Modeling and Analysis of Non-Diffusive Structural Phase Transitions in Crystals

Patrick Dondl

December 20, 2002

To my parents

Acknowledgements

I thank Dr. Johannes Zimmer for his outstandingly great supervision and support in the course of production of this work and for the enormous amount of enlightening comments, dicussions and corrections. I thank PD Dr. Walter Schirmacher for his great support and his insightful suggestions. I am very greatful to Prof. Kaushik Bhattacharya for generous support and encouragement and to the Division of Engineering and Applied Science at California Institute of Technology for hospitality.

Contents

Chapter 1

Introduction

1.1 The Shape Memory Effect

1. Macroscopic Phenomenology
Shape memory alloys are materials that, after they have been plastically deformed, return to their original shape when they are heated up above a certain temperature; they seem to "remember" this shape. Being cooled down again, the material remains in the original shape. Figure 1.1 illustrates this behavior. As opposed to this *one way effect* there also is a *two way effect* where a later decrease in thermic energy leads to the deformed state again. More information on the two way effect which will not be discussed here can be found in [27].

The most commonly known alloys that show the shape memory behavior are Nickel-Titanium compounds and compounds based on Copper or Nickel, e.g. $Au_{23}Cu_{30}Zn_{47}$. Mostly Nickel-Titanium alloys have also found their way into applications, for they are bio-compatible and not corrosive. These applications of shape memory alloys often build on the fact that high forces can result from this behavior. The deformations that can be reverted are comparatively large, ranging up to 8% length difference. For further applications and material properties of shape memory alloys see, for example, [40].

2. Stress-Strain Relations
Macroscopically, shape memory alloys can be understood by their highly temperature dependent stress-strain relation. A typical low temperature stress-strain diagram can be seen in Figure 1.2 (a). After an elastic region (1) below a certain stress one can see a yield surface as for a plastic material. This is the horizontal line (2) in the diagram where the strain can increase without increased stress. Other than for a regular plastic material there is another elastic region (5) when the strain increases even more. For this reason, and the fact that for a real plastic material the yield would come from dislocations in the system which one ideally does not have here, this behavior is said to be *quasi-plastic*. Another important material property is hysteresis. When the stress

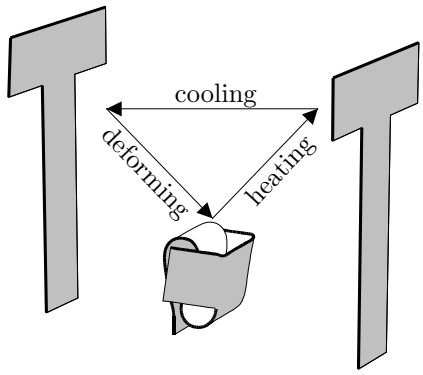

Figure 1.1: Macroscopic illustration of the shape memory effect.

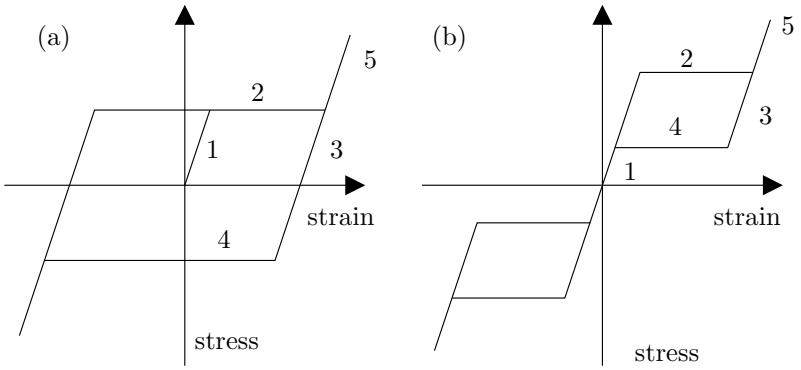

Figure 1.2: Stress-Strain relation at (a) $\theta < \theta_c$ and (b) $\theta > \theta_c$

is reduced to zero again, a deformation remains and one obtains a hysteresis loop. Under pressure this deformation can, after another elastic region (3), be returned to the original shape (4). The properties change dramatically if the temperature is increased above a certain critical value θ_c. A high temperature diagram is shown in Figure 1.2 (b). One can still see the elastic region (1), the yielding (2) and the second elastic region (5), but now complete unloading of the body leads back to zero strain. This behavior is called *pseudoelastic*. It is similar to an elastic material because zero stress always leads to zero strain, but in the course of loading and unloading one goes through a hysteresis loop via (3) and (4) which does not exist in an elastic material.

Considering these diagrams, the shape memory effect can easily be understood. Starting with an undeformed body below the temperature θ_c, a load is applied such that the material undergoes a plasticity like deformation. After unloading, the body remains strained. Now the temperature is increased above the transition temperature θ_c. The material must return to the undeformed state since, without stress, this is the only stable state.

3. Crystallographic Background

Microscopically, the shape memory effect is a solid-solid phase transition between phases of different symmetries. In the common case, described above, the high temperature phase is the phase with the higher symmetry, called *austenite*. At lower temperature phases with lower symmetry, known as *martensites*, become stable. In the load-free situation, due to symmetry properties, there is always more than one stable martensitic phase. These martensite *variants* relate to each other by the symmetry operations of the high symmetry phase, see, for example [7], Section 4.3.

To illustrate the underlying processes one can for example consider a cubic-tetragonal phase transition as it is found in Indium-Thallium alloys. This phase transition has been examined in [4]. Figure 1.3 describes the crystallographic process. Here, the austenitic phase is cubic, and there are three tetragonal martensitic variants. In two space dimensions, there are only two variants. In Figure 1.3 (a), only these two martensitic variants are stable. One of them, variant 1, is stretched along the x-axis and compressed along the y-axis, compared to the austenitic phase. For variant 2, the situation is exactly reversed in the sense that one has a compression along the x-axis and the lattice is stretched along the y-axis. In the unstrained reference configuration, both variants occupy an equal volume fraction. They are typically arranged in layers, this behavior is known as *twinning*. Now, by applying stress, as in Figure 1.3 (b), one can transform the variant that is less compatible with the stress to the one that is aligned with the applied stress. This, due to the hysteresis, macroscopically leads to the yielding in the plastic strain region. In our example we apply a pulling stress along the y-axis. This leads to a favoring of variant 2 and the material is deformed. After unloading, the body remains, due to its hysteresis, in this macroscopically deformed state. Then, in Figure 1.3 (c), the temperature is increased above the critical temperature θ_c. The high symmetry (cubic) austenite phase is the stable phase and the body macroscopically returns to the shape

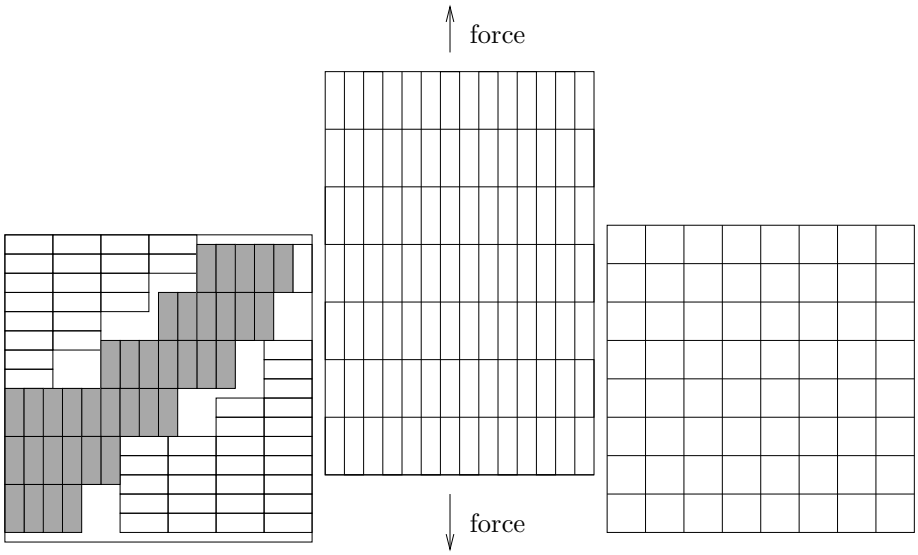

Figure 1.3: Microscopic illustration of the shape memory effect.

where the two martensitic phases were of equal volume fraction. After cooling the specimen down below the critical temperature, the twinning recurses with equal mixing of the two variants by the forming of phase boundaries between the variants and the process can start over again from Figure 1.3 (a).

1.2 Models for Shape Memory Alloys

A natural distinction can be made between macro-, meso- and micro models for shape memory alloys. In [45] one can find a survey on models for shape memory alloys that is structured in such a way.

Macro models are based on the principles of thermodynamics. The variables that are sought after are the temperature and the volume fraction of the austenite and martensite phases. In particular, these include the models by Achenbach and Müller [37], [2], [1] which were further developed by Müller and Seelecke [38]. These models are mainly used to compute macroscopic effects of the phase transition, i.e., shape changes of specimen. The macro model has also been extended to a two- or three-dimensional space. A mathematical analysis does not exist for either of them.

Models to study measure-valued solutions resolve the problem on an intermediate meso-scale. The problems that arise from using a nonconvex energy functional are treated by giving solutions as a probability (Young-) measure describing infinitely fine microstructure. See, for example [29] and [39].

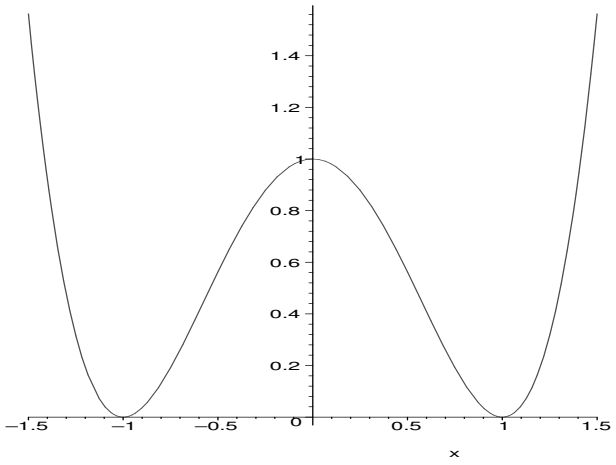

Figure 1.4: Double well potential in one Dimension, x-axis is strain.

A class of macroscopic models, so-called Frémond models use an energy function for each phase. These energies are added according to the volume fractions which gives the total energy. The result is a coupled system of differential equations for the temperature, displacement and the volume fractions. Existence and uniqueness have been studied in [17] for one space dimension, and in [16] for a higher dimensional space.

A different approach using Landau- and Landau-Ginzburg theory is to create a multi-well free energy, depending on an order parameter of the phase transition. In shape memory alloys this order parameter can be taken to be the strain. A one-dimensional illustration of such an energy can be found in Figure 1.4. Additionally, the energy of the system can have a higher order term to represent a surface energy. An early reference for these models is Falk [22]. The equations of motion describing dynamics of such a system then can be found as the Euler-Lagrange equations for this potential and a kinetic energy. For a displacement u and the temperature θ, a Landau-Ginzburg potential Φ can have the form

$$\Phi(u_x, u_{xx}, \theta) := \Phi_0(\theta) + \alpha\theta\Phi_1(u_x) + \Phi_2(u_x) + \frac{\gamma}{2}u_{xx}^2,$$

and σ is defined as

$$\sigma(u_x, \theta) := \frac{\partial\Phi(u_x, u_{xx}, \theta)}{\partial u_x}.$$

The resulting equations describing the dynamics of the system are

$$\begin{aligned} \rho u_{tt} &= (\sigma(u_x, \theta))_x - \gamma u_{xxxx} + f, \\ -\theta\Phi_0''(\theta)\theta_t &= \kappa\theta_{xx} + \theta\sigma_\theta(u_x)u_{xt} + g. \end{aligned}$$

with material constants α, ρ, γ and κ. For this one dimensional problem, adding suitable initial and boundary conditions, existence and uniqueness have been shown in [49].

The same Landau strain energy functionals can be used in a theory without the Ginzburg surface energy term $\frac{\gamma}{2}u_x^2 x$ but with viscosity added to the equation of motion. Using the material parameter β for the viscosity such a system of equations has the form

$$\begin{aligned} \rho u_{tt} &= (\sigma(u_x,\theta)+\beta u_{xt})_x + f \\ -\theta\Phi_0''(\theta)\theta_t &= \kappa\theta_{xx} + \theta\sigma_\theta(u_x)u_{xt} + \beta u_{xt}^2 + g. \end{aligned}$$

Existence and uniqueness for this system have been studied, for example, in [14], [19] and [28]. Similar equations for a wider range of systems—not only shape memory alloys—have been studied in [53]. In [43], one can find results on the asymptotic behavior of a one dimensional system. It is also possible to add the Ginzburg surface energy term $\frac{\gamma}{2}u_{xx}^2$ to the model again. For such a system existence and uniqueness have been proven in [26]. In [13], the asymptotics for $\gamma \to 0$ are examined. All these models consider only one space dimension.

Fewer results exist for the two- or three-dimensional case of these systems. First of all, due to symmetry constraints, the creation of a two- or three-dimensional multiwell Landau potential becomes a non trivial task, especially if a triple point is considered (see [52], [21]). A solution of this problem, based on a theory by Zimmer (see [56], [55]) is presented in Chapter 4.

In [41], a three dimensional generalization of the Falk model is presented by Paus. Local existence and uniqueness are proven for the system

$$\begin{aligned} u_{tt} &= \mathrm{Div}(\sigma(\nabla u,\theta)) - \gamma\triangle^2 u + f, \\ -\theta\Phi_0''(\theta)\theta_t &= \kappa\triangle\theta + \theta\sigma_\theta(\nabla u)\cdot\nabla u_t + g, \end{aligned}$$

an energy of the form

$$\Phi(\nabla u, \triangle u, \theta) := \Phi_0(\theta) + \alpha\theta\Phi_1(\nabla u) + \Phi_2(\nabla u) + \frac{\gamma}{2}|\triangle u|^2,$$

and homogeneous boundary conditions for u and $\triangle u$ and general Dirichlet boundary conditions for θ on a bounded spatial region Ω and initial data for u, u_t and θ.

Pawłow and Żochowski prove existence and uniqueness for a two- and three-dimensional system with viscosity and surface energy in [42] using a parabolic decomposition of the differential equations. This method makes an added viscosity necessary, it does not work for a system without dissipation.

A two or three dimensional system without the Ginzburg surface energy term but with viscosity is examined by Zimmer in [56] where existence of a solution is proved.

In this work we present a generalization of the work by Spies (see [48]) to two or three space dimensions. As there, the proof is built on a more general work by Chen and Triggiani presented in [12]. It was possible to obtain a global

existence and uniqueness result for systems both with *and* without viscosity. The system examined here is assumed to be isothermal, even if a time dependent temperature field could easily be added. As in [48] the linear part of the equations would decouple and the proof would not change substantially.

1.3 Numerical Computations

One dimensional numerical examinations of the systems of differential equations for Landau and Landau-Ginzburg models are presented in [11] and [3].

In the field of two- or three-dimensional simulations of elastic bodies with microstructures one has to distinguish anti-plane shear models and "true" two- or three-dimensional computations. Anti-plane shear simulations consider a displacement $u\colon \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$. The displacement is assumed to be orthogonal to the plane in which Ω lies. A simulation using this model showing formation of microstructure can be found in [51]. There, a finite difference scheme is used to compute the dynamics of a system with viscosity but without surface energy.

Of course, true two- or three-dimensional simulations regarding a displacement field $u\colon \Omega \to \mathbb{R}^n$ with $\Omega \subset \mathbb{R}^n$ and $n = 2, 3$ are computationally much more expensive. A finite difference scheme with viscosity and capillarity is implemented in [44]. They use a polynomial three well energy for a system in two space dimensions. The time development of the formed microstructure is also evaluated.

