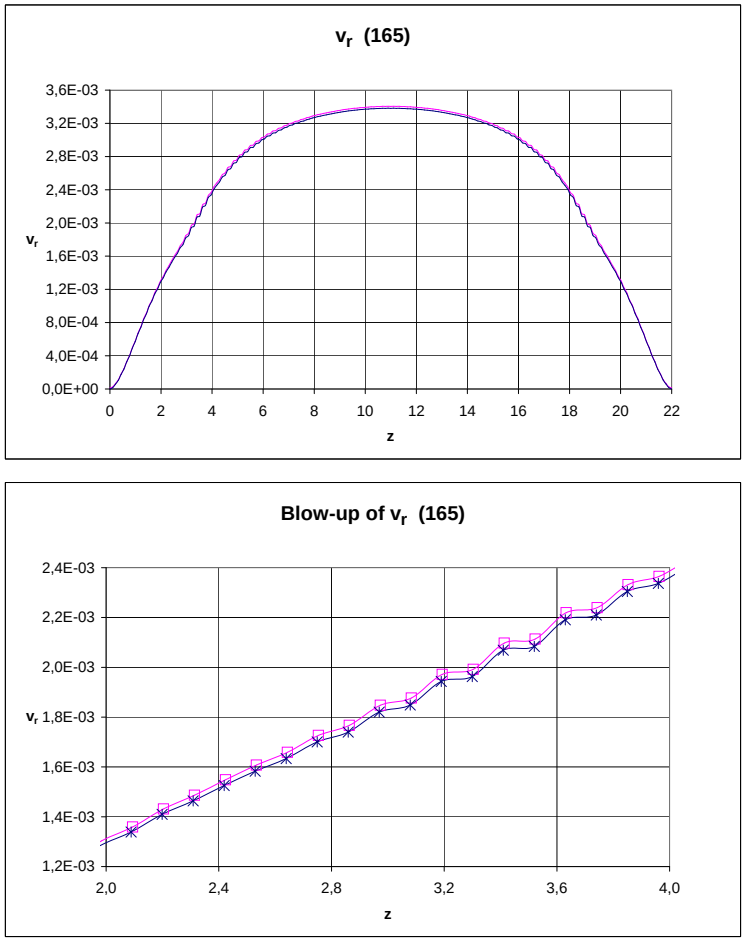


Figure 3.2.4.32: v_r and a blow-up for time step 165, stars: upper, squares: lower surface.

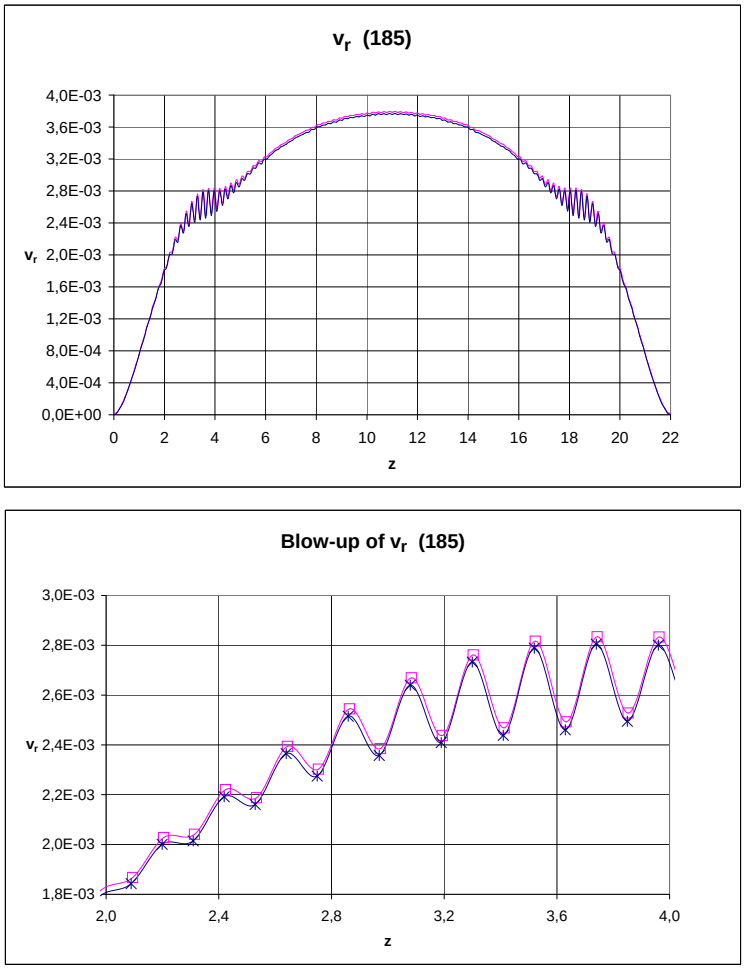


Figure 3.2.4.33: v_r and a blow-up for time step 185, stars: upper, squares: lower surface.

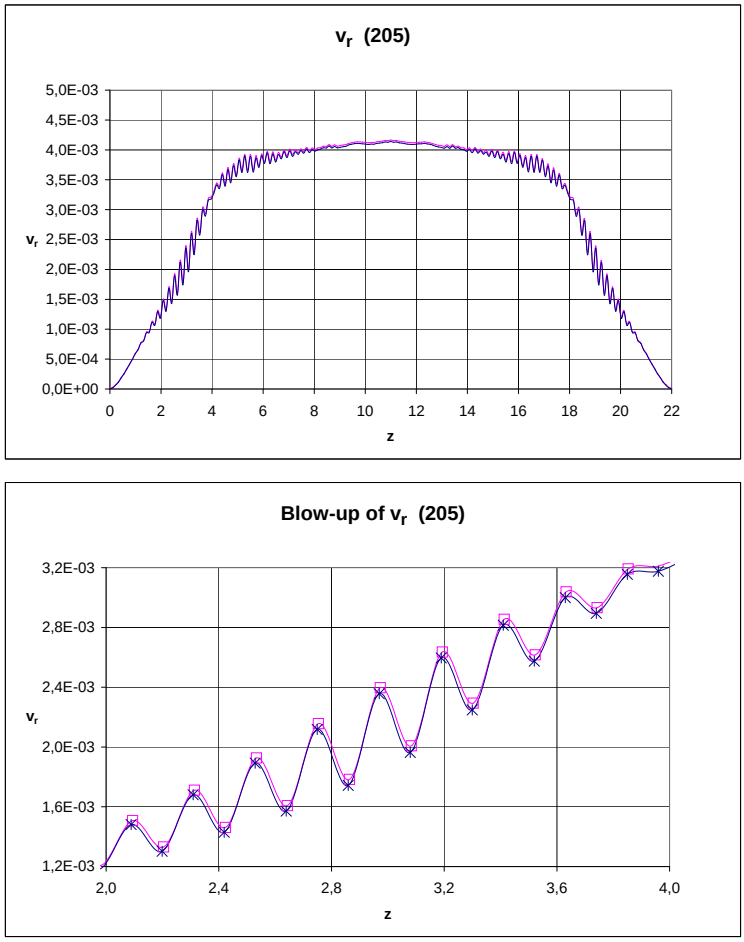


Figure 3.2.4.34: v_r and a blow-up for time step 205, stars: upper, squares: lower surface.

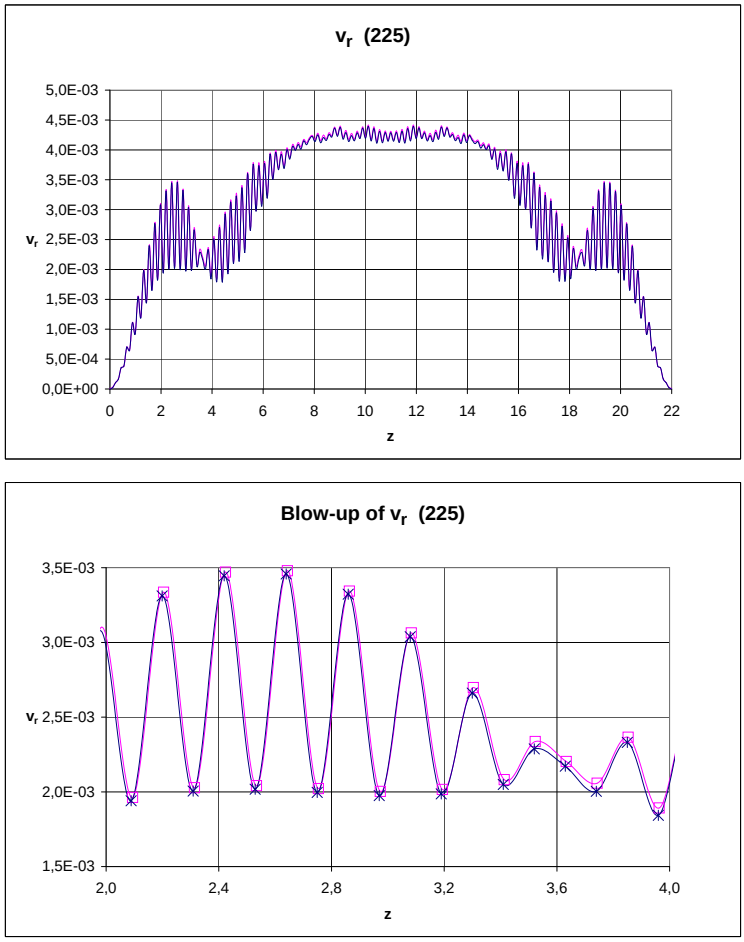


Figure 3.2.4.35: v_r and a blow-up for time step 225, stars: upper, squares: lower surface.

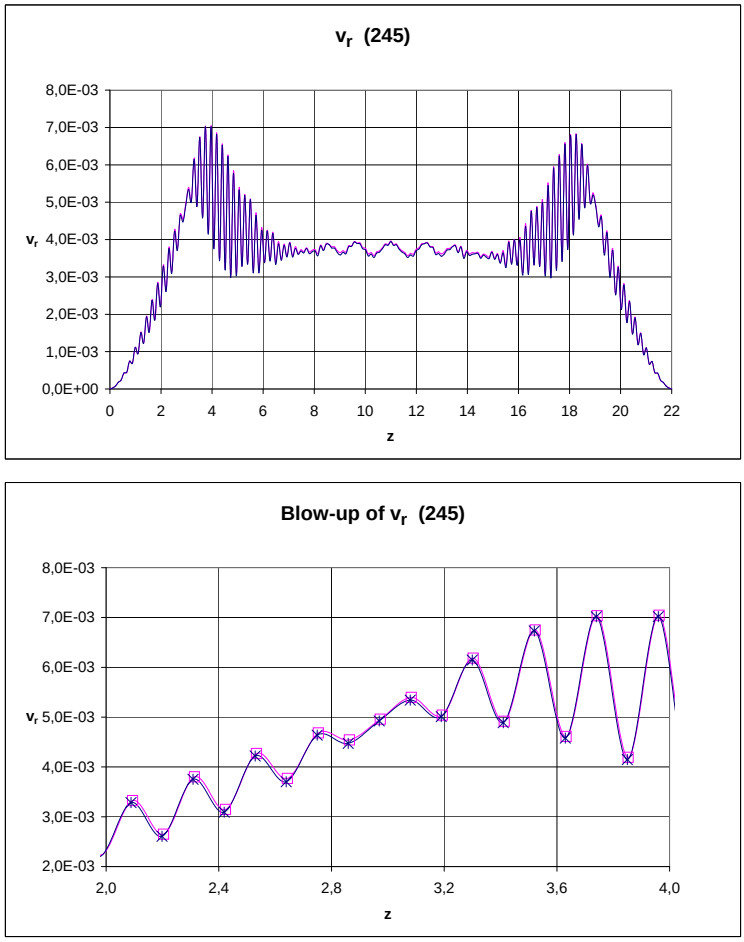


Figure 3.2.4.36: v_r and a blow-up for time step 245, stars: upper, squares: lower surface.

3.2.5 Concluding remarks to Section 3.2

The intention of the cooperation with the IWKA was the numerical simulation of the manufacturing of metal bellows. The IFU of the University of Stuttgart should deliver the PDEs that describe this process, empirical parameters in these equations should be determined by measurements at the IFU. Our part was to solve these equations with FDEM with a reliable error estimate.

However, the IFU did not have the PDEs for elastic/plastic deformation. They themselves made computations for metal forming processes, but they used commercial FEM codes where the equations that describe the forming process are hidden in a variational formulation. Therefore the IFU developed a fundamental theory for the PDEs of plastic deformation. Unfortunately all attempts to solve problems with this set of PDEs failed. These fruitless attempts consumed a large part of the project time. Then the IFU simplified the model PDEs until a "usable" system of PDEs for elastic/plastic problems was developed. At that time the cooperative project time ended and we were left alone without further support by the IFU. Up to that time IFU and IWKA believed that the forming of the metal bellows was basically hydroforming until then finally the tool closed and there was final force-forming. As mentioned above we tried one and a half years to simulate numerically this process until we had to recognize that the metal tube exploded into the tool. As we are no metallurgists we invented by phantasy model equations that could eventually describe this explosion forming. However, we had now to recognize that these equations are unstable in time because they had no damping terms.

At this point we had to give up because our financial and personal resources were exhausted. To solve the original problem we needed a cooperation partner that can deliver the stable PDEs for this explosion forming process.

Our part in the common research project with IFU and IWKA was to demonstrate that FDEM can solve the PDEs that describe the numerical simulation of metal bellow manufacturing. We have demonstrated that FDEM can solve all types of PDEs that we got from the IFU. What nobody else can do was possible by FDEM: to give for all these different types of solutions an error estimate. So we are not happy that we could not simulate the manufacturing process, but we are quite satisfied by the fact that FDEM can simulate all processes for which there are the corresponding PDEs. If we should simulate the manufacturing process of metal bellows somebody must give us the PDEs and we will solve them, —with error estimate.

3.3 Simulation of the lubrication gap of a Diesel High Pressure Injection Pump

3.3.1 The Piston and the Housing

In modern Diesel High Pressure Injection Pumps there is at the high pressure end a pressure of $2000\ bar$ or $2 \cdot 10^8\ N/m^2 = 200\, N/mm^2$. The width of the lubrication and caulking gap between piston and housing is only a few micrometers. Under the high pressure the housing is widened and the piston compressed so that the gap widens and changes its form. The housing is for simplicity taken as a tube. So we have symbolically the configuration of Fig. 3.3.1.1.

If we look at the effect of the pressure p at the high pressure end we have the situation of Fig. 3.3.1.2: by the effect of the pressure of $p = 200\, N/mm^2$ at the high pressure end the housing extends and the piston shrinks. Although we have a rotationally symmetric configuration so that

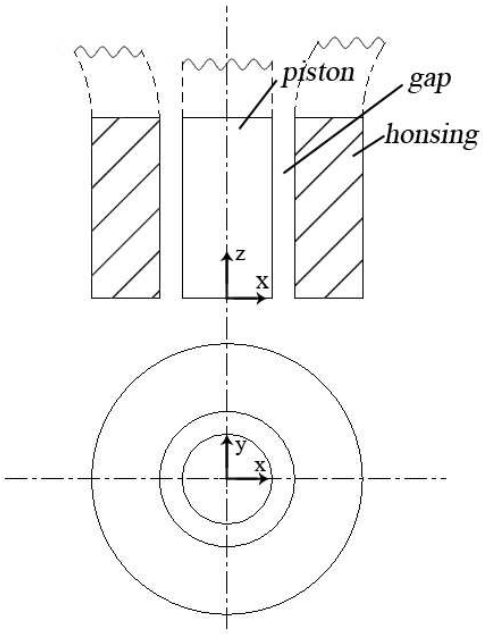


Figure 3.3.1.1: Symbolic Configuration. In reality the gap is extremely thin.

we could use cylindrical coordinates we use cartesian coordinates x, y, z that we can treat also unsymmetric configurations, e.g. the piston not in the center of the tube or an arbitrary housing. We have 3 domains: the piston, the housing (which is in our case a tube) and the lubrication gap. We will treat at first the 3 domains separately to gain experience for the needed grid spacing, and where there are problems we treat at first the 2-D case because 3-D with fine grid is very expensive w.r.t. computation and memory. Then we will combine the 3 domains to a single domain with two dividing lines (that are in 3-D in reality dividing plains) between housing and lubrication gap and between piston and lubrication gap. Then we will get a global solution over the whole domain although in the different domains hold different PDEs.

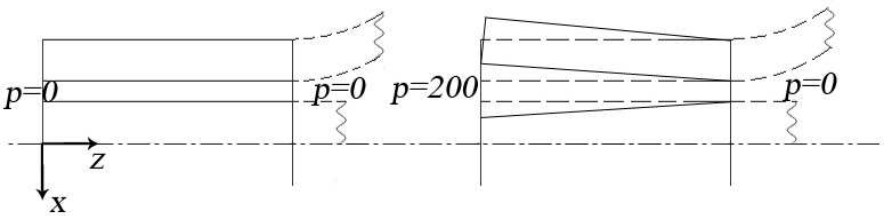


Figure 3.3.1.2: Effect of the pressure of $200\ N/mm^2$ at the high pressure end.

We now want to discuss the separate solution of the elasticity equation for the piston and the housing. We denote the displacement in x, y, z-direction by u, v, w. The normal stresses are denoted by σ_x, σ_y, σ_z (in Section 3.2.1 they were denoted by σ_{xx}, σ_{yy}, σ_{zz}), the shear stresses are denoted by τ_{xy}, τ_{yz}, τ_{xz} with $\tau_{xy} = \tau_{yx}$ etc. (in Section 3.2.1 they were denoted by σ_{xy}, σ_{yz}, σ_{xz}). The elasticity equations for isotropic material that can be obtained from [12] in the same way as in Section 3.2.1 are with the elasticity module E and Poisson's ratio ν:

$$\frac{1}{E}(\sigma_x - \nu\sigma_y - \nu\sigma_z) - \frac{\partial u}{\partial x} = 0, \tag{3.3.1.1}$$

$$\frac{1}{E}(-\nu\sigma_x + \sigma_y - \nu\sigma_z) - \frac{\partial v}{\partial y} = 0, \tag{3.3.1.2}$$

$$\frac{1}{E}(-\nu\sigma_x - \nu\sigma_y + \sigma_z) - \frac{\partial w}{\partial z} = 0, \tag{3.3.1.3}$$

$$\frac{1+\nu}{E}\tau_{xy} - \frac{1}{2}(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}) = 0, \tag{3.3.1.4}$$

$$\frac{1+\nu}{E}\tau_{yz} - \frac{1}{2}(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}) = 0, \tag{3.3.1.5}$$

$$\frac{1+\nu}{E}\tau_{xz} - \frac{1}{2}(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}) = 0. \tag{3.3.1.6}$$

The equilibrium equations are

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} = 0, \tag{3.3.1.7}$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} = 0, \tag{3.3.1.8}$$

$$\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} = 0. \tag{3.3.1.9}$$

We use the values

$$E = 2.1 \cdot 10^{11}\ N/m^2 = 210\,000\ N/mm^2, \qquad \nu = 0.3. \tag{3.3.1.10}$$

In Table 3.3.1.1 we give the sequence of the variables and corresponding equations for the interior nodes for the generation of the matrix.

Before we discuss the BCs we give the stress vector with its components

$$\begin{array}{rcl} \sigma & = & \begin{pmatrix} \sigma_x n^x + \tau_{xy} n^y + \tau_{xz} n^z \\ \tau_{xy} n^x + \sigma_y n^y + \tau_{yz} n^z \\ \tau_{xz} n^x + \tau_{yz} n^y + \sigma_z n^z \end{pmatrix} \cdot \begin{array}{l} x\text{-component} \\ y\text{-component} \\ z\text{-component}. \end{array} \\ \text{surface normal in} & & x\text{-} \quad y\text{-} \quad z\text{-direction} \end{array} \tag{3.3.1.11}$$

If we denote by n the normal and by t the tangential vector in the x, y-plane ($n^z = 0$) we have (see Fig. 3.2.1.3)

$$n = \begin{pmatrix} n^x \\ n^y \end{pmatrix}, \quad t = \begin{pmatrix} t^x \\ t^y \end{pmatrix} = \begin{pmatrix} -n^y \\ n^x \end{pmatrix}. \tag{3.3.1.12}$$

Table 3.3.1.1: Sequence of variables and equations.

no.	variable	equation
1	u	(3.3.1.1)
2	v	(3.3.1.2)
3	w	(3.3.1.3)
4	σ_x	(3.3.1.7)
5	σ_y	(3.3.1.8)
6	σ_z	(3.3.1.9)
7	τ_{xy}	(3.3.1.4)
8	τ_{yz}	(3.3.1.5)
9	τ_{xz}	(3.3.1.6)

We get in this notation with the 2-D restriction of σ the normal stress σ_n and tangential stress σ_t in the x, y-plane.

$$\sigma_n = \sigma^T \cdot n = \sigma_x (n^x)^2 + 2\tau_{xy} n^x n^y + \sigma_y (n^y)^2, \tag{3.3.1.13}$$

$$\sigma_t = \sigma^T \cdot t \; = \; -\sigma_x n^x n^y + \tau_{xy}((n^x)^2 - (n^y)^2) + \sigma_y n^x n^y. \tag{3.3.1.14}$$

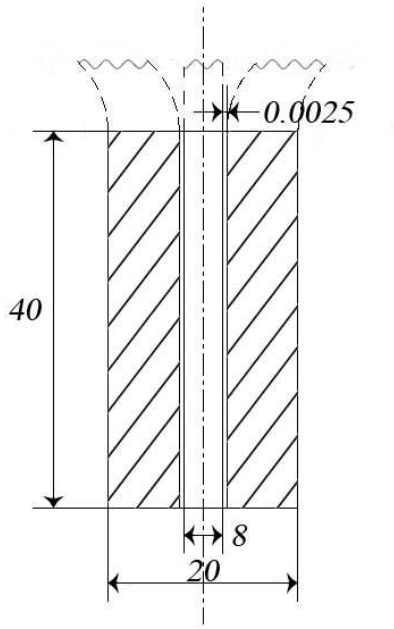


Figure 3.3.1.3: Dimensions of housing, piston and lubrication gap in mm.

Fig. 3.3.1.3 shows the dimensions of the computational domain in mm. The width of the lubrication gap is $2.5\ \mu m = 0.0025\ mm$. We want at first to discuss the BCs of the housing. Fig. 3.3.1.4 shows a view of the housing with the 4 fixed nodes on the bottom. At these nodes we admit only radial displacement: We have at node

$$\begin{aligned} 1+2 &: v = 0,\ w = 0, \\ 3+4 &: u = 0,\ w = 0. \end{aligned} \tag{3.3.1.15}$$

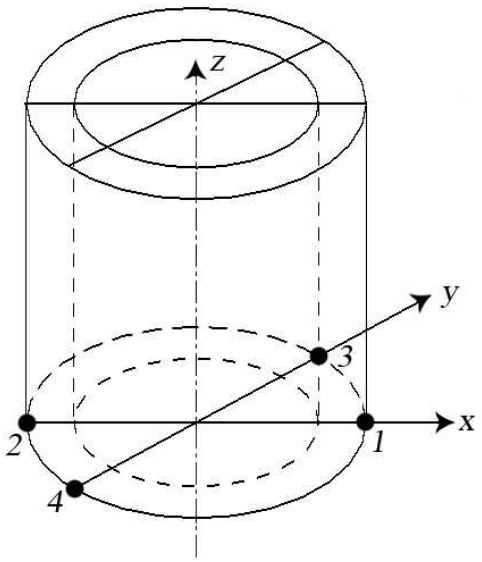


Figure 3.3.1.4: View of the housing with the 4 fixed nodes, computational domain.

At the bottom we assume the pressure $p = 200\ N/mm^2$, i.e. the normal stress is $\sigma_n = -p$. The normal vector of the bottom is $n^x = n^y = 0,\ n^z = -1$, thus with σ from (3.3.1.11) we get

$$\sigma_n = \sigma \cdot n = \sigma_z = -p.$$

The tangential stress in x- and y-direction is zero. The tangential vectors in x- and y-direction are

$$t^x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad t^y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},$$

thus the stresses are

$$\sigma_{t^x} = \sigma \cdot t^x = -\tau_{xz} = 0, \quad \sigma_{t^y} = \sigma \cdot t^y = -\tau_{yz} = 0.$$

For the other components we must use appropriate PDEs. So the BCs for the bottom are

$$\begin{array}{rll} u: & PDE\ (3.3.1.6), & \left.\begin{array}{l}\\\\\end{array}\right\rceil \textit{special values (3.3.1.5) for} \\ v: & PDE\ (3.3.1.5), & \left.\begin{array}{l}\\\\\end{array}\right| \textit{nodes 1-4 of Fig. 3.3.1.4} \\ w: & PDE\ (3.3.1.3), & \left.\begin{array}{l}\\\\\end{array}\right\rfloor \\ \sigma_x: & PDE\ (3.3.1.1), & \\ \sigma_y: & PDE\ (3.3.1.2), & \\ \sigma_z: & \sigma_z = -p\ (p = 200\,N/mm^2), & \\ \tau_{xy}: & PDE\ (3.3.1.4), & \\ \tau_{yz}: & \tau_{xz} = 0, & \\ \tau_{xz}: & \tau_{yz} = 0. & \end{array} \tag{3.3.1.16}$$

As shown in Fig. 3.3.1.1 the housing and piston must be fixed by a continuation of the material that take over the forces created by the $200\,N/mm^2$ at the bottom. We restrict the computational domain as shown in Fig. 3.3.1.3. Therefore the "lid" (upper boundary of the computational domain) is an "artificial" boundary where no values are prescribed. Thus there hold the PDEs of Table 3.3.1.1, i.e. the same equations like in the interior of the domain. The outer shell (without the nodes of bottom and lid) is a free surface where the stress σ is zero. Because there we have the normal in the x, y-plane, i.e. $n^z = 0$, this means that we retain in (3.3.1.11) in each row the first two terms which then are zero, e.g. $\sigma_x n^x + \tau_{xy} n^y = 0$. This can be used as equation for σ_x or τ_{xy}. However, where $n^x = 0$ we cannot use this equation for σ_x and where $n^y = 0$ we cannot use it for τ_{xy}. Therefore we must use different BCs depending on n^x and n^y. So the BCs for the outer shell (without bottom and lid nodes) are:

$$\begin{array}{rll} \text{for} & |n^x| \geq |n^y| & |n^x| < |n^y| \\ & & \\ u: & PDE\ (3.3.1.1), & PDE\ (3.3.1.4), \\ v: & PDE\ (3.3.1.4), & PDE\ (3.3.1.5), \\ w: & PDE\ (3.3.1.6), & PDE\ (3.3.1.6), \\ \sigma_x: & \sigma_x n^x + \tau_{yx} n^y = 0, & PDE\ (3.3.1.1), \\ \sigma_y: & PDE\ (3.3.1.2), & \tau_{xy} n^x + \sigma_y n^y = 0, \\ \sigma_z: & PDE\ (3.3.1.3), & PDE\ (3.3.1.3), \\ \tau_{xy}: & \tau_{xy} n^x + \sigma_y n^y = 0, & \sigma_x n^x + \tau_{xy} n^y = 0, \\ \tau_{yz}: & PDE\ (3.3.1.5), & \tau_{xz} n^x + \tau_{yz} n^y = 0, \\ \tau_{xz}: & \tau_{xz} n^x + \tau_{yz} n^y = 0, & PDE\ (3.3.1.6). \end{array} \tag{3.3.1.17}$$

At the inner shell we have the normal stress component equal to the negative hydrostatic pressure p (opposite to p of the fluid) which means with equ. (3.3.1.13)

$$\sigma_x (n^x)^2 + 2\tau_{xy} n^x n^y + \sigma_y (n^y)^2 + p = 0. \tag{3.3.1.18}$$

There is no tangential stress because there is no circumferential force acting on the inner shell. Therefore we put the tangential stress to zero which is with equ. (3.3.1.14)

$$-\sigma_x n^x n^y + \tau_{xy}((n^x)^2 - (n^y)^2) + \sigma_y n^x n^y = 0. \tag{3.3.1.19}$$

In the z-direction we have the frictional force by the fluid in the lubrication gap. This force is very small relative to the other forces so that we can neglect it. So we put the stress component in z-direction to zero which is with (3.3.1.11) and with $n^z = 0$ (shell is orthogonal to the z-axis)

$$\tau_{xz} n^x + \tau_{yz} n^y = 0. \tag{3.3.1.20}$$

If we want to use these relations (3.3.1.18) to (3.3.1.20) as BCs we can e.g. not use (3.3.1.18) as equation for σ_x or τ_{xy} if we have $n^x = 0$ or (3.3.1.19) not for τ_{xy} if $n^x = n^y$. These restrictions are shown in Fig. 3.3.1.5 Therefore we subdivide the "disk" in the x, y-plane in 3 sectors I, II, III as shown in Fig. 3.3.1.6. We use in the sectors the equations in the following way

$$\begin{array}{rl}
\text{Sector I} & (3.3.1.18) \text{ for } \sigma_x, \\
& (3.3.1.19) \text{ for } \tau_{xy}, \\
& (3.3.1.20) \text{ for } \tau_{xz}, \\
\text{Sector II} & (3.3.1.18) \text{ for } \sigma_x, \\
& (3.3.1.19) \text{ for } \sigma_y, \\
& (3.3.1.20) \text{ for } \tau_{yz}, \\
\text{Sector III} & (3.3.1.18) \text{ for } \sigma_y, \\
& (3.3.1.19) \text{ for } \tau_{xy}, \\
& (3.3.1.20) \text{ for } \tau_{yz}.
\end{array} \tag{3.3.1.21}$$

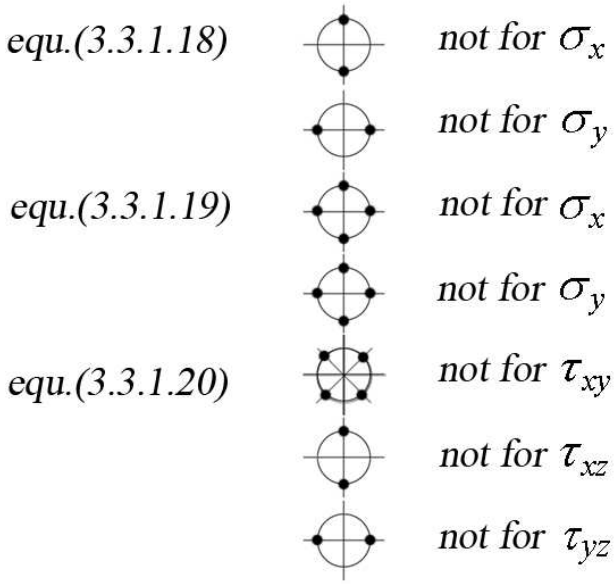


Figure 3.3.1.5: Usage of equs.(3.3.1.18)–(3.3.1.20).

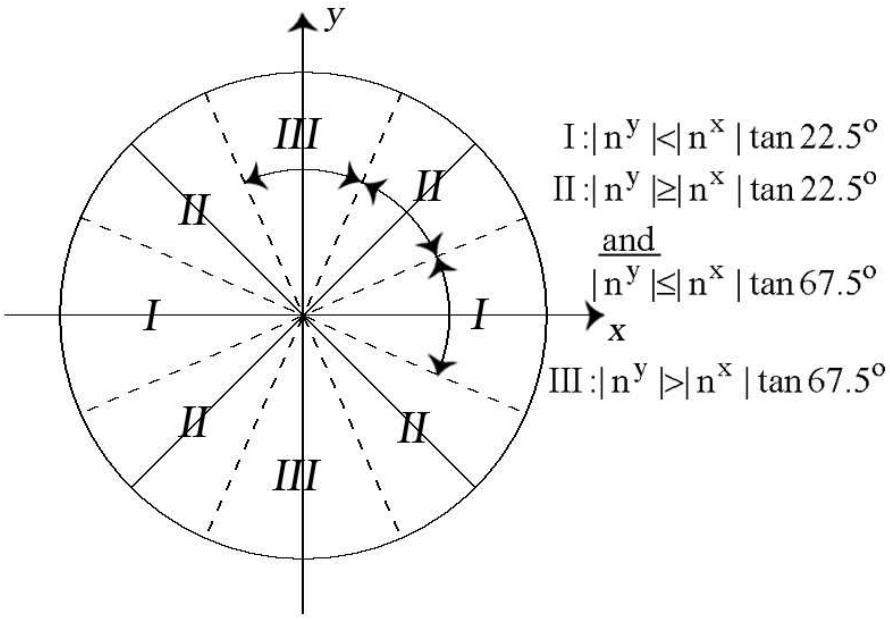


Figure 3.3.1.6: Sectors I-III.

Therefore we have for the inner shell of the tube (without bottom and lid nodes) the following BCs:

for	Sector I	Sector II	Sector III	
$u:$	PDE (3.3.1.1),	PDE (3.3.1.1),	PDE (3.3.1.4),	
$v:$	PDE (3.3.1.4),	PDE (3.3.1.2),	PDE (3.3.1.2),	
$w:$	PDE (3.3.1.6),	PDE (3.3.1.5),	PDE (3.3.1.5),	
$\sigma_x:$	$equ.$ (3.3.1.18),	$equ.$ (3.3.1.18),	PDE (3.3.1.1),	(3.3.1.22)
$\sigma_y:$	PDE (3.3.1.2),	$equ.$ (3.3.1.19),	$equ.$ (3.3.1.18),	
$\sigma_z:$	PDE (3.3.1.3),	PDE (3.3.1.3),	PDE (3.3.1.3),	
$\tau_{xy}:$	$equ.$ (3.3.1.19),	PDE (3.3.1.4),	$equ.$ (3.3.1.19),	
$\tau_{yz}:$	PDE (3.3.1.5),	$equ.$ (3.3.1.20),	$equ.$ (3.3.1.20),	
$\tau_{xz}:$	$equ.$ (3.3.1.20),	PDE (3.3.1.6),	PDE (3.3.1.6).	

Fig. 3.3.1.7 shows the piston with the two fixed nodes. There we have at fixed nodes:

$$v = 0, \quad w = 0. \tag{3.3.1.23}$$

The BCs for the bottom are the same as for the tube (3.3.1.16), but now with the special values (3.3.1.23) at the bottom. At the lid (artificial boundary) hold the equations of Table 3.3.1.1. The BCs for the outer shell of the piston (without nodes of the bottom and lid) are the same as for the inner shell of the tube (3.3.1.22).

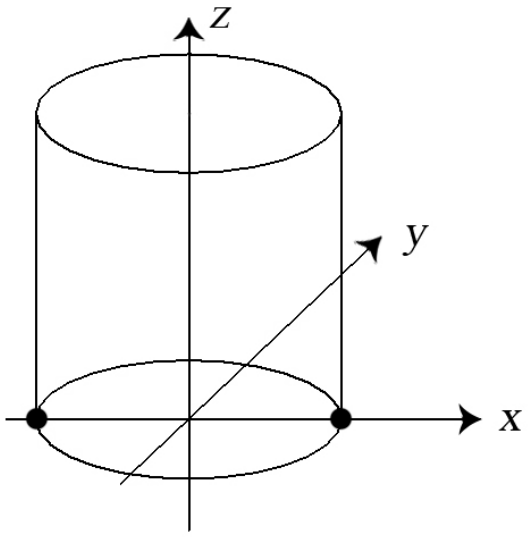


Figure 3.3.1.7: The piston with the two fixed nodes, computational domain.

For the numerical solution of the elasticity equations (3.3.1.1)–(3.3.1.6) we have the following values of the E-module E and Poisson's ratio ν:

$$E = 210\,000\ N/mm^2 (= 2.1 \cdot 10^{11}\ N/m^2), \quad \nu = 0.3. \tag{3.3.1.24}$$

Because 3-D is very expensive in computation and storage, we made at first 2-D experiments with a cross section orthogonal to the z-axis in Fig. 3.3.1.1 or 3.3.1.3. This is an annulus with inner radius $r_i = 4.0025\ mm$ and outer radius $r_a = 10\ mm$. It is exposed to an inner pressure $p_i = 200\ N/mm^2$ and an outer pressure 0, see Fig. 3.3.1.8. For this configuration the exact solution is given in [13], p. 60, equ. (46), which is in our notation, but with polar coordinates r, φ:

$$\begin{aligned} \sigma_r &= \frac{r_i^2 \cdot p_i}{r_a^2 - r_i^2}(1 - \frac{r_a^2}{r^2}), \\ \sigma_\varphi &= \frac{r_i^2 \cdot p_i}{r_a^2 - r_i^2}(1 + \frac{r_a^2}{r^2}). \end{aligned} \tag{3.3.1.25}$$

The value of $\sigma_{r\varphi} = 0$ because of the symmetry. The max. stress σ_φ occurs at the inner ring, $r = r_i$. There we have for our configuration $\sigma_\varphi = 276.3039\ N/mm^2$. We compute in cartesian coordinates. Therefore at node A $\sigma_\varphi = \sigma_y$, or similarly at node B $\sigma_\varphi = \sigma_x$. For $r = r_i$ we get from (3.3.1.25) $\sigma_r = 200\ N/mm^2$, i.e. the BC of the inner pressure.