A more common approach for computations in elasticity is the use of finite elements. In [32], this method is used to study a fully three-dimensional system with the Erickson-James energy density (see [20]). There, viscosity is used to dissipate energy from the system, but no surface energy is considered. This is changed in the later work [31]. There, the regularity of the finite elements used is increased—this is necessary because the surface energy term has higher derivatives—but they are still not fully C^1, the displacement field is not everywhere continuously differentiable. This means that the finite elements are nonconforming and convergence is not necessarily guaranteed. In [30] the same approach is applied, but a temperature field is introduced and the elastic energy is made temperature dependent.

In this work, a two dimensional finite element approach is used. As opposed to the referenced finite element simulations a fully conformal C^1 element has been implemented. This allows to conduct computations without a priori assumptions on the alignment of phase boundaries where the surface energy then has to be treated separately as in [5]. Great care was taken to make the simulation code as easily understandable as possible using the `WEB` documentation system (see [33]). The program is also very flexible and can be used for a wide variety of dynamic systems by making it very convenient to plug in different elastic energies. This shows in the different types of simulations carried out, including a traveling phase boundary obstructed by a non-transforming element, the relaxation of a triple point material and the examination of the dependence

of the length scale of microstructure on the surface energy and specimen size.

1.4 Outline of the Work

First, in Chapter 2, a potential and kinetic energy is presented for the elastic system and the equation of motion is derived from that. Also, a non-dimensionalization of the equation of motion is carried out there. Then, for the initial-boundaryvalue problem belonging to this differential equation, existence and uniqueness of a solution are proved in Chapter 3. For a special material with a triple point, namely zirconia (ZrO_2), an energy functional is constructed in Chapter 4. The numerical part of this work is presented in Chapter 5. Some remarks on notation can be found in Appendix A. In Appendix B there is an explanation concerning the function spaces used in this work. The program code together with its documentation is presented in Appendix C.

Chapter 2

Derivation of the Equation of Motion

2.1 The Continuum Model

We will be concerned with the dynamics in crystals, therefore it is natural to start with a lattice model. Given a single point o, an n-dimensional lattice ($n = 2, 3$) can be generated by translation with a set of n vectors $\{e_i\}_{1 \leq i \leq n}$ in $\mathbb{R}^n$. The lattice then consists of the points

$$\left\{ x \,\middle|\, x = o + \sum_i \nu_i e_i, \text{ with } \nu_i \in \mathbb{N} \right\}.$$

One can now turn from the lattice description of the crystal to a continuum model. To study the dynamics of a body in $\mathbb{R}^n (n = 2, 3)$, we first choose a reference configuration $\Omega \subset \mathbb{R}^n$. The position of the material body at time t is described by the function

$$\begin{aligned} y\colon \Omega \times \mathbb{R} &\to \mathbb{R}^n \\ y\colon (x, t) &\mapsto y(x, t). \end{aligned}$$

To avoid physically impossible situations like self-penetration it is necessary for y to be injective and orientation conserving. Furthermore, if one assumes y to be sufficiently smooth, then it is possible to compute the spatial derivative $F = \nabla y$, $F_{ij} = \frac{\partial y_j}{\partial x_i}$. The tensor F is called the *deformation gradient*. The orientation conservation then transforms into the condition $\det(F) > 0$. Assume that underlying each point $x \in \Omega$ there is a lattice with lattice vectors $\{e_i(x)\}$. The Cauchy-Born assumption states that locally the lattice vectors transform according to the deformation gradient F such that

$$e_i(x) \mapsto F(x) e_i(x).$$

In the further calculations, it will be more convenient to use the displacement

$$u(x) := y(x) - x.$$

It is also necessary to introduce the variables $u_t(x,t)$ (the velocity) and $u_{tt}(x,t)$ (the acceleration).

2.2 The Energy Function

We will now turn to the task of constructing a kinetic and potential energy depending on u whose Euler-Lagrange equations will describe the dynamics of our elastic system.

Kinetic Energy: The kinetic energy term obviously has to be the integral

$$E_{\text{kin}} = \int_\Omega \frac{\rho(x)}{2} |u_t(x,t)|^2 \mathrm{d}V, \tag{2.1}$$

with $\rho(x)$ being the mass density of the specimen.

Strain Energy: The deformation gradient $F(x)$ at a certain point $x \in \Omega$ locally describes the strain of the specimen, therefore the lattice parameters. For hyperelastic materials the strain energy density locally only depends on the value of F. Therefore the strain energy term for the specimen has to be of the form

$$E_{\text{strain}} = \int_\Omega W(\nabla u(x,t))\mathrm{d}V = \int_\Omega W(F(u(x,t))\mathrm{d}V, \tag{2.2}$$

where $W(F)$ is the strain energy. Now one can also define the *stress tensor* σ,

$$\begin{aligned} \sigma(F) &:= \frac{\partial W(F)}{\partial F} \\ \text{or, in components, } \sigma_{ij}(F) &= \frac{\partial W(F)}{\partial F_{ij}}. \end{aligned} \tag{2.3}$$

For the description of phase transformations this energy must have multiple minima. We will take a closer look at the properties of this function in Chapter 4.

Surface Energy: Up to this point the energy would provoke the formation of infinitely fine microstructure, because, considering a differentiable deformation u, in general the infimum of the energy is not attained—not even if one considers weak differentiability (see Appendix B). This leads to the lack of an inherent length scale in the solutions. The use of Young measure solutions (see [29], [39]) would be necessary. To avoid this, one

must penalize the creation of phase boundaries, that is penalize the spatial change in ∇u. A commonly used surface energy term

$$E_{\text{surface}} = \int_\Omega \frac{\gamma}{2} |\triangle u(x,t)|^2 \mathrm{d}V \tag{2.4}$$

will provide this property.

2.3 The Equation of Motion

Now one can write down the Lagrange function for this system, which is

$$\begin{aligned} L(x,t) &= E_{\text{kin}} - E_{\text{pot}} \\ &=: \int_\Omega \mathcal{L}(x,t)\mathrm{d}V \\ &= \int_\Omega \frac{\rho(x)}{2}|u_t(x,t)|^2 - W(\nabla u(x,t)) - \frac{\gamma}{2}|\triangle u(x,t)|^2 \mathrm{d}V \end{aligned}$$

The Lagrange density $\mathcal{L}$ in this function leads to the equation of motion (see [23], the Div operator used in the third line is explained in Appendix A)

$$\begin{aligned} & 0 = \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{L}_{u_t} + \sum_i \frac{\mathrm{d}}{\mathrm{d}x_i}\mathcal{L}_{u_{x_i}} - \sum_{i,j} \frac{\mathrm{d}^2}{\mathrm{d}x_i x_j}\mathcal{L}_{u_{x_i x_j}} \\ \Leftrightarrow \quad & 0 = \rho(x)u_{tt} - \sum_i \frac{\mathrm{d}}{\mathrm{d}x_i}\frac{\partial W(F)}{\partial u_{x_i}} + \gamma \sum_i \frac{\mathrm{d}}{\mathrm{d}x_i^2} u_{x_i x_i} \\ \Leftrightarrow \quad & 0 = \rho(x)u_{tt}(x,t) - \mathrm{Div}(\sigma(F(x))) + \gamma \triangle^2 u(x,t). \end{aligned}$$

Phenomenologically, another term can be added in order to get dissipation is a viscosity . Obviously, this can not easily be written in a potential form, therefore it is necessary to introduce this term directly in the equation of motion. The viscosity that will be used here is of the form

$$\beta \triangle u_t(x,t).$$

The term does not really describe any physical process. Instead it is an artificial dissipative term in the equation that nevertheless is widely used. Another disadvantage of this term is that it is not frame invariant.

It is also possible to add a body force $f(x,t)$ to the equation of motion. Therefore, the entire dynamics of the system we want to examine is governed by the equation

$$\rho(x)u_{tt}(x,t) = \mathrm{Div}\sigma(F(x)) - \gamma\triangle^2 u(x,t) + \beta\triangle u_t(x,t) + f(x,t).$$

As usual the reference configuration Ω will be taken to be an unstrained austenite material. Therefore it is a reasonable assumption that the mass density is homogeneous throughout the body,

$$\rho(x) = \rho = \text{const.}$$

Even if a different reference configuration was chosen, the mass density is not essential for the dynamics of the systems investigated here.

2.4 Non-Dimensionalization

It is possible to choose a scale for the units *kilogram*, *meter* and *second* in the differential equation. A suitable normalization of these units will give us the opportunity to set most of the coefficients in the differential equation equal to 1. Consider the transformation

$$\begin{aligned} \rho &\mapsto \tilde{\rho} := a\rho \\ t &\mapsto \tilde{t} := bt \\ x &\mapsto \tilde{x} := cx. \end{aligned}$$

This leads after a division by a to the transformed equation of motion (differential operators with tilde denote differentiation with respect to the transformed coordinates)

$$\begin{aligned} \frac{\rho}{b^2} u_{\tilde{t}\tilde{t}}\left(\frac{\tilde{x}}{c}, \frac{\tilde{t}}{b}\right) &= \widetilde{\mathrm{Div}} \frac{1}{a}\sigma\left(\frac{1}{c}F\left(\frac{\tilde{x}}{c}\right)\right) - \frac{\gamma}{ac^4}\widetilde{\triangle}^2 u\left(\frac{\tilde{x}}{c}, \frac{\tilde{t}}{b}\right) \\ &\quad + \frac{\beta}{abc^2}\widetilde{\triangle} u_{\tilde{t}}\left(\frac{\tilde{x}}{c}, \frac{\tilde{t}}{b}\right) + \frac{1}{a} f\left(\frac{\tilde{x}}{c}, \frac{\tilde{t}}{b}\right). \end{aligned}$$

So first one can include the factor $\frac{1}{a}$ in the strain energy density W, for example, to normalize the elastic moduli, and also in the additional body force term f. Then one can obviously choose the parameters b and c such that $\frac{\rho}{b^2} = \frac{\gamma}{ac^4} = 1$. Then, the quantity $\frac{\beta}{abc^2}$ is fixed and cannot be normalized. From now on, this transformation is implicitly assumed, and for simplicity, the original notation without the tilde is used. This leads to the non-dimensionalized equation of motion

$$u_{tt}(x,t) = \mathrm{Div}\sigma(x,t) - \triangle^2 u(x,t) + \beta\triangle u_t(x,t) + f(x,t)$$

which will be used from now on.

Chapter 3

Existence of a Solution

3.1 Introduction

We want to prove the existence and uniqueness of a solution for the system derived in Chapter 2 in three steps. First, in Section 3.3, the equation of motion derived in Section 2.3 is used to formulate the initial-boundary value problem. In Section 3.4, the partial differential equation is transformed into an ordinary differential equation on a suitable function space. Some properties of the resulting linear differential operator are shown there. By transforming the partial differential equation (PDE) into an ordinary differential equation (ODE) one can deal with it a lot easier—for a regular ODE on a finite dimensional space, the existence of a solution is known, provided the possible nonlinear part is Lipschitz continuous. Our equation, however, is defined on an infinite dimensional Banach space and one has to invoke some more functional analysis to solve this problem. To achieve this the notion of semigroups has to be introduced.

The results presented here generalize those obtained in the first part of [48] to the higher dimensional case (space dimension $n = 2, 3$). The proof given here follows the lines of the one in [48]. However, the method used here has so far not been used in the higher dimensional case, where few existence results are known. Additionally, more sophisticated estimates are needed to control the nonlinearity. The deepest existence result for nonconvex energies and the non isothermal case with capillarity is the recent one in [42]. There, a parabolic decomposition of the equation of motion is used. This technique requires an added viscosity and does therefore not completely cover our case. The system considered by Pawłow and Żochowski in [42] is more complex than our model and they include a temperature field. It would be worthwhile to investigate whether our methods apply to their system. However, the main assumption in our theorem is stronger than that in [42], as we need quadratic growth rate for the energy functional W in the region of large strains. From the physical point of view, such an assumption is not restrictive, because—with physically reasonable initial conditions—it is not possible to get into the region of strains

where this behavior of W would show. Another advantage of the proof presented here is that it allows to derive eigenvectors and -values of the linear part of the equation for suitable domains. This eigenvalue expansion can, for example, be used in a numerical computation.

Finally, one has to consider the nonlinear part of the differential equation and show that this function is Lipschitzian with respect to a suitable norm. This is achieved in Section 3.5.

3.2 Semigroups

Consider the ODE

$$\dot{x}(t) = Ax(t)$$

in $\mathbb{R}^n$, with A being a linear map from $\mathbb{R}^n$ to $\mathbb{R}^n$ and $x(0) = x_0$. This Cauchy problem obviously has the solution

$$x(t) = \exp(At)x_0$$

for $t \in \mathbb{R}$. The mappings $\exp(At), t \geq 0$, have the properties

$$\exp(At_1) \circ \exp(At_2) = \exp(A(t_1 + t_2))$$

and

$$\exp(A0) = \mathrm{Id} = \exp(A(-t)) \circ \exp(At).$$

Definition 1 *Let $S(t)$ for $t \in \mathbb{R}$ a family of operators over a Banach space X. $S(t)$ is called a one-parameter group if the following conditions are satisfied:*

$$\begin{aligned} S(t+s) &= S(t)S(s) \text{ for all } t, s \in \mathbb{R}, \\ S(0) &= 0. \end{aligned}$$

Note that the existence of the inverse element is thus inherited from the additive group of real numbers. One can now see that $S(t) := \exp(At)$ for $t \in \mathbb{R}$ has the properties of a one-parameter group. The operator A is called the *generator* of $\{S(t)\}_{t\in\mathbb{R}}$.

However, it is not necessarily the case that a definition of $S(t)$ makes sense for $t < 0$. But, given the existence of a solution for the ODE, one can speak of $\{S(t)\}_{t\geq 0}$ as a *semigroup*.

Definition 2 *A one-parameter semigroup over a Banach space X is a family of bounded linear operators $S(t), t \geq 0$, each mapping $X \to X$, with the following properties:*

$$\begin{aligned} S(t+s) &= S(t)S(s) \text{ for all } t, s \geq 0, \\ S(0) &= 0. \end{aligned}$$

$S(t)$ is called strongly continuous if it is a strongly continuous function of t. $S(t)$ is called analytic if it is analytic in t.

For a differential linear operator A the situation is more difficult, because it is *a priori* not clear if operators that are defined to act on infinite dimensional Banach spaces generate a semigroup in a similar way. But as we will see there are theorems in functional analysis that help us to achieve exactly that and also define $\{S(t)\}_{t\geq 0}$ in a useful way.

3.3 The Initial-Boundary Value Problem

The problem that has been motivated in the previous sections can be summarized to the following equations. Let Ω an open, bounded and nonempty region in $\mathbb{R}^n$, with $n = 2$ or $n = 3$ with C^2 boundary $\partial\Omega$. Further let $g\colon \partial\Omega \to \mathbb{R}^n$ and $u_0, v_0\colon \Omega \to \mathbb{R}^n$ given functions of specified regularity, $T \in \mathbb{R}^+$ a fixed but finite final time and $f = f(u,t)$ a Lipschitzian function.

Now for a function $u\colon \Omega \times [0,T]$ one can consider the initial-boundary value problem

$$
\begin{aligned}
u_{tt} &= \beta\triangle u_t - \triangle^2 u + \mathrm{Div}(\sigma(\mathrm{D}u)) + f, && (3.1)\\
u &= 0 \text{ on } \partial\Omega \times [0,T], && (3.2)\\
\triangle u &= g \text{ on } \partial\Omega \times [0,T], && (3.3)\\
u &= u_0 \text{ on } \Omega \times 0, && (3.4)\\
u_t &= v_0 \text{ on } \Omega \times 0. && (3.5)
\end{aligned}
$$

Here, equation (3.1) is the differential equation of motion, (3.2) and (3.3) describe the boundary values for u and $\triangle u$. The initial conditions for the value of u and its time derivative are given in (3.4) and (3.5). A temperature field can be added to the set of equations. In this case the temperature dependence would essentially decouple from the equations here and the results would be similar.

3.4 Transformation of the PDE

In order to formulate the system (3.1)–(3.5) as a Cauchy problem one defines the *state space* $Z := H_0^1(\Omega) \cap H^2(\Omega) \times L_2(\Omega)$ (see Appendix B),

$$
z(t) := \begin{pmatrix} u(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} u(t) \\ \dot{u}(t) \end{pmatrix}.
$$

The inner product $\langle \cdot, \cdot \rangle$ in Z is

$$
\langle z_1, z_2 \rangle = \left\langle \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \right\rangle := \int_\Omega \triangle u_1 \triangle u_2 \mathrm{d}V + \int_\Omega v_1 v_2 \mathrm{d}V.
$$

It can be shown that Z endowed with this inner product is a Hilbert space (see Appendix B). The corresponding norm in Z is denoted $||\cdot|| := \sqrt{<\cdot,\cdot>}$. This choice of the inner product has been made to ensure that the norm on Z

actually is the square root of the sum of E_{kin} and E_{surface}, the kinetic and the surface energy for a given u. Now let

$$\begin{aligned} A(\beta)z &= A(\beta)\begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} v \\ \beta\triangle v - \triangle^2 u \end{pmatrix} && (3.6) \\ &= \begin{pmatrix} 0 & \mathrm{Id} \\ -\triangle^2 & \beta\triangle \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} && (3.7) \end{aligned}$$

for $z \in \mathrm{dom}(A)$, the domain of definition of A (defined below). The β dependence will not be explicitly stated from now on. The domain of A is

$$\mathrm{dom}(A) := \left\{ \begin{pmatrix} u \\ v \end{pmatrix} \in Z \;\middle|\; \begin{array}{c} u \in H^4(\Omega), u|_{\partial\Omega} = \triangle u|_{\partial\Omega} = 0, \\ v \in H_0^1(\Omega) \cap H_2(\Omega) \end{array} \right\}. \tag{3.8}$$

The function g in (3.3) therefore has to be in H^2 and $g|_{\partial\Omega} = 0$. Now it is possible to write the initial-boundary value problem system (3.1)–(3.5) as

$$\begin{aligned} \dot{z}(t) &= Az(t) + \mathfrak{F}(z(t), t) \text{ for } 0 \le t \le T, && (3.9) \\ z(0) &= z_0 && (3.10) \end{aligned}$$

for $z(t) \in \mathrm{dom}(A)$. The nonlinear part has been collected in the function $\mathfrak{F}(t, z(t))$, which is defined as

$$\mathfrak{F}(z(t), t) := \begin{pmatrix} 0 \\ f_2(z(t), t) \end{pmatrix}, \tag{3.11}$$

where

$$f_2(z(t), t) = f_2\left(\begin{pmatrix} u(t) \\ v(t) \end{pmatrix}, t\right) := \mathrm{Div}(\sigma(\mathrm{D}u(t))) + f(u(t), t).$$

It can be shown that the domain of definition of A is dense in the space Z. This means that the linear differential operator A is densely defined in Z. Now it is possible to proof three theorems concerning the properties of the operator A.

Theorem 1 *The operator A is dissipative, that means that $\langle Az, z\rangle \le 0$ for all $z \in \mathrm{dom}(A)$.*

Proof: Let $z = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathrm{dom}(A)$. Then, by integration by parts and by using the fact that the boundary terms vanish due to the choice of boundary conditions $u|_{\partial\Omega} = \triangle u|_{\partial\Omega} = v|_{\partial\Omega} = 0$,

$$\begin{aligned} \langle Az, z\rangle &= \left\langle \begin{pmatrix} v \\ \beta\triangle v - \triangle^2 u \end{pmatrix}, \begin{pmatrix} u \\ v \end{pmatrix} \right\rangle \\ &= \int_\Omega \triangle v \triangle u \mathrm{d}V + \int_\Omega \beta\triangle v \cdot v \mathrm{d}V - \int_\Omega \triangle^2 u \cdot v \mathrm{d}V \\ &= -\beta \int_\Omega (\nabla v)^2 \mathrm{d}V. \end{aligned}$$

Thus, A is dissipative. Note that the equality holds if $\beta = 0$. $\square$

The notion of dissipativity becomes understandable if one considers that the norm $||\cdot||$ on Z is an energy norm. Since β is the coefficient of the viscosity term, it is also clear that the energy should be conserved if $\beta = 0$. Some properties of the adjoint operator A^* of A will be needed in the later proofs. It is especially important that A^* is defined on the same set as A is. Therefore, we collect some properties of A^*.

Theorem 2 *The domain* $\mathrm{dom}(A^*)$ *of the adjoint* A^* *of* A *is given by* $\mathrm{dom}(A^*) = \mathrm{dom}(A)$*, and for*

$$z = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathrm{dom}(A^*), \text{ one has}$$

$$A^* z = \begin{pmatrix} -v \\ \beta\triangle v + \triangle^2 u \end{pmatrix} = \begin{pmatrix} 0 & -\mathrm{Id} \\ \triangle^2 & \beta\triangle \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Proof: The proof will be divided into two steps. First it is shown that $\mathrm{dom}(A) \subset \mathrm{dom}(A^*)$ and that $A^* z = \begin{pmatrix} 0 & -\mathrm{Id} \\ \triangle^2 & \beta\triangle \end{pmatrix} z$. Then the opposite inclusion $\mathrm{dom}(A^*) \subset \mathrm{dom}(A)$ is shown.

i) We start with $\langle Az_1, z_2\rangle$ for $z_1, z_2 \in \mathrm{dom}(A)$ and show that this equals $\langle z_1, A^* z_2\rangle$, with A^* from the theorem. Let $z_2 = \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \in \mathrm{dom}(A)$. This yields for any $z_1 = \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} \in \mathrm{dom}(A)$, using partial integration with the boundary conditions $u|_{\partial\Omega} = \triangle u|_{\partial\Omega} = v|_{\partial\Omega} = 0$

$$\begin{aligned} \langle Az_1, z_2\rangle &= \left\langle \begin{pmatrix} v \\ \beta\triangle v_1 - \triangle^2 u_1 \end{pmatrix}, \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \right\rangle \\ &= \int_\Omega \triangle v_1 \cdot \triangle u_2 + \int_\Omega \left(\beta\triangle v_1 - \triangle^2 u_1\right) v_2 \\ &= \int_\Omega \triangle u_1(-\triangle v_2) + \int_\Omega v_1 \left(\beta\triangle v_2 + \triangle^2 u_2\right) \\ &= \left\langle \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \begin{pmatrix} -u_2 \\ \beta\triangle v_2 + \triangle^2 u_2 \end{pmatrix} \right\rangle. \end{aligned}$$

Therefore, if $z \in \mathrm{dom}(A)$ then $z \in \mathrm{dom}(A^*)$ and

$$A^* z = \begin{pmatrix} -v \\ \beta\triangle v + \triangle^2 u \end{pmatrix}.$$

ii) Now let

$$z = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathrm{dom}(A^*).$$

By definition of A^* there exists

$$\tilde{z} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} \in Z$$

such that for all

$$\zeta = \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \in \mathrm{dom}(A)$$

one has

$$\begin{aligned} 0 &= \langle A\zeta, z\rangle - \langle \zeta, \tilde{z}\rangle \\ &= \int_\Omega \triangle\zeta_2 \triangle u + \left(\beta\triangle\zeta_2 - \triangle^2\zeta_1\right) v \mathrm{d}V - \int_\Omega \triangle\zeta_1\triangle\tilde{u} + \zeta_2\tilde{v}\mathrm{d}V. \end{aligned}$$

By rearranging terms and integration by parts (considering boundary conditions as usual), this is leads to

$$0 = \int_\Omega -(\triangle\tilde{u}\triangle\zeta_1 + v\triangle^2\zeta_1)\mathrm{d}V + \int_\Omega (-\tilde{v}\zeta_2 + (\triangle u + \beta v)\triangle\zeta_2 \mathrm{d}V.$$

Since this equality must hold for all $\zeta \in \mathrm{dom}(A)$, both terms in the above equation must vanish. Therefore,

$$\begin{aligned} \int_\Omega (\triangle\tilde{u}\triangle\zeta_1 + v\triangle^2\zeta_1) &= 0 \text{ and} \\ \int_\Omega (-\tilde{v}\zeta_2 + (\triangle u + \beta v)\triangle\zeta_2 &= 0. \end{aligned}$$

These integrals can be used to define the second weak derivatives of v and $\triangle u$. We need the existence of those derivatives to make sure that $z = \begin{pmatrix} u \\ v \end{pmatrix}$ is in $\mathrm{dom}(A)$. Formally, an integration by parts yields (3.12) and (3.13). Instead, we use these two equations as a definition. For uniqueness of this definition see, for example, [10], Chapitre VIII.

$$\triangle v := -\triangle\tilde{u} \text{ a.e. and} \tag{3.12}$$

$$\triangle^2 u := \tilde{v} - \beta\triangle v. \text{ a.e.} \tag{3.13}$$

Therefore, $u \in \mathrm{dom}(\triangle^2) = H^4$ and $v \in \mathrm{dom}(\triangle) = H^2$. This proves the inclusion $\mathrm{dom}(A^*) \subset \mathrm{dom}(A)$.

Note that A is skew-selfadjoint ($A^* = -A$) for $\beta = 0$. □

Theorem 3 *i) For $\beta > 0$ the linear differential operator A, as defined in equations (3.6) and (3.8) generates an analytic contraction semigroup.*
ii) For $\beta = 0$, A generates a group of unitary operators.

Proof: *i*) To prove this one can use Theorem 1.2 in [12]. The way to achieve the necessary conditions to apply this theorem will be very similar to [48], Theorem

3.5. Let $\mathcal{A} = \triangle^2$ and $\mathcal{B} = \triangle$. Both of these operators are selfadjoint, and so $\mathcal{A}$ has a unique fractional power $\mathcal{A}^{\frac{1}{2}}$. Since $\mathcal{B}^2 = \mathcal{A}$, it is obvious that $\mathcal{A}^{\frac{1}{2}} = \mathcal{B}$. One can now see that the operator $A = \begin{pmatrix} 0 & \mathrm{Id} \\ -\mathcal{A} & \mathcal{B} \end{pmatrix}$ corresponds to the elastic system treated in [12] and that Theorem 1.2 there applies. Therefore A is the generator of an analytic semigroup.
ii) Since A is a skew-selfadjoint operator, $\frac{1}{i}A = -iA$ (i being the imaginary unit) is a self-adjoint operator. Therefore, by Stone's Theorem, (see [36], Theorem 35.1) one can see that A generates a strongly continuous group of unitary operators. □

3.5 The Nonlinear Part

To conclude the proof of existence of a solution it remains to show that the function $\mathfrak{F}(z,t)$ as defined in (3.11) is Lipschitzian in t and Lipschitzian with respect to the graph norm $||z||_A := ||z|| + ||Az||$ in z. This property is obviously dependent on the properties of the potential energy $W(F)$ (with $F = \nabla u$), namely on its growth rate in the region of large strains.

Theorem 4 *Let $W\colon F \mapsto W(F) \in \mathbb{R}$ a C^2 function and R a compact region containing the identity in* $\mathrm{Mat}(n,\mathbb{R})$. *Further let $W(F) = |F|^2$ for $F \notin R$. Then $\mathfrak{F}(z,t) := \begin{pmatrix} 0 \\ f_2(z(t),t) \end{pmatrix}$ with $f_2(z(t),t) = f_2\left(\begin{pmatrix} u(t) \\ v(t) \end{pmatrix}, t\right) :=$ $\mathrm{Div}(\sigma(F(u))) + f(u(t),t)$ and $\sigma(F(u)) = \frac{\partial W(F(u))}{\partial F}$ is Lipschitzian in t and Lipschitzian in z with respect to the graph norm of A.*

Proof: Since it was assumed that $f(u(t),t)$ is Lipschitzian in u and t, this part of the function $\mathfrak{F}$ is Lipschitzian. We only have to prove that $\mathrm{Div}(\sigma(F(u)))$ is Lipschitzian with respect to the graph norm $\left|\left|\begin{pmatrix} u \\ v \end{pmatrix}\right|\right|_A$. To achieve this, we first have to make sure that all derivatives of all the components of the tensor σ with respect to the elements of F are bounded. They are, being continuous functions on a compact set, certainly bounded in R. Outside of R, the function W is quadratic in the components of F, therefore σ is a linear function. So, the derivatives of σ are constant for $F \notin R$. Therefore, their absolute value is bounded on $\mathrm{Mat}(n,\mathbb{R})$. We denote the bound $C_{\sigma'}$.

Now one can examine the term $\mathrm{Div}(\sigma(F(u)))$. Denoting the columns of the tensor σ with σ_i, one can find

$$||\mathrm{Div}(\sigma(F(u))) - \mathrm{Div}(\sigma(F(\tilde{u})))|| = \left|\left|\sum_i \sigma_{i,x_i}(F(u)) - \sigma_{i,x_i}(F(\tilde{u}))\right|\right|$$

$$= \left|\left|\sum_i \sum_j \sum_k \sigma_{i,u_{j,x_k}}(F(u)) u_{j,x_k x_i} - \sigma_{i,\tilde{u}_{j,x_k}}(F(\tilde{u}))\tilde{u}_{j,x_k x_i}\right|\right|$$

$$\leq C_{\sigma'} \left|\left| \sum_i \sum_j \sum_k u_{j,x_k x_i} - \tilde{u}_{j,x_k x_i} \right|\right|$$
$$\leq C_{\sigma'} ||u - \tilde{u}||_{H^2}.$$

From [24] (Theorem 8.12) one can obtain that, for $u|_{\partial\Omega} = \tilde{u}|_{\partial\Omega}$ and $\partial\Omega$ of class C^2 one has

$$||u - \tilde{u}||_{H^2} \leq C_\Omega(||\triangle u - \triangle \tilde{u}||_{L^2} + ||u - \tilde{u}||_{L^2}.$$

Since

$$||\triangle u - \triangle \tilde{u}||_{L^2} \leq ||u - \tilde{u}||_A$$

the proof is completed. □

Chapter 4

Zirconia

4.1 Introduction

This chapter is concerned with modeling of the energetic landscape of a material with a triple point. Specifically, we carry out calculations for zirconia (ZrO_2). In this case, the three coexisting phases are tetragonal, orthorhombic and monoclinic, respectively. Figure 4.1 reproduces phase diagrams collected in [47]. The traditional approach to derive energy functions for such a material is to use polynomials with the proper invariance. Following a widespread terminology, we will call this approach Landau theory, even if Landau's original work was broader in scope, pointing out the necessity of multiwell energies. Truskinovsky and Zanzotto recently presented in [52] an energy function for ZrO_2. They use a Landau ansatz of lowest order. In [55] some inherent difficulties of the Landau approach are mentioned. This can be explained with an example which will show that even in the simplest case, polynomials are too rigid to derive energies with prescribed elastic moduli, energy barriers between the wells and growth rates. The lowest order polynomial for a one-dimensional function $f(x)$ with two minimizers $x = \pm 1$ and a maximum at $x = 0$ is given by $f(x) := C(x^2-1)^2$. Now, fitting the elastic modulus fixes the parameter C and therefore automatically determines the height of the energy barrier between the two minima. Vice versa, adjusting the energy barrier determines the elastic moduli. Of course, in reality, these two quantities are unrelated. One could use a higher-order polynomial ansatz to overcome these difficulties. But then, minima and maxima are usually degenerate, and the growth rate of the energy for large deformations, that already poses a problem for lowest order polynomial energies with many wells, will be even larger. The high forces associated with such a growth rate can easily lead to stability problems in numerical simulations. For these reasons, polynomials energy functions are usually too rigid for numerical simulations (see the discussion in Section 4.4).

Non-polynomial functions, such as splines, are suitable to resolve these problems, as long as they have the right symmetry. However, constructing such a

function is difficult if one works on the strain space. This problem is addressed in [55], where the orbit space method is presented as an alternative approach. The precise definition of the orbit space will be given later; intuitively, the orbit space takes care of all the symmetry requirements. It allows to define the essential part of the energy function in a convenient, geometric way on the orbit space. Here, we use a modular approach using piecewise defined functions.