The 2-D model results from the PDEs and BCs for $w = \sigma_z = \tau_{yz} = \tau_{xz} = 0$ and $\frac{\partial}{\partial z} = 0$. We have made numerical experiments with different grids for the consistency order $q = 6$. In Table 3.3.1.2

we present for each grid the displacement u and the stress σ_y of node A of Fig. 3.3.1.8 together with the estimated global relative error E.

These results are very instructive. As we know the exact value for $\sigma_y = 276.3$, we can compare the numerical value to the exact value and see the corresponding estimated error. It should be mentioned that this problem is a 1-D problem in polar coordinates, but we compute in cartesian coordinates that we can treat arbitrary unsymmetric configurations. If we go in cartesian coordinates in φ-direction around the annulus, σ_x, σ_y change continuously so that we need a corresponding fine grid for high accuracy.

If we go down the first column in Table 3.3.1.2 we can see that we get the exact value for σ_y for the grid 159×21, but the error estimate is 9% for u and 20% for σ_y. This means that the grid is not yet fine enough for the consistency order $q = 8$ that is used for the error estimate, i.e. the order $q = 8$ is still "overdrawn". If we refine the grid further in circumferential direction, the solution becomes worse and the error estimates are valueless. This shows the built-in self-control of the error estimate. If we go from 159×21 to 159×41 the error estimates increase, thus the finer grid in the thickness direction of the annulus does not improve the solution. The error estimates for the grid 319×41 are better than those of its upper and left neighbor. Finally the grid 319×81 gives error estimates for u as 0.24% and for σ as 0.85% so that we can well trust the solution. However, if we want to go from the 2-D annulus to the 3-D tube we cannot use such a fine grid because the storage becomes prohibitively large for a full LU solution of the linear system. Therefore we will restrict to the grid 159×21, but we cannot expect a usable error estimate because for this coarse grid the order $q = 8$ for the error estimate is overdrawn.

From Table 3.3.1.2 we can learn some interesting points. It is useless or even harmful to refine the grid only in one direction. The grid must be "balanced" in both directions. The error estimates, even if they are very large, indicate which grid is better. So the error estimate helps to select better grids. Finally, if we have a small error estimate, we know that our solution is in the corresponding range of

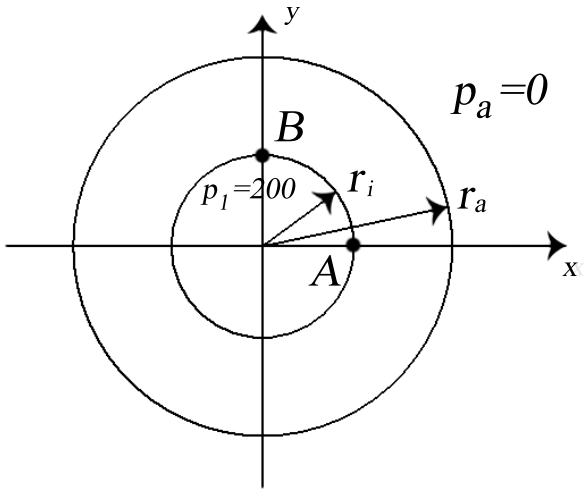


Figure 3.3.1.8: 2-D annulus.

Table 3.3.1.2: Results for node A of Fig. 3.3.1.8 for different grids. The first value of the grid gives the number of nodes in circumferential, the second in radial direction, E is the estimated global relative error. Consistency order $q = 6$. The exact value is $\sigma_y = 276.3$.

Var.	solution	error E	solution	error E	solution	error E
grid	79×21					
u	$0.6413{\cdot}10^{-2}$	0.31				
σ_y	277.0	1.13				
grid	159×21		159×41			
u	$0.6411{\cdot}10^{-2}$	$0.9{\cdot}10^{-1}$	$0.6412{\cdot}10^{-2}$	0.33		
σ_y	276.3	0.20	276.2	1.12		
grid	319×21		319×41		319×81	
u	$0.6426{\cdot}10^{-2}$	1.50	$0.6410{\cdot}10^{-2}$	$0.9{\cdot}10^{-1}$	$0.6410{\cdot}10^{-2}$	$0.24{\cdot}10^{-2}$
σ_y	277.0	6.27	276.3	0.17	276.3	$0.85{\cdot}10^{-2}$
grid	639×21					
u	$0.6710{\cdot}10^{-2}$	1480				
σ_y	5376.0	1794				

accuracy. In almost all cases the error is overestimated.

The values of Table 3.3.1.2 are computed with the consistency order $q = 6$. We made experiments with the orders $q = 4$ and $q = 2$, but the errors were considerably larger. To obtain solutions with comparable accuracy the grid must be very fine. Before we further discuss the treatment of housing and piston, we will discuss the equations for the flow in the lubrication gap.

3.3.2 The fluid flow in the lubrication gap

We solve in the lubrication gap, see Fig. 3.3.1.1, the incompressible Navier-Stokes equations with the velocity components u, v, w, the pressure p and constant values for density ρ and dynamical viscosity η, see e.g. [14]:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial x} - \\ -\frac{\eta}{\rho}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) = 0, \tag{3.3.2.1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial y} - \\ -\frac{\eta}{\rho}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) = 0, \tag{3.3.2.2}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial z} - \\ -\frac{\eta}{\rho}(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) = 0, \tag{3.3.2.3}$$

and the continuity equation

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0. \tag{3.3.2.4}$$

For FDEM it would be no problem to include the energy equation for the temperature T for the heat conduction or to solve the compressible equations with ρ as unknown. There are 4 equations for the 4 variables u, v, w, p. Which equation should be used for which variable in the discretization process? For the configuration of Fig. 3.3.1.1 we selected the following sequence

no.	variable	equation
1	u	(3.3.2.1)
2	v	(3.3.2.2)
3	w	(3.3.2.4)
4	p	(3.3.2.3)

(3.3.2.5)

Fig. 3.3.2.1 shows the fluid region. At the inner and outer mantle, i.e. the border to the piston and housing, we have for the velocities the no-slip condition: $u = v = w = 0$. For the pressure there is no prescribed value, therefore we take for p equ. (3.3.2.3). The problem are the inlet with the high

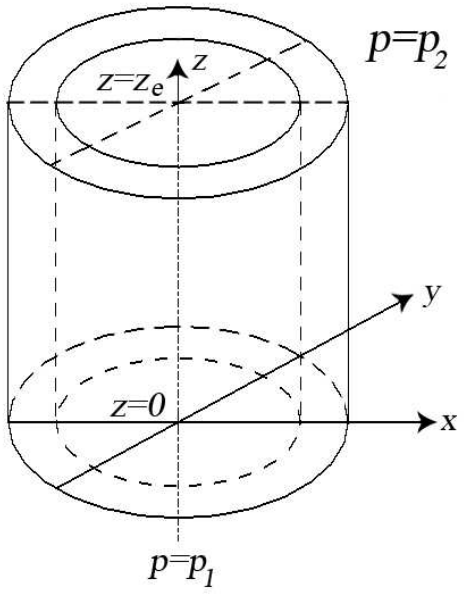


Figure 3.3.2.1: Fluid region.

pressure $p_1 = 200\ N/mm^2$ and the outlet with $p_2 = 0$. These are "artificial" boundaries because the computational domain of Fig. 3.3.2.1 has been cut out of the whole fluid flow. What we find at the inlet at $z = 0$ is determined by the conditions from where the fluid comes, and what we get at the outlet at $z = z_e$ depends in a certain sense also from the conditions that follow to the exit, although

in a quite less degree than at the inlet. Therefore the conditions that we describe at inlet and outlet are artificial ones. In this sense we prescribe at the inlet ($z = 0$):

$$\begin{aligned} u &= 0, \\ v &= 0, \\ w &= \text{parabola with } w_{max}, \\ p &= p_1 = 200\, N/mm^2, \end{aligned} \tag{3.3.2.6}$$

where w is a parabola of Fig. 3.3.2.2 with zero values at r_i, r_a and w_{max} in the middle. We establish

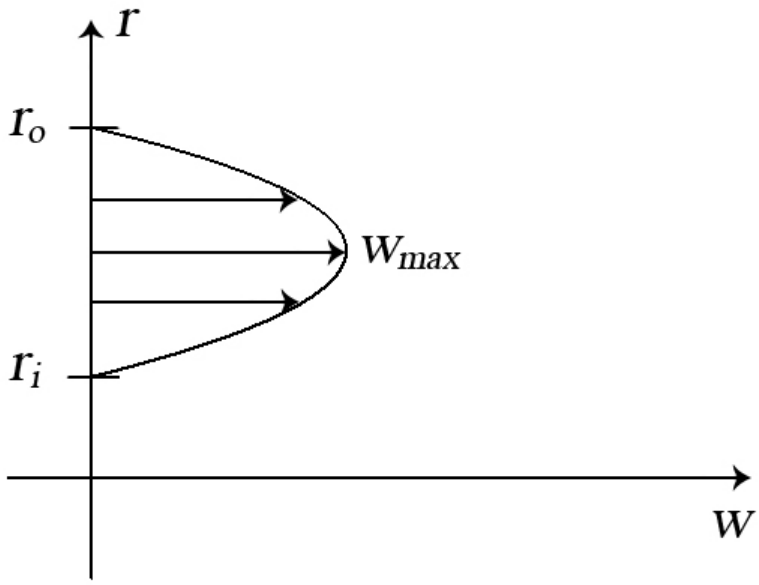


Figure 3.3.2.2: Parabolic form of w at the inlet.

w as $w(r)$, but then express r by x, y. At the outlet we want to let to the fluid as much freedom as possible and thus we prescribe for

$$\begin{aligned} u: &\quad \tfrac{\partial u}{\partial z} = 0, \\ v: &\quad \tfrac{\partial v}{\partial z} = 0, \\ w: &\quad \tfrac{\partial w}{\partial z} = 0, \\ p: &\quad \text{equ. } (3.3.2.3). \end{aligned} \tag{3.3.2.7}$$

The last condition for p is unexpected. We would expect that we have at the exit $p = p_2 = 0$ as shown in Fig. 3.3.2.1. However, we have for p only first derivatives and we prescribe $p = p_1$ at the inlet. Then from the inlet to the outlet there is a pressure drop which results from the solution of the

Navier-Stokes equations, i.e. by the friction at the walls. How can we get the desired value $p = 0$ at the outlet?

The pressure drop is determined by the velocity component w in the lubrication gap and this in turn is determined by w_{max}, see Fig. 3.3.2.2. If w_{max} is too large, the pressure drop is too large and the exit pressure is negative. If w_{max} is too small, the pressure drop is too small and the pressure at the exit is too large. But what means $p = 0$ at the exit? The exit is the lid of the computational domain of Fig. 3.3.1.2, the velocity at the exit and thus the exit pressure depends on the location. How to solve this problem? The method that we propose and that we call "w_{max}-iteration" is the following: We select an initial value w_{max} and solve the fluid equations. Then we select a control node in the middle of the exit and take the pressure from this node. We know (without computation) that for exit pressure $p_2 = p_1$, no pressure drop, $w_{max} = 0$. So we have two pairs $w_{max},\ p_{exit}$ and can extrapolate a third value for w_{max} and compute its p_{exit} by the solution of the PDEs with this value of w_{max} at the entry. From now on we have 3 pairs of $w_{max},\ p_{exit}$ and can with a parabola through these values start a Newton iteration to determine w_{max} for exit pressure to be zero. This explanation means that we can fulfil the pressure condition $p_2 = 0$ at the exit (in a "mean" pressure sense) only by an iterative procedure. The basic reason is that in the Navier-Stokes equations the pressure has only first derivatives, thus is of an initial value problem type for p. For the numerical solution of the Navier-Stokes equations the following constant parameters are prescribed:

$$\begin{aligned} \text{dynamical viscosity} \quad & \eta = 2 \cdot 10^{-3}\ Pa \cdot s, \\ \text{density} \quad & \rho = 800\ kg/m^3. \end{aligned}$$

As we take the length scale in mm we have

$$\begin{aligned} & \eta = 2 \cdot 10^{-9}\ Ns/mm^2, \\ & \rho = 8 \cdot 10^{-7}\ kg/mm^3, \\ & \tfrac{\eta}{\rho} = 2.5 \cdot 10^{-3}\ mm^2/s. \end{aligned} \tag{3.3.2.8}$$

See Erratum on page 151.

3.3.3 The combination of piston, housing and fluid flow

The problem that we want to solve is a fluid-structure interaction problem. As shown in Fig. 3.3.1.2 the pressure widens the lubrication gap (the fluid domain) at the high pressure end. The computational domain is the whole domain combined of housing, gap and piston, see Fig. 3.3.1.1. In the three domains hold different PDEs: in the housing and piston hold the elasticity equations with 9 variables, see Table 3.3.1.1, in the fluid gap hold the Navier-Stokes equations with 4 variables, see (3.3.2.5). As we must have in the whole domain the same number of variables, we add in the fluid domain 5 dummy equations of type: variable $= 0$ (in the interior and at the boundary). So e.g. in the domain of the housing and piston variale 1 has the meaning of the displacement u and in the fluid domain variable 1 has the meaning of velocity u, variable 4 has the meaning σ_x and fluid pressure p, variable 9 is τ_{xz} and 0, respectively.

This seems to be strange, but the different domains are separated by dividing lines (DLs) which are in 3-D in reality dividing areas as mentioned in the general part of FDEM. The DLs are internal boundaries over which will not be differentiated. As we have also sliding dividing lines (SDLs)

which allow a sliding of the domains relative to each other, we may have non-matching grids at the SDLs (at DLs the grids must match).

The solutions in the different domains are coupled across the DLs or SDLs by coupling conditions (CCs). In our case of housing, gap and piston we have the following simple CC: at the inner boundary of the housing and at the outer boundary of the piston the normal stress is given by the negative fluid pressure p, see (3.3.1.18) and (3.3.1.22). Thus we have a "one-sided" coupling: the structure couples to the fluid, but the fluid does (seemingly) not couple to the structure, i.e. the fluid has no explicit coupling with a structure variable. However, there is a much more complicated (hidden) back-coupling of the fluid to the structure: The fluid pressure widens the housing and compresses the piston. This widens the gap and thus changes the flow. The change in the flow on its part changes the fluid pressure that changes the structure etc.

This results in a "grid iteration": we iterate, starting with the grid for the constant gap, see left part of Fig. 3.3.1.2, until we have obtained a final grid, see right part of Fig. 3.3.1.2. For each intermediate grid we must iteratively determine w_{max}, as discussed in Section 3.3.2, thus we have a nested iteration: the innermost iteration is the Newton iteration to solve the global equations for the whole domain, the next outer iteration is the w_{max}-iteration to obtain the zero exit pressure and the outermost iteration is the grid iteration until the structure has reached its final position. This last iteration can also be controlled by w_{max}: If the relative change of w_{max} from one grid to the other is below a given limit, we stop.

The whole algorithm will be described in detail in Section 3.3.4 below for matching grid in cylindrical rotationally symmetric coordinates. The 3-D algorithm has not been implemented because our available supercomputer has not sufficient memory for the 3-D case. Nevertheless we now want to explain how the "breathing" of the fluid domain by the movement of housing and piston would be implemented in 3-D for arbitrary non-matching grid. As it is not possible to show the procedure by figures for a 3-D tetrahedral grid, we explain it at first for 2-D and then expand it to 3-D.

Fig. 3.3.3.1 shows how the nodes at the boundary of the housing and piston move by the fluid pressure and take thus the nodes of the fluid boundary with them. The shifted grid of housing and piston is shown by dashed lines, the shifted grid of the fluid is not indicated to avoid confusion in the figure.

Fig. 3.3.3.2 illustrates the situation if the piston moves. Then the computational domain of the piston has fixed size, but moves. The computational domain of the fluid is the space between piston and housing and changes with the movement of the piston. The left end position 1 is fixed, the right end position oscillates with the piston between the positions 2 and 3. However, in the present investigation we consider only fixed piston.

Now we want to explain the "breathing" of the fluid grid by the displacement of the boundaries to housing and piston as illustrated schematically in Fig. 3.3.3.1. Fig. 3.3.3.3 shows our procedure. The solid lines are the boundaries of piston and housing, they move by the displacement caused by the pressure to the positions indicated by the dashed lines. For the displacement d_A of a node on the piston boundary we take the fixed point A on the opposite housing boundary, for the displacement d_B of a node on the housing boundary, we take the opposite fixed point B on the piston boundary. For nodes in the interior we take an intermediate displacement proportional to its distance from the fixed point as illustrated in Fig. 3.3.3.4. This is made for the displacements of piston and housing and both displacements are added, i.e. the two displacements are superposed. If the piston would move, the grid of the fluid would move as shown in Fig. 3.3.3.2 which results in a similar axial displacement

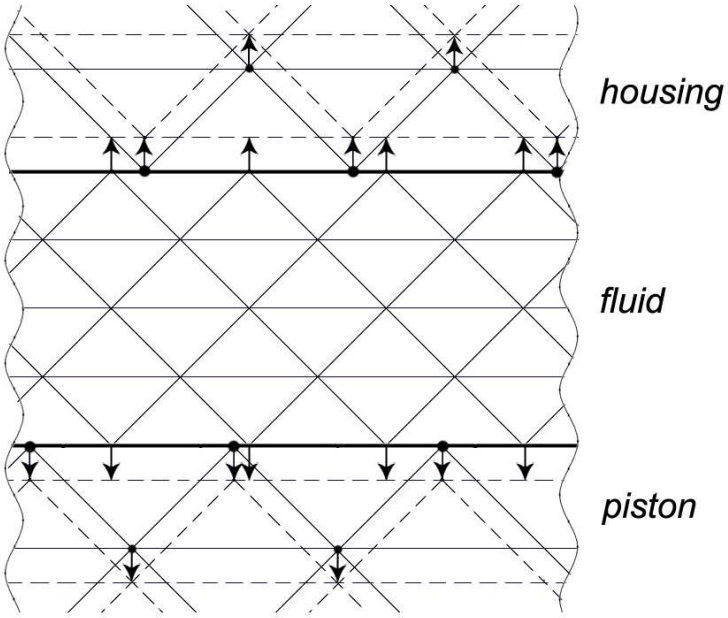


Figure 3.3.3.1: Shifting of the grid by the fluid pressure.

that would also be superposed.

Fig. 3.3.3.5 shows the situation for non-matching grid. For a node C of the fluid grid orthogonal to the axis the fixed point B on the piston and the corresponding point D on the housing are determined as points of intersection between the orthogonal direction (orthogonal to axis of piston) and the

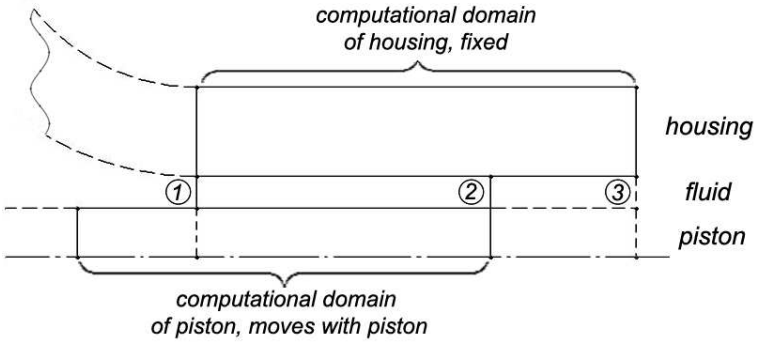


Figure 3.3.3.2: Situation if piston moves. The left end 1 of the fluid domain is fixed, the right end moves with the minimal position 2 and maximal position 3 of the piston.

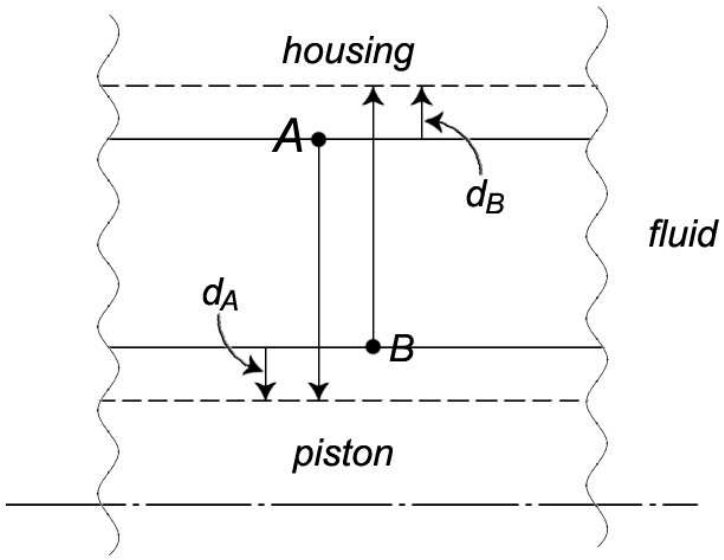


Figure 3.3.3.3: Illustration for the change in the width of the fluid channel.

straight line between the neighboring contour nodes. Then the displacement d_B is known and we can proceed as shown in Figs. 3.3.3.3 and 3.3.3.4. In the same way d_A can be determined for node C and the two local displacements in C are superposed.

As mentioned above the illustration for the 3-D case is much more difficult. Fig. 3.3.3.6 shows the basic principle. It extends Fig. 3.3.3.5 to 3-D. Each node, in this case node C, must store the

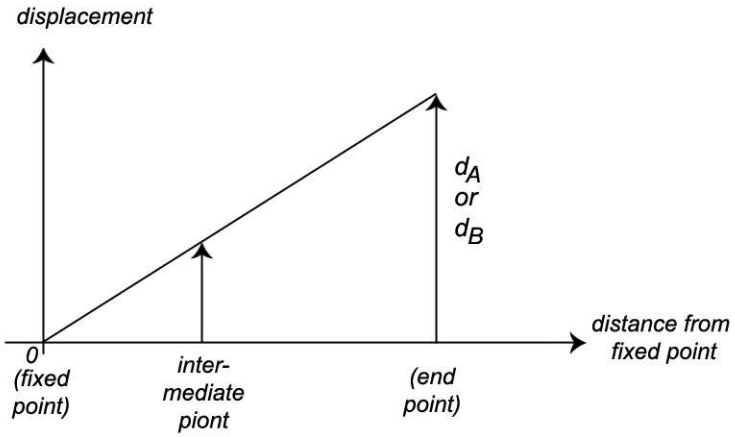


Figure 3.3.3.4: Illustration for the computation of intermediate displacement.

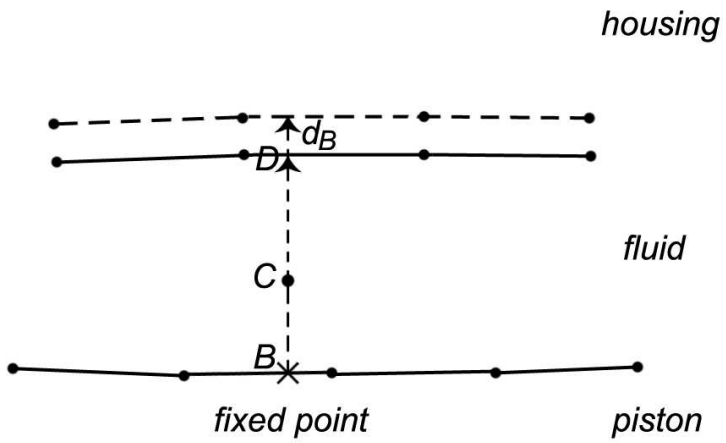


Figure 3.3.3.5: Illustration for non-matching grid.

information for the two relevant boundary triangles on the boundaries of piston and housing. Here the orthogonal line (radial from axis of piston) must be intersected with the triangles. Note that in 3-D we use tetrahedrons, therefore the boundary is composed from triangles which are the "footprints" of

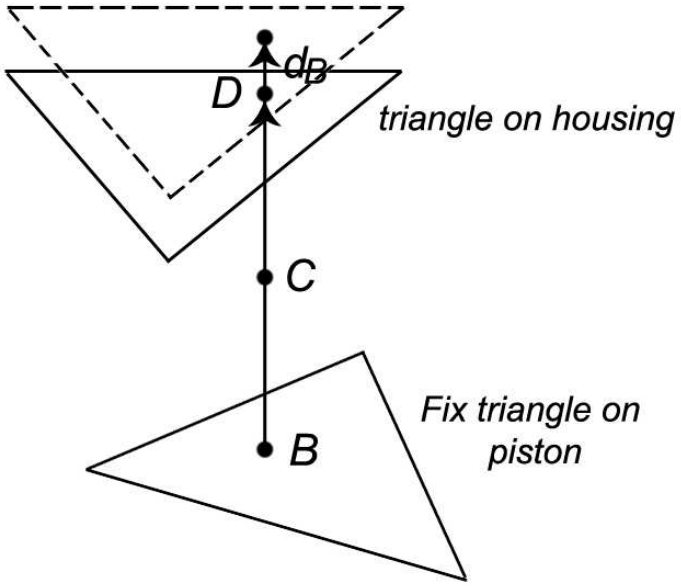


Figure 3.3.3.6: Illustration for 3-D case.

the tetrahedrons. If the points B and/or D drop out of the triangles that have been stored as relevant neighbors, by a search process the suited triangles must be determined. The displacement, e.g. d_B is determined as the mean value of the interpolation polynomials from the 3 corners of the triangle extrapolated to the point D and taking the mean value. As the extrapolation is computed with the same consistency order as the difference formulas, the consistency order is maintained.

These 3-D algorithms have not (yet) been implemented because the memory of the presently for us available supercomputers is not large enough for those 3-D problems. However, these algorithms are closely related to the algorithms for the 3-D SDLs to determine if a node is a free surface node or a SDL node. So the basic algorithmic building blocks are already available.

3.3.4 Solution in axisymmetric cylindrical coordinates

The investigations which grid is needed for 2-D simplifications of the housing in order to obtain an accuracy below 1% error showed that in x, y-coordinates in a plane $z = const$, a grid of 319×81 in circumferential and radial direction is needed only for the housing, see Table 3.3.1.2. The extension to 3-D in the z-direction resulted in a very large sparse matrix. However, the condition of the matrix for this discretization (order $q = 6$) is so bad, that even the most robust iterative CG solver ATPRES in the LINSOL program package converged so slowly or failed completely that it could not be applied for this type of matrices. Therefore only full LU preconditioning can be used for this type of linear equations. However, this results in heavy fill-in between the outermost diagonals (which are reduced by a "parallelized" bandwidth optimizer) and the factors L and U do no longer fit in the memory.

In order to show that the fluid-stucture coupling problem for piston, fluid gap and housing works for FDEM with global solution and global error estimate, we decided to solve the given problem in axisymmetric cylindrical coordinates, i.e. in 2-D, see Fig. 3.3.4.1. Here the final configuration is

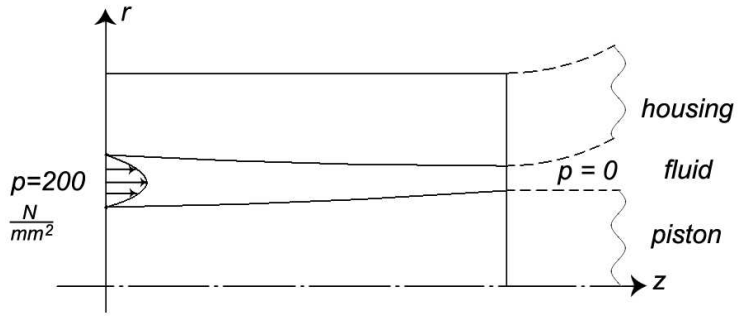


Figure 3.3.4.1: Configuration of problem.

shown with the fluid gaps at the left entry widened by the influence of the pressure of $200\ N/mm^2$ or $2000\ bar$. The starting dimensions are those of Fig. 3.3.1.3.

Now we want to discuss the equations for the structural components housing and piston. As we do not know if we can get directly the solution for the entry pressure $p = 200\ N/mm^2$, we design the algorithm for incremental procedure: we assume we have computed the stresses and displacements for an entry pressure p_1, index "old", and we want to compute the solution for a higher pressure p_2,

thus we use as in Section 3.2.4 incremental equations. The elasticity equations of Section 3.2.4 can directly be used for the housing and piston with the following changes: we now use isotropic steel with a single E and ν, we use directly the $displacement = \Delta t \cdot displacement\ velocity$ as variable (and not displacement velocity) and we use the notation:

$$\begin{array}{rl} \text{coordinates:} & z, r, \\ \text{displacements:} & w\ (z\text{-direction}),\ u\ (r\text{-direction}), \\ \text{stresses:} & \sigma_z, \sigma_r, \sigma_\varphi \text{ and shear stress } \tau_{rz}(=\tau_{zr}). \end{array}$$

Then we get from equations (3.2.4.4)–(3.2.4.9) the following equations with $2G = E/(1+\nu)$, see [12](3.81):

$$\frac{1}{E}[\sigma_z - \sigma_{z,old} - \nu(\sigma_\varphi - \sigma_{\varphi,old}) - \nu(\sigma_r - \sigma_{r,old})] - \frac{\partial w}{\partial z} = 0, \tag{3.3.4.1}$$

$$\frac{1}{E}[-\nu(\sigma_\varphi - \sigma_{\varphi,old}) - \nu(\sigma_z - \sigma_{z,old}) + \sigma_r - \sigma_{r,old}] - \frac{\partial u}{\partial r} = 0, \tag{3.3.4.2}$$

$$\frac{1}{E}[-\nu(\sigma_z - \sigma_{z,old}) + \sigma_\varphi - \sigma_{\varphi,old} - \nu(\sigma_r - \sigma_{r,old})] - \frac{u}{r} = 0, \tag{3.3.4.3}$$

$$\frac{1+\nu}{E}(\tau_{rz} - \tau_{rz,old}) - \frac{1}{2}(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial r}) = 0, \tag{3.3.4.4}$$

$$\frac{\partial \sigma_r}{\partial r} + \frac{\partial \tau_{rz}}{\partial z} + \frac{1}{r}(\sigma_r - \sigma_\varphi) = 0, \tag{3.3.4.5}$$

$$\frac{\partial \tau_{rz}}{\partial r} + \frac{\partial \sigma_z}{\partial z} + \frac{\tau_{rz}}{r} = 0, \tag{3.3.4.6}$$

These are 6 equations for the 6 variables w, u, σ_z, σ_r, σ_φ, τ_{rz}. We use in the interior of the domain the following equations for the variables:

no.	variable	equation
1	w	(3.3.4.1)
2	u	(3.3.4.2)
3	σ_z	(3.3.4.6)
4	σ_r	(3.3.4.5)
5	σ_φ	(3.3.4.3)
6	$\tau_{r,z}$	(3.3.4.4)

(3.3.4.7)

Before we discuss the boundary conditions we note like in Section 3.2.4 the stress vector

$$\sigma = \begin{pmatrix} \sigma_z n^z + \tau_{rz} n^r \\ \tau_{rz} n^z + \sigma_r n^r \end{pmatrix} \begin{array}{l} z\text{-component}, \\ r\text{-component}, \end{array} \tag{3.3.4.8}$$

and normal and tangential stress

$$\sigma_n = \sigma_z (n^z)^2 + 2\tau_{rz} n^z n^r + \sigma_r (n^r)^2, \tag{3.3.4.9}$$

$$\sigma_t = -\sigma_z n^z n^r + \tau_{rz}((n^z)^2 - (n^r)^2) + \sigma_r n^z n^r, \tag{3.3.4.10}$$

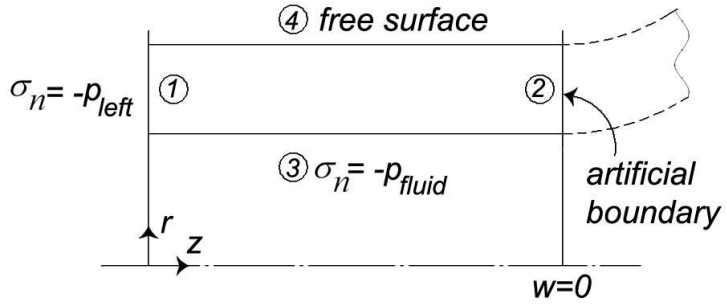


Figure 3.3.4.2: Illustration to boundary conditions for the housing.

with n^z, n^r the components of the normal vector n to the outside of the surface.

Fig. 3.3.4.2 shows the 4 boundaries for the housing. At boundary ① we assume the normal pressure $\sigma_n = -p_{left}$, with p_{left} the given entry pressure. Boundary ② is an "artificial" boundary, it is the limit of the computational domain to the continuation of the housing. The condition $\partial u/\partial z = 0$ gives horizontal tangent of the grid at the boundary ②. Here we assume $w = 0$, i.e. here the domain is "fixed". Boundary ③ is the limit to the fluid domain, here the normal stress is given by the fluid pressure p_{fluid}. We neglect the tangential stress by the friction of the fluid because it is very small relative to the other stresses. Boundary ④ is a free boundary, it is assumed to be force-free. For all variables for which no condition is prescribed we take a suited PDE.

Table 3.3.4.1 shows the assignment of variables and equations for the four boundaries, but excluding the corners. For the four corners we have the following conditions:

	Upper left corner	Upper right corner
w	$(3.3.4.1)$	$w = 0$
u	$(3.3.4.2)$	$\partial u/\partial z = 0$
σ_z	$\sigma_n = -p_{left}$, normal to left	$(3.3.4.1)$
σ_r	$\sigma_n = 0$, normal upwards	$\sigma_n = 0$, normal upwards
σ_φ	$(3.3.4.3)$	$(3.3.4.3)$
$\tau_{r,z}$	$\sigma_t = 0$, normal to left	$\sigma_t = 0$, normal upwards.

(3.3.4.11)

One has to observe for the formulas for σ_n (3.3.4.9) und σ_t (3.3.4.10) which normal is to be used in the corner.

	Lower left corner	Lower right corner
w	$(3.3.4.1)$	$w = 0$
u	$(3.3.4.2)$	$\partial u/\partial z = 0$
σ_z	$\sigma_n = -p_{left}$, normal to left	$(3.3.4.1)$
σ_r	$\sigma_n = -p_{fluid}$, normal downwards	$\sigma_n = -p_{fluid}$, normal downwards
σ_φ	$(3.3.4.3)$	$(3.3.4.3)$
$\tau_{r,z}$	$\sigma_t = 0$, normal to left	$\sigma_t = 0$, normal downwards.

(3.3.4.12)

Table 3.3.4.1: Assignment of variables and equations for the boundaries of Fig. 3.3.4.2, excluding the corners.

no.	var.	Bd. ①	Bd. ②	Bd. ③	Bd. ④
1	w	(3.3.4.1)	$w = 0$	(3.3.4.4)	(3.3.4.4)
2	u	(3.3.4.4)	$\partial u/\partial z = 0$	(3.3.4.2)	(3.3.4.2)
3	σ_z	$\sigma_n = -p_{left}$	(3.3.4.1)	(3.3.4.1)	(3.3.4.1)
4	σ_r	(3.3.4.2)	(3.3.4.2)	$\sigma_n = -p_{fluid}$	$\sigma_{rz}n^z + \sigma_r n^r = 0$
5	σ_φ	(3.3.4.3)	(3.3.4.3)	(3.3.4.3)	(3.3.4.3)
6	τ_{rz}	$\sigma_t = 0$	(3.3.4.4)	$\sigma_t = 0$	$\sigma_z n^z + \tau_{rz} n^r = 0$

Fig. 3.3.4.3 shows similarly the boundaries for the piston. The boundary conditions without the corners are shown in Table 3.3.4.2.

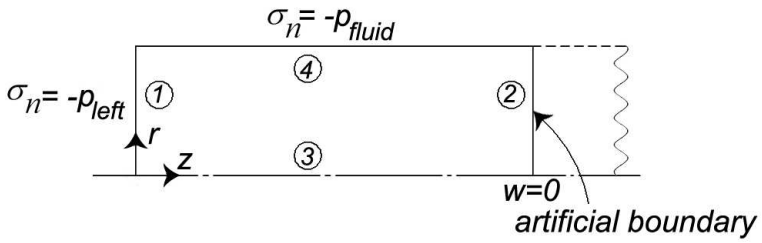


Figure 3.3.4.3: Illustration to boundary conditions for the piston.

At the axis we have at boundary ③ the value $r = 0$ which causes difficulties for the term u/r. We expand u in a power series for r:

$$\frac{u}{r} = \left.\frac{u(r) + \frac{\partial u}{\partial r}r + \frac{1}{2}\frac{\partial^2 u}{\partial r^2} + \ldots}{r}\right|_{r=0} = \frac{\partial u}{\partial r} + \frac{1}{2}\frac{\partial^2 u}{\partial r^2}r + \ldots$$

because for $r = 0 \quad u(r = 0) = 0$. Thus we use in the equation (3.3.4.3)* in Table 3.3.4.2

$$\left.\frac{u}{r}\right|_{r=0} = \frac{\partial u}{\partial r}. \tag{3.3.4.13}$$

In (3.3.4.6) we have τ_{rz}/r. This term must be regular on the axis, which only holds for $\tau_{rz} = 0$ on the axis.