We construct the function in three steps: First, we take special local functions to create the minima and allow for fitting of the elastic moduli. Second, the space between the minima is filled. One could use splines, but we use the solution of a finite element simulation that results in a piecewise defined C^1 function, giving control over the height of the energy barriers. Third and finally, we define the behavior for large strains with a third function. Note that all steps are independent of each other. Fitting of parameters, such as elastic moduli, will be comparatively simple, since one has a distinguished set of parameters for each minimum. The fitting will not influence the height of the energy barriers and the growth of the function for large strains, as it would for a polynomial (Landau) energy function. It is easy to change the height of energetic barriers and growth conditions for large strains.

This shows an advantage of the orbit space method. With this method, it is possible to use a variety of different techniques to construct energy functions. One could for example use splines instead of our finite element approach to fit the parameters. In this case, one would get an explicit representation of the energy as for a polynomial ansatz. Most of the computations would remain unchanged. However, we decided to present the finite element approach since it is a simple way to find energy functions. The idea of constructing surfaces by solving partial differential equations has been used in geometric modeling for some time, see for example [8].

Although the computations carried out are specific to the phase transformations in zirconia (or, apart from the fitting, any other material with a tetragonal-orthorhombic-monoclinic phase transition), the orbit space method can be applied to other phase transitions, as shown in [55].

The problem of fitting experimental data is addressed in [21], where the transformation mechanism is also carefully discussed. This chapter will build on [52] and [21]. In particular, we will make use of the data collected there.

The Chapter is organized as follows: In Section 4.2, it is shown that the phase transition can be analyzed in a two-dimensional framework. In Section 4.3, an energy function is constructed and fitted to the elastic moduli of the different phases. We close with a discussion in Section 4.4.

4.2 Reduction to a Two-Dimensional Problem

As usual, we take the high symmetry phase as reference configuration. For zirconia, this is the tetragonal phase, denoted T_3. To fix the notation, we list the elements of T_3, following [52] (the axes are shown in Figure 4.2; R_a^α stands

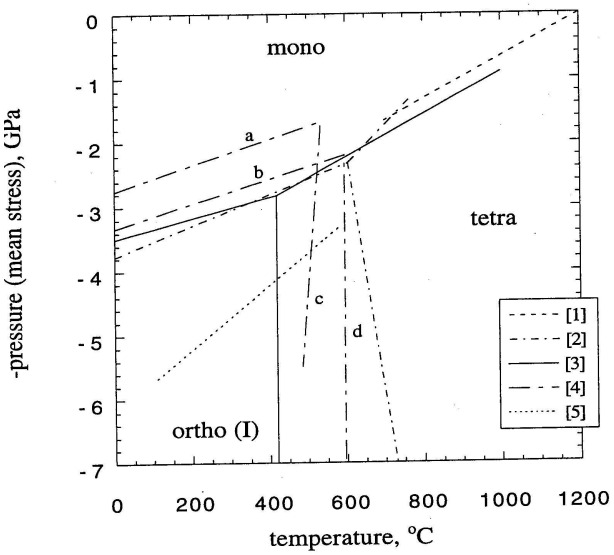

Figure 4.1: Phase diagrams, reproduced from [47]. Shown are regions of stability of tetragonal, orthorhombic and monoclinic phases of zirconia in temperature-pressure space, gathered from 5 references.

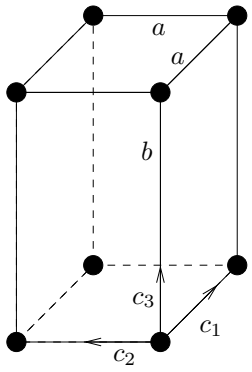

Figure 4.2: Tetragonal reference configuration. The axes c_1, c_2 and c_3 of rotations in the tetragonal point group are shown.

for the rotation with angle α and axis a):

$$T_3 = \left\{1, R^\pi_{c_1}, R^\pi_{c_2}, R^\pi_{c_3}, R^\pi_{c_1+c_2}, R^\pi_{c_1-c_2}, R^{\frac{\pi}{2}}_{c_3}, R^{\frac{3\pi}{2}}_{c_3}\right\}.$$

In zirconia, besides the tetragonal phase, both orthorhombic and monoclinic phases can be stable. This corresponds to orthorhombic and monoclinic subgroups of the tetragonal point group T_3. The orthorhombic subgroups are

$$O_{1,2,3} := \left\{1, R^\pi_{c_1}, R^\pi_{c_2}, R^\pi_{c_3}\right\}$$

and

$$O_{3,1\pm2} := \left\{1, R^\pi_{c_3}, R^\pi_{c_1+c_2}, R^\pi_{c_1-c_2}\right\}$$

(see [52]. Both orthorhombic groups form their own conjugacy class in T_3.

There are three conjugacy classes of monoclinic subgroups, from which we list one representative each:

$$M_{1,2} := \left\{1, R^\pi_{c_1}\right\}, \qquad M_{1\pm2} := \left\{1, R^\pi_{c_1+c_2}\right\}, \qquad M_3 := \left\{1, R^\pi_{c_3}\right\}.$$

Of course, there is also the trivial triclinic subgroup {Id}. A schematic representation of the point groups can be found in fig. 3 of [52].

As in [52], we assume the symmetry breaking in ZrO_2 occurs along the path

$$T_3 \longrightarrow O_{123} \longrightarrow M_3. \tag{4.1}$$

Experimental evidence for this assumption is collected in [52] and [21].

First let us take a closer look at symmetry breaking deformation gradients that, when applied to the tetragonal reference configuration, result in the desired phases with lower symmetry. According to (4.1), these are phases with point groups conjugate to O_{123} and M_3.

If F is a symmetry breaking deformation gradient generating a certain subgroup, all deformations

$$R^T F R \text{ for } R \in T_3 \tag{4.2}$$

generate a crystal of the same subgroup. Therefore, these are different variants of the lower order phases. This means these variants are related to each other by elements of the symmetry group of the high symmetry phase. It is well known that the number of variants is given by the quotient of the order of the high symmetry phase and the order of the low symmetry group (see, for example, [7], Section 4.3).

For O_{123}, there are two variants. It is easy to read off the associated deformation gradients from Figure 4.3 (see also [52]). They are

$$F = \begin{pmatrix} 1+u_{11} & & \\ & 1+u_{22} & \\ & & 1+u_{33} \end{pmatrix} \tag{4.3}$$

and

$$F = \begin{pmatrix} 1+u_{22} & & \\ & 1+u_{11} & \\ & & 1+u_{33} \end{pmatrix}. \tag{4.4}$$

Similarly, for M_3, there are four variants. It is easy to see that the corresponding deformation gradients F are given by the following four matrices:

$$\begin{pmatrix} 1+u_{11} & \pm u_{12} & \\ \pm u_{12} & 1+u_{22} & \\ & & 1+u_{33} \end{pmatrix} \tag{4.5}$$

and

$$\begin{pmatrix} 1+u_{22} & \pm u_{12} & \\ \pm u_{12} & 1+u_{11} & \\ & & 1+u_{33} \end{pmatrix}. \tag{4.6}$$

Finally, deformation gradients preserving the tetragonal symmetry are of the form

$$F = \begin{pmatrix} 1+u_{11} & & \\ & 1+u_{11} & \\ & & 1+u_{33} \end{pmatrix}. \tag{4.7}$$

From (4.3)–(4.7), it is immediate that the symmetry breaking takes place only in the two-dimensional subspace spanned by c_1 and c_2 (see Figure 4.2). In this plane, the tetragonal phase T_3 is characterized by a C_4 symmetry (the symmetry of a square). The two orthorhombic phases have a planar C_2 symmetry, since their restriction to the c_1-c_2 plane is a rectangle. Finally, parallelograms correspond to monoclinic variants. The planar point group of the parallelogram is again C_2. But now there are no longer three-dimensional rotations by 180 degrees along any axis in the c_1-c_2 plane that are a self-mapping of the monoclinic phase. This can be seen in Figure 4.3, since such a three-dimensional rotation acts as a reflection when restricted to the c_1-c_2 plane. The planar symmetry group C_4 is generated by a counterclockwise rotation by 90°. This generator will be denoted by σ,

$$\sigma := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

In summary: the preceding considerations have shown that the relevant part of the free energy density can be written as a function of an in-plane deformation in the c_1-c_2 plane.

4.3 Derivation of an Energy Function

We will use the orbit space approach, as presented in [55]. The key idea of this method is that the energy function can be written as a function defined on the 'orbit space'. Intuitively, the orbit space identifies all variants of the same phase, while separating unrelated variants. The precise definition is given later. This property of the orbit space automatically induces the right symmetries for the energy landscape, if the energy is defined on the orbit space (see below). In the special case considered here, where the symmetry breaking occurs in a two-dimensional subspace of $\mathbb{R}^3$, the derivation will be particularly easy.

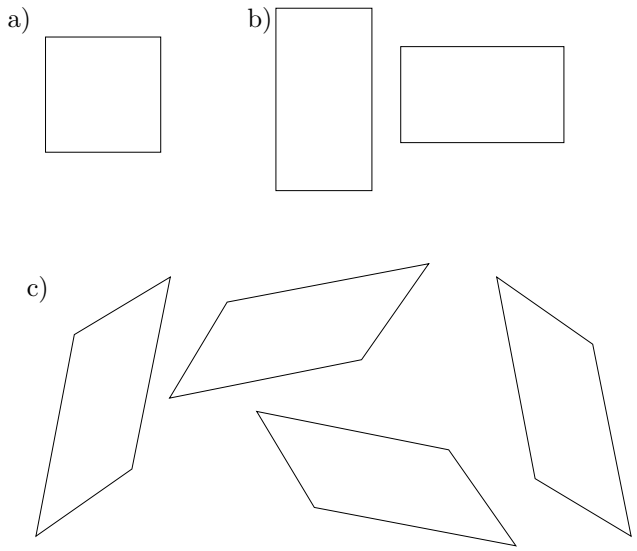

Figure 4.3: Variants: a) Tetragonal phase. b) Orthorhombic phase. c) Monoclinic phase.

It is a well known consequence of the axiom of frame indifference and the polar decomposition theorem that the energy function can be written as a function of $E := \frac{1}{2}\left(F^T F - \mathrm{Id}\right) \in \mathrm{Sym}(2,\mathbb{R})^+$. Here, E is the Green-St. Venant strain tensor, a symmetric real matrix with positive determinant. Point groups act on this set by conjugation:

$$\begin{array}{rcl} P \times \mathrm{Sym}(2,\mathbb{R})^+ & \to & \mathrm{Sym}(2,\mathbb{R})^+ \\ (P,E) & \mapsto & PEP^{-1}. \end{array}$$

It will be useful to see how the generator σ of the high symmetry point group operates on the strain tensor E. $\mathrm{Sym}(2,\mathbb{R})^+$ is not a vector space, but, concerning conjugation with the point group P, is an invariant subset of the vector space $\mathrm{Sym}(2,\mathbb{R})$. So, the operation can be extended to this vector space. Let us write the matrix E as

$$E = \begin{pmatrix} e_1 & \frac{1}{2}e_6 \\ \frac{1}{2}e_6 & e_2 \end{pmatrix}$$

with $e_i \in \mathbb{R}$. This is the upper left 2×2 submatrix of the 3×3 strain tensor of a three-dimensional deformation. The conjugation with $\sigma = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ yields

$$\sigma E \sigma^{-1} = \begin{pmatrix} e_2 & -\frac{1}{2}e_6 \\ -\frac{1}{2}e_6 & e_1 \end{pmatrix}.$$

We now use the fact that $\mathrm{Sym}(2,\mathbb{R})$ is isomorphic to $\mathbb{R}^3$, with components e_1, e_2, e_6. Then, the representation of σ is given by

$$\tilde{\sigma} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}.$$

Considering that $\tilde{\sigma}^2 = \mathrm{Id}$, it is immediate that the action of the point group on E is isomorphic to C_2. The orthorhombic and monoclinic subgroups coincide on this space and both act as identity.

The next step is to find the invariant polynomials in e_1, e_2 and e_6 under the action of the high symmetry point group. Instead of considering the infinite-dimensional vector space of invariant polynomials, we consider the algebra of invariant polynomials (that is, the multiplication of invariant polynomials is defined). It is a classic theorem due to Hilbert that this algebra is finitely generated (see, for example, Theorem 2.1.3 in [50], or [54] as the classical reference).

Here, the computation of a set of generators is particularly easy: By Theorem (A) in [15], for reflection groups in an n-dimensional space, there is a basis of the algebra of invariant polynomials that consists of n elements.

In principle, these invariants can be computed with special software packages, such as Singular, see [25]. Here, the situation is so easy that one can compute a basis by hand. First, one can literally guess 3 invariants:

$$\begin{aligned} \rho_1(e_1,e_2,e_6) &:= e_1+e_2 \quad \text{(the trace of } E\text{)}, \\ \rho_2(e_1,e_2,e_6) &:= e_1^2+e_2^2 \quad \text{(the radius squared)}, \\ \rho_3(e_1,e_2,e_6) &:= e_6^2. \end{aligned}$$

It is immediate that none of them can be expressed as a combination of the two remaining invariants. Therefore, they are independent. We only have to show that they form a basis. According to Chevalley's Theorem, there is a basis of 3 invariants. Since the polynomials listed above are of the lowest possible degree, they constitute this basis.

The fact that these three polynomials form a basis of the algebra of polynomials invariant under C_2 means that every such polynomial $\tilde{\rho} = \tilde{\rho}(e_1,e_2,e_6)$ can be written as $\tilde{\rho} = P(\rho_1,\rho_2,\rho_3)$, where P is a polynomial.

The *Hilbert map* ρ is defined as

$$\mathbb{R}^3 \to \mathbb{R}^3$$

$$(e_1,e_2,e_6) \mapsto (\rho_1(e_1,e_2,e_6),\rho_2(e_1,e_2,e_6),\rho_3(e_1,e_2,e_6)).$$

We will call the image of $\mathbb{R}^3$ under the Hilbert map the *orbit space*. The reason is that the Hilbert map identifies exactly the points corresponding to related variants, while separating points corresponding to other variants (see [55] for more details). One could say that the Hilbert map strips off the symmetry; the image is similar to a fundamental domain for the given group. The advantage of the orbit space, compared to a fundamental domain, is that no care has to be taken about identifications on the boundary.

To familiarize ourselves with the orbit space, and to proceed with the construction of the energy function, we locate the position of the different phases of zirconia in the orbit space $\rho(\mathrm{Sym}(2,\mathbb{R}))$.

The tetragonal reference configuration is particularly simple. Here, obviously $e_1 = e_2 = e_6 = 0$. Consequently, the tetragonal phase is mapped to the origin $(\rho_1(0,0,0), \rho_2(0,0,0), \rho_3(0,0,0)) = (0,0,0)$ of the orbit space.

Next, consider the orthorhombic phase. In Table 4.1, the data of one orthorhombic variant is given as $e_1 = 0.01, e_2 = 0, e_6 = 0$. By applying the tetragonal generator σ to this element, we find the other variant as $e_1 = 0, e_2 = 0.01, e_6 = 0$. The key point of the orbit space method is: both variants are mapped to the same point in the orbit space,

$$\begin{aligned} (0.01,0,0) &\mapsto (0.01,0.0001,0) \\ (0,0.01,0) &\mapsto (0.01,0.0001,0)\,. \end{aligned} \tag{4.8}$$

One can prove that this is always true: the Hilbert map collapses corresponding variants to one point in the orbit space, and no other variant is mapped to that point (see [55]) for more details). Therefore, the simple idea to construct energy functions is as follows: determine the image of the stable variants in the orbit space; define a function Φ on the orbit space with minimizers precisely at these points. The function $\Phi(\rho(E))$ will be an energy function with the correct symmetry. We will later address the question of how to fit elastic moduli and how to create energy barriers between stable phases.

The previous considerations motivate the idea to construct a function on the orbit space that has minima at the right places and fits the elastic moduli from experimental results. The locations of the minima in the strain space and in the orbit space can be found in Table 4.1. The values we used are taken from [21]. In Table 4.1, only one variant is shown (in the strain space; in the orbit space, corresponding variants are collapsed to one point). All other variants can be found by applying the generator $\tilde{\sigma}$ of the tetragonal symmetry group to the given variant. To create precisely one minimum in this three dimensional orbit space, one can use a C^1 function $f_\alpha = f_\alpha(x_1,x_2,x_3)$, with α being a given symmetric 3×3 matrix, as follows: If $\sum_{i,j}\alpha_{ij}x_ix_j < 1$, set

$$f_\alpha(x_1,x_2,x_3) := \frac{1}{500}\left[2\left(\sum_{i,j}\alpha_{ij}x_ix_j\right) - \left(\sum_{i,j}\alpha_{ij}x_ix_j\right)^2 - 1\right]$$

and $f_\alpha(x_1,x_2,x_3) := 0$ otherwise.

Since zirconia has at the critical temperature not only one, but three stable phases, we define a function on the orbit space as

$$\begin{aligned} \Phi_{\mathrm{minima}}(\rho_1,\rho_2,\rho_3) \;:=\; & f_\alpha\left(\rho_1,\sqrt{\rho_2},\sqrt{\rho_3}\right) \\ & + f_\beta\left(\rho_1-0.01,\rho_2-0.0001,\sqrt{\rho_3}\right) \\ & + f_\gamma\left(\rho_1-0.0534,\rho_2-0.00232,\rho_3-0.0256\right) \end{aligned}$$

(We will use this function to fit the experimental data available for the three phases). This function obviously has minimizers precisely at the locations where the three phases can be found in the orbit space. Since ρ_1 and ρ_2 are quadratic terms, one has to take the square root of the respective parameter for minima where this parameter is zero, since otherwise the second derivatives would vanish. Obviously, care has to be taken that the resulting function is still differentiable. Here the square root will cancel with the quadratic minimizer function f_α, as long as α is diagonal.

It is straightforward to choose the parameters in this function such that the function $\Phi_{\text{minima}}(\rho(e_1, e_2, e_6))$ defined as the composition of Φ_{minima} and the Hilbert map ρ fits the elastic moduli. Tables 4.2 and 4.3 show the values we used for the coefficients and the resulting elastic moduli. The values we fitted to are taken from [21]. As we can see, all the elastic moduli for deformations in the two dimensional subspace where the symmetry breaking occurs could be exactly reproduced. The function could also easily be extended to the entire three dimensional space.

So far, we have found an energy function with minimizers exactly at the right positions. Yet, it still looks phenomenologically wrong: there are no energy barriers between the minima. In the next step, we will glue in humps. The advantage of this cut-and-paste approach is that the height of the barrier can be arbitrarily chosen. When filling in a hump, care has to be taken: First, the resulting function has to be C^1. Second, we may not create an additional minimum. Third, the hump must not interfere with the neighborhoods of the minima where we already defined the energy function. Of course, it is cumbersome to construct such a function on the three dimensional orbit space. But since a triple point is characterized by three minima on the orbit space, there is a common plane in which the minima lie. Therefore, we first construct a linear function that maps the location of the tetragonal minimum to the origin, the orthorhombic minimum to the point $(1, 0, 0)$, the monoclinic minimum to the point $(0, 1, 0)$ and the direction orthogonal to the plane to the third direction. A short calculation shows that the matrix representing this linear function is given by

$$A := \begin{pmatrix} 100.1 & -13.49 & -207.7 \\ -0.02712 & 2.712 & 38.87 \\ 38.87 & -3887 & 271.2 \end{pmatrix}. \tag{4.9}$$

The new coordinates obtained by this transformation will be denoted $\bar{x}_i$. Assume for the moment that a function Φ_{fill} in the $\bar{x}_1$-$\bar{x}_2$ plane, defining energy barriers between the minima, is given. Then we can deal with the third direction $\bar{x}_1$ by setting

$$\Phi_{\text{3dfill}}(\bar{x}_1, \bar{x}_2, \bar{x}_3) := \Phi_{\text{fill}}(\bar{x}_1, \bar{x}_2) + \delta \bar{x}_3^2.$$

The parameter δ does affect the elastic moduli. Its numerical value can be found in Table 4.2. To construct a hump-function Φ_{fill} meeting all of the above requirements, we use a finite element method to solve the Laplace equation with appropriate boundary conditions and a few constrained degrees of freedom. The idea behind this is that the third requirement for the energy barriers, namely

C^1 smoothness, is the most restrictive one; by solving Laplace's equation, one has the maximum principle at hand to ensure that there are no minimizers or maximizers in the interior of the domain. Strictly speaking, one had to check whether the assumptions of the maximum principle are met; but here—in the two-dimensional space—it is easy to inspect the solution once it is found (Note that this is the only step in the construction where we do not write down an explicit expression for the function; if one wants to do so, splines or Bézier functions would be a natural choice to define an energy barrier).

To get a continuously differentiable solution, one can use the Bogner-Fox-Schmit rectangle (see for example [9], II.5.10) as element. Here a 8×8 element grid defining a rectangular region R with corner coordinates $(-1,-1)$, $(-1,2)$, $(2,-1)$, $(2,2)$ in the plane containing the minima constructed above is used. Note that all three minima are contained in the rectangular region R. We chose Dirichlet conditions on the entire boundary such that the function value is set to 0.004; partial derivatives are set to zero. All degrees of freedom for the three elements containing the coordinates $(0,0)$, $(1,0)$ and $(0,1)$ (the location of the minima) are set to be zero (i.e., there are no degrees of freedom for these nodes). These elements have also been carefully sized so that their boundaries are still inside the region where the Φ_{minima} function is nonzero, to avoid the creation of plateaus. Finally, the function values for a few nodes between the minima are set to a positive value. In this way, we create the humps in Φ_{fill}. This results in a function with the desired properties. Our example can be seen in Figure 4.4.

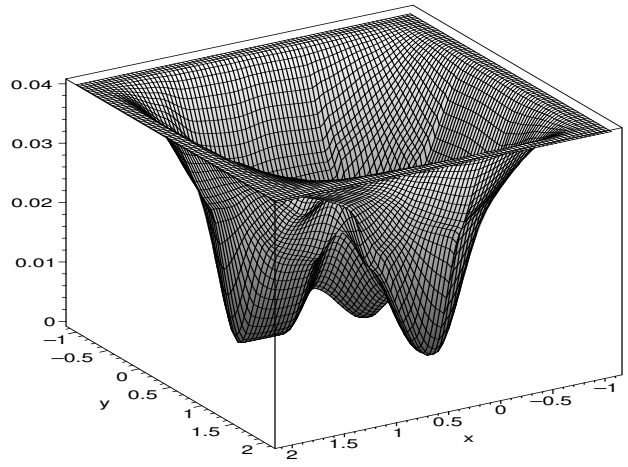

Figure 4.4: Plot of the function Φ_{fill}

In the last step, the growth of the energy function for large strains is controlled. To do so, we define a C^1 function

$$g(\bar{x}_1, \bar{x}_2) := \begin{cases} \frac{1}{500}(-2(\bar{x}_1^2 + \bar{x}_2^2) + (\bar{x}_1^2 + \bar{x}_2^2)^2 + 1) & \text{for } \bar{x}_1^2 + \bar{x}_2^2 > 1 \\ 0 & \text{otherwise} \end{cases} .$$

Table 4.1: Locations of the minima: The minima in the e_1-e_2-e_6 space are taken from [21]. The values in the orbit space follow by evaluating the Hilbert map $\rho = (\rho_1, \rho_2, \rho_3)$ at these points.

	tetragonal	orthorhombic	monoclinic
e_1	0	0.01	0.0479
e_2	0	0	0.0055
e_6	0	0	0.1600
$\rho_1(e_1, e_2, e_6)$	0	0.01	0.0534
$\rho_2(e_1, e_2, e_6)$	0	0.0001	0.00232
$\rho_3(e_1, e_2, e_6)$	0	0	0.0256

Table 4.2: Choice of parameters to fit the elastic moduli

parameter	value	parameter	value		
α_{11}	3936	α_{22}	38377	α_{33}	11881
α_{12}	0	α_{13}	0	α_{23}	0
β_{11}	12100	β_{22}	12100	β_{33}	12100
β_{12}	0	β_{13}	0	β_{23}	0
γ_{11}	54896	γ_{22}	8381025	γ_{33}	71717
γ_{12}	-556900	γ_{13}	415.3	γ_{23}	126900
δ	$5 \cdot 10^{-4}$				

This function will be added to Φ_{fill}. When we move the center of this function from $(0, 0)$ to $(0.5, 0.5)$, it does not interfere with the minima and still starts to grow inside the area where Φ_{fill} is defined. This function thus controls the growth of the potential in the plane toward infinity. So we have

$$\Phi_{\text{3dfill}}(\bar{x}_1, \bar{x}_2, \bar{x}_3) := \Phi_{\text{fill}}(\bar{x}_1, \bar{x}_2) + \delta \bar{x}_3^2 + g(\bar{x}_1 - 0.5, \bar{x}_2 - 0.5)$$

The total energy then has the following form:

$$\Phi(E) := \Phi_{\text{orbit}}(\rho(E)), \tag{4.10}$$

where

$$\Phi_{\text{orbit}} := \Phi_{\text{minima}}(\rho_1, \rho_2, \rho_3) + \Phi_{\text{3dfill}}(A(\rho_1, \rho_2, \rho_3)). \tag{4.11}$$

4.4 Discussion

The energy derived in Section 4.3 meets the symmetry requirements of the tetragonal-orthorhombic-monoclinic phase transition in zirconia, and it interpolates experimental data that is available for this material. The energy constructed here will be used in the finite element simulation presented in the next

Table 4.3: Elastic Moduli: The values resulting from our energy function are, modulo round off errors in the parameters, the same as in [21], Tables IIb, IIIb. Note also the re-labeling of the indices in the monoclinic phase, Table IV. Here, the tetragonal phase's labeling is always used.

	tetragonal	orthorhombic	monoclinic
C_{11}	340	98.3	312
C_{22}	340	98.3	350
C_{66}	95.0	96.8	66.3
C_{12}	33.0	95.3	35.2
C_{16}	0	0	3.2
C_{26}	0	0	4.3

Chapter; in particular, the energy barriers between different wells are of a reasonable height, and the growth of the function for large strains can be controlled easily.

It was not only the aim to derive a specific energy function for zirconia; one goal was to present the orbit space approach as an alternative to the polynomial ansatz in Landau theory because polynomials tend to be too rigid to be used in numerical simulations when they are constrained to have minimizers only at preassigned spots. Often, the minima tend to be degenerate, or the region between the minima is too shallow to be distinguishable from a true minimum in a finite element simulation; the rapid growth of polynomials for large values (strains) poses another problem in simulations. One possibility to avoid this would be to work in a fundamental domain (see [18] for this approach on an atomistic level). The orbit space presents an alternative. The fact that the generating polynomials can be computed automatically provides a convenient way to deal with the symmetry constraints.

To advocate this approach, it was the aim to present the orbit space method in a toolbox-like style so it should prove to be applicable in other situations as well.

The idea to derive energy functions by using a finite element simulation seems to be new in this field; in geometric modeling, these ideas have been used successfully to construct surfaces without unwanted minimizers (by solving Laplace's equation), or surfaces with minimal curvature (by solving the biharmonic equation); see, e.g., [8] for the use of partial differential equations in geometric modeling.

Chapter 5

Numerics

5.1 Introduction

The main focus in the finite element simulation implemented in this work has been laid on flexibility and robustness. Since the equation of motion for the systems regarded here includes a capillarity term coming from the surface energy, the Galerkin formulation (see [46], Chapter 1) of the variational problem introduced in Section 5.2 formally needs a continuously differentiable displacement function u. In many other works (for example in [5]), however, the finite elements are not conforming (C^1). This makes special treatment of the gradients necessary, therefore usually the elements used are aligned according to the phase boundaries that one would expect. In this work we make no a priori assumptions on the formation of phase boundaries. This was be achieved by using the fully conformal Bogner-Fox-Schmit element (see, for example, [9], II.5.10).

To make the simulation code written in MATLAB easily applicable to a wide variety of problems, the MATLAB symbolic math package has been used on several occasions. This makes it, for example, possible to change the strain energy in the system easily since the stress tensor σ is automatically computed from the energy. Also, setting of the initial conditions is simplified by this approach.

To make the program readily understandable, the `WEB` documentation system (see [33]) has been used. The program code is divided into small sections that have individual LaTeX documentation. The code can be found in Appendix C.

This part of the work is organized as follows: First, in Section 5.2 the variational formulation of the differential equation is introduced. Then the discretization of the displacement u is formally made in Section 5.3 and the spatial part of the partial differential equation is turned into a system of linear equations. The finite element is introduced in Section 5.4 and the various boundary conditions for the finite element simulation are discussed in Section 5.5. The results of the numerical simulations carried out are shown in Section 5.6.

5.2 Variational formulation of the problem

We will again start with the equation of motion that we want to solve,

$$u_{tt}(x,t) = \mathrm{Div}(\sigma(\nabla u(x,t))) + \beta \triangle u_t(x,t) - \triangle^2 u(x,t) + f(x,t) \tag{5.1}$$

equipped with appropriate initial and boundary conditions. To obtain the finite element formulation of this problem, it is first necessary to turn to the equivalent variational formulation. One has to find a function

$$\begin{aligned} u\colon \Omega \times \mathbb{R} &\to \mathbb{R}^n \\ u\colon (x,t) &\mapsto u(x,t). \end{aligned}$$

such that for all test functions

$$\zeta\colon \Omega \to \mathbb{R}^n$$

with $\zeta \in C_0^\infty(\Omega)$[1] the following condition, together with the boundary conditions for u, holds:

$$\begin{aligned} &\int_\Omega u_{tt}(x,t)\zeta \mathrm{d}V = \\ &\int_\Omega \left(\mathrm{Div}(\sigma(\nabla u(x,t)))\zeta + \beta \triangle u_t(x,t)\zeta - \triangle^2 u(x,t)\zeta + f(x,t)\zeta\right) \mathrm{d}V. \end{aligned}$$

Since ζ has compact support, integration by parts yields

$$\begin{aligned} &\int_\Omega \triangle u \triangle \zeta \mathrm{d}V + \int_\Omega u_{tt}\zeta \mathrm{d}V = \\ &- \int_\Omega \sigma(\nabla u(x,t)) \cdot \nabla \zeta \mathrm{d}V - \int_\Omega \beta \nabla u_t \cdot \nabla \zeta \mathrm{d}V + \int_\Omega f(x,t)\zeta \mathrm{d}V. \end{aligned} \tag{5.2}$$

5.3 The Discretization

Equation (5.2) is formulated in an infinite-dimensional function space. We approximate the solution by approximating this space with a finite dimensional space, spanned by $\xi_1(x), \ldots, \xi_K(x)$ with $K \in \mathbb{N}$. Let

$$u(x,t) = \sum_i u_i(t)\xi_i(x)$$

with coefficients u_i and a finite basis $\mathcal{B} = \{\xi_i\}_{1 \le i \le K}, K \in \mathbb{N}$ and restrict the test functions to be the elements ξ_i of $\mathcal{B}$, too. This yields the following equation:

$$\begin{aligned} \int_\Omega \sum_i u_i \triangle \xi_i \triangle \xi_j \mathrm{d}V \quad + \quad & \int_\Omega \sum_i (u_{i,tt}\xi_i)\xi_j \mathrm{d}V = -\int_\Omega \sigma\left(\sum_i u_i \nabla \xi_i\right) \cdot \nabla \xi_j \mathrm{d}V \\ & - \int_\Omega \sum_i u_{i,t} \nabla \xi_i \cdot \nabla \xi_j \mathrm{d}V + \int_\Omega f \xi_j \mathrm{d}V \end{aligned}$$

[1] See Appendix A for the notation.

which is equivalent to (by exchanging the summation and the integration)

$$\sum_i u_i \int_\Omega \triangle\xi_i \triangle x i_j \mathrm{d}V + \sum_i u_{i,tt} \int_\Omega \xi_i \xi_j \mathrm{d}V = \tag{5.3}$$
$$-\int_\Omega \sigma \left(\sum_i u_i \nabla \xi_i \right) \cdot \nabla \xi_j \mathrm{d}V - \sum_i u_{i,t} \int_\Omega \nabla \xi_i \cdot \nabla \xi_j \mathrm{d}V + \int_\Omega f \xi_j \mathrm{d}V.$$

Now define the matrices

$$K_{ij} := \int_\Omega \triangle\xi_i \triangle \xi_j \mathrm{d}V$$

and

$$M_{ij} := \int_\Omega \xi_i \xi_j \mathrm{d}V.$$

The matrix K will be called the *stiffness matrix* and M is the *mass matrix*. Furthermore, let

$$\mathbf{u}(t) = \begin{pmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_N(t) \end{pmatrix}.$$

Then one can write equation (5.3) as

$$K\mathbf{u} + M\ddot{\mathbf{u}} = \mathbf{g} \tag{5.4}$$

with

$$g_j := -\int_\Omega \left[\sigma \left(\sum_i u_i \nabla \xi_i \right) + \sum_i u_{i,t} \nabla \xi_i \right] \cdot \nabla \xi_j + f \xi_j \mathrm{d}V.$$

The spatial derivatives of the PDE have been transformed to a system of linear equations. The partial differential equation has, by this method, thus been transformed into an ordinary differential equation in time, and a regular numerical time integration scheme can be used to solve this problem.