Table 3.3.4.2: Assignment of variables and equations for the boundaries of Fig. 3.3.4.3, excluding the corners.

no.	var.	Bd. ①	Bd. ②	Bd. ③	Bd. ④
1	w	(3.3.4.1)	$w = 0$	(3.3.4.4)	(3.3.4.4)
2	u	(3.3.4.4)	$\partial u/\partial z = 0$	$u = 0$	(3.3.4.2)
3	σ_z	$\sigma_n = -p_{left}$	(3.3.4.1)	(3.3.4.1)	(3.3.4.1)
4	σ_r	(3.3.4.2)	(3.3.4.2)	(3.3.4.2)	$\sigma_n = -p_{fluid}$
5	σ_φ	(3.3.4.3)	(3.3.4.3)	$(3.3.4.3)^*$	(3.3.4.3)
6	τ_{rz}	$\sigma_t = 0$	(3.3.4.4)	$\tau_{rz} = 0^*$	$\sigma_t = 0$

*) see text

The boundary condition at the four corners for the piston are:

	Upper left corner	Upper right corner
w	(3.3.4.1)	$w = 0$
u	(3.3.4.2)	$\partial u/\partial z = 0$
σ_z	$\sigma_n = -p_{left}$, normal to left	(3.3.4.1)
σ_r	$\sigma_n = -p_{fluid}$, normal upwards	$\sigma_n = -p_{fluid}$, normal upwards
σ_φ	(3.3.4.3)	(3.3.4.3)
$\tau_{r,z}$	$\sigma_t = 0$, normal to left	$\sigma_t = 0$, normal upwards

(3.3.4.14)

	Lower left corner	Lower right corner
w	(3.3.4.1)	$w = 0$
u	$u = 0$	$u = 0$
σ_z	$\sigma_n = -p_{left}$, normal to left	(3.3.4.1)
σ_r	(3.3.4.2)	(3.3.4.2)
σ_φ	$(3.3.4.3)^*$	$(3.3.4.3)^*$
$\tau_{r,z}$	$\sigma_t = 0$, normal to left	$\tau_{rz} = 0^*$.

(3.3.4.15)

*) see text

With these boundary conditions the definition of the problem for the structural parts housing and piston is terminated. As we proceed incrementally, we increase the entry pressure at the left end by an initial increment Δp_1 and then in steps by an increment Δp_2 until finally $p = 200\ N/mm^2$ or a higher value has been reached. The displacements (3.3.4.1)–(3.3.4.4) are in reality Δw und Δu that must be added to the "old" coordinates. Therefore we must shift the nodes of the grid in the following way after each pressure step:

$$\begin{aligned} z &= z_{old} + w, \\ r &= r_{old} + u. \end{aligned} \tag{3.3.4.16}$$

Now we want to discuss the PDEs and boundary conditions for the fluid region, i.e. for the lubrication gap. The Navier-Stokes equations in axisymmetric coordinates can be found in [14]. They

are the transformed equations (3.3.2.1)–(3.3.2.4), reduced to 2-D. If we denote by w, u the velocity components in z-direction (axial) and r-direction (radial), see Fig. 3.3.4.1, the steady equations are

$$u\frac{\partial u}{\partial r} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial r} + \frac{\eta}{\rho}(\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} - \frac{u}{r^2} + \frac{\partial^2 u}{\partial z^2}), \tag{3.3.4.17}$$

$$u\frac{\partial w}{\partial r} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \frac{\eta}{\rho}(\frac{\partial^2 w}{\partial r^2} + \frac{1}{r}\frac{\partial w}{\partial r} + \frac{\partial^2 w}{\partial z^2}), \tag{3.3.4.18}$$

$$\frac{\partial u}{\partial r} + \frac{u}{r} + \frac{\partial w}{\partial z} = 0. \tag{3.3.4.19}$$

The values of ρ and η are given in (3.3.2.7).

Fig. 3.3.4.4 shows the computational domain for the fluid. In the interior we arrange the variables

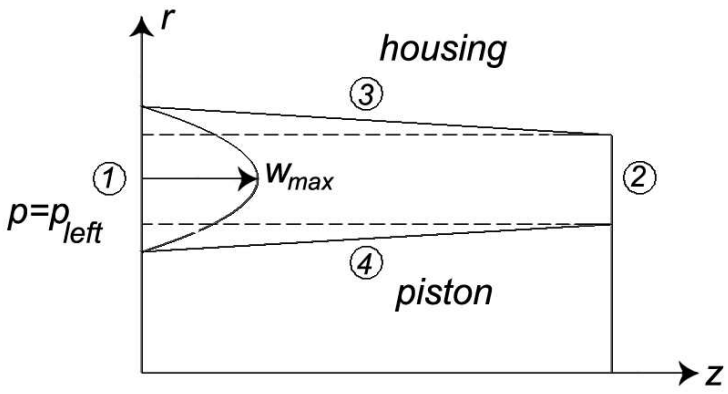


Figure 3.3.4.4: Illustration to boundary conditions for the fluid.

and equations as follows:

no.	variable	equation
1	w	$(3.3.4.19)$
2	u	$(3.3.4.17)$
3	p	$(3.3.4.18)$

(3.3.4.20)

As explained in Section 3.3.2 the boundaries ① and ② are artificial boundaries. At boundary ① we prescribe

$$\begin{aligned} &w \text{ as parabola with max.value } w_{max} \text{ in the center of the gap,} \\ &u = 0, \\ &p = p_{left}. \end{aligned} \tag{3.3.4.21}$$

Here p_{left} denotes the actual prescribed pressure at the entry side, its final value in an incremental pressure step procedure is $200\ N/mm^2$. At the exit, boundary ②, we prescribe as explained in the

context of equation (3.3.2.6)

$$\begin{aligned} &\text{for } w: \quad \tfrac{\partial w}{\partial z} = 0, \\ &\text{for } u: \quad \tfrac{\partial u}{\partial z} = 0, \\ &\text{for } p: \quad (3.3.4.18). \end{aligned} \tag{3.3.4.22}$$

At the boundaries ③ and ④ we have for the velocity components the no-slip condition. As there is no prescription for p, we take an appropriate PDE:

$$\begin{aligned} w: &\quad w = 0, \\ u: &\quad u = 0, \\ p: &\quad (3.3.4.18). \end{aligned} \tag{3.3.4.23}$$

As explained in Section 3.3.2 the desired condition $p_{exit} = 0$ can be fulfilled only by an appropriate choice of w_{max} at the entry. This leads to the "w_{max}-iteration", that has been explained in Section 3.3.2. We define as "exit pressure" the value of the pressure at the center of the gap of the boundary ② in Fig. 3.3.4.4.

The combination and coupling of the 3 domains: piston, housing and fluid has been discussed for the 3-D case in Section 3.3.3. In the 2-D case with axisymmetric cylindrical coordinates we use for simplicity a matching grid as shown in Fig. 3.3.4.5 for the initial position for entry and exit pressure zero. Such a grid can be generated easily "by hand" and allows flexibility for accuracy tests with

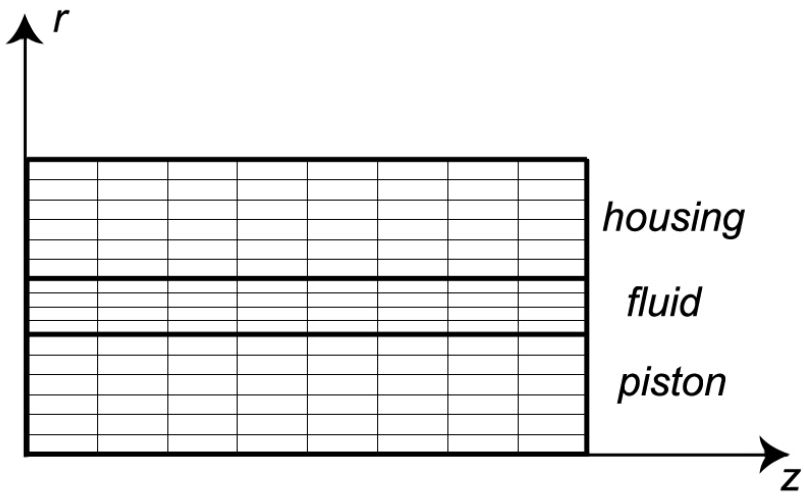


Figure 3.3.4.5: Type of grid used for the solution. In reality the fluid gap is very thin, see Fig. 3.3.1.3. The diagonals that make from quadrilaterals the triangles are not shown.

different grids. As this is a matching grid, we have between fluid and housing or piston a "normal" dividing line (DL) and not a SDL which would be needed for non-matching grid.

As explained in Section 3.3.3 we have a direct coupling of housing and piston to the fluid only by the normal stress which is equal to the fluid pressure p_{fluid}, see boundary ③ in Table 3.3.4.1

and boundary ④ in Table 3.3.4.2. There is no direct coupling of the fluid to the housing and piston, however, the grid of the structure and thus the width of the fluid gap changes with (3.3.4.16). This changes on its turn the fluid flow.

The displacement of the nodes of the fluid grid is determined by the displacement of the nodes of the housing and piston at the boundary to the fluid, see Figs. 3.3.3.1–3.3.3.4 and the corresponding context. As we now have matching 2-D grid, the boundary nodes of housing and piston are explicitly known for each node of the fluid domain. Thus it is easy to compute Δz and Δr for each node. This results is the new fluid grid. Fig. 3.3.4.6 shows the nested structure of the solution process.

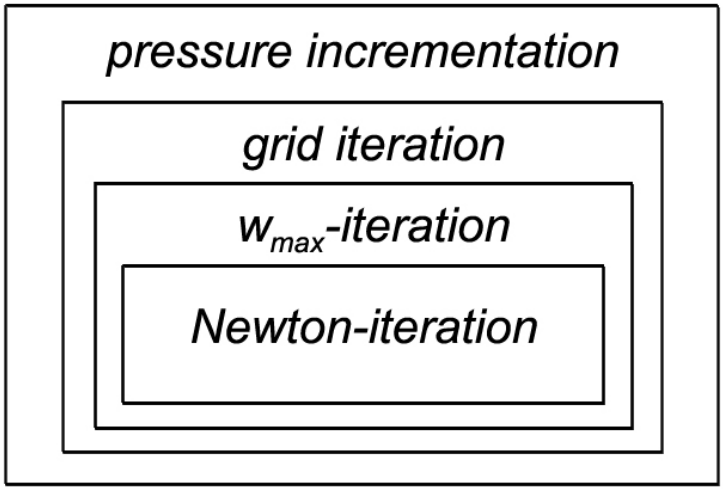


Figure 3.3.4.6: Illustration to nested character of the solution process.

For the computation of the global solution for housing, piston and fluid we must execute a Newton iteration. Here w_{max} and grid are fixed. For this solution the "exit pressure" is checked and w_{max} is corrected as explained at the end of Section 3.3.2 by a type of Newton iteration. This defines the w_{max}-iteration. Now w_{max} is fixed for this grid and from the solution we know the displacements for the boundaries ③ and ④ of Fig. 3.3.4.4 and we can compute the new fluid grid according to Figs. 3.3.3.3 and 3.3.3.4.

Now we start for the new grid (housing, piston, fluid) the same process with starting value w_{max} of the previous grid and we must again adapt w_{max} for exit pressure zero in a w_{max}-iteration. Finally we have a new solution for this grid that determines a new grid for housing, piston and fluid. This can be repeated and this defines the grid iteration in Fig. 3.3.4.6. If the grid is changed, w_{max} will change. Therefore we use w_{max} to control the grid iteration and we stop the grid iteration if

$$\frac{w_{max} - w_{max,old}}{w_{max}} \leq \varepsilon_{grid}. \tag{3.3.4.24}$$

Now we have the solution and grid for the actual pressure p_{left} of Figs. 3.3.4.2–3.3.4.4. If we increase the pressure p_{left} in an incremental procedure by Δp_1 and Δp_2 we start the whole procedure for $p_{left} = p_{left,old} + \Delta p_2$ and repeat until we have reached the prescribed entry pressure $p_{left} = 200\ N/mm^2\ (2000\,bar)$ or a requested higher value.

Here we have described the "fully coupled" solution. This means that in each w_{max}-iteration step the equations for housing, piston and fluid are solved. However, we do not make use of the displacements of housing and piston during the w_{max}-iteration because the grid is fixed. Only after the stopping of this iteration the displacements are applied to the old grid, giving the new grid. To save computation time we therefore defined the "pseudo-uncoupled" solution: We solve separately the PDEs for the 3 domains housing, piston and fluid. During the w_{max}-iteration only the fluid equations are solved. After the stopping of the w_{max}-iteration the displacements are computed by the solution for housing and piston, using the actual fluid pressure as BC. Then we get a new grid and start a new w_{max}-iteration, if the stopping criterion (3.3.4.24) is not yet fulfilled. This pseudo-uncoupled procedure gives the same result as the fully coupled solution, but it avoids the unnecessary repeated solution of the structural equations and thus saves much computation time.

Here we want to discuss briefly how the algorithm would be executed if the piston moved/oscillated. Then we have a sliding dividing line (SDL) instead of a simple DL because the grid of the fluid domain changes, see Fig. 3.3.3.2. The changes of the grid and of the fluid pressure depend on the time increment Δt. Basically we have the same nested algorithm like in Fig. 3.3.4.6. However, for small Δt one could fix in the optimal case to one Newton iteration, one w_{max}-iteration and one grid iteration, a pressure iteration in this case is not needed. The errors that are generated by this cutting of the iterations are $O(\Delta t)$ and one can see by numerical experiments with different values of Δt how the solution changes with Δt. In the same way one could see how the solution changes if 2 or 3 Newton steps are prescribed. Eventually then a larger Δt is possible so that the overall computation time is reduced. The same check can be made for more than one w_{max}-iteration or grid iteration. This is a complicated optimization problem for the 4 parameters Δt, $n{-}it_{Newton}$, $n{-}it_{w_{max}}$, $n{-}it_{grid}$, where $n{-}it$ denotes the number of iterations.

3.3.5 Results for the axisymmetrical cylindrical coordinates

We have solved the fluid-structure interaction problem of housing, piston and fluid with the pseudo-uncoupled method that gives the same result as the fully coupled method. However, we could get a solution only up to 1500 or $1600\ bar$. There are extreme differences in scale between housing and piston that are in the range of cm at the one side, and between the lubrication (fluid) gap that is in the range of micrometer at the other side. We made numerical experiments with different length scales between m and mm and looked at the errors for $1500\ bar$. We could see that for the scale in cm we got the best results. Here the error estimate told us what is the best length scale for this problem that includes also extreme coefficients ρ and η that vary in a wide range between m and mm. So we solved the PDEs in the length scale cm. Therefore the displacements are in cm, the velocities in cm/s, the stresses in N/cm^2 and the volume flow in cm^3/s.

The volume flow is determined from the prescribed entry parabola with w_{max} for the velocity w, see Fig. 3.3.4.4:

$$vol = 2\pi \int_{r_i}^{r_a} w(r) r\, dr = 2\pi w_{max} \left[\frac{a_0}{2} \left(r_a^2 - r_i^2\right) + \frac{a_1}{3} \left(r_a^3 - r_i^3\right) + \frac{a_2}{4} \left(r_a^4 - r_i^4\right) \right], \quad (3.3.5.1)$$

with a_0, a_1, a_2 are the coefficients of the parabola $a_0 + a_1 r + a_2 r^2$ of the prescribed entry profile and with r_i, r_a the inner and outer radius of the fluid gap. These values are known during the computation.

But why could we not get results for entry pressure $p > 1600\, bar$? We were really desperate. After long investigations we finally could find the cause: during the grid iteration at the inner and outer boundaries of the housing and at the outer boundary of the piston tiny waves in the size of a fraction of a micrometer built up. This made the fluid channel "rough", see Fig. 3.3.4.4. As a consequence the Newton iteration for the fluid diverged. In the structural equation there are no damping terms, thus the roughness could build up. This roughness is in the range of the discretization errors of the structural equations. We at first tried to solve the problem by a finer grid which helped a little and brought us up to $1800\, bar$, but then the same effect occured. So what to do now?

The idea that solved the problem was to smooth the inner wall of the housing and outer wall of the piston. The smoothing is made in the following way, see Fig. 3.3.5.1: We determine the parabola for node B with the two neighbor nodes A and C.

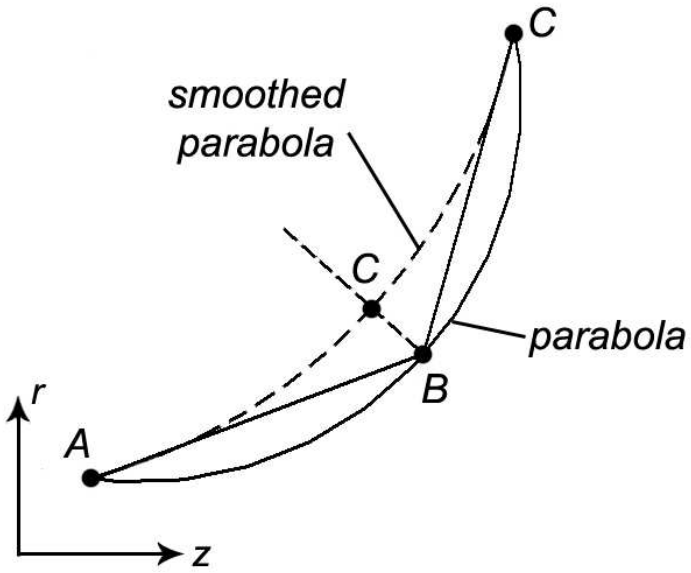


Figure 3.3.5.1: Illustration for the smoothing of the surface.

This has the form

$$r_{par}(z) = a_0 + a_1 z + a_2 z^2 \,, \tag{3.3.5.2}$$

and its second derivative is

$$r''_{par} = 2a_2 \,. \tag{3.3.5.3}$$

Now we determine a new parabola with

$$r''_{par,\,new} = \beta r''_{par} = \beta \cdot 2a_2 \,, \tag{3.3.5.4}$$

i.e. a parabola that is for $0 < \beta < 1$ more "flat", see dashed parabola in Fig. 3.3.5.1. Now we determine the new "smoothed" node C as the intersection of the normal to the old parabola with the new smoothed parabola. End points of the boundaries remain fixed. If we do this smoothing for all nodes of a boundary we have executed one smoothing sweep. Now we can execute further sweeps. We denote by

$$n_{smooth} \quad \text{the number of smoothing sweeps.} \tag{3.3.5.5}$$

With the smoothing parameters $\beta = 0.5$ and $n_{smooth} = 2$ we could solve the coupled problem for arbitrary entry pressure p without problems. We have computed with consistency order $q = 2$ (higher order caused problems for the used grid). The grid for the 3 domains was:

housing: 401 (z-direction) $\times 80$ r-direction),
piston: 401×40,
fluid: 401×81.

We have used our distributed memory supercomputer HP XC6000 with Itanium 2 processors, 1.5 GHz, 2-processor nodes with Quadrics interconnect. We computed in parallel on 16 processors. If we solved the pseudo-uncoupled problem directly for the entry pressure $2000\ bar$, without writing the information for the result pictures to disk, we needed 3354 sec. In this timing is the part for the linear equation solver LINSOL 3296 sec, i.e. most of the time is spent in LINSOL. We used full LU preconditioning.

For the result table and result plots we computed with $\Delta p_1 = 1500\ bar$ and $\Delta p_2 = 500\ bar$ for the values 1500, 2000, 2500 and $3000\ bar$ for the geometry of Fig. 3.3.1.3. Additionally (for curiosity) we solved the problem for $2000\ bar$ for the housing with twice the wall thickness, i.e. $12\ mm$ instead of $6\ mm$, or $32\ mm$ outer diameter instead of $20\ mm$. Table 3.3.5.1 shows the maximum value, the maximum relative error and the mean relative error for all solution components in the 3 domains housing, piston and fluid, and the volume flow.

Table 3.3.5.1: Maximum value, max. relative error and mean relative error for the solution components in the three domains, and the volume flow through the gap, for different entry pressures.

```
p = 1500 bar

Housing
                    max. solution max. error mean error
w              cm    0.3101E-02    0.51E-04   0.80E-05
u              cm    0.5685E-03    0.35E-03   0.84E-04
sigma-z     N/cm^2 0.1677E+05    0.21E-02   0.79E-05
sigma-r     N/cm^2 0.1500E+05    0.31E-02   0.20E-04
sigma-phi   N/cm^2 0.2079E+05    0.69E-03   0.71E-04
tau-rz      N/cm^2 0.6591E+03    0.29E-01   0.47E-04
```

```
Piston
                  max. solution max. error mean error
w             cm   0.1611E-02    0.27E-04   0.20E-05
u             cm   0.1143E-03    0.63E-03   0.27E-05
sigma-z     N/cm^2 0.1633E+05    0.27E-02   0.50E-05
sigma-r     N/cm^2 0.1501E+05    0.32E-02   0.22E-05
sigma-phi   N/cm^2 0.1501E+05    0.12E-02   0.23E-05
tau-rz      N/cm^2 0.5803E+03    0.33E-01   0.40E-04

Fluid
                  max. solution max. error mean error Volume [cm^3/s]
w            cm/s  0.1722E+04    0.69E+00   0.13E-01            1.05
u            cm/s  0.1391E+00    0.16E+03   0.19E+01
p           N/cm^2 0.1500E+05    0.62E-01   0.43E-02


p = 2000 bar

Housing
                  max. solution max. error mean error
w             cm   0.4143E-02    0.10E-03   0.11E-04
u             cm   0.7584E-03    0.32E-03   0.82E-04
sigma-z     N/cm^2 0.2244E+05    0.30E-02   0.12E-04
sigma-r     N/cm^2 0.2000E+05    0.28E-02   0.22E-04
sigma-phi   N/cm^2 0.2772E+05    0.65E-03   0.67E-04
tau-rz      N/cm^2 0.9837E+03    0.24E-01   0.87E-04

Piston
                  max. solution max. error mean error
w             cm   0.2096E-02    0.19E-03   0.18E-04
u             cm   0.1524E-03    0.97E-03   0.86E-05
sigma-z     N/cm^2 0.2196E+05    0.37E-02   0.19E-04
sigma-r     N/cm^2 0.2001E+05    0.33E-02   0.30E-05
sigma-phi   N/cm^2 0.2001E+05    0.16E-02   0.32E-05
tau-rz      N/cm^2 0.9253E+03    0.28E-01   0.43E-04

Fluid
                  max. solution max. error mean error Volume [cm^3/s]
w            cm/s  0.3339E+04    0.87E+00   0.14E-01            2.40
u            cm/s  0.3810E+00    0.96E+02   0.10E+01
p           N/cm^2 0.2000E+05    0.94E-01   0.60E-02


p = 2500 bar

Housing
                  max. solution max. error mean error
w             cm   0.1047E-02    0.12E-02    0.69E-04
u             cm   0.1900E-03    0.39E-02    0.39E-03
sigma-z     N/cm^2 0.2817E+05    0.10E-01    0.39E-04
sigma-r     N/cm^2 0.2500E+05    0.24E-02    0.28E-04
sigma-phi   N/cm^2 0.3467E+05    0.18E-02    0.63E-04
tau-rz      N/cm^2 0.1393E+04    0.16E-01    0.29E-03
```

```
Piston
                  max. solution max. error mean error
w             cm   0.4589E-03    0.43E-02    0.40E-03
u             cm   0.3818E-04    0.53E-02    0.16E-03
sigma-z     N/cm^2 0.2771E+05    0.98E-02    0.74E-04
sigma-r     N/cm^2 0.2502E+05    0.14E-02    0.58E-05
sigma-phi   N/cm^2 0.2502E+05    0.36E-02    0.71E-05
tau-rz      N/cm^2 0.1382E+04    0.15E-01    0.49E-04

Fluid
                  max. solution max. error mean error Volume [cm^3/s]
w            cm/s  0.5255E+04    0.13E+01    0.17E-01           4.54
u            cm/s  0.8386E+00    0.12E+03    0.91E+00
p           N/cm^2 0.2500E+05    0.23E+00    0.17E-01


p = 3000 bar

Housing
                  max. solution max. error mean error
w             cm   0.1052E-02    0.25E-02    0.13E-03
u             cm   0.1905E-03    0.11E-01    0.67E-03
sigma-z     N/cm^2 0.3391E+05    0.17E-01    0.80E-04
sigma-r     N/cm^2 0.3000E+05    0.40E-02    0.37E-04
sigma-phi   N/cm^2 0.4164E+05    0.37E-02    0.93E-04
tau-rz      N/cm^2 0.1868E+04    0.27E-01    0.52E-03

Piston
                  max. solution max. error mean error
w             cm   0.4323E-03    0.11E-01    0.11E-02
u             cm   0.3824E-04    0.12E-01    0.39E-03
sigma-z     N/cm^2 0.3355E+05    0.15E-01    0.15E-03
sigma-r     N/cm^2 0.3004E+05    0.20E-02    0.13E-04
sigma-phi   N/cm^2 0.3003E+05    0.59E-02    0.15E-04
tau-rz      N/cm^2 0.1929E+04    0.23E-01    0.84E-04

Fluid
                  max. solution max. error mean error Volume [cm^3/s]
w            cm/s  0.7042E+04    0.54E+01    0.45E-01           7.39
u            cm/s  0.1547E+01    0.60E+03    0.23E+01
p           N/cm^2 0.3000E+05    0.61E+00    0.40E-01


Housing with diameter 3,2 cm, p = 2000 bar

Housing
                  max. solution max. error mean error
w             cm   0.3931E-02    0.10E-03    0.10E-04
u             cm   0.6655E-03    0.11E-02    0.22E-03
sigma-z     N/cm^2 0.2099E+05    0.90E-02    0.24E-04
sigma-r     N/cm^2 0.2000E+05    0.37E-02    0.76E-04
sigma-phi   N/cm^2 0.2286E+05    0.27E-02    0.15E-03
tau-rz      N/cm^2 0.8061E+03    0.45E-01    0.15E-03
```

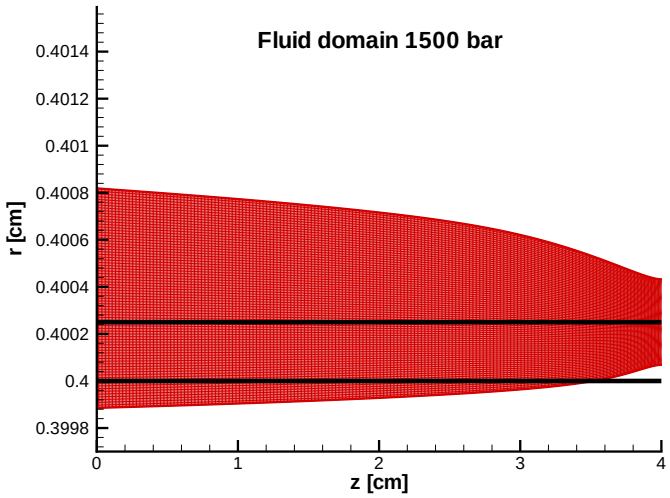


Figure 3.3.5.2: Fluid domain with the computational grid for $1500\ bar$, bold lines: original channel.

```
Piston
                    max. solution max. error mean error
w              cm    0.2122E-02    0.89E-04    0.90E-05
u              cm    0.1524E-03    0.83E-03    0.45E-05
sigma-z     N/cm^2 0.2185E+05    0.31E-02    0.97E-05
sigma-r     N/cm^2 0.2001E+05    0.32E-02    0.23E-05
sigma-phi   N/cm^2 0.2001E+05    0.14E-02    0.24E-05
tau-rz      N/cm^2 0.8511E+03    0.31E-01    0.39E-04

Fluid
                    max. solution max. error mean error Volume [cm^3/s]
w             cm/s  0.2884E+04    0.43E+00    0.16E-01           1.95
u             cm/s  0.2708E+00    0.68E+02    0.13E+01
p           N/cm^2 0.2000E+05    0.10E+00    0.78E-02
```

The figures for the results, Fig. 3.3.5.2–3.3.5.19, are black-and-white plots in the printed version and they are colored plots in the online version of the paper. For the entry pressures 1500, 2000, 2500 and $3000\ bar$ are shown the fluid channel and its grid (omitting the diagonals that make from the squares the triangles) and in bold lines the original size of the channel (for entry pressure zero bar) and contour plots of the velocity w in z-direction. Then follow for housing and piston for $2000\ bar$ contour plots for the stresses σ_z, σ_r, σ_φ and τ_{rz}. Finally follow for $2000\ bar$ the fluid channel and contour plot of velocity w for housing with $32\ mm$ diameter, i.e. double thickness of the wall. The scale for z and r is in cm.

Here we want to make some comments to the results. Fig. 3.3.5.2 shows drastically how the fluid gap widens under the influence of the entry pressure of $1500\ bar$. It is interesting to see the form of the gap. It is clear that the housing widens at the entry, this is the upper side of the fluid channel.

However, at the exit we have the pressure ≈ 0 and nevertheless the housing widens, here not by the influence of the local fluid pressure but by the stresses that exert their influence from the high pressure side: If the housing is widened at the left side, this causes also a widening at the right end, independent of the fluid pressure at the exit. Clearly the piston is compressed and therefore shrinks at the high pressure end. However it widens at the low pressure end. Why? This is a volume effect, Material is pressed to the right, but there we have displacement $w = 0$ and pressure zero and thus the material escapes into a larger diameter. Here it should be recalled that the right end of housing and piston are "artificial boundaries". Here ends the computational domain, but the housing and piston do not end here but continue to the right in an unknown form. If we could include the right continuation in the computation, the form at the right end of the gap may be (slightly) different.

If we then look at the fluid gap for $2000\ bar$, Fig. 3.3.5.4, for $2500\ bar$, Fig. 3.3.5.6, and $3000\ bar$, Fig. 3.3.5.8, we see how the gap with initially 2.5 micrometers (bold lines) is blown up. The manufacturing tolerances may be in the range of a fraction of a micrometer, but then the pressure makes these tolerances obsolete. This demonstrates drastically the problems which engineers must solve under such high pressure conditions.

Just for curiosity we also solved for $2000\ bar$ the equations for the outer housing diameter of $32\ mm$ which means doubling the thickness of the wall for the housing from $6\ mm$ to $12\ mm$. We expected a significant reduction of the widening of the fluid gap. However, if we compare Fig. 3.3.5.4 ($20\ mm$ diameter) and Fig. 3.3.5.18 ($32\ mm$ diameter) we recognize that there is not much difference in the form of the gap. From Table 3.3.5.1 we have the volume flow for $2000\ bar$ for $20\ mm$ diameter $2.40\ cm^3/s$ and for $32\ mm$ diameter $1.95\ cm^3/s$ which is not much difference for doubling the wall thickness. This shows that the severity of the problem cannot be reduced just by simply doubling the wall thickness.

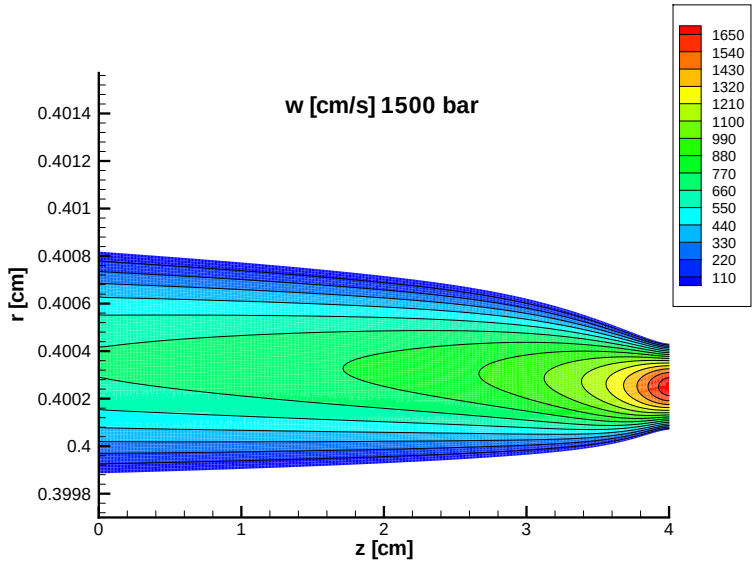


Figure 3.3.5.3: Contour plot of the velocity w in z-direction for $1500\ bar$.

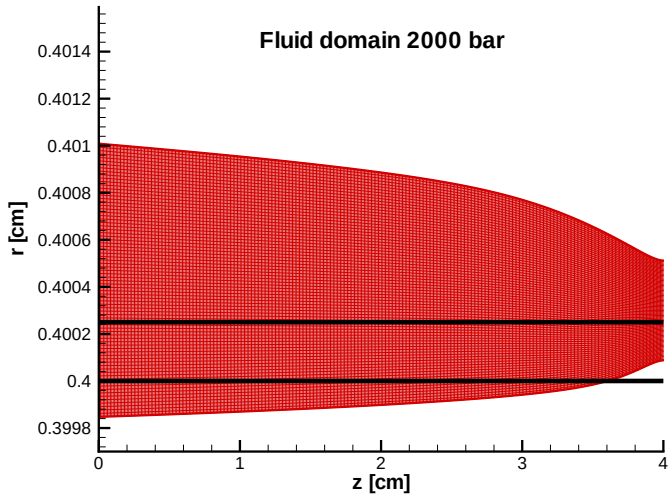


Figure 3.3.5.4: Fluid domain with the computational grid for $2000\ bar$, bold lines: original channel.

Here are some concluding remarks to this example of a fluid/structure interaction. Seemingly the problem for high entry pressure above $1500\ bar$ were the fluid equations (Navier-Stokes equations), because the Newton iteration diverged for the fluid domain. However, the reason were the equations

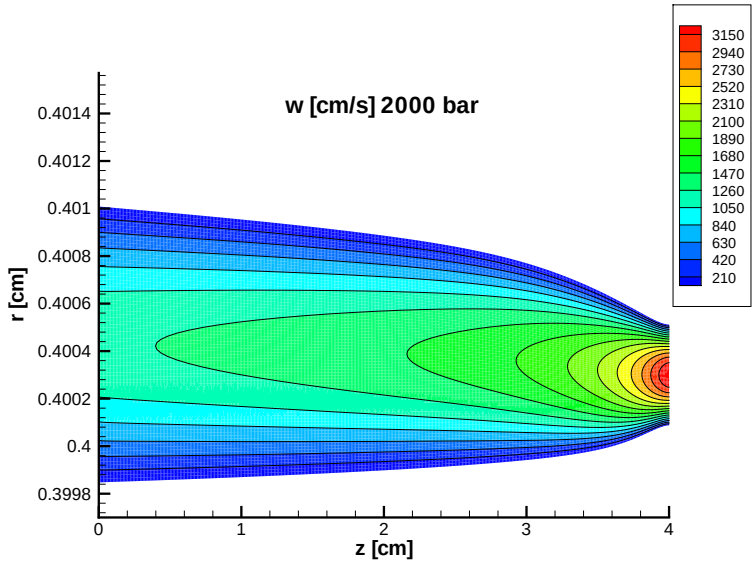


Figure 3.3.5.5: Contour plot of the velocity w in z-direction for $2000\ bar$.