5.4 Introduction of the Finite Element

Now one has to find a convenient form of the discretized displacement u. Take a rectangular material domain $\left[-\frac{a}{2}, \frac{a}{2}\right] \times \left[-\frac{b}{2}, \frac{b}{2}\right]$ and divide this domain into $N \times M$ rectangular subdomains of size $r \times s$ ($r = \frac{a}{N}, s = \frac{b}{M}$),

$$\Omega_{ij} := [(i-1)r, ir] \times [(j-1)s, js]$$

for $i, j \in \mathbb{N}$, $1 \leq i \leq N, 1 \leq j \leq M$. These subdomains will be the domains of the *finite elements* that are to be introduced. On the corner points of the subdomains Ω_{ij} are the 4 *nodes* of each element, numbered counterclockwise starting on the lower left corner. For each of these nodes we introduce four

degrees of freedom. These will be named d_k ($k = 1, \ldots, 16$ or more precisely $k = 4(n-1)+m$ with the node number n and the number of the degree of freedom m). Now define the polynomials

$$\begin{aligned} p_1(x) &:= (1-x^2)(1+2x), \\ p_2(x) &:= x(1-x^2), \\ p_3(x) &:= x^2(3-2x), \\ p_4(x) &:= -x^2(1-x) \end{aligned}$$

for $x \in [0,1]$. These polynomials form a basis for the polynomials up to order 3 on $[0,1]$. On the end point $x = 0$, one has $p_1 = 1$, $p_{1,x} = 0$, $p_2 = 0$ and $p_{2,x} = 1$. The values and the first derivatives of p_3 and p_4 vanish there. At $x = 1$, both the function value and the value of the derivative of p_1 and p_2 are equal to zero, but there p_3 and $p_{4,x}$ are equal to 1. Therefore, one can control the function value and the value of the derivative at $x = 0$ and $x = 1$ very easily by a linear combination of these polynomials. On the domain $[0,r] \times [0,s]$ one can now define the polynomials

$$\begin{aligned} \varphi_1(x_1,x_2) &:= p_1\left(\frac{x_1}{r}\right) p_1\left(\frac{x_2}{s}\right), \\ \varphi_2(x_1,x_2) &:= p_2\left(\frac{x_1}{r}\right) p_1\left(\frac{x_2}{s}\right), \\ \varphi_3(x_1,x_2) &:= p_1\left(\frac{x_1}{r}\right) p_2\left(\frac{x_2}{s}\right), \\ \varphi_4(x_1,x_2) &:= p_2\left(\frac{x_1}{r}\right) p_2\left(\frac{x_2}{s}\right), \\ \varphi_5(x_1,x_2) &:= p_3\left(\frac{x_1}{r}\right) p_1\left(\frac{x_2}{s}\right), \\ \varphi_6(x_1,x_2) &:= p_4\left(\frac{x_1}{r}\right) p_1\left(\frac{x_2}{s}\right), \\ \varphi_7(x_1,x_2) &:= p_3\left(\frac{x_1}{r}\right) p_2\left(\frac{x_2}{s}\right), \\ \varphi_8(x_1,x_2) &:= p_4\left(\frac{x_1}{r}\right) p_2\left(\frac{x_2}{s}\right), \\ \varphi_9(x_1,x_2) &:= p_1\left(\frac{x_1}{r}\right) p_3\left(\frac{x_2}{s}\right), \\ \varphi_{10}(x_1,x_2) &:= p_2\left(\frac{x_1}{r}\right) p_3\left(\frac{x_2}{s}\right), \\ \varphi_{11}(x_1,x_2) &:= p_1\left(\frac{x_1}{r}\right) p_4\left(\frac{x_2}{s}\right), \\ \varphi_{12}(x_1,x_2) &:= p_2\left(\frac{x_1}{r}\right) p_4\left(\frac{x_2}{s}\right), \\ \varphi_{13}(x_1,x_2) &:= p_3\left(\frac{x_1}{r}\right) p_3\left(\frac{x_2}{s}\right), \\ \varphi_{14}(x_1,x_2) &:= p_4\left(\frac{x_1}{r}\right) p_3\left(\frac{x_2}{s}\right), \\ \varphi_{15}(x_1,x_2) &:= p_3\left(\frac{x_1}{r}\right) p_4\left(\frac{x_2}{s}\right), \end{aligned}$$

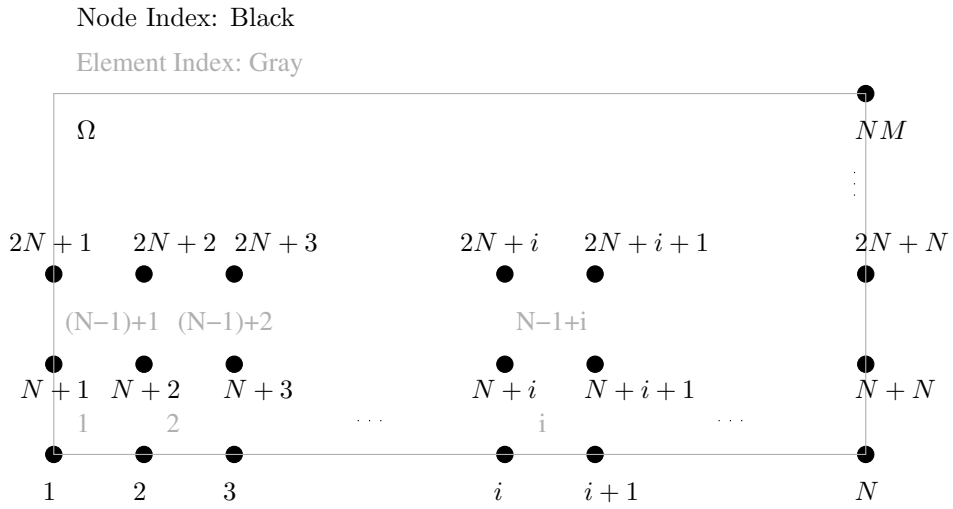

Figure 5.1: The numbering of the nodes and elements for the finite element simulation

$$\varphi_{16}(x_1, x_2) \quad := \quad p_4\left(\frac{x_1}{r}\right) p_4\left(\frac{x_2}{s}\right).$$

These are the so called *shape functions* for the Bogner-Fox-Schmit finite element (see, for example, [9], II.5.10). Being the tensor product of the four polynomials p_i from above, one can see that this is a complete bicubic basis for the rectangle $[0, r] \times [0, s]$. Now for each corner point of this rectangle, only one function φ_i has a nonvanishing function value, x-derivative, y-derivative or xy-derivative respectively. These four values are, for the first corner point controlled by the first four polynomials, for the second corner point by the next four polynomials and so on.

Now we can define the displacement $u_{ij}(x_1, x_2)$ on one subdomain Ω_{ij} to be

$$u_{ij}(x_1, x_2) := \sum_{i=1}^{16} d_i \varphi_i(x_1 - ir, x_2 - js).$$

with d_i being the 16 degrees of freedom for the element with the domain Ω_{ij}. In Figure 5.1 one can see how adjectant elements share the same nodes and the degrees of freedom that belong to those nodes. The total number of nodes in this system is $(N + 1)(M + 1)$. So the total number of degrees of freedom is $4(N+1)(M+1)$. One can easily check that the displacement u is a continuously differentiable function on Ω. Since u has been constructed from base functions that have been moved to each element one can write u as

$$u(x_1, x_2) = \sum_i u_i \xi_i,$$

with coefficients u_i and all the base functions for the whole domain, constructed by moving the base functions from $[0, r] \times [0, s]$ to each element. The arrangement of the degrees of freedom in the vector $\mathbf{u}$ is done by node, so the first 4 entries belong to node 1, the next 4 entries belong to node 2, and so on. The arrangement of the nodes can be seen in Figure 5.1. Since we want a displacement with values in $\mathbb{R}^2$, we have introduce two vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ as described above. With this discretization one can compute the matrices K and M from equation (5.4). The integrations for the remaining vector $\mathbf{g}$ in this equation can now be done by working with one element at a time because every base function ξ_i is nonzero only on the four elements that share its corresponding node.

5.5 Boundary Conditions for the Numerical Simulation

For the finite element simulation one has to specify conditions for the degrees of freedom of the boundary nodes. The Dirichlet boundary conditions for the differential equation, however, namely u and $\triangle u$ on $\partial\Omega$, can not be directly transferred to the degrees of freedom that one has available on the nodes, which are u, u_x, u_y and u_{xy}. One can see that Dirichlet boundary conditions for the value of u can be set by prescribing the degrees of freedom for u and for the value of the derivative tangential to the boundary in question (automatically, if u is fixed on the boundary, the tangential derivative of u is fixed, too). Additionally, it is possible to prescribe the derivatives normal to the boundary. If nothing is set for a certain part of the boundary, this boundary is said to have *free* boundary conditions. *Simply supported* boundary conditions mean a prescribed value of u at the boundary (and therefore also the tangential derivative is fixed). *Clamped* condition means that also the value for the normal derivative is prescribed. For the Bogner-Fox-Schmit element used here one does not prescribe the values of the mixed derivative u_{xy} on the boundary nodes.

5.6 Results of the Numerical Computations

Generally, due to our non dimensionalization of the system, one can not really compare the absolute times from our simulation to other simulations. We will in this work speak of the "time step" from the time integration loop. For all simulations the initial condition for the velocity $u_t(0)$ was chosen to be equal to zero. Also, in the figures, the coordinates are labeled as the element coordinates. The x- and y-coordinates in the simulation range from $-\frac{1}{2}$ the size of the specimen to $\frac{1}{2}$ the size of the specimen. The end points of the coordinates will be denoted $x_{\min}$ and $x_{\max}$ (same for y).

1. Reproduction of the Simulations by Swart and Holmes
To test the validity of the results of the program code two, of the anti-plane shear simulations presented in [51] were reproduced. In Section 6.3 of [51], an

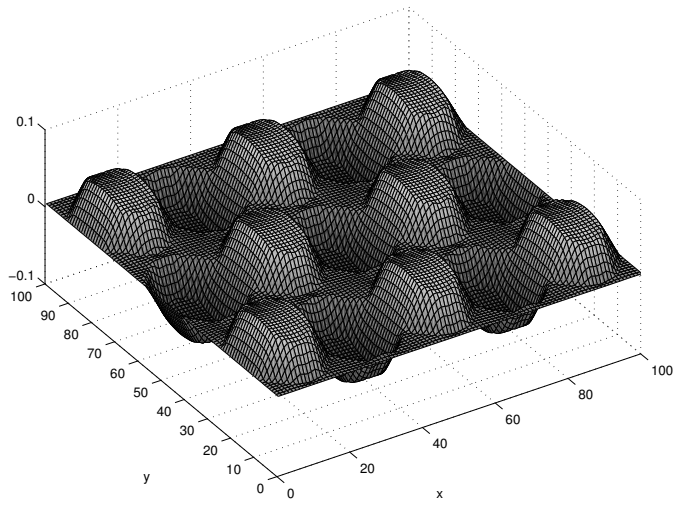

Figure 5.2: Displacement for the reproduction of [51], Section 6.3. Relaxed state.

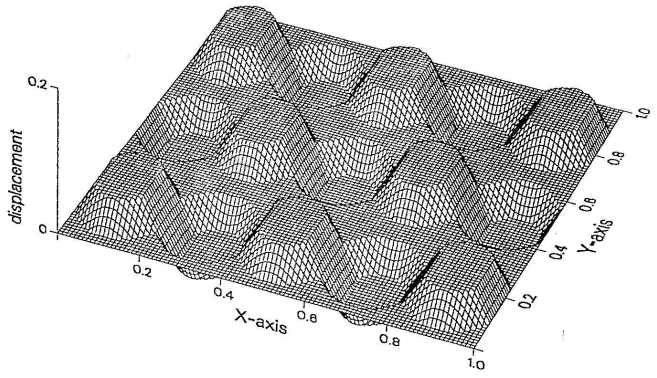

Figure 5.3: Original image from [51], Section 6.3

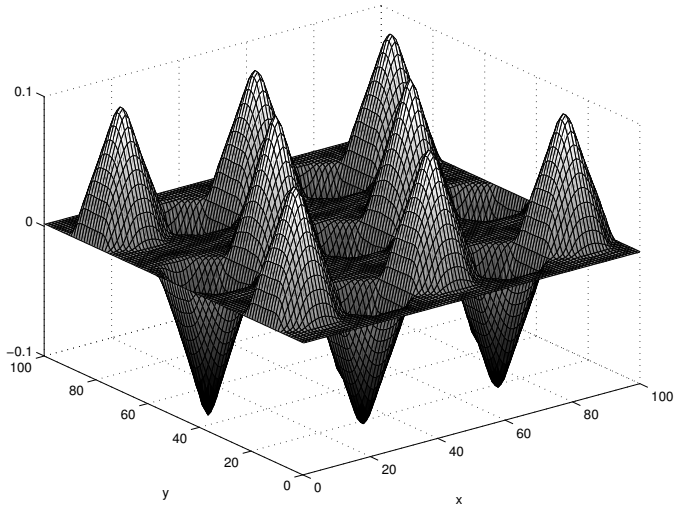

Figure 5.4: Displacement for the reproduction of [51], Section 6.3 with added capillarity. Time step 600.

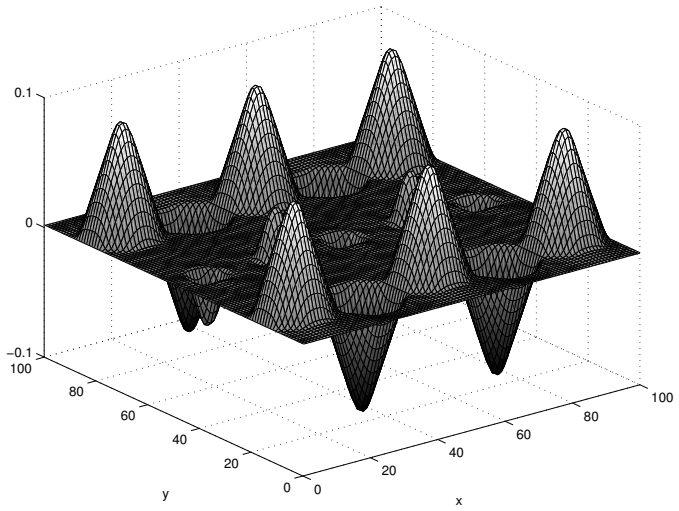

Figure 5.5: Displacement for the reproduction of [51], Section 6.3 with added capillarity. Time step 1600.

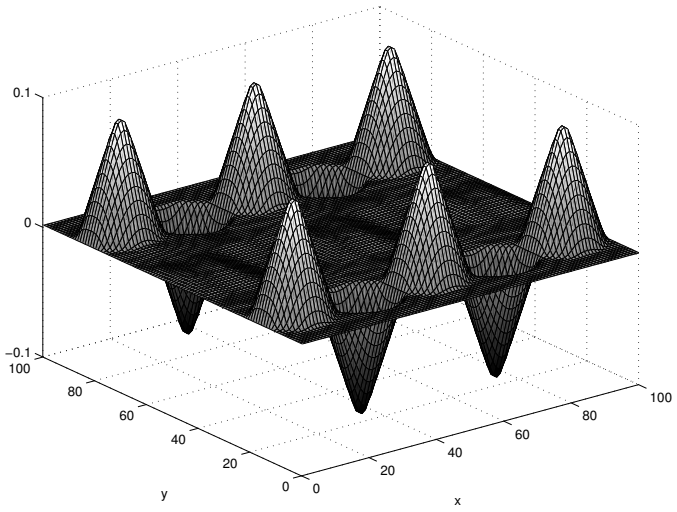

Figure 5.6: Displacement for the reproduction of [51], Section 6.3 with added capillarity. Time step 3000.

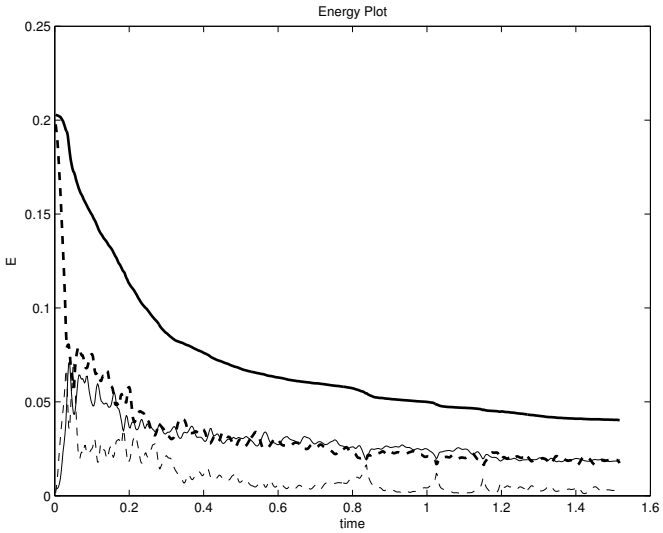

Figure 5.7: Energy versus time for the reproduction of [51], Section 6.3 with added capillarity. The thick black line is the total energy, the thick dashed line below strain energy. Close to zero start the surface energy (solid) and the kinetic energy (dashed).

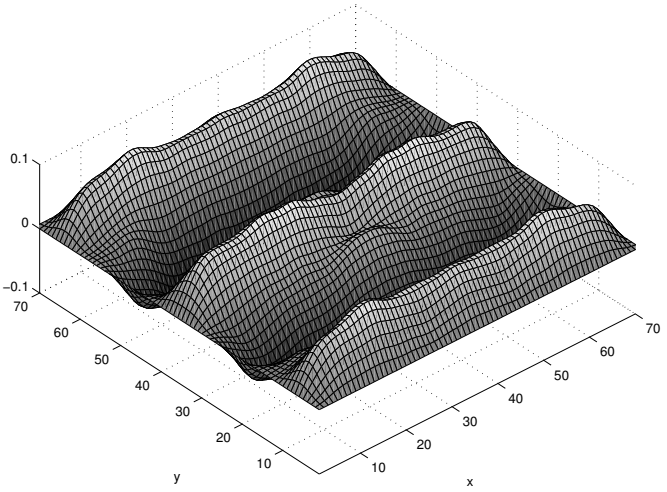

Figure 5.8: Displacement for the reproduction of [51], Section 6.4.

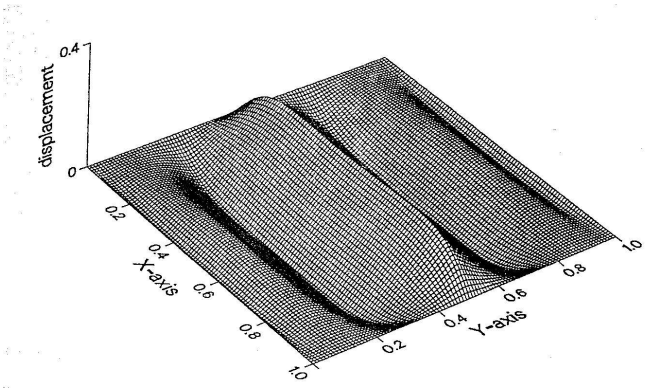

Figure 5.9: Original image from [51], Section 6.4

energy function of the form

$$W(F) = 4||F||^2(||F|| - 1)^2$$

is used. With this energy function, a 100×100 element grid, no capillarity and a viscosity factor of 0.02 we repeated this simulation. The initial conditions were the same as in [51], namely

$$u_0(x, y) = 0.05 \sin(5\pi x) \sin(3\pi y).$$

The result is optically indistinguishable the one that Swart and Holmes obtain and can be seen in Figure 5.2. The image from [51] has been included for comparison. It can be seen in Figure 5.3. The parameters for the reproduction of the simulation [51], Section 6.3, were

Grid	100×100,
Element Size	0.01×0.01,
Boundary Conditions	Clamped everywhere, values are from initial conditions on the boundary,
Initial Conditions	$u_0(x, y) = 0.05 \sin(5\pi x) \sin(3\pi y)$,
Strain Energy	$W(F) = 4\|\|F\|\|^2(\|\|F\|\| - 1)^2$,
Capillarity	$\gamma = 0$,
Viscosity	$\beta = 0.02$.

However, the piecewise almost-constant gradients found in their and our work seem to be a result of the spatial discretization, since a simulation with the exact same parameters, but with capillarity (Factor 10^{-4}) reveals a different picture. Figures 5.4, 5.5 and 5.6 show similar, but rounded humps that oscillate and the ones that are not "held up" by the clamped boundary conditions vanish with time. The rounding of the humps seems reasonable, since the energy itself does not favor any specific value of F as long as the norm of it is a minimizer, i.e., if $F = 0$ or $||F|| = 1$. The humps at the boundary remain—they become smaller, though—because clamped boundary conditions means that the value of the normal derivative of u is prescribed on the boundary. With the boundary conditions taken to be the initial conditions on the boundary, these normal derivatives are not zero everywhere on the boundary. The time dependence of the potential-, surface-, kinetic- and total energy can be seen in Figure 5.7. There, also an approximately equipartitioning of the total energy in the potential- and surface energy in the fully relaxed state can be seen. The parameters for the reproduction of the simulation [51], Section 6.3, with added capillarity, were

Grid	100×100,
Element Size	0.01×0.01,
Boundary Conditions	Clamped everywhere, values are from initial conditions on the boundary,
Initial Conditions	$u_0(x, y) = 0.05 \sin(5\pi x) \sin(3\pi y)$,
Strain Energy	$W(F) = 4\|\|F\|\|^2(\|\|F\|\| - 1)^2$,
Capillarity	$\gamma = 10^{-4}$,
Viscosity	$\beta = 0.02$.

In Section 6.4 of [51], a simulation using an anisotropic energy function, depending on $F = (u_x, u_y)$, of the form

$$W(F) = \frac{1}{2}u_x^2 + \frac{1}{4}(u_y^2 - 1)^2$$

is used. We carried out a simulation using this energy function and the same domain as in the previous simulation. The factor for the capillarity was again set to 10^{-4}. Starting with a small hump

$$u_0(x, y) = 0.1 \exp(-30x^2 - 30y^2),$$

a similar result as Swart and Holmes have is obtained (Figure 5.8—compare with the original result in Figure 5.9—our figure shows a not fully relaxed state, one can see this by the "extra" edge on the central peak). The important feature of this result is the refinement of the microstructure at the incompatible boundaries This will be examined in greater detail in the simulation "Energy Scaling of Microstructure". The parameters for the reproduction of the simulation found in [51], Section 6.4, were

Grid	100×100,
Element Size	0.01×0.01,
Boundary Conditions	Clamped everywhere, values are from initial conditions at the boundary,
Initial Conditions	$u_0(x, y) = 0.1 \exp(-30x^2 - 30y^2)$
Strain Energy	$W(u_x, u_y) = \frac{1}{2}u_x^2 + \frac{1}{4}(u_y^2 - 1)^2$,
Capillarity	$\gamma = 10^{-4}$,
Viscosity	$\beta = 0.02$.

2. Reproduction of the Reid and Gooding Simulation
Reid and Gooding present in [44] a true two dimensional (u has values in $\mathbb{R}^2$, therefore the deformation gradient F is a 2×2 matrix) numerical simulation using a three well energy. The energy has the form

$$W(q_2, q_3) = -1000(q_2^2 + q_3^2) + 5515.9(q_2^3 - 3q_2q_3^2) + 29583(q_2^2 + q_3^2)^2$$

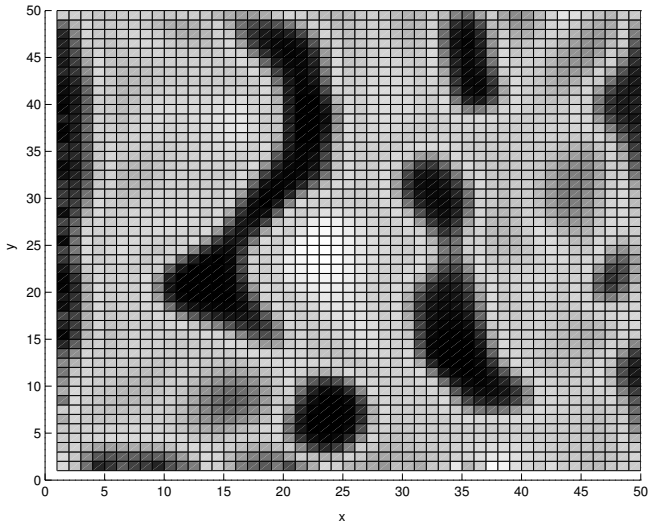

Figure 5.10: Strain energy density for the Reid and Gooding simulation at time step 100. Darker means less energy.

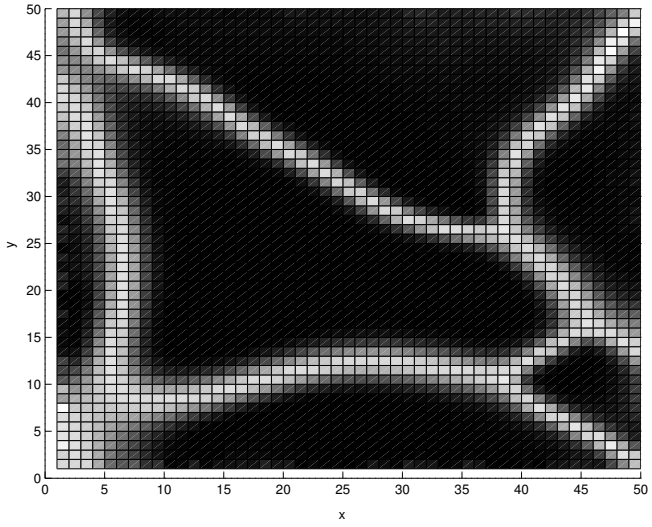

Figure 5.11: Strain energy density for the Reid and Gooding simulation at time step 1500. Darker means less energy.

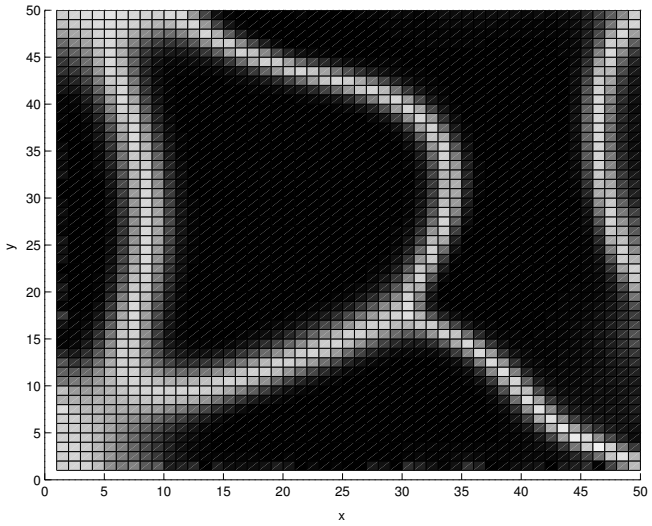

Figure 5.12: Strain energy density for the Reid and Gooding simulation at time step 4500 (Fully Relaxed State). Darker means less energy.

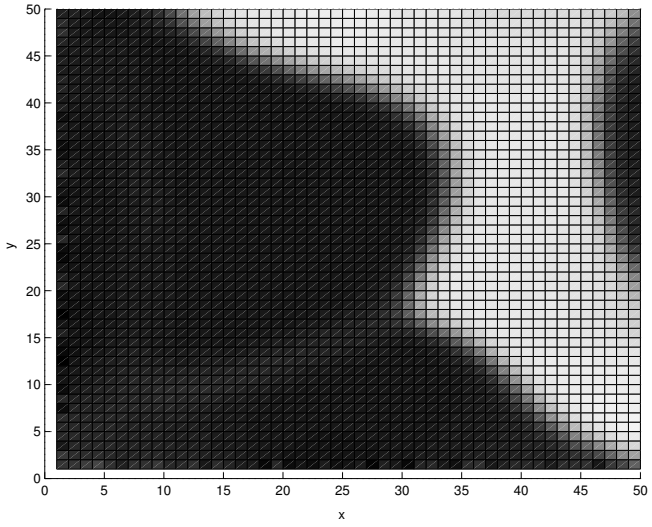

Figure 5.13: The variable q_2 at time step 4500. Black: $q_2 = -0.1$, White: $q_2 = 0.2$.

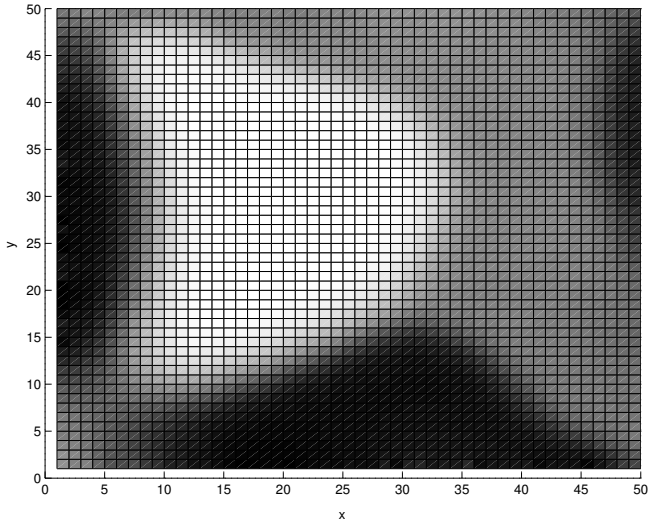

Figure 5.14: The variable q_3 at time step 4500. Black: $q_3 = -0.2$, Gray: $q_3 = 0$, White: $q_3 = 0.2$.

with $q_2 := E_{11} - E_{22}$ and $q_3 := E_{12}$, the entries in the Green-St. Venant strain tensor $E := \frac{1}{2}(F^t F - \mathrm{Id})$ already introduced in Section 4.3. The variable $q_1 = E_{11} + E_{22}$ is also used in [44], but the strain energy does not depend on it. This energy function has minimizers at the points $(q_2, q_3) = (-0.1, 0.2)$, $(q_2, q_3) = (-0.1, -0.2)$ and $(q_2, q_3) = (0.2, 0)$ (values are rounded). The simulation was carried out using a 50×50 element grid with a specimen size of 1×1. The factor for the capillarity was chosen very high, in rough agreement with the Reid and Gooding simulation, namely 0.05. The exact value of the capillarity used by Reid and Gooding could not be reproduced, because they do not directly use a capillarity term, but instead penalize gradients in their derived variables q_i. Viscosity was not included in the system. Figures 5.10, 5.11 and 5.12 show the strain energy of the system at various times. One can see the time development of the phase boundaries, they are the areas with higher strain energy (white lines in the Figures). Figures 5.13 and 5.14 show the variables q_1 and q_2 at the final relaxed state. The values of the variables q_2 and q_3 correspond to minima of the strain energy function. All 3 minima can be seen. The fully relaxed state obtained here does, however, look different from the one that Reid and Gooding compute. They have only two phases in the fully relaxed state. This is obviously due to the fact that the shape of our specimen is not compatible with the symmetry of the strain energy, and theirs was chosen diamond shaped to favor exactly the result with two phases. The alignment of the phase boundaries is the same, though. The parameters for our simulation were

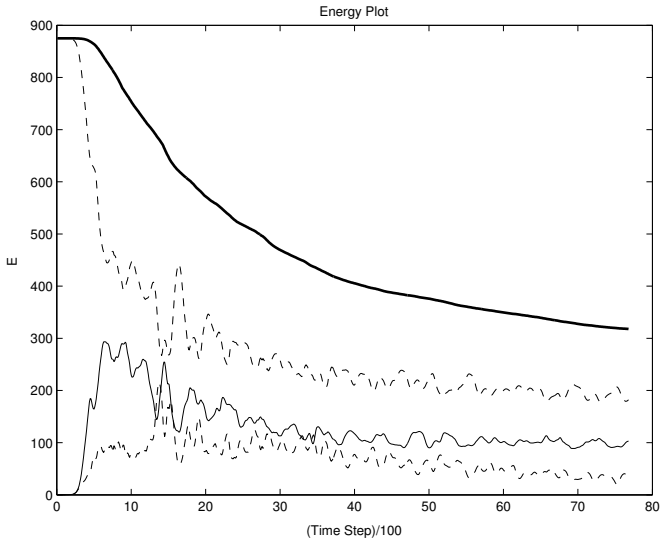

Figure 5.15: Energy versus time step for the Müller/Kohn Simulation. The thick black line is the total energy, dashed line below the strain energy. Close to zero start the surface energy (solid) and the kinetic energy (dashed).