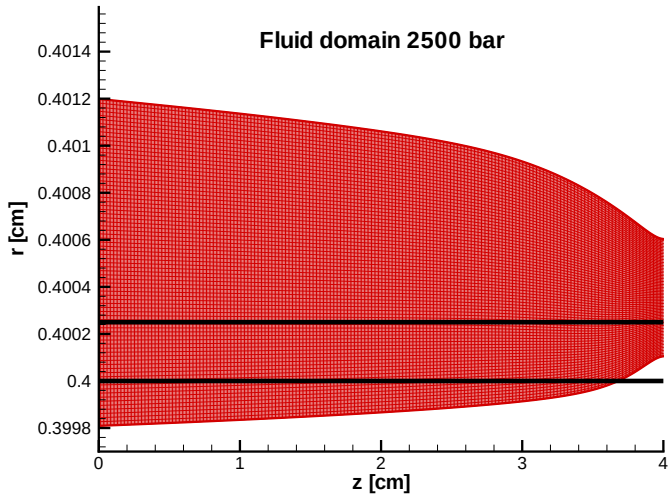


Figure 3.3.5.6: Fluid domain with the computational grid for $2500\ bar$, bold lines: original channel.

for the structural components housing and piston. The structural equations, i.e. the elasticity equations, do not have damping terms. This property leads in the frame of our solution algorithm, i.e. by the grid iteration, to "tiny" oscillations of the displacements w, u at the surface. This does no harm

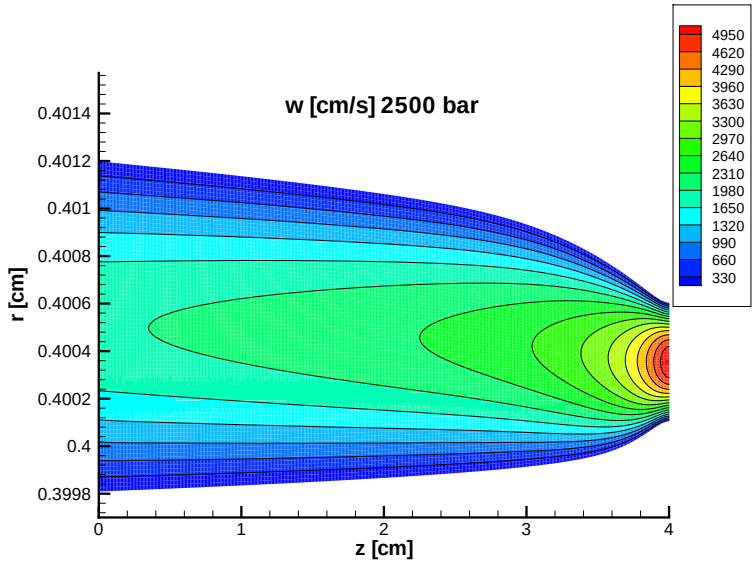


Figure 3.3.5.7: Contour plot of the velocity w in z-direction for $2500\ bar$.

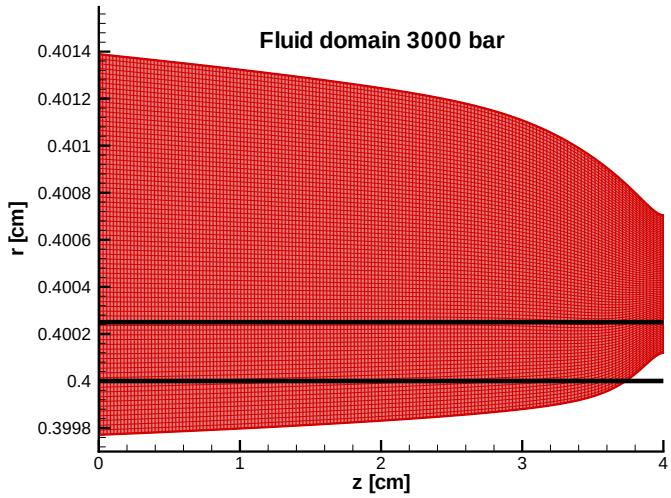


Figure 3.3.5.8: Fluid domain with the computational grid for $3000\ bar$, bold lines: original channel.

to the solution of the structural components. However, the scale of the fluid domain is by a factor $1/1000$ smaller, thus these "tiny" oscillations of the structural components (outer surface of piston, inner surface of housing) are "large" oscillations of the fluid domain: the wall becomes "rough" in a

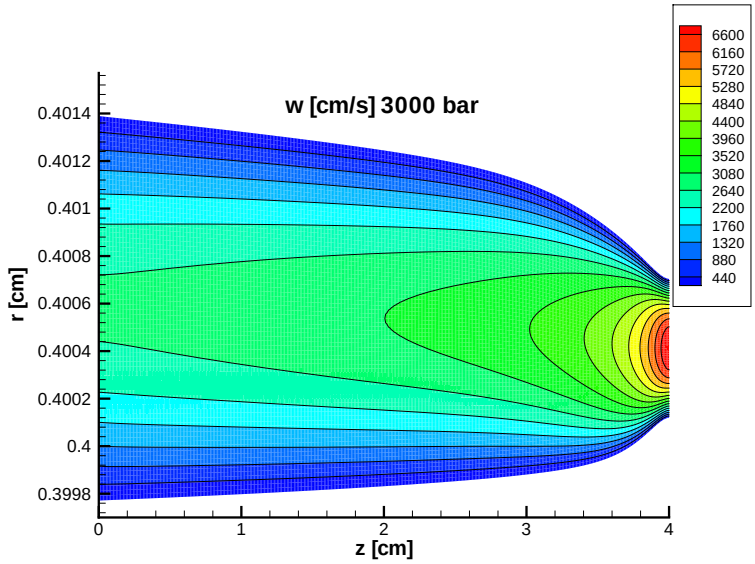


Figure 3.3.5.9: Contour plot of the velocity w in z-direction for $3000\ bar$.

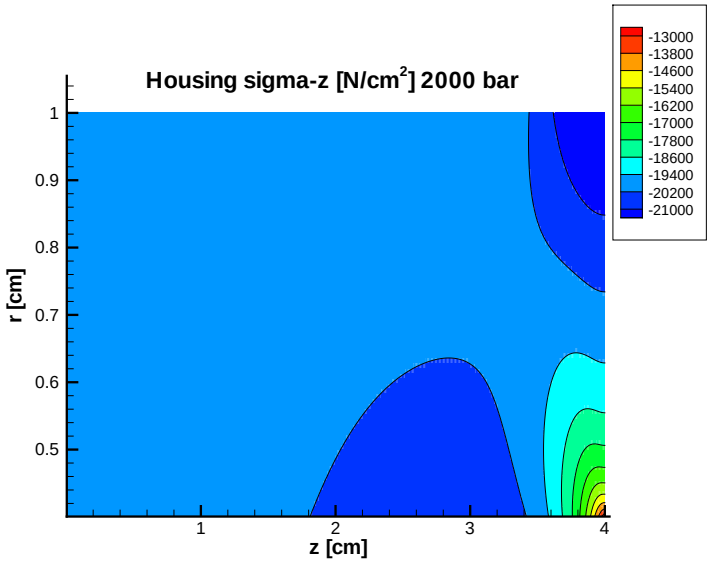


Figure 3.3.5.10: Contour plot for the stress component σ_z for the housing for 2000 bar.

very bad manner. In the appropriate scale the fluid surface which is in the discretized form a polygon, looks like a sawtooth curve and causes the Newton iteration for the fluid to diverge. We needed a rather long and frustrating time to recognize these interrelations. We investigated many different

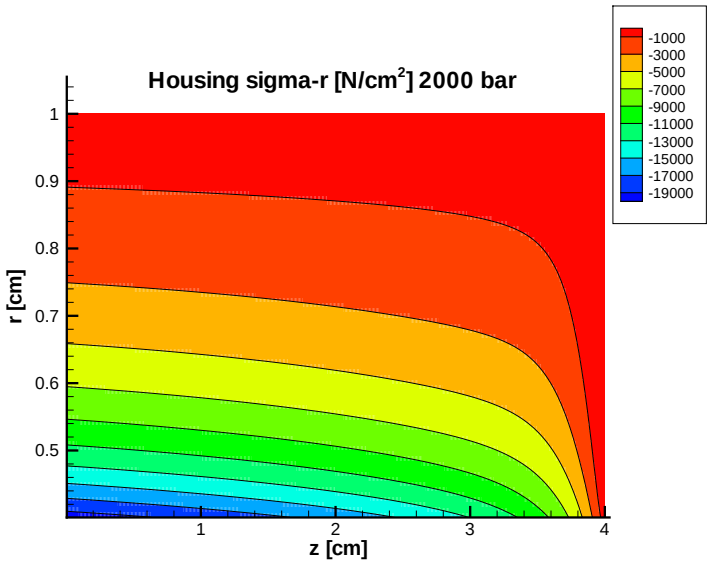


Figure 3.3.5.11: Contour plot for the stress component σ_r for the housing for 2000 bar.

methods to cure this situation: finer grids, more iterations etc., but nothing helped. Finally, as explained above, smoothing of the surface cured the problem. This is a typical example how problems with quite different scales like housing and piston at the one side and fluid domain at the other side cause completely unexpected difficulties. Again our error estimate showed us merciless the quality

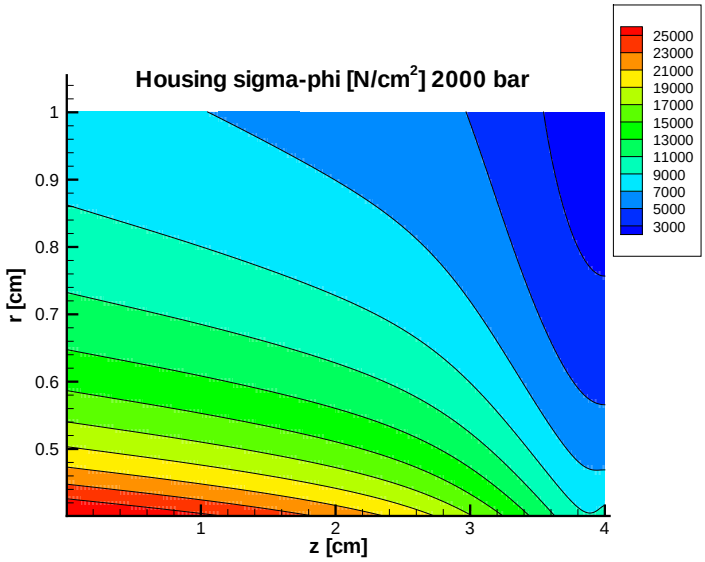


Figure 3.3.5.12: Contour plot for the stress component σ_φ for the housing for 2000 bar.

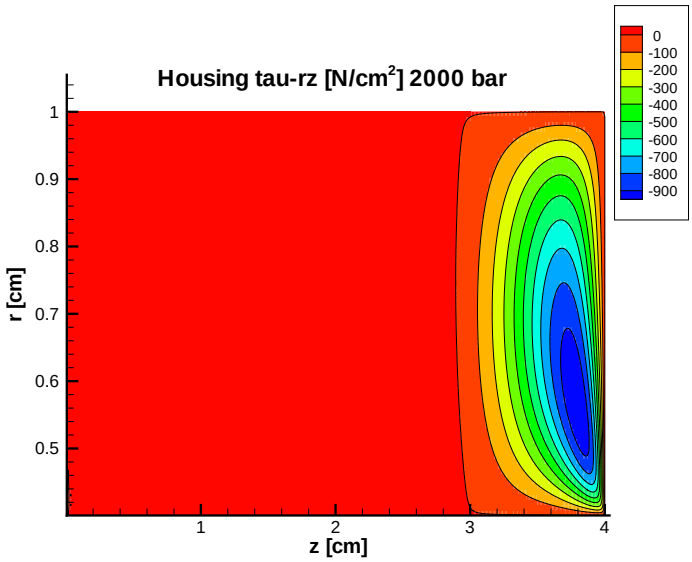


Figure 3.3.5.13: Contour plot for the stress component τ_{rz} for the housing for 2000 bar.

of the solution. When the oscillations occured the errors of the fluid domain became large. The natural action in such a situation is to refine the grid for the fluid domain, but this did not help because the errors were caused by the oscillations of the elasticity equations. This shows also drastically how "dirty" computational mathematics may be.

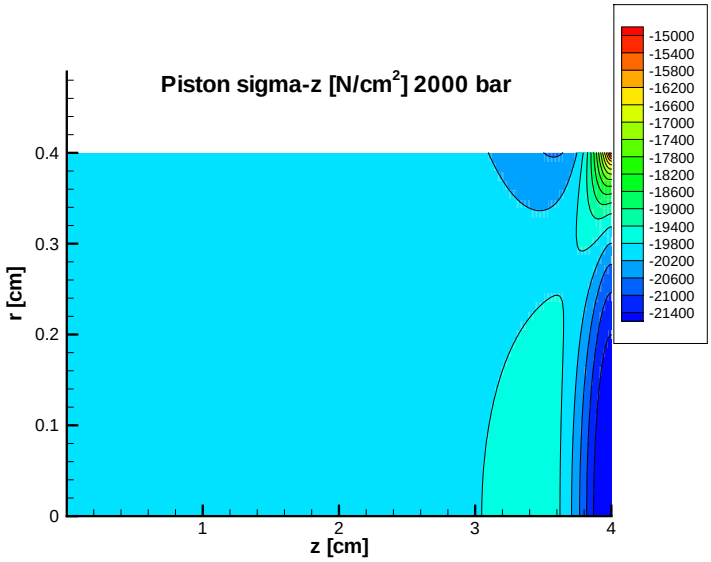


Figure 3.3.5.14: Contour plot for the stress component σ_z for the piston for 2000 bar.

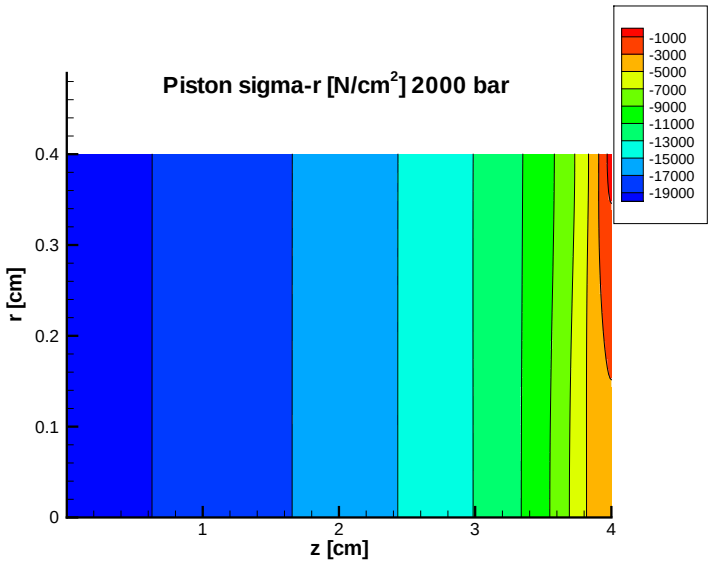


Figure 3.3.5.15: Contour plot for the stress component σ_r for the piston for 2000 bar.

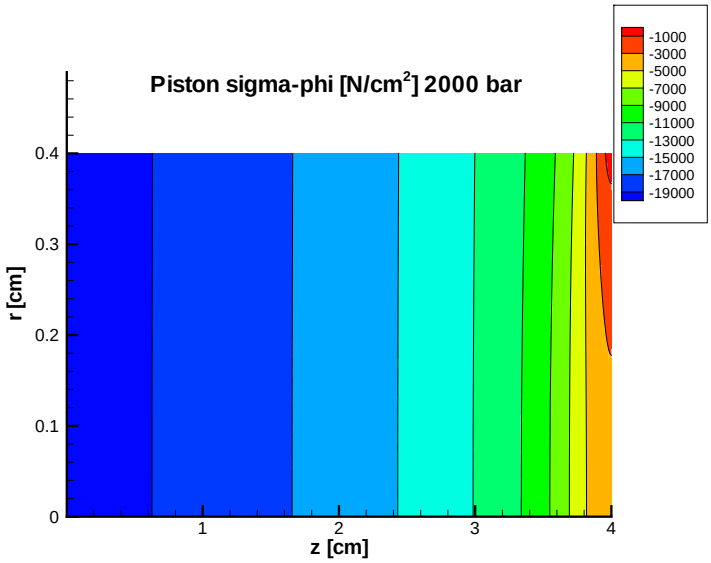


Figure 3.3.5.16: Contour plot for the stress component σ_φ for the piston for $2000\ bar$.

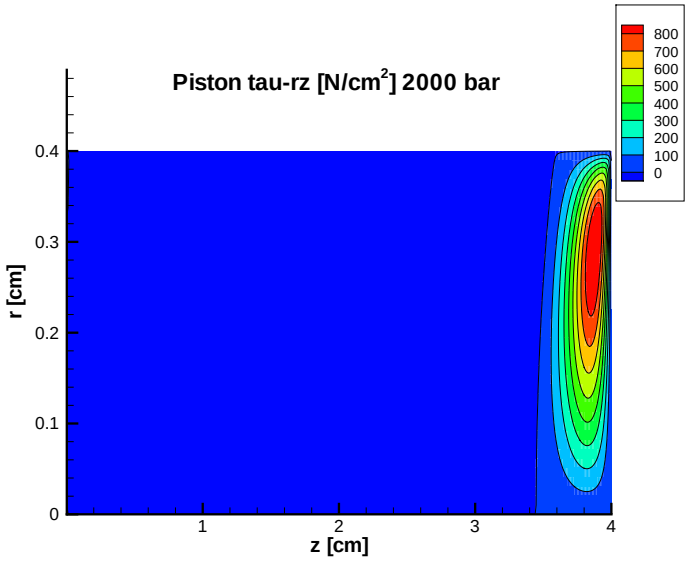


Figure 3.3.5.17: Contour plot for the stress component τ_{rz} for the piston for $2000\ bar$.

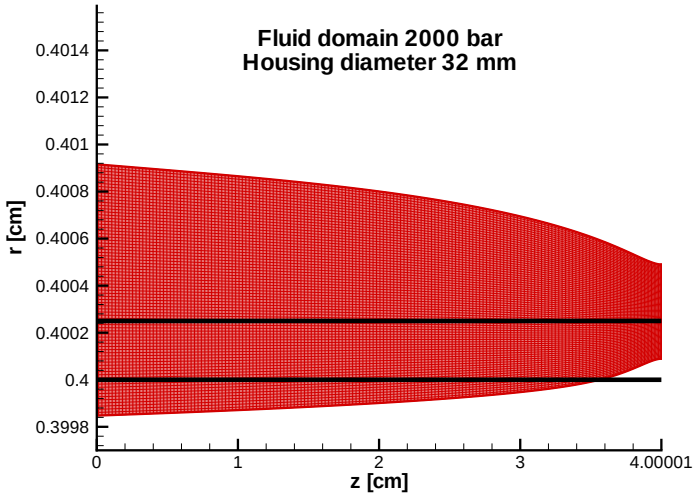


Figure 3.3.5.18: Fluid domain with the computational grid for $2000\ bar$ for housing diameter $32\,mm$, bold lines: original channel.

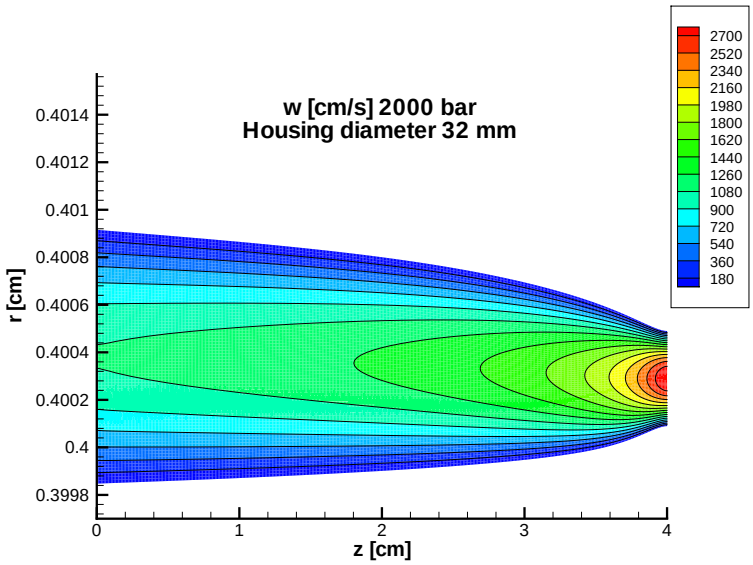


Figure 3.3.5.19: Contour plot of the velocity w in z-direction for $2000\ bar$ for housing diameter $32\,mm$.

Erratum

In (3.3.2.8) we gave the value $\eta/\rho = 2.5 \cdot 10^{-3}\ mm^2/s$. Dr. Martin Petry of Bosch told us that this value is wrong! In the transformation of N and kg we did an error. The correct value is

$$\frac{\eta}{\rho} = 2.5\ mm^2/s.$$

So the value in (3.3.2.8) is by a factor 10^3 too small. However, we computed in cm (not in mm) as explained above, because for this scale the errors were the smallest. We used in the cm scale the value $\eta/\rho = 2.5 \cdot 10^{-2}\ mm^2/s = 2.5 \cdot 10^{-4}\ cm^2/s$ which is by a factor of 100 too small. The correct value is $\eta/\rho = 2.5 \cdot 10^{-2}\ cm^2/s$. So the computations for the results of Section 3.3.5 were computed with η/ρ that is by a factor 100 too small. Then we repeated for $p = 2000\ bar$ the calculations with the correct value of η/ρ. The values corresponding to Table 3.3.5.1 are

p = 2000 bar, **corrected values**

Housing

		max. solution	max. error	mean error
w	cm	0.4140E-02	0.57E-04	0.20E-05
u	cm	0.7583E-03	0.46E-03	0.23E-04
sigma-z	N/cm^2	0.2238E+05	0.22E-02	0.29E-05
sigma-r	N/cm^2	0.2000E+05	0.41E-02	0.55E-05
sigma-phi	N/cm^2	0.2771E+05	0.93E-03	0.20E-04
tau-rz	N/cm^2	0.9259E+03	0.36E-01	0.21E-04

Piston

		max. solution	max. error	mean error
w	cm	0.2116E-02	0.12E-03	0.28E-05
u	cm	0.1524E-03	0.70E-03	0.18E-05
sigma-z	N/cm^2	0.2186E+05	0.32E-02	0.37E-05
sigma-r	N/cm^2	0.2001E+05	0.41E-02	0.81E-06
sigma-phi	N/cm^2	0.2001E+05	0.13E-02	0.87E-06
tau-rz	N/cm^2	0.8619E+03	0.39E-01	0.12E-04

Fluid

		max. solution	max. error	mean error	Volume [cm^3/s]
w	cm/s	0.3767E+04	0.14E-01	0.39E-02	2.63
u	cm/s	0.4028E+00	0.32E+02	0.21E+01	
p	N/cm^2	0.2000E+05	0.10E+00	0.94E-02	

The values of Table 3.3.5.1 were computed with 81 nodes in radial direction in the fluid. However, for this grid we got rather large errors for the correct value of η/ρ. In order to get a maximal error in the 1% range, see the table above, we needed 641 nodes in radial direction in the fluid, i.e. we computed with 401×641 nodes in the fluid domain.

The volume flow for the wrong η/ρ was $2.40\ cm^3/s$, the value for the correct η/ρ is $2.63\ cm^3/s$. This means that the increase of η/ρ by a factor 100 changes the volume flow by 9.6% relative to the old (wrong) value and by 8.7% relative to the new (correct) value. This unvoluntary experiment shows that the volume flow depends only marginally on the value of η/ρ. Therefore we did not repeat all the examples of Section 3.3.5. The volume flows given there should be increased by roughly 10% to get a better value.

The Figs. 3.3.5.4 for the fluid domain and 3.3.5.5 for w are basically the same for the correct value of η/ρ.

3.4 The simulation of the oxygen diffusion in a PEM fuel cell

The PEM (proton exchange membrane) fuel cell is a “cold” cell, in contrast to the “hot” SOFC (solid oxyde fuel cell). In its simple form it uses hydrogen and oxygen (air) as propellant. The manufacturer Freudenberg (Weinheim, Germany) produces non-woven (fleece) material that is used as GDL (gas diffusion layer) in PEM fuel cells. As mentioned in Section 3.1 Freudenberg was primarily interested in the gas diffusion properties of the GDL. Therefore the problem was reduced to the simulation of the oxygen diffusion in the GDL of a test fuel cell.

In the Addendum A1 there are 19 slides (in German) that describe the problem and give the PDEs and BCs. In the Addendum A2 there is again a summary of the problem and above all there are given the values of the material coefficients for the simulation. The values are given in the standard units kg, m, s, K and in grey background in g, mm, s, K. We use the latter scale for the computation.

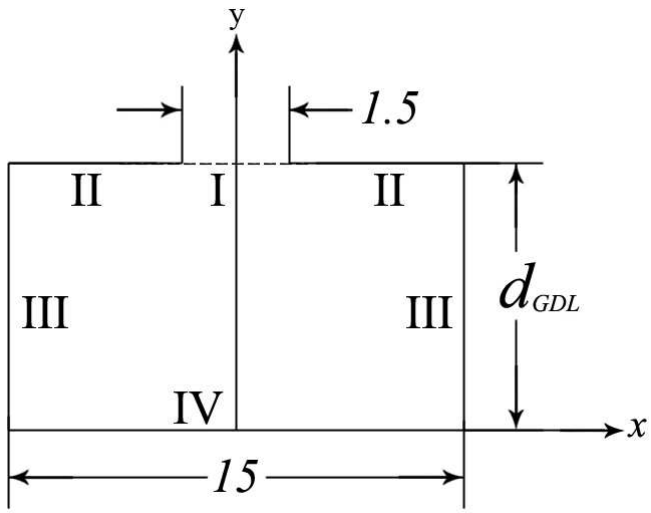


Figure 3.4.1: Definition of the GDL, which is the computational domain. Observe the strongly exaggerated opening I to the channel and of the GDL with $d_{GDL} = 150\ \mu m = 0.15\ mm$.

Fig. 3.4.1 shows the computational domain which combines the figures A1, Folie 1+3. On Folie 4 the used model is described. The unknown functions are (see A1, Folie 10):

u_x, u_y:	velocity components in x- and y-direction,
p:	pressure,
ϱ:	total density,
T:	temperature,
C_{O_2}:	mass concentration of oxygen.

On A1, Folie 3–8 the 6 equations for the 6 unknowns are derived and compiled on Folie 9, with the definition of the used symbols on Folie 10–11. On A1, Folie 12–17 the BCs for the boundaries I-IV, see Fig. 3.4.1, are compiled, with the used additional symbols on Folie 18.

The sequence of the 6 variables and equations in the interior of the computational domain is as

follows

Variable	equation
	(A1, Folie9)
u_x	x-momentum (Impuls),
u_y	y-momentum (Impuls),
p	constitutive equation,
ϱ	mass conservation
	(products differentiated),
T	energy equation,
C_{O_2}	O_2 transport equation.

The BCs for boundary I, see Fig. 3.4.1, the opening to a gas channel, are given in A1, Folie 13. However, as there are given only 3 BCs, we must take additionally 3 of the PDEs. The setting is as follows:

u_x:	x-momentum, from $p = p_K = const.$,
	see below, follows $\partial p/\partial x = 0$, thus $u_x = 0$,
u_y:	y-momentum equation,
p:	$p = p_K$, channel pressure,
ϱ:	constitutive equation,
T:	thermal condition,
	see 2. on Folie 13,
C_{O_2}:	diffusive O_2 transport in y-direction,
	see 3. on Folie 13.

The BCs for boundary II, the upper wall, are given in A1, Folie 14. Again we need additionally 3 PDEs:

u_x:	x-momentum equation,
u_y:	$u_y = 0$, impermeability condition,
p:	from the y-momentum equation
	follows with $u_y = 0$ the BC $\partial p/\partial y = 0$,
ϱ:	constitutive equation,
T:	heat conduction equation,
	see 2. on Folie 14,
C_{O_2} :	impermeability for O_2 gives $\partial C_{O_2}/\partial y = 0$,
	see 3. on Folie 14.

The BCs for boundary III, side walls, are given in A1, Folie 15. Again we need additionally 3 PDEs:

u_x:	$u_x = 0$, impermeability condition,
u_y:	y-momentum equation,
p:	from x-momentum equation follows
	with $u_x = 0$ the BC $\partial p/\partial x = 0$,
ϱ:	constitutive equation,
T	isolated wall, $\partial T/\partial x = 0$,
	see 2. on Folie 15,
C_{O_2}	impermeability $\partial C_{O_2}/\partial x = 0$,
	see 3. on Folie 15.

The BCs for boundary IV, membrane with catalytic layer are given in A1, Folie 16 and 17. Here we also need additionally 3 PDEs:

u_x:	x-momentum equation,
u_y:	equation of point 1. on Folie 16,
	balancing of species density transport
	in y-direction of water vapor and oxygen,
p:	y-momentum equation,
ϱ:	constitutive equation,
T:	equation of point 2. on Folie 17,
	heat conduction caused by the exothermal catalytic reaction,
C_{O_2} :	equation of point 3. on Folie 17,
	instantaneous irreversible reaction
	at catalytic layer.

All the necessary coefficients of the PDEs and BCs are given in the Addendum A2. We use the values in the units g, mm, s, K.

Of principal interest is also the mass concentration of water vapor C_{H_2O}. As we have 3 species, O_2, H_2O and N_2, we have

$$C_{O_2} + C_{H_2O} + C_{N_2} = 1.$$

We have $C_{N_2} = \gamma = const. = 0.7$, see last line on A2, Folie 5, so we have

$$C_{H_2O} = 1 - C_{O_2} - \gamma,$$

thus C_{H_2O} can be easily computed if C_{O_2} is known.

Fig. 3.4.1 shows the computational domain. We used for the first test computations a triangular grid of 101×21 nodes in x- and y-diretion and consistency order $q = 2$. The error estimates for the velocities u_x, u_y were in the range of 30%, those of the other variables below 0.01%. More nodes in the y-direction did not significantly change the results. More nodes in the x-direction changed significantly the values of u_x, u_y but did not improve the error estimates. The other values remained unchanged. This is a clear indication for a singularity in the velocities. The large errors occured at the left and right end of boundary I, see Fig. 3.4.1, i.e. where the BC of boundary II, the channel opening, changes to the upper wall, see Fig. 3.4.2.

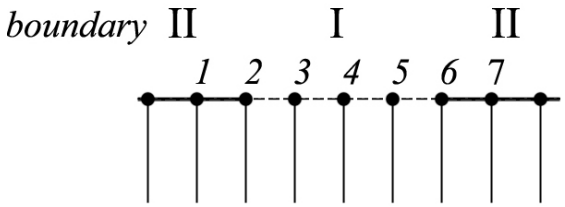


Figure 3.4.2: Illustration to BCs.

At nodes 1, 2, 6, 7 we have the BCs of boumdary II, at nodes 3, 4, 5 those of boundary I. At nodes 2 and 6 we have $u_y = 0$ (impermeability), at the neighboring nodes 3 and 5 we have the y-momentum equation, i.e. $u_y = -(K_y/\eta)\partial p/\partial y \neq 0$ which causes the fluid flow through the opening. The basic reason for this incompatibility is Darcy's law, see A1, Folie 5, for the momentum equations that allows such a jump.

If we look at Fig. 3.4.2. we see that the nodes 2 and 3 or 5 and 6 come closer together if we refine the grid in the x-direction. This means that the "length" of the opening where we apply the BCs of boundary I, changes with the grid spacing. The solution of this problem would be to "collapse" nodes 2 and 3 or 5 and 6 and to apply at the same geometrical node the conditions of the channel for the inside of the channel and the conditions of the wall at the outside. This possibility is included in FDEM by the concept of the dividing lines. In Fig. 3.4.3 the situation is depicted.

Here the geometrical nodes 2 and 6 are split up into two logical nodes each, where for $2'$ and $6''$ the BCs of the wall and for $2''$ and $6'$ the BCs of the channel are applied. In the interior of the computational domain we need coupling conditions for the two logical nodes that result from one geometrical node, e.g. $8', 8''$ and $9', 9''$ in Fig. 3.4.3. Because in the interior the solution goes in the x-direction continuously through the dividing line we use the coupling conditions:

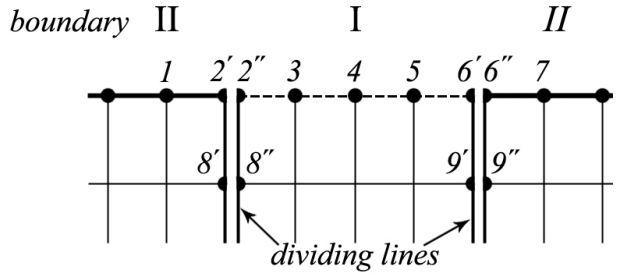


Figure 3.4.3: Illustration with dividing line.

$$\text{variable}_{left} = \text{variable}_{right},$$
$$\left(\frac{\partial \text{variable}}{\partial x}\right)_{left} = \left(\frac{\partial \text{variable}}{\partial x}\right)_{right}.$$

Table 3.4.1: Results for 6 different grids. Error denotes the max. estimated global relative error of the solution component. Solution is the max. absolute value.

Var.	solution	error	solution	error	solution	error
grid	101×21		201×21		401×21	
u_x	0.776	0.256	0.852	0.238	0.999	0.270
u_y	15.7	8.54E-2	8.83	0.189	5.45	0.242
p	1.40E5	3.78E-5	1.40E5	2.33E-5	1.40E5	1.28E-5
ϱ	1.22E-6	1.04E-4	1.22E-6	5.22E-5	1.22E-6	2.91E-5
T	348	3.15E-8	348	1.48E-8	348	1.90E-8
C_{O_2}	8.73E-2	1.43E-3	8.74E-2	6.51E-4	8.74E-2	3.87E-4
grid	101×41		201×41		401×41	
u_x	0.778	0.274	0.860	0.241	1.02	0.266
u_y	31.3	0.114	17.5	0.180	10.7	0.226
p	1.40E5	3.83E-5	1.40E5	2.32E-5	1.40E5	1.28E-5
ϱ	1.22E-6	7.07E-4	1.22E-6	2.29E-4	1.22E-6	7.09E-5
T	348	9.84E-7	348	3.03E-7	348	8.45E-8
C_{O_2}	8.73E-2	1.34E-2	8.74E-2	4.33E-3	8.74E-2	1.23E-3

In Table 3.4.1 we present the results for 6 different grids: the max. absolute function values and the max. estimated global relative errors. We solve with consistency order $q = 2$ and expect from half the grid size an error reduction by a factor $(1/2)^2 = 1/4$. If we go in Table 3.4.1 to the right we have a doubling of the meshes, i.e. a halving of the mesh size, in the x-direction, if we go down we have a halving in the y-direction, in the diagonal we have halving in both directions. However, if we look at Table 3.4.1 we cannot find a convergent sequence of function values for u_x and u_y and the estimated errors are relatively large and do not decrease with finer grid. For the other variables the function values are practically the same for all grids and the error estimates are very small, but do not decrease in the expected way. This behaviour is a clear indication that there is a singularity for the velocities in the problem.

If we look in Fig. 3.4.3 at nodes $2'$ and $2''$, we have at $2'$ for u_y the impermeability condition $u_y = 0$ and at $2''$ Darcy's law $u_y = -(K_y/\eta)\partial p/\partial y$. Similarly we have for u_x at $2'$ Darcy's law $u_x = -(K_x/\eta)\partial p/\partial x$ and at $2''$ we have from Darcy's law and constant p in the opening that $u_x = 0$. So we have a jump in u_x and u_y between $2'$ and $2''$. The same holds for $6'$ und $6''$.

This inherent singularity cannot be removed by any measure of the numerical method. If for the momentum equations a Navier-Stokes model with zero velocities at walls had been used, there were no singularities.

In order to attenuate the singularities we have changed the BCs at $2'$ und $6''$ for p from $\partial p/\partial y = 0$ to $\partial p/\partial x = 0$ so that from Darcy's law follows $u_x = 0$ which fits to the channel condition, and at $2''$ and $6'$ we similarly change from $p = p_K$ to $\partial p/\partial y = 0$ which gives from Darcy's law $u_y = 0$. Note that these measures do not remove the singularity caused by Darcy's law, they can only attenuate it.

Table 3.4.2: Results with modified BCs for 6 different grids. Error denotes the max. estimated global relative error of the solution component. Solution is the max. absolute value.

Var.	solution	error	solution	error	solution	error
grid	101×21		201×21		401×21	
u_x	0.715	0.352	0.673	0.388	0.598	0.462
u_y	0.170	0.698	0.285	0.651	0.435	0.541
p	1.40E5	3.78E-5	1.40E5	2.33E-5	1.40E5	1.28E-5
ϱ	1.22E-6	1.04E-4	1.22E-6	5.22E-5	1.22E-6	2.91E-5
T	348	3.15E-8	348	1.48E-8	348	1.90E-8
C_{O_2}	8.73E-2	1.43E-3	8.74E-2	6.51E-4	8.74E-2	3.87E-4
grid	101×41		201×41		401×41	
u_x	0.715	0.358	0.673	0.396	0.598	0.473
u_y	0.170	0.710	0.285	0.654	0.436	0.542
p	1.40E5	3.87E-5	1.40E5	2.36E-5	1.40E5	1.27E-5
ϱ	1.22E-6	7.07E-4	1.22E-6	2.29E-4	1.22E-6	7.15E-5
T	348	9.83E-7	348	3.02E-7	348	8.39E-8
C_{O_2}	8.73E-2	1.33E-2	8.74E-2	4.31E-3	8.74E-2	1.22E-3

Table 3.4.2 shows the results for these modified BCs in the same form as Table 3.4.1. The function values and error estimates for the last 4 variables are practically the same in both tables, obviously these variables are not affected by the singularity of the velocities at the ends of the channel. For the velocities u_x and u_y the situation is quite different from that of Table 3.4.1. In Table 3.4.2 the function values of u_x und u_y and even the error estimates do not change with the y-grid, the function values are quite different of those of Table 3.4.1 and they change in a different way with the x-grid. The error estimates tell us again that the values are not accurate. Note that we discuss here only the maxima.

The results of Table 3.4.2 show that also with the modified BCs at the end nodes of the channel opening the basic singular behaviour (caused by Darcy's law) is still present. The solution method with the error estimate merciless shows the consequence: the inaccuracy of the velocities u_x and u_y. At the same time the error estimates show the accuracy of the other 4 variables in spite of the inaccuracy of the velocities. So the error estimates tell us that we must look for a better, singularity-free model for the velocities, but this was not the task for us.

Here we must discuss the results of Table 3.4.1 and 3.4.2 in more detail. What we see in the tables is not the whole information: these are the maximal absolute values of the solution components and of the global relative error components (max. error component over the domain divided by max. solution component over the domain). The velocities have a singularity at nodes 2 und 6 of Fig. 3.4.2 or nodes $2'$, $2''$ and $6'$, $6''$ of Fig. 3.4.3. There, at these singular nodes, the function values are "arbitrary" (at a singularity there is no unique value), and quite naturally there are also the maxima of the errors. If we look at nodes that are some grid spacings away from the singularity, the function values do no longer change with the grid, and the errors are correspondingly smaller. Obviously, by the internal structure of the PDEs, the values of the velocity components at the singularities do

not affect the solution of u_x and u_y in the interior of the domain which is visible by the smaller errors there. As the maximal values of the other variables and the maximal values of their errors occur away from the singularity in the interior, these values are not affected by the values of u_x, u_y at the singularity. So only the "whole" information about the solution and the errors reveals the ultimate behaviour of the solution. Without the knowledge of the error estimate the explanation of Tables 3.4.1 and 3.4.2 would be difficult.

In order to give an information about the computation time on 8 processors of our IBM SP with Power3 processors of 375 MHz we have executed the grid 201×41 of Table 3.4.2 in batch mode (the other results have been computed in interactive mode that gives a little larger timings). The CPU time of the master processor 1 is $88.3\,sec$. In this timing the part of the linear solver LINSOL for the computation of 2 Newton corrections and of the error estimate for $201 \times 41 \times 6 = 49446$ unknowns is $86.5\,sec$. So most of the solution time is used for the linear solver which is in this case our CG solver PRES20 with full LU preconditioning.

Tables 3.4.1 and 3.4.2 have been computed with consistency order $q = 2$. We know from our experience that higher order gives worse results if there are singularities in the solution because higher orders are much more sensitive with respect to singularities. Therefore we did not expect better results for higher order. We solved the PDEs of Table 3.4.2 for the grid 401×41 with consistency order $q = 4$ and $q = 6$. For $q = 4$ we got $u_x = 0.670$, $u_y = 0.543$, but the error estimates were 56.9 and 18.2. The values of the other variables were the same as for $q = 2$, but the error estimates were roughly by a factor of 10 larger. For $q = 6$ the result was nonsense and the error estimates of the velocities were 1500. This confirms our experience that higher order is not useful in the presence of singularities.

For the illustration of the solution we show the results for each variable and also C_{H_2O} in two figures: at first as a greyscale plot and then as contour plot for the grid 201×41 of Table 3.4.2. Observe the different y-scale.

In conclusion to this problem of oxygen diffusion at the cathode side of the test configuration of a PEM fuel cell we can say that FDEM worked immediately without any problem of the numerical method. The built-in error estimate revealed merciless the singularity of the used model for the velocities. It showed the inaccuracy of the velocities and the accuracy of the remaining 4 variables. Better results for the velocities can be obtained only by a better singularity-free model.

Originally it was intended to simulate a whole PEM fuel cell, but Freudenberg had not the corresponding model equations with the values of the necessary coefficients available. We had discussed such a model. The problem here is the modeling of the catalytic membrane between anode and cathode. There are two possibilities: either to model the membrane as an infinitely thin layer and to model this by a single dividing line with a jump in some variable, or to resolve the membrane by a grid of a certain thickness and to separate it by two dividing lines from the two gas diffusion layers on both sides. For both models of equations there would be no problem for FDEM to compute a global error estimate over the gas diffusion layers and the membrane.

As PEM fuel cells are combined in a stack to deliver correspondingly more power, FDEM could compute the global solution over a whole stack, with global error estimate. With such a model one could "play" with different configurations and optimize the whole stack. The error estimate tells us if we can trust our numerical results and shows eventual weak points in the model as we have seen it above. In this sense FDEM is a unique tool for the simulation of PEM fuel cells. Quite naturally the same arguments hold for SOFC, the "hot" fuel cells.

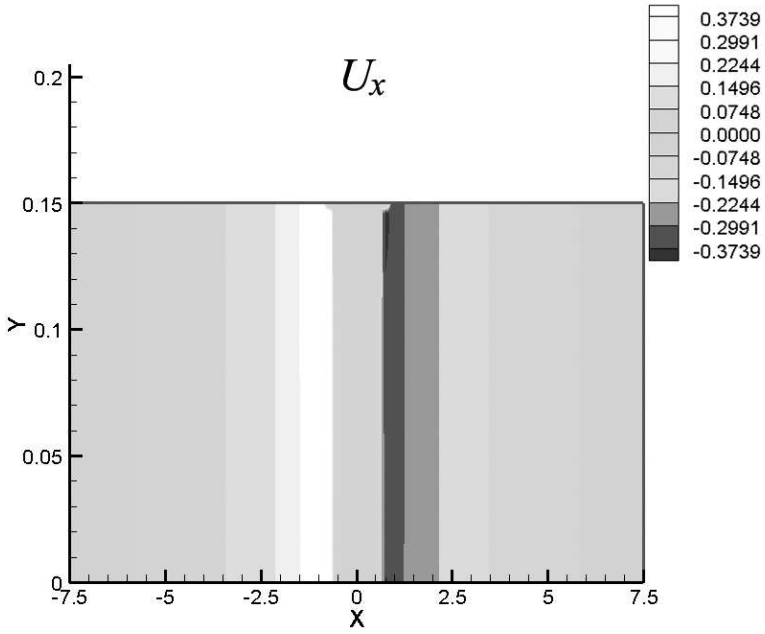


Figure 3.4.4: Grayscale result for u_x.

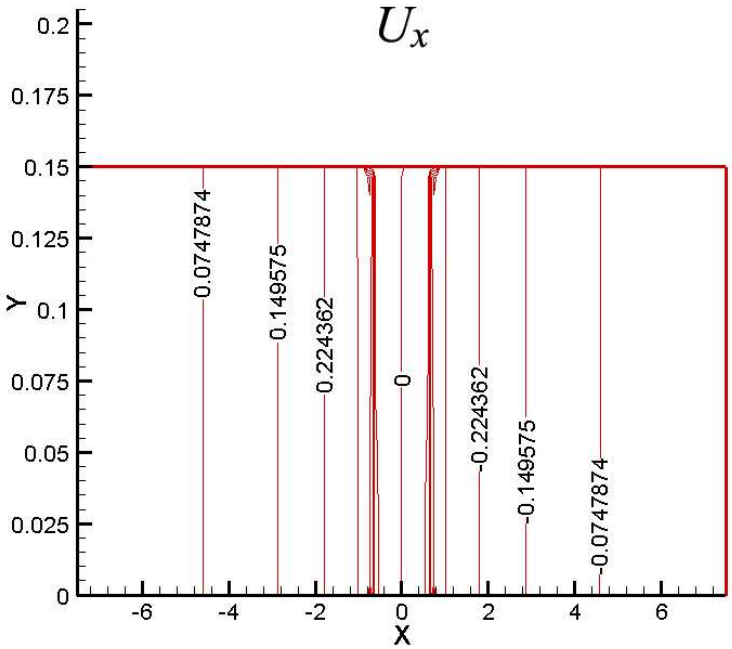


Figure 3.4.5: Contour plot result for u_x.

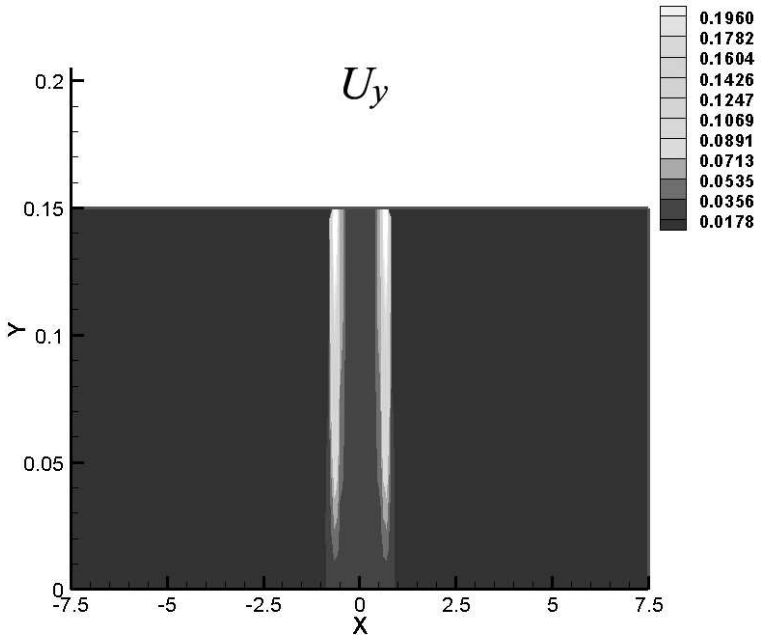


Figure 3.4.6: Grayscale result for u_y.

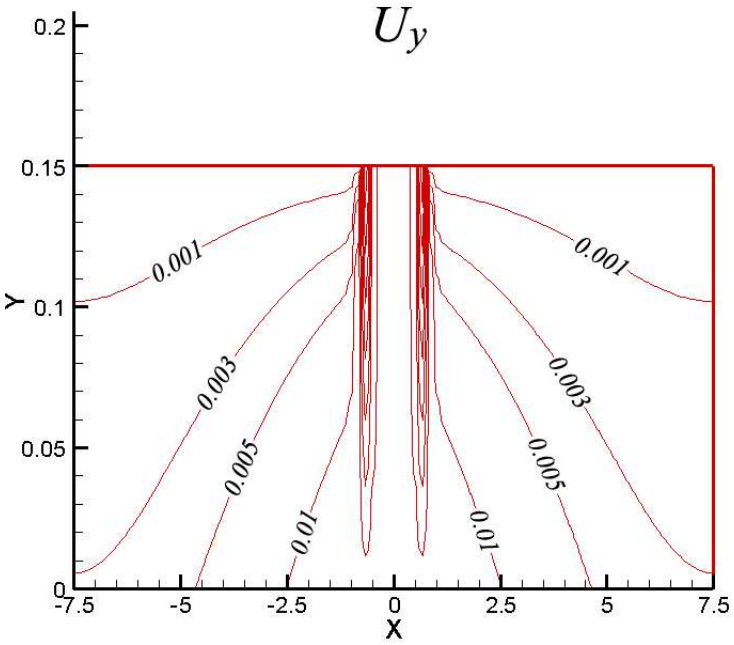


Figure 3.4.7: Contour plot result for u_y.

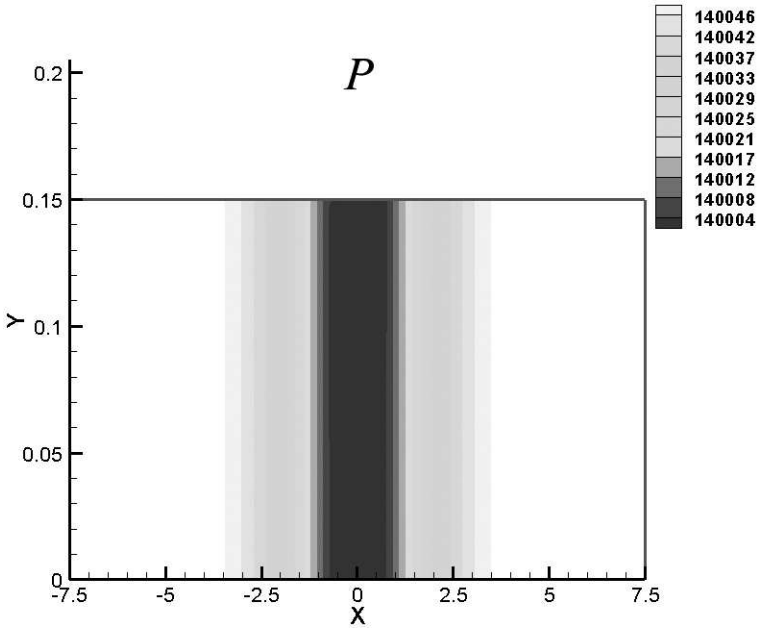


Figure 3.4.8: Grayscale result for p.

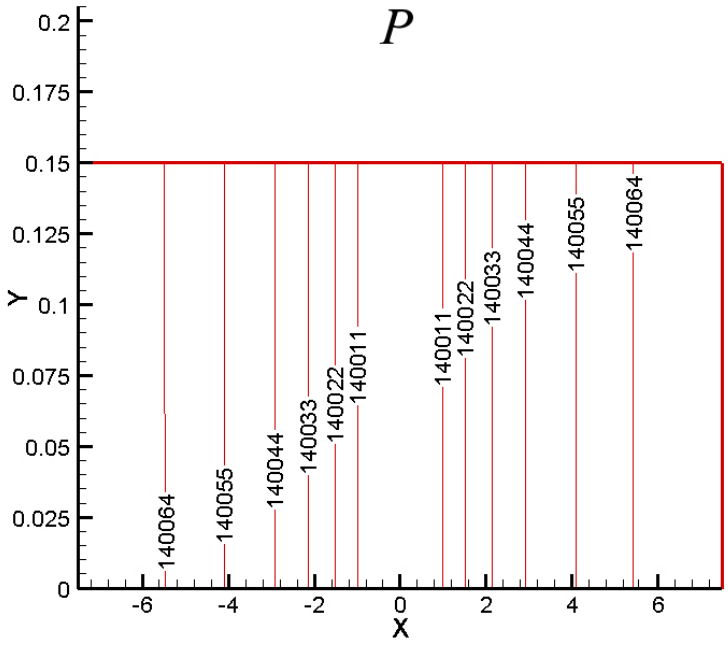


Figure 3.4.9: Contour plot result for p.

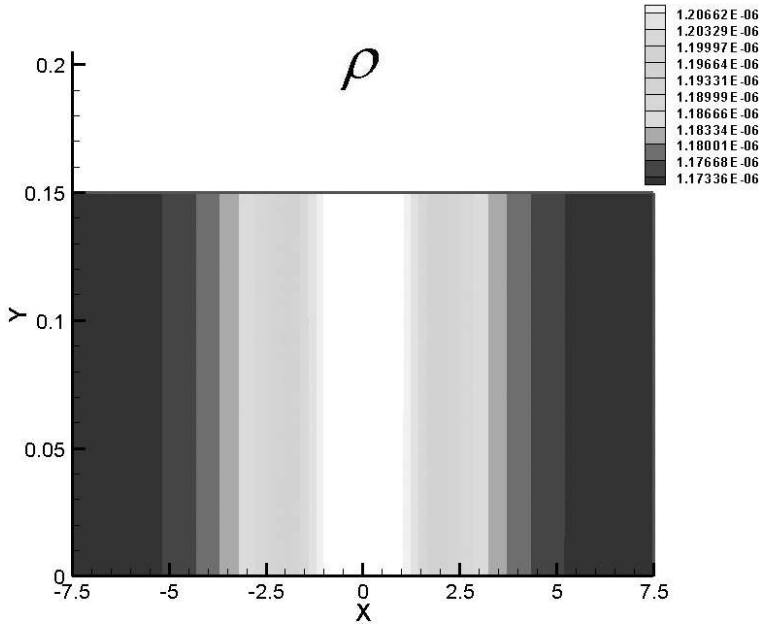


Figure 3.4.10: Grayscale result for ϱ.

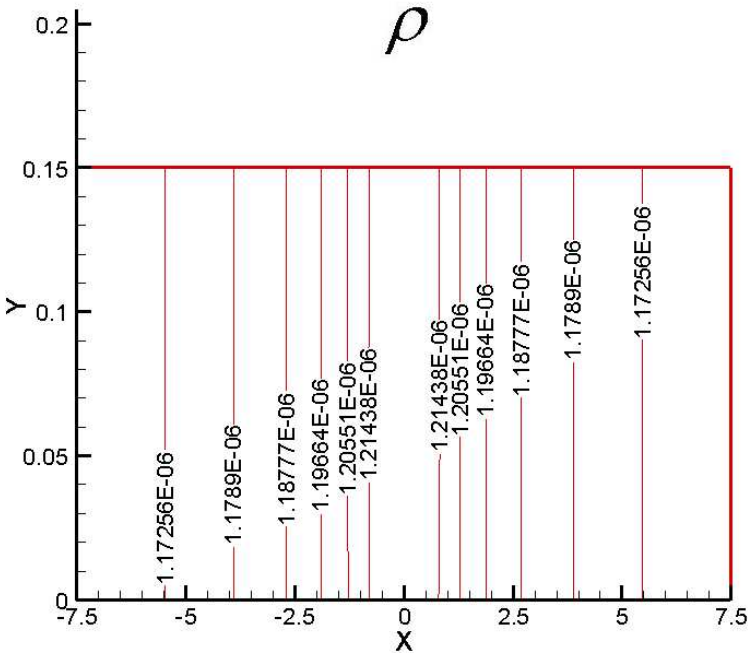


Figure 3.4.11: Contour plot result for ϱ.

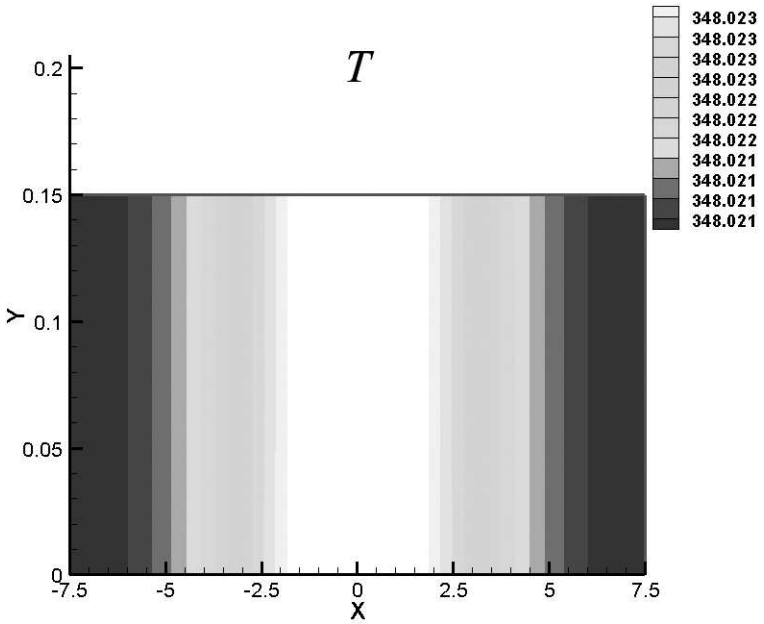


Figure 3.4.12: Grayscale result for T.

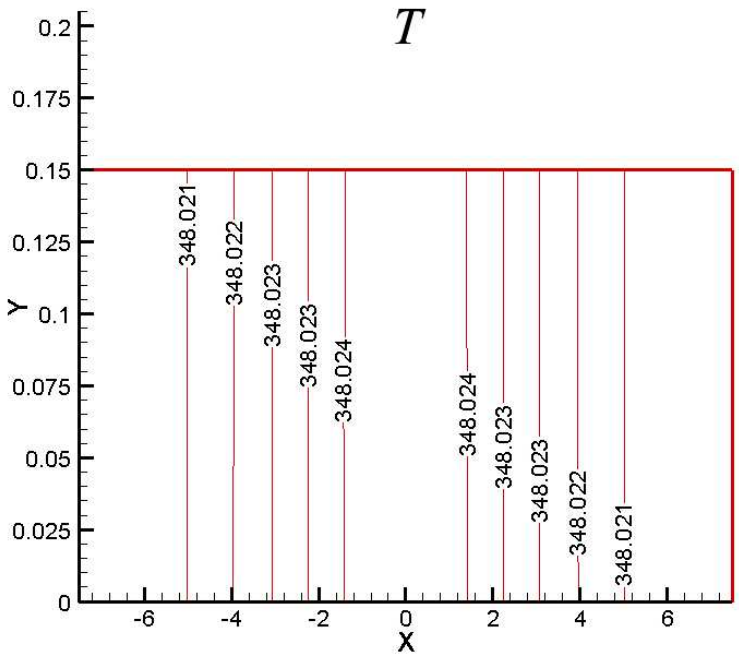


Figure 3.4.13: Contour plot result for T.

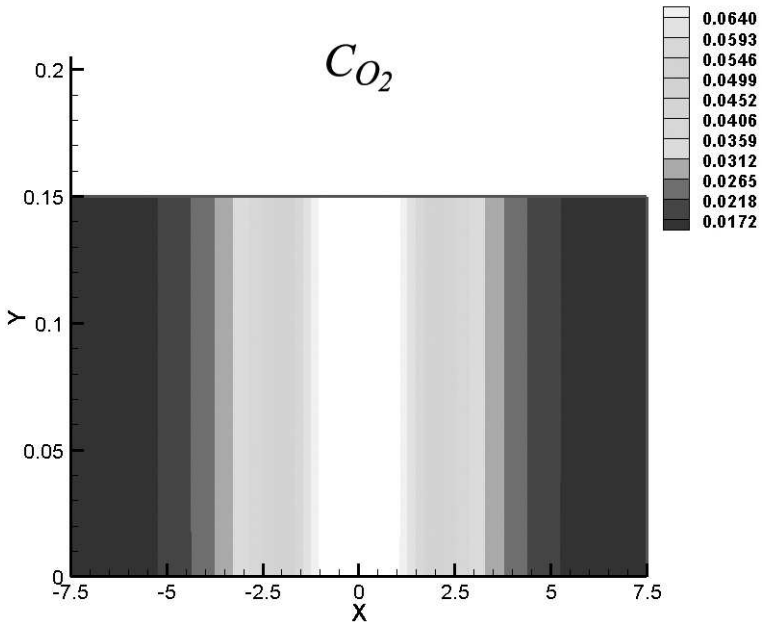


Figure 3.4.14: Grayscale result for C_{O_2}.

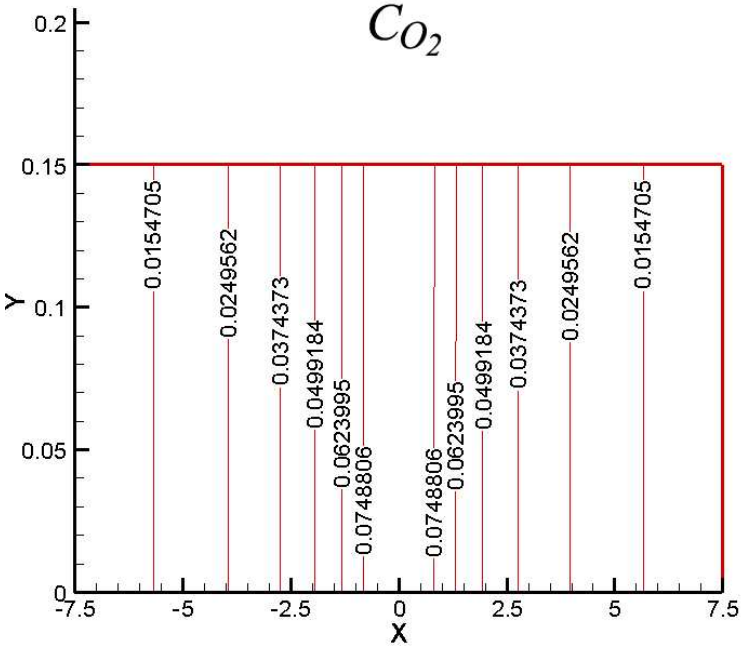


Figure 3.4.15: Contour plot result for C_{O_2}.

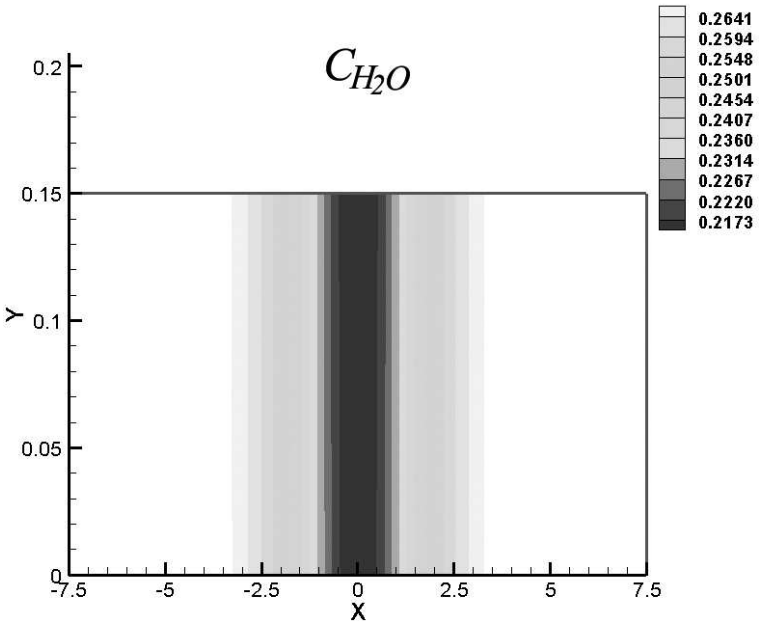


Figure 3.4.16: Grayscale result for C_{H_2O}.

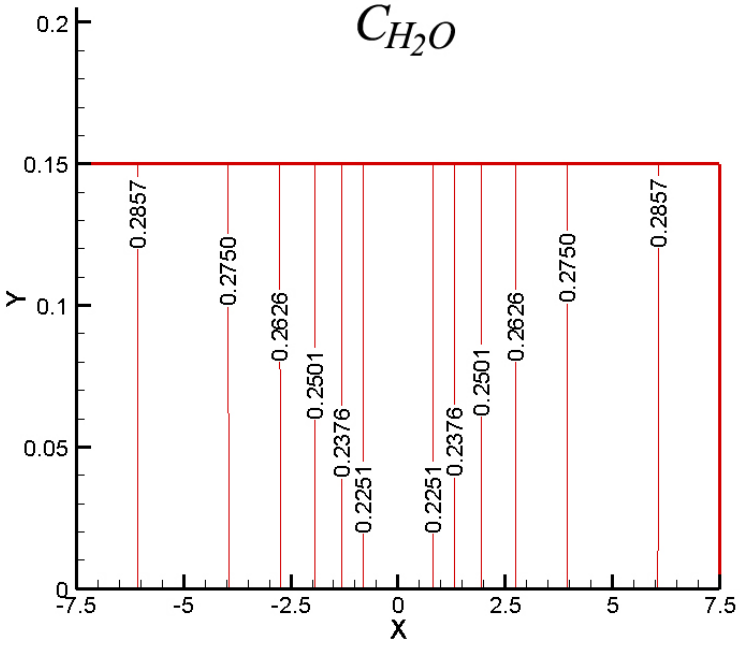


Figure 3.4.17: Contour plot result for C_{H_2O}.

4 Remarks to a User's Guide

When we started this research project we had a "black-box" solver FDEM with which we could solve arbitrary non-linear systems of elliptic and parabolic PDEs with error estimate on an arbitrary 2-D or 3-D domain for which a discretization by triangles or tetrahedrons was available. Then we got from the IWKA the problem to simulate the manufacturing of metal bellows, from Bosch a problem from the area of high pressure Diesel injection pumps and from Freudenberg the fuel cell problem.

For the IWKA problem we had to design a program that described in the tiniest details the manufacturing process in its different stages. The grid moves with the deformation, the equations change between elastic and plastic and a moving tool forces the metal sheet into the form. The algorithm, that simulates this process, is closely interleaved with the FDEM code. Nobody else than the developer of the FDEM code could generate this code. However, FDEM itself is such a complicated code, that it would not be "suitable" to explain the code in the necessary details to a general user of FDEM. In this case an interested user must be personally instructed by the developer how to use the code for different parameters of different problems of the same type.

The Bosch problem is only a cutout of the larger problem, namely the simulation of a whole injection pump. It could be seen in the early talks with Bosch, that on the available supercomputers this problem could not be solved with satisfying accuracy. The partial problem for which we finally agreed is a fluid-structure interaction problem for which we offered a global solution with global error estimate. However, in the detailed analysis of the problem it turned out that the solution of the seemingly simple problem needed a 4-fold nested iteration (Fig. 3.3.4.6) where FDEM was the innermost core iteration. Again it would not be "suitable" to explain this code to a general user of FDEM so that for an interested user only a personal instruction is useful.

The Freudenberg problem of the solution for the oxygen diffusion in a fuel cell, see Chapter 3.4, could be solved basically with the standard version of FDEM. So for this problem we will describe in detail in the form of a user's guide how to use FDEM. This is a 2-D problem.

4.1 Structure of the grid files

One of the key parameters for the solution of PDEs with FDEM is the structured or unstructured FEM grid. In 2-D we use linear triangles, in 3-D linear tetrahedrons. As explained above in the general part this grid serves only for the structure of the 2-D or 3-D space, i.e. by the grid the neighboring nodes are known. We generate from the nodes difference formulas (we do not use a FEM for the solution).

For the simple geometries that we use in our examples we generate the grid explicitly "by hand", i.e. by an own explicit grid generator. However, we could—and we did it also in some examples—use a commercial grid generator. As at the time of the early development of FDEM at our computer center an I-DEAS grid generator was available, we structured our data according to the rules of that (older) I-DEAS grid generator (we do not know if actual versions still accept the data in this form). As the computer center later changed to the PATRAN grid generator, we wrote also a program that translated PATRAN data to I-DEAS data. With the hand-generated I-DEAS data we can flexibly change the number of nodes for accuracy tests which would not be possible with a commercial mesh generator. In the following we describe the used data structure.

All lists with the grid data are stored in p parts on the p processors of a parallel computer. The

data have on the processor a local number and they have also their global number. For the nodes we have two lists: nnr (global node number) and $coord$ (coordinates), nl is the local number of the last (local) node (i is local node number):

i	$nnr(i)$
1	216
2	312
3	24
$\vdots$	$\vdots$
nl	754

$\mapsto$ 3-D

i	$coord(i,1)$	$coord(i,2)$	$coord(i,3)$
1	first	second	third
2	coord.	coord.	coord.
3			
$\vdots$	$\vdots$	$\vdots$	$\vdots$
nl			

For the elements we have also two lists: $nenr$ (global element number) and nek (global node number), nel is the local element number of the last (local) element (i is local element number):

i	$nenr(i)$
1	25
2	101
3	37
$\vdots$	$\vdots$
nel	871

$\mapsto$ 3-D

i	$nek(i,1)$	$nek(i,2)$	$nek(i,3)$	$nek(i,4)$
1	238	240	261	231
2	global	node	number	of
3	first	second	third	forth
$\vdots$		node		
nel	$\vdots$	$\vdots$	$\vdots$	$\vdots$

We have three types of boundaries:

1. external boundaries,
2. internal boundaries of DL (dividing line) type,
3. internal boundaries of SDL (sliding dividing line) type.

A boundary may be a single node if there are given special BCs, e.g. at a corner of the domain. It is important to know that later a list can be supplied that determines which boundary overwrites which other boundary, e.g. at a corner where two boundaries meet, see below.

For each of the three types of boundaries there is a list. These lists are of the same structure and they are named

- $bnod$ for type 1,
- $tnod$ for type 2,
- $snod$ for type 3.

As example we show the $bnod$ list, nbl is the local number of the last boundary node of type 1 (i is local node number):

i	$bnod(i,1)$	$bnod(i,2)$
1	238	1
2	240	1
3	321	2
$\vdots$	global node	$\vdots$
nbl	number	$nexb$

$bnod(i,2)$ is the number of the boundary, $nexb$ is the max. boundary number. The arrays for $tnod$ (type 2) and $snod$ (type 3) have the same structure.

If we have a coupled domain that consists of $nsect$ (sub)domains, there is a list nod that tells which node belongs to which domain (i is local node number):

i	$nod(i,1)$	$nod(i,2)$
1	256	1
2	312	3
$\vdots$	global node number	number of domain

Because nodes of internal boundaries belong to two or more domains, there are more than nl rows in this list.

In the same way we have a list el that tells which element belongs to which domain (i is local element number):

i	$el(i,1)$	$el(i,2)$
1	25	1
2	250	2
⋮	global element number	number of domain

The data set for the entering of the grid data has in the style of the I-DEAS structure the following shape:

-1	start of a list
2411	identifier for nodes
$\{a1\}$	1^{st} row of nnr list, here 216
$\{b1\}$	1^{st} row of $coord$ list (2 or 3 values)
$\{a2\}$	2^{nd} row of nnr list
$\{b2\}$	2^{nd} row of $coord$ list
.	
.	
.	
-1	end of a list
-1	start of a list
2412	identifier for elements
$\{c1\}$	1^{st} row of $nenr$ list
$\{d1\}$	1^{st} row of nek list (3 or 4 values)
$\{c2\}$	
$\{d2\}$	
.	
.	
.	
-1	end of a list

[Now follows data for the boundaries. Each boundary has an own list between -1 and -1, boundaries have a number and a name. Below are given the rules for the names of boundaries.]

-1	start of a list
791	identifier for boundaries
{number of the boundary}	usually $1, 2, 3, ...$
{name of the boundary}	rules for names for FDEM see below, characterizes type of boundary
{e1}	1^{st} element of 1^{st} row of bound list, i.e. the node number
{f1}	3 values ⎫ *These values (forces etc.) are not*
{g1}	3 values ⎬ *needed for FDEM, we store zeros*
{h1}	6 values ⎭ *in the required format*
{e2}	1^{st} element of 2^{nd} row of $bnod$-list
{f2}	
{g2}	
{h2}	
.	
.	
.	
-1	end of list
-1	start of list
{next boundary}	
-1	end of list
.	
.	
.	
-1	end of list
{until all boundaries are listed}	

[Now follows data for the (sub)domains. At first there is the information for the nodes of the (sub)domains. A (sub)domain has a number and a name. Below are given the rules for the names of domains.]

-1 start of a list
2417 identifier for domains
{number of the domain}{number of nodes in the domain}
{name of the domain} starts with NO for nodes, see rules below.
7 256 7 268 7 243 7 212 [row with 4 node numbers, with a 7 before each node number]
{further rows with 4 node numbers}
.
.
.
{last row, eventually with zeros as node numbers if there is only a remainder}

{same type of node lists for further domains, starting with {number of the domain}, until all nodes of all domains are listed}

[Now follows in the same style the information for the elements of the (sub)domains. A (sub)domain has a number and a name, see rules below for names.]

{number of the domain}{number of nodes in the domain}
{name of the domain} starts with EL for elements, see rules below
8 21 8 43 8 24 8 31 [row with 4 element numbers, with an 8 before each element number]
{further the rows with 4 element numbers for this domain}
.
.
.
{last row, eventually with zeros as element numbers if there is only a remainder}
{same type of element lists for further domains, starting with {number of the domain}, until all elements of all domains are listed}
-1 identifier to denote the end of the domain list

For FDEM there are rules for the names of boundary and domain names. We have, as mentioned above, 3 types of boundaries: external, DLs and SDLs. Correspondingly there are 3 types of names:

EXname	for external boundaries, i.e. the first 2 letters of an external boundary name are EX.
INname	for DLs, internal boundaries, i.e. the first 2 letters of a DL boundary name are IN.
SLname	for SDLs, sliding DLs, i.e. the first 2 letters of a SDL boundary name are SL.

For domains we have 2 types of lists: lists for nodes and lists for elements. Correspondingly there are 2 types of names:

NOname	for lists with node numbers, i.e. the first 2 letters of a domain list with node numbers are NO.
ELname	for lists with element numbers, i.e. the first 2 letters of a domain list with element numbers are EL.

At the corner e.g. of a rectangular domain we have the intersection of 2 boundaries. So the corner belongs to 2 boundaries. When we generate the matrix and r.h.s. of the large sparse linear system for the computation of the Newton correction, the values for the corner depend on the sequence which boundary is treated as last one, because the last one overwrites the values of the first one at the corner. As we treat boundaries according to their number, boundaries with higher boundary number overwrite those with lower boundary number. Note that a "boundary" may be a single node, e.g. a corner, if there are given special BCs. In order to get more flexibility we have introduced the possibility to change this standard rule by a bd_order list so that we can prescribe which boundary overwrites which other one. The number of rows in this list is the number of external boundaries plus 2 times the number of DLs (a DL is doubled by the program, each copy belongs uniquely to one of the domains that are separated by the DL) plus the number of SDLs. The structure of the bd_order list is the following one:

i	$bd\_order(i,1)$	$bd\_order(i,2)$
1	boundary	boundary
2	number	type
$\vdots$		

boundary type has the value 1 for external boundaries, 2 for DLs, 3 for SDLs. The boundaries with lower position in the list (larger i) overwrite boundaries with higher position (smaller i). The bd_order list belongs to the list of entry parameters that control the execution of FDEM. If there is no bd_order list, the standard rule holds that boundaries with larger boundary number overwrite boundaries with lower boundary number.

As we want to use the grid file on parallel computers, for the parallel reading of the file there must be done some preparation. The file is generated and stored sequentially by a single processor. The preparation program reads the file sequentially but stores it back as a direct access file. Before the file is stored back a header is generated that contains the detailed information which data are stored where in the direct access file. When the file is used on a parallel computer, processor 1 reads the header and broadcasts it to the other processors. Then in parallel all p processors read their part of the lists, i.e. processor 1 the first p^{th} part, processor 2 the second p^{th} part etc.

4.2 Entering the PDEs into the program frame

As mentioned above we want to take as example the PDEs of Section 3.4, the oxygen diffusion in a fuel cell.

The PDEs must be entered in the program frame of the subroutine FDEMU1. The PDEs and variables for a system of m PDEs are

$$Pu = \begin{pmatrix} P_1 u \\ P_2 u \\ . \\ . \\ P_m u \end{pmatrix} = 0, \quad u = \begin{pmatrix} u_1 \\ u_2 \\ . \\ . \\ u_m \end{pmatrix},$$

i.e. we arrange the PDEs so that the r.h.s. is zero. In Table 4.2.1 the variables and equations as prescribed in Section 3.4 are compiled. In the Addendum A3 all the subroutines for the entering of the PDEs, BCs and the corresponding Jacobian matrices are printed. For the PDEs we look at the subroutine FDEMU1. All the constants that appear in the equations are defined in a module "probconst" that is accessed by the USE statement. At "start of calculation" there is a part that is active only if $ljac$ is true. This is used for the Jacobi test that is described in a later section. The entering of the PDEs starts with the loop

$$do \quad i = 1,\ mv$$

where mv is the maximal local node number. "local" means here on the actual processor of a (distributed memory) parallel computer. As mentioned in the main part of this report the data are distributed (with their overlap data) onto the processors so that each processor can compute its part of the r.h.s. of the discretized system, which is just Pu, completely independent of the other processors.

Table 4.2.1: Numbering of variables and equations.

no	var	name	equation on A1, Folie 9	name
1	u_1	u_x	x-momentum	P_1u
2	u_2	u_y	y-momentum	P_2u
3	u_3	p	constitutive equation	P_3u
4	u_4	ρ	mass conservation	P_4u
5	u_5	T	energy equation	P_5u
6	u_6	C_{O_2}	O_2 transport equation	P_6u

The first PDE, x-momentum on Addendum A1, Folie 9 is

$$P_1u = u_x + \frac{K_x}{\eta} \cdot \frac{\partial p}{\partial x} = 0.$$

This is entered in our notation as

$$p(i,1) = u(i,1) + K\_x * ux(i,3)/ns.$$

Here i denotes the (local) node number, in $p(i,1)$ the "1" denotes the first equation of the i-block and in $u(i,1)$ the "1" denotes the first solution component u_1, see Table 4.2.1. Similarly $\partial p/\partial x$ is denoted by $ux(i,3)$, which means the x-derivative of the solution component u_3. Below the row for $p(i,1)$ (and similarly below all other entering rows) is a row that is marked by "!" as comment and contains f, $f2$, fx, $f2x$, etc. These rows are used for the test problem that is explained below. The second equation is entered as $p(i,2)$ in a quite similar way. The third equation is the constitutive equation, see Table 4.2.1. It is

$$p - \rho T(R_{H_2O}(1-\gamma-C_{O_2}) + R_{N_2}\gamma + R_{O_2}C_{O_2}) = 0$$

which is entered as

$$p(i,3) = u(i,3) - u(i,4) * u(i,5) * (RR\_H2O * (1 - gamma - u(i,6)) + \\ + RR\_N2 * gamma + RR\_O2 * u(i,6)).$$

Now it should be clear, at least with the printout of FDEMU1, how the PDEs are entered in FDEM.

4.3 Entering the BCs into the program frame

In Fig. 3.4.1 are depicted the four boundaries of the computational domain. The BCs are entered into the program frame of the subroutine FDEMU2. Which equations are used for which variable at which boundary has been reported in Section 3.4. In FDEMU2 after "start of calculation" at first there is the part for the Jacobi tester which is used only if $ljac = true$. The proper calculation starts with the computation of 20 auxiliary variables $p1(i)$ to $p20(i)$ that cover all the functions of

the variables that appear in the BCs. These are computed in the first do-loop after the Jacobi tester. The do-loop runs from $i = 1$ to nb, where nb is the (local) number of boundary points for the actual call of FDEMU2. If we have 4 boundaries ($nexb = 4$) FDEMU2 is called 4 times. At the first call the variable $irand$ that denotes the boundary, has the value $irand = 1$, i.e. it is for boundary 1 or boundary I in Fig. 3.4.1. For the second call for boundary II the variable $irand = 2$ and similarly for boundaries 3 and 4 resp. III and IV. The value nb is usually different for each call.

The computation of all the auxiliary functions for all boundaries as it is programmed in FDEMU2 by our student programmer is not an economic way because for a certain boundary values are computed that are not needed for that boundary. This procedure is only justified by the fact that this demonstration program was executed only a few times. If it would be used many times, e.g. for parameter variations, one should compute only those values that are needed for the actual boundary. The auxiliary variables again are accompanied by the expressions in f, fx, $f2$, $f2x$ etc. for the test PDE and they are marked as comment by a "!".

The proper part of delivering the BCs now is quite simple and it starts with

! zum Kanal (to the channel)

$$
\begin{array}{l}
\text{if} \quad (irand == 1) \quad \text{then} \\
\quad \text{do} \quad i = 1,\ nb \\
\qquad p(i,1) = p14(i) \\
\qquad p(i,2) = p2(i) \\
\qquad \vdots
\end{array}
$$

Because of $(irand == 1)$ these are the BCs for boundary 1 or I. In $p(i,1)$ the "i" is the number of the i^{th} block of equation, the "1" denotes the first equation in this block of m equations for a system of m PDEs. Similarly $p(i,2)$ denotes the second equation in the i^{th} block etc., see the print-out of the code in A3.

The BCs for boundary 2 or II in Fig. 3.4.1 start with

! neben dem Kanal (oben) (besides the channel(at the top))

$$
\begin{array}{l}
\text{if} \quad (irand == 2) \quad \text{then} \\
\quad \text{do} \quad i = 1,\ nb \\
\qquad p(i,1) = p1(i) \\
\qquad p(i,2) = p10(i) \\
\qquad \vdots
\end{array}
$$

Thus the BCs for boundary 2 are entered into the program frame of FDEMU2. After the statements

if $(irand == 3)$ then
if $(irand == 4)$ then

the BCs for the boundaries 3 and 4 are entered in the same way.

4.4 Entering the Jacobians for the PDEs into the program frame

The next problem that we will discuss is the entering of the Jacobian matrices for the PDEs in the interior of the domain which is done by the subroutine FDEMU3. The Jacobian matrix $\partial Pu/\partial u_x$ is shown in equ. (2.4.6). If we have $m \times m$ elements in each Jacobian matrix for node i, e.g.

$$\left(\frac{\partial Pu}{\partial u}\right)_i = \left(\frac{\partial P_{i,iequ}}{\partial u_{i,icom}}\right) = pu(i),$$

where $iequ$ denotes the number of the equation in the block of m equations, e.g. $iequ = 2$ denotes the second equation, and $icom$ denotes the solution component, e.g. $icom = 3$ denotes the variable $u(i,3)$ in our notation. $pu(i)$ denotes the corresponding component in the terminology of FDEMU3. FDEMU3 is called for $iequ = 1$ to m and for each $iequ$ for $icom = 1$ to m, thus m^2 times. At each call the $pu(i)$ is stored by FDEMU3 in the right position of a corresponding array. In the same way the other Jacobians are treated:

$$\left(\frac{\partial Pu}{\partial u_x}\right)_i = \left(\frac{\partial P_{i,iequ}}{\partial u_{x,i,com}}\right) = pux(i),$$

$$\left(\frac{\partial Pu}{\partial u_{xx}}\right)_i = \left(\frac{\partial P_{i,iequ}}{\partial u_{xx,i,com}}\right) = puxx(i),$$

and similarly $puy(i)$, $puyy(i)$ etc.

In the printout Addendum A3 in FDEMU3 at

!*** start of calculation
do $i = 1, mv$
auxiliary variables for the Jacobians are computed. At
if $(iequ == 1)$ then
are computed all Jacobians for the first equation (of the block of m equations). At
if $(icom == 1)$ then
do $i = 1, mv$
$pu(i) = p1\_1(i)$
enddo
endif

are computed for all nodes i the $\partial P_{i,1}/\partial u_{i,1}$, where $p1\_1(i)$ is one of the precomputed auxiliary variables. Here it should be repeated that all the arrays for the Jacobians are pre-assigned with zeros and only nonzero elements must be entered. In a later part of the printout we have

if $(iequ == 4)$ then
if $(icom == 1)$ then
do $i = 1, mv$
$pux(i) = p4\_1x(i)$
$pu(i) = p4\_1(i)$
enddo
endif
$\vdots$

Here are entered $\partial P_{i,4}/\partial u_{x,i,1}$ and $\partial P_{i,4}/\partial u_{i,1}$. So by the m^2 repeated calls of FDEMU3 by the main program of FDEM all the nonzero elements of the Jacobian matrices for the PDEs in the interior of the domain are stored in a corresponding array of FDEM.

4.5 Entering the Jacobians for the BCs into the program frame

The Jacobians for the BCs are entered by the subroutine FDEMU4. If we have $nexb$ exterior boundaries, FDEMU4 is called $nexb \cdot m \cdot m$ times, namely for all boundaries $irand$, all equations $iequ$ and all solution components $icom$. In the printout Addendum A3 in FDEMU4 at "start of calculation" again auxiliary variables are computed. The computation of the elements of the Jacobian matrices starts at the comment lines.

```
!*******
!Rand 1 (boundary 1)
!*******
!Zum Kanal (to the channel)
```

```
if  (irand == 1)  then
  if  (iequ == 1)  then
    do  i = 1, nb
      pu(i) = p14_1(i)
    enddo
  endif
endif
```

This gives for boundary 1 $\partial P_{i,1}/\partial u_{i,1}$. The meaning of the remaining text of the program code of FDEMU4 should now be clear. At

```
if (irand == 2) then
```

the Jacobians for boundary 2 are entered etc.

4.6 Entering the coupling conditions (CCs) at dividing lines (DLs) into the program frame

The CCs for the DLs are entered by the subroutine FDEMU5. We assume that we have $ninb$ (n inner boundaries) DLs. For FDEMU5 we do not include a printout for an example, therefore we explain the entering here. In the list of formal parameters there is ib for the number of the actual DL, $iequ$ and $icom$ for equation and component as in the other subroutines described above, nk the number of components in the system (that we usually denote by m for a system of m PDEs), nt the number of nodes on the actual DL, and mlt that is equal to nt if there is no crossing of DLs, but has vector information if there are crossing DLs that e.g. generate quadruple nodes, this parameter mlt is not explained here. Like for the exterior boundaries FDEMU5 is called for each DL ib. However, in contrast to the exterior boundaries each node is "doubled", see Fig. 2.6.1, so that we need two CCs in each node. A DL separates two domains and the two domains (left and right, or upper and lower)

are denoted by "1" and "2". If we have e.g. for a DL that is parallel to the y-axis the conditions $u_{left} = u_{right}$, $u_{x,left} = u_{x,right}$, we have for DL with number 1 (first DL) the code

```
if   (ib == 1)   then   (for first DL)
  do   j = 1, nk   (for all components)
    do   i = 1, nt   (for all nodes of DL)
      p(i,j,1) = u(i,j,1) − u(i,j,2)
      p(i,j,2) = ux(i,j,1) − ux(i,j,2)
    enddo
  enddo
else if   (ib == 2)   then   (for second DL)
  ⋮
```

In the parentheses of the assignments the "1" and "2" in the last position denote the two sides of the DL.

The Jacobian matrices for the DLs are entered in the program frame of the subroutine FDEMU6. For the Jacobians we have now $ninb \cdot m^2$ calls for $ninb$ DLs and a system of m PDEs. Which equation is called and to which component is derived is the same type of information as for the Jacobians for exterior boundaries. For the example of CCs that is given above the j in $p(i,j,1)$ denotes the equation $iequ$ and the j in $u(i,j,1)$ denotes the component $icom$, so we have in this example entries only for $iequ = icom$. Therefore the code for the Jacobians is

```
if   (ib == 1)   then
  if   (iequ == icom)   then
    do   i = 1, nt
      pu(i,1,1) = 1.d0
      pu(i,2,1) = −1.d0
      pux(i,1,2) = 1.d0
      pux(i,2,2) = −1.d0
    enddo
  endif
else if   (ib == 2)   then
  ⋮
```

Here in the index parentheses of pu the second position denotes the "left" (1) or "right" (2) variable, the third position denotes the first (1) or second (2) coupling condition.

This completes the explanation how the PDEs, BCs, and CCs and their Jacobian matrices must be entered in FDEM by writing the corresponding Fortran code into prescribed program frames of the subroutines FDEMUi.

4.7 Test problem

In Section 2.10 we have discussed how we generate from our problem $Pu = 0$ a test problem $Pu - P\bar{u} = 0$, equ. (2.10.2), and we have explained this in more detail in the equs. (2.10.3) to (2.10.11).

In this example of fuel cell simulation we have the 6 variables u_1 to u_6, see Table 4.2.1. For u_1 we use the test function f, for u_2 we use f_2, for u_3 we use f_3 etc. The derivatives of the test functions are denoted fx, $f2x$, $f3x$, ..., fy, $f2y$, $f3y$, ..., fxx, $f2xx$, $f3xx$, etc. For this 2-D case they are functions of x_i, y_i for node i, and because such test functions might be used also for unsteady problems, they depend also on t (although the fuel cell problem is steady). In the print-out of the subroutine FDEMU1 for entering the PDEs of interior nodes we have the code segment

$$\begin{aligned}&\text{do} \quad i = 1, mv\\ &\quad p(i,1) = u(i,1) + K\_x * ux(i,3)/ns\\ &\qquad - (f(x(i),y(i),t) + K\_x * f3x(x(i),y(i),t)/ns\end{aligned}$$

Here we have in the first assignment row the $P_1 u$ and in the second row the $P_1 \bar{u}$, which is in this case changed to comment by the indicator "!", so we would solve the physical problem. If we want to solve the test problem, we only have to erase the "!". It is easy to see how one gets from Pu the $P\bar{u}$. This can be seen also in the subroutine FDEMU2 for the BCs. There the $P\bar{u}$ terms are already included in the auxiliary variables so that at the assignments for Pu there is not visible if the physical or the test problem is solved. It should be recalled that the Jacobian matrices are not changed by $P\bar{u}$, because this is an explicit function of x, y.

The choice of the test functions f, $f2$, $f3$, ... is made by the choice of the corresponding subroutines at the binding of the problem. Usually we select for the f's polynomials of order $2, 4, 6, 8$ or the sugar-loaf function as presented in the examples of Section 2.10.

We recommend never to use FDEM without at first testing the PDEs by a test problem. The test problem not only tests if the PDEs are entered correctly, but it also shows you the basic properties of the solution. Often for technical problems it is difficult to see what are the correct BCs so that the problem is well-posed. Then the test PDE shows immediately when there is no solution.

4.8 The Jacobi tester

The basic idea to check the Jacobi matrices by a difference quotient has been explained in the context of equ. (2.10.12). In FDEM there is an entry parameter $ljac$. If $ljac = true$ the Jacobi tester is switched on, if $ljac = false$ it is switched off. If it is switched on the testing of the Jacobi matrices is running integrated in the solution process which is stopped if there is detected an assumed error in the Jacobis. Here we recall that the Jacobi tester gives the exact value of the derivative only for linear functions of u, else it gives the derivative only up to an error $O(\varepsilon)$.

We want to explain how the Jacobi tester works. The PDEs for a system of m PDEs are:

$$P = \begin{pmatrix} P_1 \begin{pmatrix} u_1 & u_{1,x} & u_{1,y} & u_{1,yy} \\ \vdots & \vdots & \vdots & \vdots \\ u_m & u_{m,x} & u_{m,y} & u_{m,yy} \end{pmatrix} \\ \vdots \\ P_m \begin{pmatrix} u_1 & u_{1,x} & u_{1,y} & u_{1,yy} \\ \vdots & \vdots & \vdots & \vdots \\ u_m & u_{m,x} & u_{m,y} & u_{m,yy} \end{pmatrix} \end{pmatrix}. \tag{4.8.1}$$

We have e.g. the following Jacobis:

$$\frac{\partial P}{\partial u} = \begin{pmatrix} \frac{\partial P_1}{\partial u_1}, \ldots, \frac{\partial P_1}{\partial u_m} \\ \vdots \\ \frac{\partial P_m}{\partial u_1}, \ldots, \frac{\partial P_m}{\partial u_m} \end{pmatrix}, \tag{4.8.2}$$

$$\frac{\partial P}{\partial u_x} = \begin{pmatrix} \frac{\partial P_1}{\partial u_{1,x}}, \ldots, \frac{\partial P_1}{\partial u_{m,x}} \\ \vdots \\ \frac{\partial P_m}{\partial u_{1,x}}, \ldots, \frac{\partial P_m}{\partial u_{m,x}} \end{pmatrix} \tag{4.8.3}$$

and similarly $\partial P/\partial u_y, \ldots, \partial P/\partial u_{yy}$. If we change according to (2.10.12) in (4.8.1) $u_1 \to u_1 + \varepsilon$ we check the first column of (4.8.2), if we change $u_m \to u_m + \varepsilon$, we check the last column of (4.8.2). Similarly, if we change $u_{1,x} \to u_{1,x} + \varepsilon$ we check the first column of (4.8.3) etc.

In the printout of the subroutine FDEMU1 in the Addendum A3 we have after "! $* * * *$ start of calculation" a code segment with $fex(\ldots)$. The elements of the vector fex are the ε's for the checking of the corresponding Jacobians as described above. In this code "it" is the number of components in the system of m PDEs which is in this case also equal to the value of nk in the j-loop. If $ljac = true$ the Jacobi tester calls $6 \times m$ times FDEMU1 with the appropriate component of $fex(\ldots) = \varepsilon$ and the other components zero. The basic call of FDEMU1 for the solution of the PDEs is executed with all components of $fex(\ldots)$ equal to zero. So for $ljac = true$ the Jacobi tester acts as control program that is integrated into the regular solution algorithm.

In Section 4.3 we have presented how the BCs are entered by the subroutine FDEMU2 and in Section 4.6 how the CCs, the internal BCs, are entered. The code of these subroutines starts like FDEMU1 with the $fex(\ldots)$ assignments and thus the Jacobians for the BCs and the CCs are checked in the same way as for the PDEs in FDEMU1. Thus the Jacobi tester is a very sophisticated program part that is elegantly interleaved with the solution of the PDEs itself.

4.9 Remarks to LINSOL

The linear solver LINSOL [7] has been developed initially together with the FIDISOL program package, see [2], and since that time has been continuously improved and extended, see the references given in [7]. LINSOL was originally a pure iterative solver with several generalized CG (conjugate gradient) methods. The essential improvement was the implementation of an (I)LU (incomplete LU factorization) preconditioner, together with (effectively) two bandwidth optimizers. All these codes were optimized for sparse matrices on distributed memory parallel computers. Unfortunately, it turned out that all the hard industrial problems that are mentioned in this report have for the needed large number of unknowns so badly conditioned matrices that all CG solvers, even the most robust ones, do not converge. These problems need full LU, all attempts with ILU with different types of dropping strategies failed. However, full LU for sparse matrices means fill-in between the outermost diagonals for our algorithm, therefore a bandwidth optimizer is essential. Full LU increases for large problems, above all in 3-D, considerably the needed storage and computation time. In 3-D problems

the storage for the factors L and U ultimately limit the size of a problem that can be solved on a given computer.

FDEM is a black-box solver. In the PDE operator (2.4.1) therefore appear all possible derivatives, e.g. u_{xy}. However, if in the actual system of m PDEs u_{xy} does not occur, the corresponding Jacobian $m \times m$ matrix $\partial P/\partial u_{xy}$ has only zero elements. In the process of the generation of the matrix Q_d of the linear system for the computation of the Newton correction, see (2.4.10), the elements of the difference formulas for u_{xy} in this case are multiplied by zeros and added to the corresponding position in Q_d. If there is no contribution of other terms of the PDEs to this position, there remains a zero. These zeros are denoted as "computed zeros". The matrix Q_d is handed to LINSOL (in reality each processor hands its part of the matrix to LINSOL by its call of LINSOL) with the computed zeros and in a first step LINSOL takes out of the matrix these zeros.

For the full LU preconditioning a "Gauss factor" tells LINSOL, that the user expects a storage space for $L + U$ that is "Gauss factor" times the storage of Q_d (including the computed zeros). If the selected "Gauss factor" is too small, the LU factorization stops when the storage limit is attained and tells to which row of Q_d it has proceeded up to then. We usually needed several iterations to find out the appropriate "Gauss factor" for a new problem. Below in the list of parameters of FDEM are also mentioned the parameters for the call of LINSOL. There they are explained only very shortly. For a detailed understanding the user must consult the documentation of LINSOL [7]. It should be mentioned that in our examples of industrial problems LINSOL needed 80 to 95% of the total computation time so that the efficiency of LINSOL is decisive for the solution of the PDEs.

4.10 Computational parameters

As FDEM is not "frozen" but still in continuous development, the list of entry parameters may still change. Therefore it is advisable to check the actual situation. Above all the parameters for the selection of the nodes for the difference formulas may change because some internal parameters may become external parameters and thus become directly accessible to the user. The situation that is described below is that of September 2004.

The computational parameters are entered by at least two mandatory and 6 optional files. The mandatory files are the basic parameters of FDEM and the parameters for the use of LINSOL.

Fig. 4.10.1 shows the structure of the basic input file *fdem_input* with example parameter values. Here we explain only those parameters whose meaning is not directly visible from the comment in Fig. 4.10.1. The first 3 parameters (in the text of nb_max "bound." means "boundary") are used for the declaration of the corresponding arrays if grid refinement is used, "load" is $.true.$ if a computation is continued from a previous computation, "store" is $.true.$ if at the end of a computation all data are saved so that the computation can be continued. "mnls" gives the number of simultaneously solved systems $M \cdot A = I$ (2.2.6) for the computation of the polynomial coefficients $a_{j,i}$ in (2.2.4) for the difference formulas. Each system is small and recursive so that vectorization is inefficient. If $mnls$ systems (in the example $mnls = 100$) are solved simultaneously, the computation is fully vectorizable, see Section 13.3 in [4].

"idoku" is a mixture of FDEM and LINSOL parameters:

for FDEM: idoku ≥ 0: only processor 1 prints information,
idoku < 0: all processors print information.

for LINSOL: idoku$= 500$ means that in LINSOL every 500^{th} iteration step the LINSOL defect is printed.

```
13000    *     i   *  n_max     -> max. number of grid points on one processor
26000    *     i   *  ne_max    -> max. number of elements on one processor
1000     *     i   *  nb_max    -> max. number of bound. points on one processor
14       *     i   *  nexb      -> number of external boundaries
2        *     i   *  ninb      -> number of internal boundaries
0        *     i   *  nslb      -> number of sliding boundaries
10       *     i   *  lv_max    -> max. number of refinements per element
.false.  *     l   *  load      -> load an old computation
.false.  *     l   *  store     -> save computation at the end
2        *     i   *  dim       -> dimension of the problem
6        *     i   *  nk        -> number of equations and sol. components
100      *     i   *  mnls      -> number of simultaneous solved small systems MA=I
500      *     i   *  idoku     -> output control parameter
5        *     i   *  initsol   -> initial solution parameter
0.1      *    dp   *  is_fac    -> error of initial solution (only if initsol = 3)
0        *     i   *  maxit     -> max. number of Newton iteration steps
.false.  *     l   *  mref      -> mesh refinement on/off
.false.  *     l   *  ordctr    -> order control on/off
2        *     i   *  pd        -> consistency order
2        *     i   *  pd2a2     -> order surplus for order 2
2        *     i   *  pd2a4     -> order surplus for order 4
2        *     i   *  pd2a6     -> order surplus for order 6
226      *     i   *  isort     -> kind of sorting nodes and pivot search
300      *     i   *  nle_3     -> max. number of nodes to collect for central node
1.0d-2   *    dp   *  tol       -> requested tolerance
1.d0     *    dp   *  s_grid    -> safety factor for mesh refinement
1.d0     *    dp   *  s_24      -> safety factor for order control
0.01d0   *    dp   *  s_46      -> safety factor for order control
1.d-1    *    dp   *  eps_piv   -> reference pivot element for pivot search
1.d-1    *    dp   *  alpha     -> reference pivot element for pivot search
1        *     i   *  cycle     -> number of cycles for mesh ref. and order control
1.d+6    *    dp   *  ten       -> ten-factor for Newton iteration
.true.   *     l   *  ell       -> elliptic/parabolic problem?
.true.   *     l   *  inso      -> initial solution given?
.false.  *     l   *  gridsave  -> save refinement meshes?
4.d0     *    dp   *  s_o       -> safety factor for overlap
0.1d0    *    dp   *  redfac    -> reduce factor for Newton it. (only lean version)
.false.  *     l   *  llean     -> compute solution with lean version?
.false.  *     l   *  ljac      -> test Jacobian matrices?
.false.  *     l   *  ltest     -> only statistics for difference/error formulas
.true.   *     l   *  lphys     -> physical problem?
.false.  *     l   *  lrefg     -> refine whole subdomain?
ideas    *     c   *  interf    -> mesh generator (not used)
fdem     *     c   *  collect   -> way to collect points (not used)
stdout   *     c   *  output file name
```

Figure 4.10.1: Example for the use of the entry parameter file *fdem_input*.

"initsol" determines the initial (starting) solution. The value has the following meaning:

1: start with solution 1.0 for all nodes and all components,
2: start with the exact solution for test PDE,
3: start with the disturbed solution for test PDE, the disturbance factor is is_fac, e.g. $is\_fac = 0.1$ for 10% disturbance,
4: start with zero solution,
5: special value.

"maxit" is the maximal number of Newton iteration steps. The special value 0 means that there is no limit.

"pd2a2": $pd2a = \triangle q - 2$, where $\triangle q$ is explained in Section 2.2. If we have consistency order $q = 2$ we usually have $\triangle q = 4$ which means that we selected nodes that are sufficient for the order $2 + 4 = 6$. Because for the order 2 we need error formulas of order $q + 2 = 2 + 2 = 4$ and we want to have sufficient nodes of two orders higher $= 6$, we have $pd2a = 2$. So $\triangle q$ gives the surplus to q, $pd2a$ the surplus to the error formula order $q + 2$. Therefore we have $pd2a = \triangle q - 2$. In the parameter list we can give different surplus values for the orders $2, 4, 6$ by $pd2a2$, $pd2a4$, $pd2a6$.

"isort" is a key parameter that contains three keys in the three positions. These keys determine for the selection of the nodes the arrangement, the sorting and the search as described in the text after equ. (2.2.16).
For the arrangement we have the 3 possibilities:

arrangement:	1.	2.	3.
key value:	1	2	3

for the sorting we have the 3 possibilities

sorting:	$a.$	$b.$	$c.$
key value:	1	3	2

for the search we have the 2 possibilities

search:	1.	2.
key value:	6	7

Therefore the value 226 for $isort$ in Fig. 4.10.1 means arrangement 2., sorting c., search 1.

"nle_3" is a guess for the declaration of some arrays for this computation of the difference formulas. If the value is too small the arrays are deallocated and reallocated with a larger value. "tol" is the requested relative global tolerance in (2.5.12). "s_grid" is the safety factor for mesh refinement in (2.5.18). "s_24" and "s_46" are the safety factors for order control that are used in (2.5.16), (2.5.17) and there are named $f_{2\leftrightarrow 4}$, $f_{4\leftrightarrow 6}$.

"eps_piv" is the value ε_{pivot} used in (2.2.11) and (2.2.16) for the generation of the difference formulas, and likewise "alpha" is the value used in (2.2.16). "cycle" is the max. number of cycles for mesh refinement and order control if the accuracy requirement by tol is not attained in less cycles. "ten" is a parameter that controls the stopping of the Newton iteration. In (2.5.4) there is the safety factor 0.1. This value now has been generalized to $1/ten$. If we have $ten = 10$, the safety factor is 0.1, if we have $ten = 10^6$, the safety factor is 10^{-6}.

If "ell" is $.true.$ the problem is elliptic, if it is $.false.$ it is parabolic. If "inso" is $.true.$ an initial solution is selected by an interpolation from the boundary values.

"s_o" is the safety factor for overlap that is named $a_{overlap}$ in (2.8.1). "redfac" is the reduction factor for the residual control in the Newton iteration for the lean version, because there we do not compute error estimates that are used in (2.5.4) to stop the Newton iteration. In the lean version the Newton iteration is stopped if $||(Pu)_d|| < redfac||(Pu)_d||_0$, where $||(Pu)_d||_0$ is the initial residual norm. If "lean" is $.true.$ the solution is computed with the lean version, without error estimate, else the standard version is used.

If "ljac" is $.true.$ the Jacobi tester is switched on, see Section 4.8, else it is switched off. If "ltest" is $.true.$ then for the test polynomial the maximal difference between the exact and the numerical derivative is computed which is the exact error for linear term in u and $O(\varepsilon)$ for non-linear term and compared to the estimated error. The result is delivered in the form of a table. Further a table is printed with the information about the size of the cofficients of the difference and error formulas.

If "lphys" is $.true.$ the physical problem is solved, if it is $.false.$ the test PDEs are solved. "lrefg" $.true.$ means for the mesh refinement process: if in a sudomain at least one node is refinement node, the whole subdomain is refined. If "lrefg" is $.false.$ individual refinement is made. This closes the explanation of the mandatory parameter file *fdem_input*.

The other mandatory file is the *lsol_input* file that is shown for an example in Fig. 4.10.2. This file controls the use of LINSOL and consequently most of the parameters correspond to the param-

```
7         *     i   *  ms          -> method selection
75        *     i   *  msprec      -> method of preconditioning
1         *     i   *  msnorm      -> normalization method
100000    *     i   *  maxmvm      -> max. number of matrix-vector-multiplications
0         *     i   *  misc
11001     *     i   *  optim
1         *     i   *  bwoalg      -> bandwidth optimizer algorithm
.true.    *     l   *  lilu         -> Newton iteration with (l)LU?
0.d0      *    dp   *  threshold1
0.d0      *    dp   *  threshold2
0.d0      *    dp   *  ldrop
-300.d0   *    dp   *  gaussfac
0.d0      *    dp   *  memfac
0.d0      *    dp   *  epsmat
1.d0      *    dp   *  pivot_threshold
1         *     i   *  m for GPBiCG(m,l)
4         *     i   *  l for GPBiCG(m,l)
1.d0      *    dp   *  LINSOL_MATRIX_FACTOR
1.d0      *    dp   *  LINSOL_INDEX_FACTOR
1.d0      *    dp   *  LINSOL_INFO_FACTOR
```

Figure 4.10.2: Example for the use of the entry parameter file *lsol_input*.

eters that are presented in the User's guide of LINSOL, see Section 4.9 and above all the section "Parameters" in the User's Guide. "memfac" is not used. "epsmat" exists only from Version 1.1 of LINSOL on and has the following meaning: before the treatment of the matrix all elements with absolute value less or equal to $epsmat$ are eliminated. This serves above all to eliminate the "computed

zeros". "pivot_threshold" (also from Version 1.1 on) has the following meaning:

pivot_threshold $= 1.0$:	takes max. absol. value of a row as pivot,
$= 0.0$:	no pivoting,
$= 0.5$:	accepts pivot if it is ≤ 0.5 times max. pivot (analoguously other values $0 <$ value < 1).

The last three parameters in *lsol_input*, LINSOL_MATRIX_FACTOR, LINSOL_INDEX_FACTOR and LINSOL_INFO_FACTOR, correspond to the parameters MATRIX_FACTOR etc. described in the User's guide of LINSOL and are needed for parallel computation for the program that calls LINSOL.

Finally there is the parameter "lilu". This is a FDEM parameter. If it is $.true.$ the solution of the linear system in the Newton iteration takes place with (I)LU, if it is $.false.$ without (I)LU. This ends the discussion of the mandatory input file *lsol_input*.

There are further optional parameter input files. The file *bd_order* gives the rules for the overwriting of boundaries, e.g. in corners, as described at the end of Section 4.1. An example input is

2	2
4	2
1	1
2	2
3	1
$\vdots$	$\vdots$

This list must be read from below: boundary 3 of type 1 overwrites bd. 2 of type 2 that overwrites bd. 1 of type 1 that overwrites bd. 4 of type 2 that overwrites bd. 2 of type 2. Note that the numbering for each of the 3 types of boundaries (external, DLs, SDLs) is from 1 to max. number of boundaries of that type. If this file does not exist the standard rule holds that boundaries with higher number overwrite boundaries with lower number.

With the input file *fdem_epsp* for a domain with nd subdomains, for each subdomain different values for ε_{pivot} for the generation of the difference formulas and also separately for the error formulas and different values of α can be selected, where ε_{pivot} and α are the parameters in (2.2.16). An example file for $nd = 3$ domains would look like this:

1.E-2	** epspiv formula domain 1
1.E-2	** epspiv error domain 1
1.E-2	** epspiv formula domain 2
1.E-2	** epspiv error domain 2
1.E-2	** epspiv formula domain 3
1.E-2	** epspiv error domain 3
3.E-2	** alpha domain 1
3.E-2	** alpha domain 2
3.E-2	** alpha domain 3

If this file does not exist, the values of *fdem_input* are used everywhere.

The input file *couple_domain* is needed if there are dividing lines. The solutions of the different subdomains are coupled by CCs (coupling conditions). For a dividing line in 2-D there result from one geometrical node two logical nodes that belong each uniquely to one of the coupled domains. Consequently there must be two equations (CCs) for each node. Such CCs have two sides: side "one" and side "two". In the file *couple_domain* is determined, which domain is side "one" and side "two" in the first equation and in the second equation. This type of distinguishing information results from the fact that FDEM is a black-box solver with flexible general properties. Below is an example of *couple_domain*:

```
2     0  0  0  0  0  0  0  0  0  0  0
1 2   ** domain 1 is "one", domain 2 is "two"
2 1   ** domain 2 is "one", domain 1 is "two"
3 2   ** domain 3 is "one", domain 2 is "two"
2 3   ** domain 2 is "one", domain 3 is "two"
```

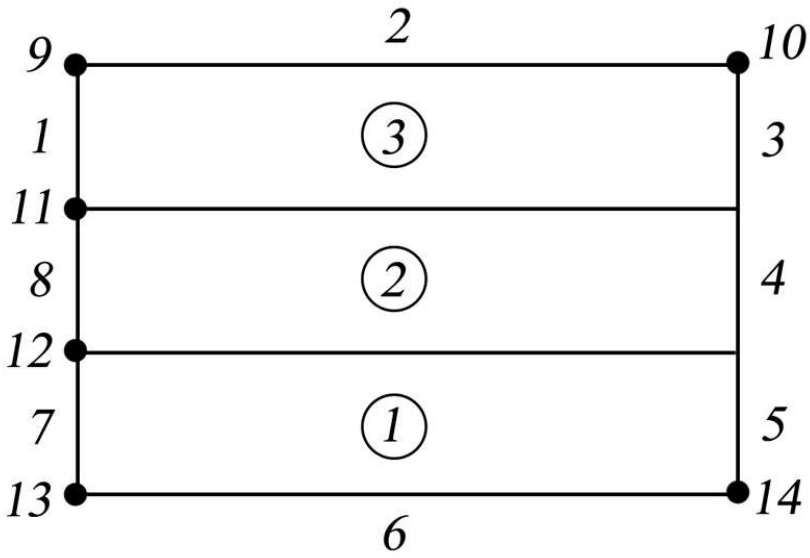


Figure 4.10.3: Domain with 3 subdomains and 14 external boundaries (one node may be a boundary).

Because in the first place of the first row is a "2" follow 2 double rows for 2 dividing lines with domain numbers "one" and "two" for equation 1 and equation 2. If e.g. in the second position of the first row, where triple nodes or lines are noted, would be 1 (instead of 0) for one triple node, then there could be e.g. the triple row:

```
1 2   ** domain 1 is “one”, domain 2 is “two”
2 3   ** domain 2 is “one”, domain 3 is “two”
3 1   ** domain 3 is “one”, domain 1 is “two”
```

for the 3 CCs of the triple node. In the third position of the first row is noted the number of quadruple nodes with 4 CCs etc. This is a very sophisticated allocation of domains to equations in the CCs.

Fig. 4.10.3 shows a domain with 3 subdomains and 14 external boundaries, note that a single node, e.g. a corner, may have its own BC and so it is an own “boundary”. There are two internal boundaries, i.e. DLs. Because the right end nodes of these DLs are not defined as separate boundaries, they belong by definition to the DLs. In the file *extbound* is defined which boundary belongs to which domain. For the example of Fig. 4.10.3 the file *extbound* looks like this:

```
3 **   pos.1,    i.e. bd1,    belongs to domain 3
3 **   pos.2,    i.e. bd2,    belongs to domain 3
3 **   pos.3,    i.e. bd3,    belongs to domain 3
2 **   pos.4,    i.e. bd4,    belongs to domain 2
1 **   pos.5,    i.e. bd5,    belongs to domain 1
1 **   pos.6,    i.e. bd6,    belongs to domain 1
1 **   pos.7,    i.e. bd7,    belongs to domain 1
2 **   pos.8,    i.e. bd8,    belongs to domain 2
3 **   pos.9,    i.e. bd9,    belongs to domain 3
3 **   pos.10,   i.e. bd10,   belongs to domain 3
3 **   pos.11,   i.e. bd11,   belongs to domain 3
1 **   pos.12,   i.e. bd12,   belongs to domain 1
1 **   pos.13,   i.e. bd13,   belongs to domain 1
1 **   pos.14,   i.e. bd14,   belongs to domain 1
```

A special remark must be made to boundaries 11 and 12. These are single geometrical nodes that lie on a DL. Therefore from node 11 result two logical nodes, one belongs to domain 3 and one to domain 2. The notation in *extbound* that $bd11$ belongs to domain 3 means that there are given BCs for the upper logical node and thus the lower logical node belongs to $bd8$ and has the BCs of $bd8$. Similarly the resulting upper logical node of $bd12$ belongs to $bd8$ and for the lower logical node of $bd12$ there are given BCs because in *extbound* is noted that $bd12$ belongs to domain 1. Quite naturally the file *extbound* is needed only if there are DLs. Without DLs all boundaries belong to the single domain.

If there are SDLs (sliding DLs) there are two further input files *sdl_assoc* and *slibound* that are not explained here.

4.11 Licensing conditions

At the time of writing this report the future of the FDEM program package (60 000 lines of Fortran code) is not finally fixed. The University of Karlsruhe does not intend to maintain the program package for a long time. Therefore we will install the code at the “Institut für Wissenschaftliches

Rechnen" (IWR, Institute for Scientific Computing) at the Forschungszentrum Karlsruhe where it will be maintained.

Presently we are looking for partners who have problems that cannot be (efficiently) solved by their standard software. In common research projects we apply FDEM to these problems. Some examples have been presented in this report. FDEM is a general purpose black-box solver. In such a cooperation we design from this base code a special code for the problem of our research partner. Basically the partner then can install the code on his computer, but this needs a corresponding training for his staff. Predominantly we solve the problem for the partner on the computers that are available to us. When the partner himself cannot deliver the PDEs a third partner will be needed that cares for the PDEs that we then solve.

It should be mentioned that the numerical solution of hard technical problems needs often exceptional numerical experience that cannot be replaced by the pure application of even the best code. Therefore we consider it to be the most efficient way that we solve the problems for our partners that then can profit directly from our experience.

References

[1] W. Schönauer, K. Raith, G. Glotz, The SLDGL program package for the selfadaptive solution of nonlinear systems of elliptic and parabolic PDEs, in Advances in Computer Methods for Partial Differential Equation-IV, edited by R. Vichnevetsky and R. S. Stepleman, IMACS, New Brunswick, 1981, pp. 117-125.

[2] W. Schönauer, Scientific Computing on Vector Computers, North-Holland, Amsterdam, 1987.

[3] M. Schmauder, W. Schönauer, CADSOL-A fully vectorized black box solver for 2-D and 3-D partial differential equations, in R. Vichnevetsky, D. Knight, G. Richter (Eds.), Advances in Computer Methods for Partial Differential Equations-VII, IMACS, New Brunswick, 1992, pp. 639-645.

[4] W. Schönauer, Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers, selfedition Willi Schönauer, Karlsruhe, Germany, 2000, ISBN 3-00-005484-7, see
http://www.rz.uni-karlsruhe.de/∼ rx03/book/

[5] W. Schönauer, T.Adolph, How We solve PDEs, J. of Computational and Applied Mathematics 131, 2001, pp. 473-492.

[6] W. Schönauer, T. Adolph, FDEM: How we make the FDM more flexible than the FEM, J. of Computational and Applied Mathematics 158, Issue 1, 2003, pp. 157-167.

[7] LINSOL, see http://www.rz.uni-karlsruhe.de/rd/linsol.php
or enter LINSOL in the search windows of the University of Karlsruhe (http://www.uni-karlsruhe.de/).

[8] H. Häfner, W. Schönauer, The Integration of different variants of the (I)LU algorithm in the LINSOL program package, Applied Numerical Mathematics 41 (2002), pp. 39-59.

[9] D. Zundel, W. Schönauer, A fast “parallelized” single pass bandwidth optimizer for sparse matrices, accessible via [7], documentation.

[10] M. H. Gutknecht, Variants of BiCGStab for matrices with complex spectrum, SIAM J. Sci. Comput. 14 (1993), pp. 1020-1033.

[11] H. A. van der Vorst, Iterative Krylov Methods for Large Linear Systems, Cambridge University Press, Cambridge, UK, 2003.

[12] H. Leser, C. Wizemann, M. Vulkan, Abschlussbericht Weiterentwicklung und Anwendung des FDEM (Finite Difference Element Method) Programmpakets zur Lösung von partiellen Differentialgleichungen, Institut für Umformtechnik, Universität Stuttgart, 18. Februar 2004. This report of the part of the IFU in the project is available in hard copy and electronic form from the TIB Hannover, the library of the Universität Hannover. Search for `Leser, Wizemann` in the catalogue.
Here is the way to get the electronic document:

- enter in a search engine: `tib ub hannover`
- click: Technische Informationsbibliothek Universitätsbibliothek Hannover
- click: Kataloge
- click: Katalog (OPAC) der TIB/UB Hannover
- enter on top in search area: `Leser Wizemann`
- click: suchen
- click: title for electronic document (not book)
- click on bottom on link for .pdf document
This report is also available at
http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/fdem-IFU-Stuttgart.pdf .

[13] S. Timoshenko, J. N. Goodier, Theory of Elasticity, McGraw-Hill Book Company, New York, 1951.

[14] H. Schlichting, Grenzschicht-Theorie, G. Braun, Karlsruhe 1965.

Remark 1: The present report will also be available in the same form at the same place like [12]. It will be available also at
http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/fdem.pdf .

Remark 2: Presently we are writing a report on the application of FDEM to PEMFCs and SOFCs (fuel cells). This report will also be available in electronic form from the TIB/UB Hannover. It is available also at
http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/fuelcells.pdf .

Remark 3: Presently a doctoral thesis is in preparation at the University of Karlsruhe: T. Adolph, The parallelization of the mesh refinement algorithm in the Finite Difference Element Method program package (2005).

Addendum A1

Slides of Freudenberg ForschungsDienste KG

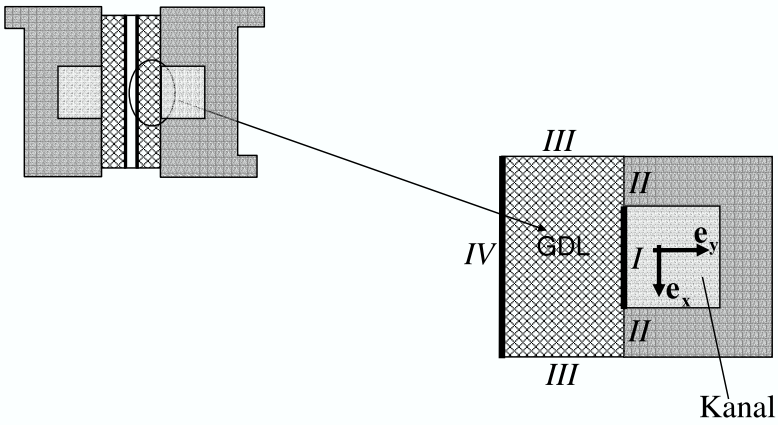
PDEs and Boundary Conditions for the Cathode Part of a PEM Fuel Cell

(in German)

Freudenberg
ForschungsDienste KG

Freudenberg

Berechnung der Kathodenseite einer Test-BSZ



CAE / Strömungsberechnung und Rheologie

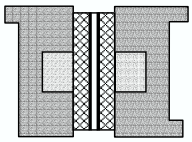
Folie 1

Freudenberg
ForschungsDienste KG

Freudenberg

Berechnung der Kathodenseite einer Test-BSZ

- Zielstellung der Berechnung:
  - Konzentrationsverteilung des Wasserdampfs in der GDL
  - Bereiche möglicher Kondensation erkennen

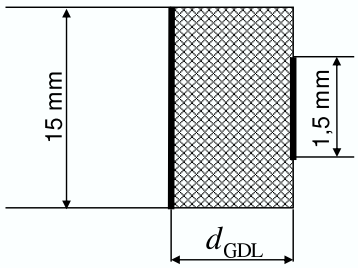


- Weg:
  - zunächst 2-dimensionales Modell der FFD
  - Simulation mit der FDE-Methode (Finite Difference Element Method) durch Prof. Schönauer Universität Karlsruhe

CAE / Strömungsberechnung und Rheologie

Folie 2

Geometrie der Test-BSZ



- Dicke der GDL d_{GDL} variiert je nach Verpressung und bestimmt die Permeabilität der GDL. Werte zwischen 140µm und 240 µm.
  Kanallänge (Dimension in z-Richtung) ist 310mm.

Modellierung der Kathode

- Zweidimensionales Modell (Annahme: Gradienten in Kanalrichtung werden vernachlässigt)
- stationär
- Gemisch idealer Gase (Sauerstoff, Stickstoff und Wasserdampf)
- Membran und Katalysatorschicht als Randbedingungen modelliert, werden nicht aufgelöst
- Freie Kanalströmung wird nicht mitberechnet sondern als Randbedingung modelliert
- Polarisationskurve wird nicht ermittelt
- Entstehendes Wasser in GDL nur gasförmig (zunächst keine Kondensation)

Freudenberg
ForschungsDienste KG

Freudenberg

Impulsbilanz (Darcy-Gesetz)

$$\mathbf{u} = -\frac{\mathbf{K}}{\eta}\nabla p$$

- Gesamtdruck p
- massengemittelte Geschwindigkeit $\mathbf{u}$
- dynamische Viskosität des Fluids η
- Permeabilität des porösen Mediums $\mathbf{K}$
  kann allgemein auch richtungsabhängig (tensorielle Größe) und Feldfunktion sein

CAE / Strömungsberechnung und Rheologie Folie 5

Freudenberg
ForschungsDienste KG

Freudenberg

O_2-Transport

$$\rho\,\mathbf{u}\cdot\nabla C_{O_2} = \nabla\cdot\left(\rho\,\mathbf{D}\cdot\nabla C_{O_2}\right)$$

- Massenkonzentration des Sauerstoffs C_{O_2}
- Dispersionstensor $\mathbf{D}$ kann allgemein auch richtungsabhängig (tensorielle Größe) und Feldfunktion sein
- Gesamtdichte ρ von Sauerstoff und Wasserdampf

Bilanz der Masse

$$\nabla\cdot\left(\rho\,\mathbf{u}\right) = 0$$

CAE / Strömungsberechnung und Rheologie Folie 6

Energiebilanz

$$\rho\, c_{\mathrm{p}}\, \mathbf{u} \cdot \nabla T = \nabla \cdot \left(\boldsymbol{\lambda} \cdot \nabla T \right)$$

- Temperatur T
- Effektive Wärmeleitfähigkeit $\boldsymbol{\lambda}$ der GDL kann allgemein auch richtungsabhängig (tensorielle Größe) und Feldfunktion sein
- spezifische Wärmekapazität c_{p} von Gemisch aus Sauerstoff, Stickstoff und Wasserdampf

Fehlende Gleichung zur Lösbarkeit des Systems: Konstitutive Gleichung (Gemisch idealer Gase)

$$p = \rho T \left(R_{\mathrm{H_2O}} \left(1 - \gamma - C_{\mathrm{O_2}} \right) + R_{\mathrm{N_2}} \gamma + R_{\mathrm{O_2}} C_{\mathrm{O_2}} \right) = \mathrm{fn}\left(\rho, T, C_{\mathrm{O_2}} \right)$$

- aus $p = p_{\mathrm{N_2}} + p_{\mathrm{O_2}} + p_{\mathrm{H_2O}}$ und $C_{\mathrm{O_2}} + C_{\mathrm{H_2O}} + C_{\mathrm{N_2}} = 1$
- Massenkonzentration des Stickstoffs im Querschnitt als konstant angenommen: $C_{\mathrm{N_2}} = \mathrm{const.} = \gamma_{\mathrm{Kanal}} = \gamma$
- individuelle Gaskonstanten $R_{\mathrm{H_2O}}, R_{\mathrm{N_2}}, R_{\mathrm{O_2}}$ mit $R_{\mathrm{i}} = \dfrac{\mathrm{R}}{M_{\mathrm{i}}}$ und $\mathrm{R}, M_{\mathrm{i}}$ universelle Gaskonstante, Molmasse Komponente i

Freudenberg
ForschungsDienste KG

Freudenberg

Zusammenfassung der Gleichungen im x,y-Hauptachsensystem

- Impuls $u_x = -\frac{K_x}{\eta}\frac{\partial p}{\partial x} \qquad u_y = -\frac{K_y}{\eta}\frac{\partial p}{\partial y}$
- Masse $\frac{\partial(\rho u_x)}{\partial x} + \frac{\partial(\rho u_y)}{\partial y} = 0$
- O_2-Transport $\rho\left(u_x \frac{\partial C_{O_2}}{\partial x} + u_y \frac{\partial C_{O_2}}{\partial y}\right) = \frac{\partial}{\partial x}\left(D_x \rho \frac{\partial C_{O_2}}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_y \rho \frac{\partial C_{O_2}}{\partial y}\right)$
- Energie $\rho\, c_p\left(u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y}\right) = \frac{\partial}{\partial x}\left(\lambda_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\lambda_y \frac{\partial T}{\partial y}\right)$
- Konstitutive Gleichung $p = \rho T\left(R_{H_2O}\left(1-\gamma-C_{O_2}\right) + R_{N_2}\gamma + R_{O_2}C_{O_2}\right)$
- 6 PDEs für die 6 Unbekannten $u_x, u_y, p, \rho, T, C_{O_2}$

CAE / Strömungsberechnung und Rheologie Folie 9

Freudenberg
ForschungsDienste KG

Freudenberg

Gesuchte Größen

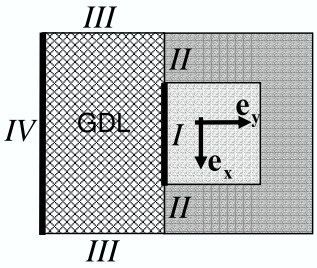
	Symbol	Einheit
Geschwindigkeit x-Komponente	u_x	m/s
Geschwindigkeit y-Komponente	u_y	m/s
Gesamtdruck	p	Pa
Gesamtdichte	ρ	kg/m³
Temperatur	T	K
Massenkonzentration des Sauerstoffs	C_{O_2}	-

CAE / Strömungsberechnung und Rheologie Folie 10

Materialdaten

	Symbol	Einheit
Permeabilität x-Komponente	K_x	µm²
Permeabilität y-Komponente	K_y	µm²
Dispersionstensor x-Komponente	D_x	m²/s
Dispersionstensor y-Komponente	D_y	m²/s
Wärmeleitfähigkeit x-Komponente	λ_x	W(Km)
Wärmeleitfähigkeit y-Komponente	λ_y	W(Km)
dynamische Viskosität	η	Pa s
individividuelle Gas-konstante Sauerstoff	R_{O_2}	J/(gK)
individividuelle Gas-konstante Stickstoff	R_{N_2}	J/(gK)
individividuelle Gas-konstante Wasserdampf	R_{H_2O}	J/(gK)
spezifische Wärmekapazität	c_p	J/(g K)
Massenkonzentration Stickstoff im Querschnitt	γ	-

Randbedingungen



- Rand I : Übergang zur freien Kanalströmung
- Rand II : Elektrodenwand
- Rand III : Seitenränder, Dichtung
- Rand IV : Membranseite/Katalysatorschicht:

- Rand I :

  1. Mechanische Randbedingung ist die Druckrandbedingung

$$p = p_{\mathrm{K}}$$

  2. Thermische Randbedingung ist der Wärmestrom

$$\lambda_y \frac{\partial T}{\partial y} = -\alpha_{\mathrm{K}} \left(T - T_{\mathrm{K}} \right)$$

  3. Stoff-Randbedingung ist die diffusive Stoffstromdichte in y-Richtung

$$D_y \frac{\partial C_{\mathrm{O}_2}}{\partial y} = -\beta_{\mathrm{K}} \left(C_{\mathrm{O}_2} - C_{\mathrm{O}_2\mathrm{K}} \right)$$

- Rand II :

  1. Mechanische Randbedingung ist die kinematische Randbedingung

$$u_y = 0$$

  2. Thermische Randbedingung ist der Wärmestrom

$$\lambda_y \frac{\partial T}{\partial y} = -\alpha_{\mathrm{W}} \left(T - T_{\mathrm{C}} \right)$$

  3. Stoff-Randbedingung ist die Undurchlässigkeit der Wand

$$\frac{\partial C_{\mathrm{O}_2}}{\partial y} = 0$$

- Rand III:
  1. Mechanische Randbedingung ist die kinematische Randbedingung

$$u_x = 0$$

  2. Thermische Randbedingung ist adiabatische Isolierung

$$\frac{\partial T}{\partial x} = 0$$

  3. Stoff-Randbedingung ist die Undurchlässigkeit der Wand

$$\frac{\partial C_{O_2}}{\partial x} = 0$$

- Rand IV:
  1. Bilanz der Stoffstromdichten von Wasser und Sauerstoff in y-Richtung

$$u_y C_{O_2} - D_y \frac{\partial C_{O_2}}{\partial y} = -\frac{M_{O_2}}{2M_{H_2O}} \left(u_y C_{H_2O} - D_y \frac{\partial C_{H_2O}}{\partial y} \right)$$

  aus der Reaktionsbilanz an der Kathode: 2mol Wasser pro 1 mol Sauerstoff. Voraussetzung: kein Wassertransport durch die Membran. Umformung ergibt:

$$\left(u_y C_{O_2} - D_y \frac{\partial C_{O_2}}{\partial y} \right) \left(1 - \frac{R_{H_2O}}{2 R_{O_2}} \right) = -\frac{R_{H_2O}}{2 R_{O_2}} (1-\gamma) u_y$$

- Rand IV:
  2. Thermische Randbedingung ist der abgeführte Wärmestrom aus der exothermen Reaktion an der Katalysatorschicht

$$\lambda_y \frac{\partial T}{\partial y} = -q\,\beta_r \rho\, C_{O_2}$$

  3. Stoff-Randbedingung ist die Stoffstromdichte

$$D_y \frac{\partial C_{O_2}}{\partial y} = \beta_r \left(C_{O_2} - C^*_{O_2}\right)$$

  die Annahme einer instantanen und irreversiblen Reaktion in der Katalysatorschicht liefert die Bedingung $C^*_{O_2} = 0$

$$D_y \frac{\partial C_{O_2}}{\partial y} = \beta_r\, C_{O_2}$$

Daten zur Modellierung der Randbedingungen

	Symbol	Einheit
Gesamtdruck im Kanal	p_K	Pa
Temperatur im Kanal	T_K	K
Massenkonzentration des Sauerstoffs im Kanal	C_{O_2K}	-
Wärmeübergangskoeffizient Kanal-GDL	α_K	W/m²
Stoffübergangskoeffizient Kanal-GDL	β_K	m/s
Temperatur der Elektrode	T_C	K
Wärmeübergangskoeffizient Elektrode-GDL	α_W	W/m²
Stoffübergangskoeffizient der Reaktionsschicht	β_r	m/s
Reaktionswärme	q	J/kg

Literatur

[1]: Bradean/ Promislow/ Wetton;
Heat and Mass Transfer in Porous Fuel Cell Electrodes;
Intern. Symposion on Advances in Computational Heat Transfer
Palm Cove Queensland/Australia May 2001

[2]: Promislow/ Stockie;
Adiabatic Relaxation of Convective-Diffusive Gas Transport in a Porous Fuel Cell Electrode; SIAM J. Appl. Math., Vol. 62/Nr.1 (2001)

[3]: Bird/Stewart/Lightfoot;
Transport Phenomena;
John Wiley New York 1960

[4]: Bejan/Nield;
Convection in Porous Media;
Springer-Verlag New York 1999

[5]: Baehr;
Thermodynamik;
Springer-Verlag Berlin, Heidelberg, New York 1996

Addendum A2

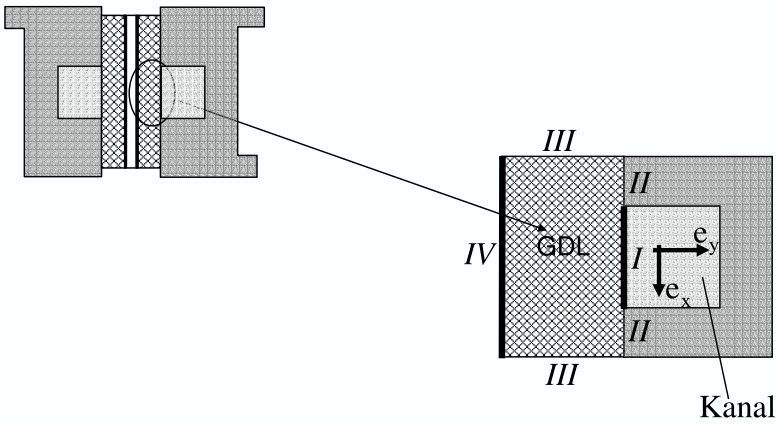
Addendum A2

Slides of Freudenberg
ForschungsDienste KG

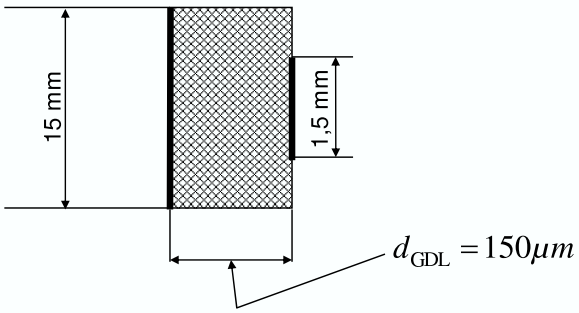
Material Parameters
for the Cathode Part of
a PEM Fuel Cell

(in German)

Berechnung der Kathodenseite einer Test-BSZ
Tabellen der Zahlenwerte



Geometrie der Test-BSZ



- Dicke der GDL d_{GDL} variiert je nach Verpressung und bestimmt die Permeabilität der GDL. Werte zwischen 140µm und 240 µm.
  Kanallänge (Dimension in z-Richtung) ist 310mm.

Freudenberg
ForschungsDienste KG

Freudenberg

Zusammenfassung der Gleichungen im x,y-Hauptachsensystem

- Impuls $u_x = -\frac{K_x}{\eta}\frac{\partial p}{\partial x} \qquad u_y = -\frac{K_y}{\eta}\frac{\partial p}{\partial y}$
- Masse $\frac{\partial(\rho u_x)}{\partial x} + \frac{\partial(\rho u_y)}{\partial y} = 0$
- O_2-Transport $\rho\left(u_x \frac{\partial C_{O_2}}{\partial x} + u_y \frac{\partial C_{O_2}}{\partial y}\right) = \frac{\partial}{\partial x}\left(D_x \rho \frac{\partial C_{O_2}}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_y \rho \frac{\partial C_{O_2}}{\partial y}\right)$
- Energie $\rho\, c_p\left(u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y}\right) = \frac{\partial}{\partial x}\left(\lambda_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\lambda_y \frac{\partial T}{\partial y}\right)$
- Konstitutive Gleichung $p = \rho T\left(R_{H_2O}\left(1-\gamma-C_{O_2}\right) + R_{N_2}\gamma + R_{O_2}C_{O_2}\right)$
- 6 PDEs für die 6 Unbekannten $u_x, u_y, p, \rho, T, C_{O_2}$

CAE / Strömungsberechnung und Rheologie Folie 3

Freudenberg
ForschungsDienste KG

Freudenberg

Gesuchte Größen

	Symbol	Einheit
Geschwindigkeit x-Komponente	u_x	m/s
Geschwindigkeit y-Komponente	u_y	m/s
Gesamtdruck	p	Pa
Gesamtdichte	ρ	kg/m³
Temperatur	T	K
Massenkonzentration des Sauerstoffs	C_{O_2}	-

CAE / Strömungsberechnung und Rheologie Folie 4

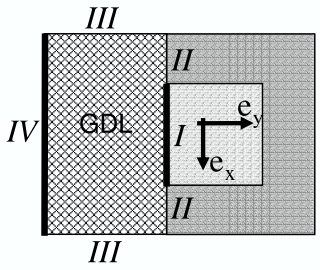
Materialdaten Zahlenwerte

Grösse	Symbol	Zahlenwert	Einheit	Quelle
Permeabilität x-Komponente bei GDL-Verpressung von 250 auf 150 µm	K_x	2,50E+00	µm²	Messung FFD
Permeabilität y-Komponente bei GDL-Verpressung von 250 auf 150 µm	K_y	2,50E+00	µm²	Messung FFD
Dispersion (hier Diffusion) x-Komponente	D_x	2,90E-05	m²/s	[7], S.651
Dispersion (hier Diffusion) y-Komponente	D_y	2,90E-05	m²/s	[7], S.651
effektive Wärmeleitfähigkeit x-Komponente	λ_x	4,00E+00	W/(Km)	[1], S.3
effektive Wärmeleitfähigkeit y-Komponente	λ_y	4,00E+00	W/(Km)	[1], S.3
dynamische Viskosität	η	2,00E-04	Pas	[6], S.432
individuelle Gaskonstante Sauerstoff	R_{O_2}	2,60E-01	J/(gK)	[5], S.422
individuelle Gaskonstante Stickstoff	R_{N_2}	2,97E-01	J/(gK)	[5], S.422
individuelle Gaskonstante Wasserdampf	R_{H_2O}	4,62E-01	J/(gK)	[5], S.422
spezifische Wärmekapazität des idealen Gases bei konstantem Druck	c_p	1,08	J/(gK)	[5], S.424
Massenkonzentration des Stickstoffs	γ	0,7	-	[5], S.207

Materialdaten Zahlenwerte Einheitensystem mm, g, s, K

Grösse	Symbol	Zahlenwert	Einheit im System g mm s K
Permeabilität x-Komponente bei GDL-Verpressung von 250 auf 150 µm	K_x	2,50E-06	mm²
Permeabilität y-Komponente bei GDL-Verpressung von 250 auf 150 µm	K_y	2,50E-06	mm²
Dispersion (hier Diffusion) x-Komponente	D_x	2,90E+01	mm²/s
Dispersion (hier Diffusion) y-Komponente	D_y	2,90E+01	mm²/s
effektive Wärmeleitfähigkeit x-Komponente	λ_x	4,00E+09	g mm/(s³K)
effektive Wärmeleitfähigkeit y-Komponente	λ_y	4,00E+09	g mm/(s³K)
dynamische Viskosität	η	2,00E-04	g/(mm s)
individuelle Gaskonstante Sauerstoff	R_{O_2}	2,60E+08	mm/(K s²)
individuelle Gaskonstante Stickstoff	R_{N_2}	2,97E+08	mm/(K s²)
individuelle Gaskonstante Wasserdampf	R_{H_2O}	4,62E+08	mm/(K s²)
spezifische Wärmekapazität des idealen Gases bei konstantem Druck	c_p	1,08E+09	mm/(K s²)
Massenkonzentration des Stickstoffs	γ	7,00E-01	-

Randbedingungen



- Rand I : Übergang zur freien Kanalströmung
- Rand II : Elektrodenwand
- Rand III : Seitenränder, Dichtung
- Rand IV : Membranseite/Katalysatorschicht:

- Rand I :

1. Mechanische Randbedingung ist die Druckrandbedingung

$$p = p_{\mathrm{K}}$$

2. Thermische Randbedingung ist der Wärmestrom

$$\lambda_y \frac{\partial T}{\partial y} = -\alpha_{\mathrm{K}} \left(T - T_{\mathrm{K}} \right)$$

3. Stoff-Randbedingung ist die diffusive Stoffstromdichte in y-Richtung

$$D_y \frac{\partial C_{\mathrm{O}_2}}{\partial y} = -\beta_{\mathrm{K}} \left(C_{\mathrm{O}_2} - C_{\mathrm{O}_2\mathrm{K}} \right)$$

- Rand II :

  1. Mechanische Randbedingung ist die kinematische Randbedingung

$$u_y = 0$$

  2. Thermische Randbedingung ist der Wärmestrom

$$\lambda_y \frac{\partial T}{\partial y} = -\alpha_W \left(T - T_C\right)$$

  3. Stoff-Randbedingung ist die Undurchlässigkeit der Wand

$$\frac{\partial C_{O_2}}{\partial y} = 0$$

- Rand III :

  1. Mechanische Randbedingung ist die kinematische Randbedingung

$$u_x = 0$$

  2. Thermische Randbedingung ist adiabatische Isolierung

$$\frac{\partial T}{\partial x} = 0$$

  3. Stoff-Randbedingung ist die Undurchlässigkeit der Wand

$$\frac{\partial C_{O_2}}{\partial x} = 0$$

- Rand IV:

  1. Bilanz der Stoffstromdichten von Wasser und Sauerstoff in y-Richtung

$$u_y C_{O_2} - D_y \frac{\partial C_{O_2}}{\partial y} = -\frac{M_{O_2}}{2M_{H_2O}} \left(u_y C_{H_2O} - D_y \frac{\partial C_{H_2O}}{\partial y} \right)$$

  aus der Reaktionsbilanz an der Kathode: 2mol Wasser pro 1 mol Sauerstoff. Voraussetzung: kein Wassertransport durch die Membran. Umformung ergibt:

$$\left(u_y C_{O_2} - D_y \frac{\partial C_{O_2}}{\partial y} \right) \left(1 - \frac{R_{H_2O}}{2 R_{O_2}} \right) = -\frac{R_{H_2O}}{2 R_{O_2}} (1-\gamma) u_y$$

- Rand IV:

  2. Thermische Randbedingung ist der abgeführte Wärmestrom aus der exothermen Reaktion an der Katalysatorschicht

$$\lambda_y \frac{\partial T}{\partial y} = -q \, \beta_r \rho \, C_{O_2}$$

  3. Stoff-Randbedingung ist die Stoffstromdichte

$$D_y \frac{\partial C_{O_2}}{\partial y} = \beta_r \left(C_{O_2} - C^*_{O_2} \right)$$

  die Annahme einer instantanen und irreversiblen Reaktion in der Katalysatorschicht liefert die Bedingung $C^*_{O_2} = 0$

$$D_y \frac{\partial C_{O_2}}{\partial y} = \beta_r \, C_{O_2}$$

Zahlenwerte zur Modellierung der Randbedingungen

Grösse	Symbol	Zahlenwert	Einheit	Quelle
Gesamtdruck im Kanal	p_K	1,40E+05	Pa	FFD
Temperatur im Kanal	T_K	7,50E+01	°C	FFD
Massenkonzentration des Sauerstoffs im Kanal	C_{O_2K}	1,90E-01	-	FFD
Wärmeübergangskoeffizient GDL-Kanal	α_K	1,50E+01	W/(K m²)	[1], S.5
Stoffübergangskoeffizient Kanal-GDL	β_K	2,00E-03	m/s	[2], S.8
Temperatur der Elektrode	T_C	7,50E+01	°C	FFD
Wärmeübergangskoeffizient GDL-Elektrode	α_W	1,10E+04	W/(K m²)	[1], S.5
Stoffübergangskonstante der Reaktionsschicht	β_r	6,00E-04	m/s	[2], S.8
Reaktionswärme	q	8,50E+03	J/(gK)	[1], S.5

Zahlenwerte zur Modellierung der Randbedingungen im mm, g, s, K System

Grösse	Symbol	Zahlenwert	Einheit im System g mm s K
Gesamtdruck im Kanal	p_K	1,40E+05	g/(mm s²)
Temperatur im Kanal	T_K	3,48E+02	K
Massenkonzentration des Sauerstoffs im Kanal	C_{O_2K}	1,90E-01	-
Wärmeübergangskoeffizient GDL-Kanal	α_K	1,50E+04	g/(s³ K)
Stoffübergangskoeffizient Kanal-GDL	β_K	2,00E+00	mm²/s
Temperatur der Elektrode	T_C	3,48E+02	K
Wärmeübergangskoeffizient GDL-Elektrode	α_W	1,10E+07	g/(s³ K)
Stoffübergangskonstante der Reaktionsschicht	β_r	6,00E-01	mm/s
Reaktionswärme	q	8,50E+12	mm/(K s²)

Freudenberg
ForschungsDienste KG

Freudenberg

Literatur

[1]: Bradean/ Promislow/ Wetton;
Heat and Mass Transfer in Porous Fuel Cell Electrodes;
Intern. Symposion on Advances in Computational Heat Transfer
Palm Cove Queensland/Australia May 2001

[2]: Promislow/ Stockie;
Adiabatic Relaxation of Convective-Diffusive Gas Transport in a Porous Fuel Cell Electrode; SIAM J. Appl. Math., Vol. 62/Nr.1 (2001)

[3]: Bird/Stewart/Lightfoot;
Transport Phenomena;
John Wiley New York 1960

[4]: Bejan/Nield;
Convection in Porous Media;
Springer-Verlag New York 1999

Freudenberg
ForschungsDienste KG

Freudenberg

Literatur

[5]: Baehr;
Thermodynamik;
Springer-Verlag Berlin, Heidelberg, New York 1996

[6]: Spurk;
Strömungslehre;
Springer-Verlag Berlin, Heidelberg, New York 1996

[7]: Baehr/Stephan;
Wärme und Stoffübertragung;
Springer-Verlag Berlin, Heidelberg 1994

Addendum A3

Program Code for the Subroutines to Enter the PDEs and BCs and their Jacobian Matrices (FDEMU1 to FDEMU4) for the Example of the Fuel Cell

```
      subroutine FDEMU1(isect,t,x,y,u,ut,ux,uy,uxx,uxy,uyy,
     &                  p,mt,mv,nk,it,ljac,fex)
!**                                                                   ***
!**********************************************************************
!**                                                                   ***
!**      F D E M U 1   subroutine                                     ***
!**      in which the pde system at inner grid points is described.   ***
!**                                                                   ***
!**********************************************************************
!**                                                                   ***
!**      formal parameters :                                          ***
!**      -------------------                                          ***
!**                                                                   ***
      USE probconst

      implicit none
!***
      integer, intent(in) :: mt, mv, nk, it, isect
      double precision,intent(in) :: t,x(mv),y(mv),fex(nk*it)
      double precision,intent(inout) :: u(mv,nk),ut(mt,nk),ux(mv,nk),
     &                                  uy(mv,nk),uxx(mv,nk),
     &                                  uxy(mv,nk),uyy(mv,nk)
      double precision, intent(out) :: p(mv,nk)
      logical, intent(in) :: ljac
!**                                                                   ***
!**-----------------------------------------------------------------***
!**                                                                   ***
!**   list of formal parameters :                                     ***
!**   ---------------------------                                     ***
!**                                                                   ***
!--------i------i-----i--------------------------------------------------
! name   i type i i/o i     meaning
!--------i------i-----i--------------------------------------------------
!--------i------i-----i--------------------------------------------------
!**                                                                   ***
!**                                                                   ***
!**                                                                   ***
!**        local parameters :  (please define all the appearing       ***
!**        ------------------   local parameters)                     ***
!**                                                                   ***
      integer i,j
      double precision f,fx,fy,fz,fxy,fyz,fxz,fxx,fyy,fzz
      double precision f2,f2x,f2y,f2z,f2xy,f2yz,f2xz,f2xx,f2yy,f2zz
      double precision f3,f3x,f3y
      double precision f4,f4x,f4y,f4z,f4xy,f4yz,f4xz,f4xx,f4yy,f4zz
      double precision f5,f5x,f5y,f5z,f5xy,f5yz,f5xz,f5xx,f5yy,f5zz
      double precision f6,f6x,f6y,f6z,f6xy,f6yz,f6xz,f6xx,f6yy,f6zz



!**                                                                   ***
!**                                                                   ***
!**** start of calculation :
!**   ---------------------
```

```
      if (ljac) then
        do j = 1,nk
          do i = 1,mv
            u(i,j)   = u(i,j)   + fex((j-1)*it+1)
            ux(i,j)  = ux(i,j)  + fex((j-1)*it+2)
            uy(i,j)  = uy(i,j)  + fex((j-1)*it+3)
            uxx(i,j) = uxx(i,j) + fex((j-1)*it+4)
            uxy(i,j) = uxy(i,j) + fex((j-1)*it+5)
            uyy(i,j) = uyy(i,j) + fex((j-1)*it+6)
          enddo
        enddo
        do j = 1,nk
          do i = 1,mt
            ut(i,j) = ut(i,j) + fex(j*it)
          enddo
        enddo
      endif

      do i = 1,mv

         p(i,1) = u(i,1)+K_x*ux(i,3)/ns

!     &            -(f(x(i),y(i),t)+K_x*f3x(x(i),y(i),t)/ns)

         p(i,2) = u(i,2)+K_y*uy(i,3)/ns

!     &           -(f2(x(i),y(i),t)+K_y*f3y(x(i),y(i),t)/ns)

         p(i,3) = u(i,3)-u(i,4)*u(i,5)*
     &            (RR_H2O*(1-gamma-u(i,6))+RR_N2*gamma+RR_O2*u(i,6))

!     &            -(f3(x(i),y(i),t)-f4(x(i),y(i),t)*f5(x(i),y(i),t)*
!     &            (RR_H2O*(1-gamma-f6(x(i),y(i),t))
!     &            +RR_N2*gamma+RR_O2*f6(x(i),y(i),t)))

         p(i,4) = ux(i,4)*u(i,1)+u(i,4)*(ux(i,1)+uy(i,2))+uy(i,4)*u(i,2)

!     &            -(f4x(x(i),y(i),t)*f(x(i),y(i),t)+f4(x(i),y(i),t)*(
!     &            fx(x(i),y(i),t)+f2y(x(i),y(i),t))+f4y(x(i),y(i),t)*
!     &            f2(x(i),y(i),t))

         p(i,5) = u(i,4)*CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))-
     &           lambda_x*uxx(i,5)-lambda_y*uyy(i,5)

!     &           -(f4(x(i),y(i),t)*CC_p*(f(x(i),y(i),t)*
!     &           f5x(x(i),y(i),t)+f2(x(i),y(i),t)*f5y(x(i),y(i),t))-
!     &           lambda_x*f5xx(x(i),y(i),t)-lambda_y*f5yy(x(i),y(i),t))


         p(i,6) = u(i,4)*(u(i,1)*ux(i,6)+u(i,2)*uy(i,6))-
     &            DD_x*(ux(i,4)*ux(i,6)+u(i,4)*uxx(i,6))-
     &            DD_y*(uy(i,4)*uy(i,6)+u(i,4)*uyy(i,6))

!     &            -(f4(x(i),y(i),t)*(f(x(i),y(i),t)*f6x(x(i),y(i),t)+
```

```
!     &            f2(x(i),y(i),t)*f6y(x(i),y(i),t))-
!     &            DD_x*(f4x(x(i),y(i),t)*f6x(x(i),y(i),t)+
!     &            f4(x(i),y(i),t)*f6xx(x(i),y(i),t))-
!     &            DD_y*(f4y(x(i),y(i),t)*f6y(x(i),y(i),t)+
!     &            f4(x(i),y(i),t)*f6yy(x(i),y(i),t)))

      enddo

!**** end of calculation
!     ------------------
!
      r e t u r n
!-----end of FDEMU1-------------------------------------------------------
      e    n    d

      subroutine FDEMU2(irand,t,x,y,u,ut,ux,uy,uxx,uxy,
     &                  uyy,p,nb,nk,it,ljac,fex)
!**                                                                    ***
!**********************************************************************
!**                                                                    ***
!**      F D E M U 2   subroutine                                      ***
!**      in which the boundary conditions for the 6 boundary areas     ***
!**      are described.                                                ***
!**      example e1                                                    ***
!**                                                                    ***
!**********************************************************************
!**                                                                    ***
!**      formal parameters :                                           ***
!**      -------------------                                           ***
!**                                                                    ***
      USE probconst

      implicit none
!***
      integer, intent(in) :: irand, nk, nb, it
      double precision, intent(in) :: t,x(nb),y(nb),fex(nk*it)
      double precision, intent(inout) :: u(nb,nk),ut(nb,nk),ux(nb,nk),
     &                                   uy(nb,nk),uxx(nb,nk),
     &                                   uxy(nb,nk),uyy(nb,nk)
      double precision, intent(out) :: p(nb,nk)
      logical, intent(in) :: ljac
!**                                                                    ***
!**--------------------------------------------------------------------***
!**                                                                    ***
!**   list of formal parameters :                                      ***
!**   ---------------------------                                      ***
!**                                                                    ***
!--------i------i-----i-----------------------------------------------
! name   i type i i/o i     meaning
!--------i------i-----i-----------------------------------------------
!--------i------i-----i-----------------------------------------------
!**                                                                    ***
!**                                                                    ***
!**                                                                    ***
```

```
!**        local parameters :  (please define all the appearing     ***
!**        ------------------   local parameters)                   ***
!**                                                                  ***
      integer i,j
      double precision f,fx,fy,fz,fxy,fyz,fxz,fxx,fyy,fzz
      double precision f2,f2x,f2y,f2z,f2xy,f2yz,f2xz,f2xx,f2yy,f2zz
      double precision f3,f3x,f3y
      double precision f4,f4x,f4y,f4z,f4xy,f4yz,f4xz,f4xx,f4yy,f4zz
      double precision f5,f5x,f5y,f5z,f5xy,f5yz,f5xz,f5xx,f5yy,f5zz
      double precision f6,f6x,f6y,f6z,f6xy,f6yz,f6xz,f6xx,f6yy,f6zz

      double precision p1(nb),p2(nb),p3(nb),p4(nb),p5(nb),p6(nb),p7(nb)
      double precision p8(nb),p9(nb),p10(nb),p11(nb),p12(nb),p13(nb)
      double precision p14(nb),p15(nb),p16(nb),p17(nb),p18(nb),p19(nb)
      double precision p20(nb)



!**                                                                  ***
!**                                                                  ***
!**** start of calculation :
!**   ---------------------



      if (ljac) then
        do j = 1,nk
          do i = 1,nb
            u(i,j)   = u(i,j)   + fex((j-1)*it+1)
            ux(i,j)  = ux(i,j)  + fex((j-1)*it+2)
            uy(i,j)  = uy(i,j)  + fex((j-1)*it+3)
            uxx(i,j) = uxx(i,j) + fex((j-1)*it+4)
            uxy(i,j) = uxy(i,j) + fex((j-1)*it+5)
            uyy(i,j) = uyy(i,j) + fex((j-1)*it+6)
            ut(i,j)  = ut(i,j)  + fex(j*it)
               enddo
        enddo
      endif



      do i = 1,nb

         p1(i) = u(i,1)+K_x*ux(i,3)/ns

!     &           -(f(x(i),y(i),t)+K_x*f3x(x(i),y(i),t)/ns)

         p2(i) = u(i,2)+K_y*uy(i,3)/ns

!     &           -(f2(x(i),y(i),t)+K_y*f3y(x(i),y(i),t)/ns)

         p3(i) = u(i,3)-u(i,4)*u(i,5)*
     &           (RR_H2O*(1-gamma-u(i,6))+RR_N2*gamma+RR_O2*u(i,6))
```

```
!     &           -(f3(x(i),y(i),t)-f4(x(i),y(i),t)*f5(x(i),y(i),t)*
!     &           (RR_H2O*(1-gamma-f6(x(i),y(i),t))
!     &           +RR_N2*gamma+RR_O2*f6(x(i),y(i),t)))

         p4(i) = ux(i,4)*u(i,1)+u(i,4)*(ux(i,1)+uy(i,2))+uy(i,4)*u(i,2)

!     &           -(f4x(x(i),y(i),t)*f(x(i),y(i),t)+f4(x(i),y(i),t)*(
!     &           fx(x(i),y(i),t)+f2y(x(i),y(i),t))+f4y(x(i),y(i),t)*
!     &           f2(x(i),y(i),t))

         p5(i) = u(i,4)*CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))-
     &           lambda_x*uxx(i,5)-lambda_y*uyy(i,5)

!     &           -(f4(x(i),y(i),t)*CC_p*(f(x(i),y(i),t)*
!     &           f5x(x(i),y(i),t)+f2(x(i),y(i),t)*f5y(x(i),y(i),t))-
!     &           lambda_x*f5xx(x(i),y(i),t)-lambda_y*f5yy(x(i),y(i),t))

         p6(i) = u(i,4)*(u(i,1)*ux(i,6)+u(i,2)*uy(i,6))-
     &           DD_x*(ux(i,4)*ux(i,6)+u(i,4)*uxx(i,6))-
     &           DD_y*(uy(i,4)*uy(i,6)+u(i,4)*uyy(i,6))

!     &           -(f4(x(i),y(i),t)*(f(x(i),y(i),t)*f6x(x(i),y(i),t)+
!     &           f2(x(i),y(i),t)*f6y(x(i),y(i),t))-
!     &           DD_x*(f4x(x(i),y(i),t)*f6x(x(i),y(i),t)+
!     &           f4(x(i),y(i),t)*f6xx(x(i),y(i),t))-
!     &           DD_y*(f4y(x(i),y(i),t)*f6y(x(i),y(i),t)+
!     &           f4(x(i),y(i),t)*f6yy(x(i),y(i),t)))

!Speziell fuer den Rand 1:
         p7(i) = u(i,3)-PP_k

!     &           -(f3(x(i),y(i),t)-PP_k)

         p8(i) = lambda_y*uy(i,5)+Alpha_k*(u(i,5)-TT_k)

!     &           -(lambda_y*f5y(x(i),y(i),t)
!     &           +Alpha_k*(f5(x(i),y(i),t)-TT_k))

         p9(i) = DD_y*uy(i,6)+Beta_k*(u(i,6)-CC_o2k)

!     &           -(DD_y*f6y(x(i),y(i),t)
!     &           +Beta_k*(f6(x(i),y(i),t)-CC_o2k))

!Speziell fuer den Rand 2:
         p10(i) = u(i,2)

!     &            -f2(x(i),y(i),t)

         p11(i) = lambda_y*uy(i,5)+Alpha_w*(u(i,5)-TT_c)

!     &          -(lambda_y*f5y(x(i),y(i),t)
!     &          +Alpha_w*(f5(x(i),y(i),t)-TT_c))
```

```
         p12(i) = uy(i,6)

!     &           -f6y(x(i),y(i),t)

         p13(i) = uy(i,3)

!     &            -f3y(x(i),y(i),t)

!Speziell fuer den Rand 3:
         p14(i) = u(i,1)

!     &            -f(x(i),y(i),t)

         p15(i) = ux(i,5)

!     &            -f5x(x(i),y(i),t)

         p16(i) = ux(i,6)

!     &            -f6x(x(i),y(i),t)

         p17(i) = ux(i,3)

!     &            -f3x(x(i),y(i),t)

!Speziell fuer den Rand 4:
         p18(i) = (u(i,2)*u(i,6)-DD_y*uy(i,6))*(1.-(RR_H2O/(2.*RR_O2)))+
     &            (RR_H2O/(2.*RR_O2))*(1.-gamma)*u(i,2)

!     &            -((f2(x(i),y(i),t)*f6(x(i),y(i),t)
!     &            -DD_y*f6y(x(i),y(i),t))*(1.-(RR_H2O/(2.*RR_O2)))+
!     &             (RR_H2O/(2.*RR_O2))*(1.-gamma)*f2(x(i),y(i),t))

         p19(i) = lambda_y*uy(i,5)+QQ*Beta_r*u(i,4)*u(i,6)

!     &            -(lambda_y*f5y(x(i),y(i),t)
!     &            +QQ*Beta_r*f4(x(i),y(i),t)*f6(x(i),y(i),t))

         p20(i) = DD_y*uy(i,6)-Beta_r*(u(i,6)-CC_O2)

!     &           -(DD_y*f6y(x(i),y(i),t)-Beta_r*(f6(x(i),y(i),t)-CC_O2))

      enddo


!     zum kanal:
      if (irand == 1) then
         do i=1,nb

            p(i,1) = p14(i)

            p(i,2) = p2(i)

            p(i,3) = p7(i)
```

```
            p(i,4) = p3(i)

            p(i,5) = p8(i)

            p(i,6) = p9(i)

         enddo
      endif
!     neben dem Kanal (oben):
      if (irand == 2) then
         do i=1,nb

            p(i,1) = p1(i)

            p(i,2) = p10(i)

            p(i,3) = p13(i)

            p(i,4) = p3(i)

            p(i,5) = p11(i)

            p(i,6) = p12(i)

         enddo
      endif
!     rechts/links:
      if (irand == 3) then
         do i=1,nb

            p(i,1) = p14(i)

            p(i,2) = p2(i)

            p(i,3) = p17(i)

            p(i,4) = p3(i)

            p(i,5) = p15(i)

            p(i,6) = p16(i)

         enddo
      endif
!     unten:
      if (irand == 4) then
         do i=1,nb

            p(i,1) = p1(i)

            p(i,2) = p18(i)

            p(i,3) = p2(i)
```

```
            p(i,4) = p3(i)

            p(i,5) = p19(i)

            p(i,6) = p20(i)

         enddo
      endif



!**** end of calculation
!     ------------------
!
      r e t u r n
!-----end of FDEMU2-----------------------------------------------------
      e    n    d

      subroutine FDEMU3(isect,iequ,icom,t,x,y,u,ut,ux,uy,uxx,uxy,
     &                  uyy,pu,put,pux,puy,puxx,puxy,
     &                  puyy,mt,mv,nk)
!**                                                                   ***
!**********************************************************************
!**                                                                   ***
!**      F D E M U 3    subroutine                                    ***
!**      in which the jacobian matrices at inner grid points are     ***
!**      described. only nonzero elements must be defined.           ***
!**                                                                   ***
!**********************************************************************
!**                                                                   ***
!**      formal parameters :                                          ***
!**      -------------------                                          ***
!**                                                                   ***
      USE probconst

      implicit none
!***
      integer, intent(in) :: iequ, icom, mt, mv, nk, isect
      double precision, intent(in) :: t,x(mv),y(mv),u(mv,nk),
     &                                ut(mt,nk),ux(mv,nk),uy(mv,nk),
     &                                uxx(mv,nk),uxy(mv,nk),
     &                                uyy(mv,nk)
      double precision, intent(out) :: pu(mv),put(mt),pux(mv),puy(mv),
     &                                 puxx(mv),puxy(mv),
     &                                 puyy(mv)
!**                                                                   ***
!**-------------------------------------------------------------------***
!**                                                                   ***
!**   list of formal parameters :                                     ***
!**   ---------------------------                                     ***
!**                                                                   ***
!--------i------i-----i-----------------------------------------------
! name   i type i i/o i     meaning
```

```
!--------i------i-----i-------------------------------------------------
!**                                                                  ***
!**                                                                  ***
!**                                                                  ***
!**        local parameters :  (please define all the appearing      ***
!**        ------------------   local parameters)                    ***
!**                                                                  ***
      integer i

      double precision p1_1(mv),p1_3x(mv)
      double precision p2_2(mv),p2_3y(mv)
      double precision p3_3(mv),p3_4(mv),p3_5(mv),p3_6(mv)
      double precision p4_1x(mv),p4_2y(mv),p4_4x(mv),p4_4y(mv)
      double precision p4_1(mv),p4_2(mv),p4_4(mv)
      double precision p5_1(mv),p5_2(mv),p5_4(mv),p5_5x(mv)
      double precision p5_5y(mv),p5_5xx(mv),p5_5yy(mv)
      double precision p6_1(mv),p6_2(mv),p6_6x(mv),p6_6y(mv)
      double precision p6_6xx(mv),p6_6yy(mv)
      double precision p6_4(mv),p6_4x(mv),p6_4y(mv)

!**                                                            ***
!**                                                                  ***
!**** start of calculation :
!**   ---------------------


      do i=1,mv
!         p1(i) = u(i,1)+K_x*ux(i,3)/ns
         p1_1(i)= 1.
         p1_3x(i)= K_x/ns

!         p2(i) = u(i,2)+K_y*uy(i,3)/ns
         p2_2(i) = 1.
         p2_3y(i) = K_y/ns

!         p3(i) = u(i,3)-u(i,4)*u(i,5)*
!     &            (RR_H2O*(1-gamma-u(i,6))+RR_N2*gamma+RR_O2*u(i,6))
         p3_3(i) = 1.
         p3_4(i) = -u(i,5)*(RR_H2O*(1-gamma-u(i,6))
     &            +RR_N2*gamma+RR_O2*u(i,6))
         p3_5(i) = -u(i,4)*(RR_H2O*(1-gamma-u(i,6))
     &            +RR_N2*gamma+RR_O2*u(i,6))
         p3_6(i) = u(i,4)*u(i,5)*(RR_H2O-RR_O2)

!         p4(i) = ux(i,4)*u(i,1)+u(i,4)*(ux(i,1)+uy(i,2))+uy(i,4)*u(i,2)
         p4_1x(i)= u(i,4)
         p4_2y(i)= u(i,4)
         p4_4x(i)= u(i,1)
         p4_4y(i)= u(i,2)
         p4_1(i) = ux(i,4)
         p4_2(i) = uy(i,4)
         p4_4(i) = ux(i,1)+uy(i,2)

!         p5(i) = u(i,4)*CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))-
```

```
!     &            lambda_x*uxx(i,5)-lambda_y*uyy(i,5)
         p5_1(i) = u(i,4)*CC_p*ux(i,5)
         p5_2(i) = u(i,4)*CC_p*uy(i,5)
         p5_4(i) = CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))
         p5_5x(i)= u(i,4)*CC_p*u(i,1)
         p5_5y(i)= u(i,4)*CC_p*u(i,2)
         p5_5xx(i)= -lambda_x
         p5_5yy(i)= -lambda_y

!         p6(i) = u(i,4)*(u(i,1)*ux(i,6)+u(i,2)*uy(i,6))-
!     &            DD_x*(ux(i,4)*ux(i,6)+u(i,4)*uxx(i,6))-
!     &            DD_y*(uy(i,4)*uy(i,6)+u(i,4)*uyy(i,6))
         p6_1(i)= u(i,4)*ux(i,6)
         p6_2(i)= u(i,4)*uy(i,6)
         p6_4(i)= u(i,1)*ux(i,6)+u(i,2)*uy(i,6)-
     &           DD_x*uxx(i,6)-DD_y*uyy(i,6)
         p6_4x(i)= -DD_x*ux(i,6)
         p6_4y(i)= -DD_y*uy(i,6)
         p6_6x(i)= u(i,4)*u(i,1)-DD_x*ux(i,4)
         p6_6y(i)= u(i,4)*u(i,2)-DD_y*uy(i,4)
         p6_6xx(i)=-DD_x*u(i,4)
         p6_6yy(i)=-DD_y*u(i,4)

      enddo

      if (iequ == 1) then
         if (icom == 1) then
            do i=1,mv
               pu(i)= p1_1(i)
            enddo
         endif
          if (icom == 3) then
            do i=1,mv
               pux(i)= p1_3x(i)
            enddo
         endif
      endif

      if ( iequ == 2 ) then
         if (icom == 2) then
            do i=1,mv
               pu(i)= p2_2(i)
            enddo
         endif
         if (icom == 3) then
            do i=1,mv
               puy(i)= p2_3y(i)
            enddo
         endif
      endif

      if (iequ == 3) then
         if (icom == 3) then
            do i=1,mv
```

```
            pu(i)= p3_3(i)
         enddo
      endif
      if (icom == 4) then
         do i=1,mv
            pu(i)= p3_4(i)
         enddo
      endif
      if (icom == 5) then
         do i=1,mv
            pu(i)= p3_5(i)
         enddo
      endif
      if (icom == 6) then
         do i=1,mv
            pu(i)= p3_6(i)
         enddo
      endif
   endif

   if (iequ == 4) then
      if (icom == 1) then
         do i=1,mv
            pux(i)= p4_1x(i)
            pu(i) = p4_1(i)
         enddo
      endif
      if (icom == 2) then
         do i=1,mv
            pu(i) = p4_2(i)
            puy(i)= p4_2y(i)
         enddo
      endif
      if (icom == 4) then
         do i=1,mv
            pux(i)= p4_4x(i)
            puy(i)= p4_4y(i)
            pu(i) = p4_4(i)
         enddo
      endif
   endif

   if (iequ == 5) then
      if (icom == 1) then
         do i=1,mv
            pu(i)= p5_1(i)
         enddo
      endif
      if (icom == 2) then
         do i=1,mv
            pu(i)= p5_2(i)
         enddo
      endif
      if (icom == 4) then
```

```
            do i=1,mv
               pu(i)= p5_4(i)
            enddo
         endif
         if (icom == 5) then
            do i=1,mv
              pux(i)= p5_5x(i)
              puy(i)= p5_5y(i)
              puxx(i)= p5_5xx(i)
              puyy(i)= p5_5yy(i)
            enddo
         endif
      endif

      if (iequ == 6) then
         if (icom == 1) then
            do i=1,mv
               pu(i)= p6_1(i)
            enddo
         endif
         if (icom == 2) then
            do i=1,mv
               pu(i)= p6_2(i)
            enddo
         endif
         if (icom == 4) then
            do i=1,mv
               pu(i)= p6_4(i)
               pux(i)= p6_4x(i)
               puy(i)= p6_4y(i)
            enddo
         endif
         if (icom == 6) then
            do i=1,mv
              pux(i)= p6_6x(i)
              puy(i)= p6_6y(i)
              puxx(i)= p6_6xx(i)
              puyy(i)= p6_6yy(i)
            enddo
         endif
      endif



!**** end of calculation
!     -----------------
!
      r e t u r n
!-----end of FDEMU3-----------------------------------------------------
      e    n    d

      subroutine  FDEMU4(irand,iequ,icom,t,x,y,u,ut,ux,uy,uxx,
     &                   uxy,uyy,pu,put,pux,puy,puxx,
     &                   puxy,puyy,nb,nk)
```

```
!**                                                                  ***
!**********************************************************************
!**                                                                  ***
!**      F D E M U 4    subroutine                                   ***
!**      in which the jacobian matrices at the 4 boundary lines     ***
!**      are described. only nonzero elements must be defined.      ***
!**      example e1                                                  ***
!**                                                                  ***
!**********************************************************************
!**                                                                  ***
!**      formal parameters :                                         ***
!**      -------------------                                         ***
!**                                                                  ***

      USE probconst

      implicit none
!**                                                                  ***
      integer, intent(in) :: irand, iequ, icom, nk, nb
      double precision, intent(in) :: t,x(nb),y(nb),u(nb,nk),
     &                                ut(nb,nk),ux(nb,nk),uy(nb,nk),
     &                                uxx(nb,nk),uxy(nb,nk),
     &                                uyy(nb,nk)
      double precision, intent(out) :: pu(nb),put(nb),pux(nb),puy(nb),
     &                                 puxx(nb),puxy(nb),
     &                                 puyy(nb)
!**                                                                  ***
!**------------------------------------------------------------------***
!**                                                                  ***
!**   list of formal parameters :                                    ***
!**   ---------------------------                                    ***
!**                                                                  ***
!--------i------i-----i-----------------------------------------------
! name   i type i i/o i     meaning
!--------i------i-----i-----------------------------------------------
!--------i------i-----i-----------------------------------------------
!**                                                                  ***
!**                                                                  ***
!**                                                                  ***
!**        local parameters :  (please define all the appearing      ***
!**        ------------------   local parameters)                    ***
!**                                                                  ***
      integer i
      double precision f,fx,fy,fz,fxy,fyz,fxz,fxx,fyy,fzz

      double precision p1_1(nb),p1_3x(nb)
      double precision p2_2(nb),p2_3y(nb)
      double precision p3_3(nb),p3_4(nb),p3_5(nb),p3_6(nb)
      double precision p4_1x(nb),p4_2y(nb),p4_4x(nb),p4_4y(nb)
      double precision p4_1(nb),p4_2(nb),p4_4(nb)
      double precision p5_1(nb),p5_2(nb),p5_4(nb),p5_5x(nb)
      double precision p5_5y(nb),p5_5xx(nb),p5_5yy(nb)
      double precision p6_1(nb),p6_2(nb),p6_6x(nb),p6_6y(nb)
      double precision p6_6xx(nb),p6_6yy(nb),p6_4(nb)
```

```
      double precision p6_4x(nb),p6_4y(nb)

      double precision p7_3(nb),p8_5(nb),p8_5y(nb),p9_6(nb),p9_6y(nb)
      double precision p10_2(nb),p11_5(nb),p11_5y(nb),p12_6y(nb)
      double precision p13_3y(nb),p14_1(nb),p15_5x(nb),p16_6x(nb)
      double precision p17_3x(nb),p18_2(nb),p18_6(nb),p18_6y(nb)
      double precision p19_4(nb),p19_5y(nb),p19_6(nb)
      double precision p20_6(nb),p20_6y(nb),p12_5y(nb)


!**                                                                  ***
!**                                                                  ***
!**** start of calculation :
!**   ---------------------

      do i=1,nb

!         p1(i) = u(i,1)+K_x*ux(i,3)/ns
         p1_1(i)= 1.
         p1_3x(i)= K_x/ns

!         p2(i) = u(i,2)+K_y*uy(i,3)/ns
         p2_2(i) = 1.
         p2_3y(i) = K_y/ns

!         p3(i) = u(i,3)-u(i,4)*u(i,5)*
!     &            (RR_H2O*(1-gamma-u(i,6))+RR_N2*gamma+RR_O2*u(i,6))
         p3_3(i) = 1.
         p3_4(i) = -u(i,5)*(RR_H2O*(1-gamma-u(i,6))
     &            +RR_N2*gamma+RR_O2*u(i,6))
         p3_5(i) = -u(i,4)*(RR_H2O*(1-gamma-u(i,6))
     &            +RR_N2*gamma+RR_O2*u(i,6))
         p3_6(i) = u(i,4)*u(i,5)*(RR_H2O-RR_O2)

!         p4(i) = ux(i,4)*u(i,1)+u(i,4)*(ux(i,1)+uy(i,2))+uy(i,4)*u(i,2)
         p4_1x(i)= u(i,4)
         p4_2y(i)= u(i,4)
         p4_4x(i)= u(i,1)
         p4_4y(i)= u(i,2)
         p4_1(i) = ux(i,4)
         p4_2(i) = uy(i,4)
         p4_4(i) = ux(i,1)+uy(i,2)

!         p5(i) = u(i,4)*CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))-
!     &            lambda_x*uxx(i,5)-lambda_y*uyy(i,5)
         p5_1(i) = u(i,4)*CC_p*ux(i,5)
         p5_2(i) = u(i,4)*CC_p*uy(i,5)
         p5_4(i) = CC_p*(u(i,1)*ux(i,5)+u(i,2)*uy(i,5))
         p5_5x(i)= u(i,4)*CC_p*u(i,1)
         p5_5y(i)= u(i,4)*CC_p*u(i,2)
         p5_5xx(i)= -lambda_x
         p5_5yy(i)= -lambda_y

!         p6(i) = u(i,4)*(u(i,1)*ux(i,6)+u(i,2)*uy(i,6))-
```

```
!     &            DD_x*(ux(i,4)*ux(i,6)+u(i,4)*uxx(i,6))-
!     &            DD_y*(uy(i,4)*uy(i,6)+u(i,4)*uyy(i,6))
         p6_1(i)= u(i,4)*ux(i,6)
         p6_2(i)= u(i,4)*uy(i,6)
         p6_4(i)= u(i,1)*ux(i,6)+u(i,2)*uy(i,2)-
     &           DD_x*uxx(i,6)-DD_y*uyy(i,6)
         p6_4x(i)= -DD_x*ux(i,6)
         p6_4y(i)= -DD_y*uy(i,6)
         p6_6x(i)= u(i,4)*u(i,1)-DD_x*ux(i,4)
         p6_6y(i)= u(i,4)*u(i,2)-DD_y*uy(i,4)
         p6_6xx(i)=-DD_x*u(i,4)
         p6_6yy(i)=-DD_y*u(i,4)


!Speziell fuer den Rand 1:
!         p7(i) = u(i,3)-PP_k
         p7_3(i)= 1.

!         p8(i) = lambda_y*uy(i,5)+Alpha_k*(u(i,6)-TT_k)
         p8_5(i)= Alpha_k
         p8_5y(i)= lambda_y

!         p9(i) = DD_y*uy(i,6)+Beta_k*(u(i,6)-CC_o2k)
         p9_6(i)= Beta_k
         p9_6y(i)= DD_y

!Speziell fuer den Rand 2:
!         p10(i) = u(i,2)
         p10_2(i)= 1.

!         p11(i) = lambda_y*uy(i,5)+Alpha_w*(u(i,5)-TT_c)
         p11_5(i)= Alpha_w
         p11_5y(i)= lambda_y

!         p12(i) = uy(i,6)
         p12_6y(i)= 1.

!         p13(i) = uy(i,3)
         p13_3y(i)= 1.

!Speziell fuer den Rand 3:
!         p14(i) = u(i,1)
         p14_1(i)= 1.

!         p15(i) = ux(i,5)
         p15_5x(i)= 1.

!         p16(i) = ux(i,6)
         p16_6x(i)= 1.

!         p17(i) = ux(i,3)
         p17_3x(i)= 1.

!Speziell fuer den Rand 4:
```

```
!         p18(i) = (u(i,2)*u(i,6)-DD_y*uy(i,6))*(1.-(RR_H2O/(2.*RR_O2)))+
!     &            (RR_H2O/(2.*RR_O2))*(1.-gamma)*u(i,2)
         p18_2(i)= u(i,6)*(1.-(RR_H2O/(2.*RR_O2)))
     &            +RR_H2O/(2.*RR_O2)*(1.-gamma)
         p18_6(i)= u(i,2)*(1.-(RR_H2O/(2.*RR_O2)))
         p18_6y(i)= -DD_y*(1.-(RR_H2O/(2.*RR_O2)))

!         p19(i) = lambda_y*uy(i,5)+QQ*Beta_r*u(i,4)*u(i,6)
         p19_4(i)= QQ*Beta_r*u(i,6)
         p19_5y(i)= lambda_y
         p19_6(i)= QQ*Beta_r*u(i,4)

!         p20(i) = DD_y*uy(i,6)-Beta_r*(u(i,6)-CC_O2)
         p20_6(i)= -Beta_r
         p20_6y(i)= DD_y

      enddo


!*********************************************************************
!   RAND 1                                                      ****
!*********************************************************************
!     Zum Kanal:

      if (irand == 1) then

      if (iequ == 1) then
         if (icom == 1) then
            do i=1,nb
               pu(i)= p14_1(i)
            enddo
         endif
      endif

      if (iequ == 2) then
         if (icom == 2) then
            do i=1,nb
               pu(i)= p2_2(i)
            enddo
         endif
         if (icom == 3) then
            do i=1,nb
               puy(i)= p2_3y(i)
            enddo
         endif
      endif

      if (iequ == 3) then
         if (icom == 3) then
            do i=1,nb
               pu(i)= p7_3(i)
            enddo
         endif
      endif
```

```
      if (iequ == 4) then
         if (icom == 3) then
            do i=1,nb
               pu(i)= p3_3(i)
            enddo
         endif
         if (icom == 4) then
            do i=1,nb
               pu(i)= p3_4(i)
            enddo
         endif
         if (icom == 5) then
            do i=1,nb
               pu(i)= p3_5(i)
            enddo
         endif
         if (icom == 6) then
            do i=1,nb
               pu(i)= p3_6(i)
            enddo
         endif
      endif

      if (iequ == 5) then
         if (icom == 5) then
            do i=1,nb
               pu(i)= p8_5(i)
               puy(i)= p8_5y(i)
            enddo
         endif
      endif

      if (iequ == 6) then
         if (icom == 6) then
            do i=1,nb
               pu(i) = p9_6(i)
               puy(i)= p9_6y(i)
            enddo
         endif
      endif

      endif

!*********************************************************************
!   RAND 2                                                        ****
!*********************************************************************
!     oberer rand (neben Kanal):
      if (irand == 2) then

      if (iequ == 1) then
         if (icom == 1) then
            do i=1,nb
               pu(i)= p1_1(i)
```

```
      enddo
   endif
   if (icom == 3) then
      do i=1,nb
         pux(i)= p1_3x(i)
      enddo
   endif
endif

if (iequ == 2) then
   if (icom == 2) then
      do i=1,nb
         pu(i)= p10_2(i)
      enddo
   endif
endif

if (iequ == 3) then
   if (icom == 3) then
      do i=1,nb
         puy(i)= p13_3y(i)
      enddo
   endif
endif

if (iequ == 4) then
   if (icom == 3) then
      do i=1,nb
         pu(i)= p3_3(i)
      enddo
   endif
   if (icom == 4) then
      do i=1,nb
         pu(i)= p3_4(i)
      enddo
   endif
   if (icom == 5) then
      do i=1,nb
         pu(i)= p3_5(i)
      enddo
   endif
   if (icom == 6) then
      do i=1,nb
         pu(i)= p3_6(i)
      enddo
   endif
endif

if (iequ == 5) then
   if (icom == 5) then
      do i=1,nb
         pu(i)= p11_5(i)
         puy(i)= p11_5y(i)
      enddo
```

```
         endif
      endif

      if (iequ == 6) then
         if (icom == 6) then
            do i=1,nb
               puy(i)= p12_6y(i)
            enddo
         endif
      endif


      endif


!*********************************************************************
!   RAND 3                                                       ****
!*********************************************************************
!     rechter & linker rand:
      if (irand == 3) then

      if (iequ == 1) then
         if (icom == 1) then
            do i=1,nb
               pu(i)= p14_1(i)
            enddo
         endif
      endif

      if (iequ == 2) then
         if (icom == 2) then
            do i=1,nb
               pu(i)= p2_2(i)
            enddo
         endif
         if (icom == 3) then
            do i=1,nb
               puy(i)= p2_3y(i)
            enddo
         endif
      endif

      if (iequ == 3) then
         if (icom == 3) then
            do i=1,nb
               pux(i)= p17_3x(i)
            enddo
         endif
      endif

      if (iequ == 4) then
         if (icom == 3) then
            do i=1,nb
               pu(i)= p3_3(i)
```

```
            enddo
         endif
         if (icom == 4) then
            do i=1,nb
               pu(i)= p3_4(i)
            enddo
         endif
         if (icom == 5) then
            do i=1,nb
               pu(i)= p3_5(i)
            enddo
         endif
         if (icom == 6) then
            do i=1,nb
               pu(i)= p3_6(i)
            enddo
         endif
      endif

      if (iequ == 5) then
         if (icom == 5) then
            do i=1,nb
               pux(i)= p15_5x(i)
            enddo
         endif
      endif

      if (iequ == 6) then
         if (icom == 6) then
            do i=1,nb
               pux(i)= p16_6x(i)
            enddo
         endif
      endif


      endif

!*********************************************************************
!   RAND 4                                                       ****
!*********************************************************************
!     unterer rand:
      if (irand == 4) then

      if (iequ == 1) then
         if (icom == 1) then
            do i=1,nb
               pu(i)= p1_1(i)
            enddo
         endif
         if (icom == 3) then
            do i=1,nb
               pux(i)= p1_3x(i)
            enddo
```

```
   endif
endif

if (iequ == 2) then
   if (icom == 2) then
      do i=1,nb
         pu(i)= p18_2(i)
      enddo
   endif
   if (icom == 6) then
      do i=1,nb
         pu(i)= p18_6(i)
         puy(i)= p18_6y(i)
      enddo
   endif
endif

if (iequ == 3) then
   if (icom == 2) then
      do i=1,nb
         pu(i)= p2_2(i)
      enddo
   endif
   if (icom == 3) then
      do i=1,nb
         puy(i)= p2_3y(i)
      enddo
   endif
endif

if (iequ == 4) then
   if (icom == 3) then
      do i=1,nb
         pu(i)= p3_3(i)
      enddo
   endif
   if (icom == 4) then
      do i=1,nb
         pu(i)= p3_4(i)
      enddo
   endif
   if (icom == 5) then
      do i=1,nb
         pu(i)= p3_5(i)
      enddo
   endif
   if (icom == 6) then
      do i=1,nb
         pu(i)= p3_6(i)
      enddo
   endif
endif

if (iequ == 5) then
```

```
         if (icom == 4) then
            do i=1,nb
               pu(i)= p19_4(i)
            enddo
         endif
         if (icom == 5) then
            do i=1,nb
               puy(i)= p19_5y(i)
            enddo
         endif
         if (icom == 6) then
            do i=1,nb
               pu(i)= p19_6(i)
            enddo
         endif
      endif

      if (iequ == 6) then
         if (icom == 6) then
            do i=1,nb
               pu(i)= p20_6(i)
               puy(i)= p20_6y(i)
            enddo
         endif
      endif


      endif
!
!**** end of calculation
!     ------------------
!
      r e t u r n
!-----end of FDEMU4---------------------------------------------------------
      e n d
```