Grid	50×50,
Element Size	0.02×0.02,
Boundary Conditions	Clamped everywhere,
	$(u, u_{\text{tangential}}, u_{\text{normal}})\|_{\partial\Omega} = 0$,
Initial Conditions	Small amplitude random noise
Strain Energy	$-1000(q_2^2 + q_3^2) + 5515.9(q_2^3 - 3q_2q_3^2) +$
	$29583(q_2^2 + q_3^2)^2$,
Capillarity	$\gamma = 10^{-4}$,
Viscosity	$\beta = 0$.

3. Energy scaling of Microstructure

In [34] Kohn and Müller study a variational problem involving a nonconvex energy function analytically. They obtain a scaling law depending on ε for the minimum of the strain- and surface-energy at the relaxed state for an energy density of the form

$$E^{\varepsilon}(u) := \int_{\Omega} u_x^2 + (u_y^2 - 1)^2 + \varepsilon^2 u_{yy}^2 \mathrm{d}V \tag{5.5}$$

in an anti-plane shear system subject to the boundary condition $u|_{x=} = 0$. Previous work has always assumed that the twinning which results from such

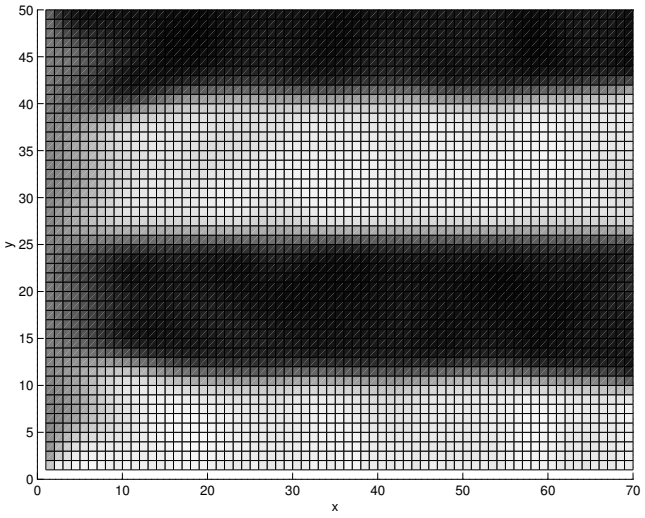

Figure 5.16: Relaxed state for the Kohn/Müller simulation. Shown is the variable u_y. Black: $u_y = -1$, White: $u_y = +1$.

Table 5.1: The energy results for the Kohn/Müller simulation

Grid	ε	Potential Energy
50×50	$\sqrt{0.5}$	215
50×50	1	210
50×50	$\sqrt{2}$	272
70×50	$\sqrt{0.5}$	220
70×50	1	285
70×50	$\sqrt{2}$	361
90×50	$\sqrt{0.5}$	257
90×50	1	223
90×50	$\sqrt{2}$	293

an energy is essentially one-dimensional. Kohn and Müller instead examine the complex patterns that are required at the austenite interface, modeled by the boundary condition $u|_{x=x_{\min}} = 0$. On a domain $\Omega = [0, L] \times [0, 1]$ they receive

$$\min_{u|_{x=x_{\min}}=0} E^{\varepsilon} \sim C\varepsilon^{\frac{2}{3}} L^{\frac{1}{3}}.$$

A calculation using only a sawtooth function of y alone away from the boundary ($x \geq h + x_{\min}$) and linearly interpolating in the regions closer to the boundary $0 \leq x < h + x_{\min}$ to satisfy $u|_{x=x_{\min}} = 0$ would result in a scaling $E^{\varepsilon} \sim C\varepsilon^{\frac{1}{2}} L^{\frac{1}{2}}$. This scaling was widely believed to be true.

The behavior of such a system is numerically regarded in this work. Simulations were carried out using Dirichlet boundary conditions for u on $x = x_{\min}$ (simply supported) and free boundary conditions for the other three boundaries. The strain energy used was the same as in [34], except for the fact that we have to add a term $\varepsilon^2 u_{xx.}^2$, too, due to the nature of our finite element method. This term should, however, not result in any major differences since the main surface energy will come from the phase boundaries which are aligned in the x-direction. A 50×50, 70×50 and a 90×50 grid of elements, each of size 0.5×0.5 length scales, was the domain for the simulation to examine the effect of various domain sizes. The factor $\varepsilon (= \sqrt{\gamma}$ in our simulation) for the capillarity was taken to be $\sqrt{0.5}$, 1.0 and $\sqrt{2.0}$, so a total of 9 simulations was run. Of those, we show the time dependence of the total strain- surface- and kinetic-energy exemplarily for the simulation with $\varepsilon = 1$ and the 50×70 grid in Figure 5.15. For the same simulation, an image of the resulting relaxed state can be found in Figure 5.16. This figure shows the important variable u_y in which the energy is non-convex. One can clearly see the two phases with $u_y = 1$ (black) and $u_y = +1$ (white) and the refinement at the boundary $x = 0$. The other simulations deliver similar images and similar energies.

The potential (surface plus strain) energies for the simulations carried out can be found in Table 5.1. It was not possible to reproduce the scaling law depending on the length L of the specimen. This is not surprising, because in order to see the self similar refinement proposed by Kohn and Müller one would have to use a much larger scale simulation. We do generally only see a few phase boundaries, and for the refinement at the austenite interface our spatial discretization is to coarse. However, except for the run with $\varepsilon = \sqrt{0.5}$ and the 50×50 grid and the run with $\varepsilon = \sqrt{0.5}$ and the 90×50 grid (see below for difficulties in such simulations) we are in very good agreement with the scaling law in ε proposed in [34]. We obtain a scaling of the energy, disregarding the two unsatisfying runs, with ε^a, where

$$a \approx 0.65.$$

An error analysis was not performed, due to the few data points collected. This result, however, is in excellent agreement with the prediction of Kohn and Müller in [34].

The difficulties that arise when carrying out these simulations mainly come from the fact that a viscosity term in the equation of motion hinders the phase

boundaries from moving. They get stuck and this is obviously what happened in the runs with $\varepsilon = \sqrt{0.5}$ and the 50×50 or 90 grid—even though we set β, the factor for the viscosity in the simulations, to be equal to zero. But in this simulation it seems that even the numerical viscosity that is always inherent in computations like these can enough to keep the phase boundaries from moving. Of course, no viscosity at all would not work either, because somehow one has to dissipate energy from the system. The refinement of the microstructure at an austenite interface has been numerically examined before in the already mentioned simulation by Swart and Holmes, see [51], Section 6.4. But their system does not include a surface energy term at all and therefore they can not obtain scaling laws for the energy. The parameters for the the simulation regarding the scaling law for the energy were

Grid	50×50, 70×50 and 90×50,
Element Size	0.5×0.5,
Boundary Conditions	Simply supported at $x = x_{\min}$, free on the other boundaries $u\|_{x=x_{\min}} = 0$,
Initial Conditions	$u_0(x, y) = 0.1 \exp(-30x^2 - 30y^2)$
Strain Energy	$W(u_x, u_y) = \int_\Omega u_x^2 + (u_y^2 - 1)^2$,
Capillarity	$\gamma = \varepsilon^2 = 0.5, 1, 2$,
Viscosity	$\beta = 0$.

4. Zirconia

In Chapter 4, a strain energy function for zirconia, a material with a triple point is constructed. This seven well (4 monoclinic, 2 orthorhombic and 1 tetragonal) potential was plugged into a two dimensional simulation and a test was performed on a 50×50 grid. We chose very rigid boundary conditions (clamped everywhere) and an initial condition that, at various points, exhibits strains for every phase the system can have. However, the relaxed state shown in Figures 5.17 and 5.18 does not show orthorhombic minima. The variable ρ_2 on the orbit space is omitted here, because it is possible to distinguish the minima without it. One can see that the largest region on the domain is occupied by the tetragonal ($\rho_1 = \rho_2 = \rho_3 = 0$) minimum. Where the boundary conditions prevent this situation, the monoclinic minima are obtained, especially visible in the lower right corner. This situation is in good agreement with what one would expect, but still there seem to be problems with this simulation and it is not completely trustworthy, because the phase boundaries are not as clearly visible as in the other simulations. Nevertheless, the energy functional constructed in Chapter 4 provides a basis for more investigations on zirconia, for example its reaction to hard loading, i.e. the changing of boundary conditions with time. The parameters used in the zirconia simulation were

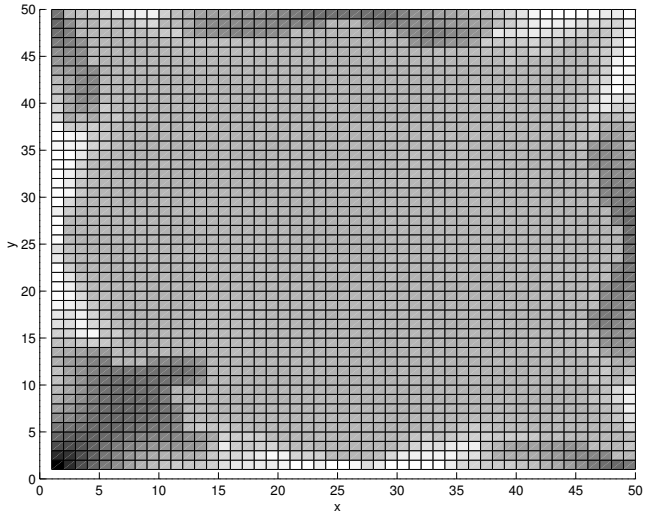

Figure 5.17: The orbit space variable ρ_1 for the relaxed state for the zirconia simulation. The light gray area shows values in the tetragonal minimum, the dark and light gray areas have values in the monoclinic minimum.

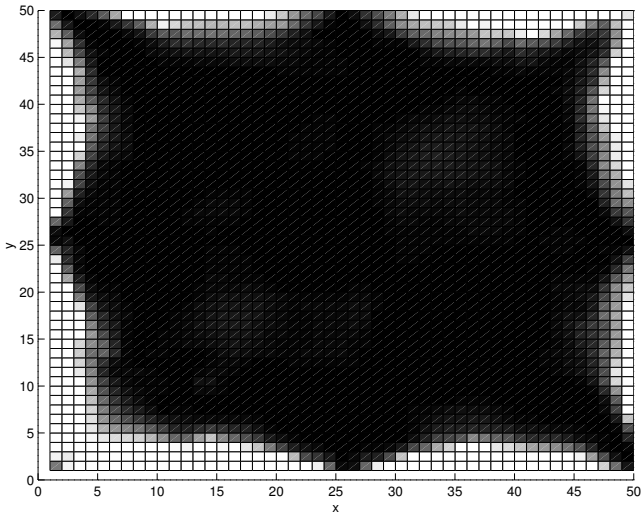

Figure 5.18: The orbit space variable ρ_3 for the relaxed state for the zirconia simulation. The dark area is $\rho_3 = 0$ (monoclinic or orthorhombic), the white areas have values in the monoclinic minimum.

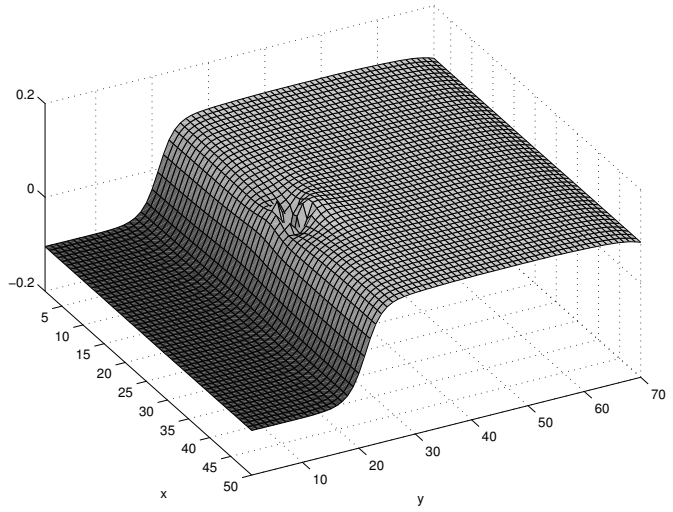

Figure 5.19: Traveling wave, shown is E_{12}. Time step 2200.

Grid	50×50,
Element Size	0.5×0.5,
Boundary Conditions	Clamped everywhere, values are taken from the initial conditions on the boundary,
Initial Conditions	$u_1 = 0.5\sin(x/4)\sin(y/2.8)$, $u_2 = 0.5\sin(x/2.8)\sin(y/4)$
Strain Energy	Constructed in Chapter 4,
Capillarity	$\gamma = 1$,
Viscosity	$\beta = 0.01$.

5. A Traveling Wave and an Obstacle

A very interesting question is the question of how traveling phase boundaries react to obstacles that are in their way. It is very important to study such a situation, because crystals are never defect-free—and the propagation of phase boundaries obviously defines the quasi-plastic behavior as well as the pseudo-elastic returning to the "memorized" shape. To model this situation, we, starting with the Green-St. Venant strain tensor E again, first created a strain energy that has only two minimizers. These minimizers were at $E_{12} = \pm 0.1$. This energy function for the two-dimensional simulation was

$$W(E_{11}, E_{22}, E_{12}) = E_{11}^2 + E_{22}^2 + 12(E_{12}^2 - 0.1^2)^2 \left(1 + \exp\left(\frac{-E_{12}^2}{0.1^2}\right)\right).$$

The exponential term only serves the purpose to increase the energy barrier between the two minima. Then, to create the obstacle, which was in this case

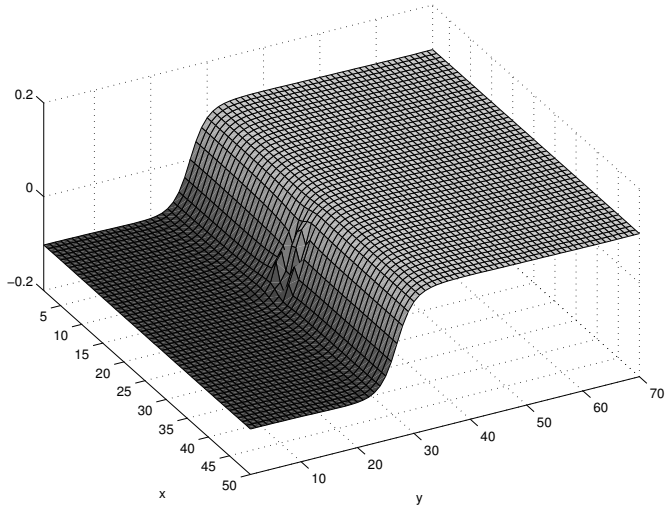

Figure 5.20: Traveling wave, shown is E_{12}. Time step 2400.

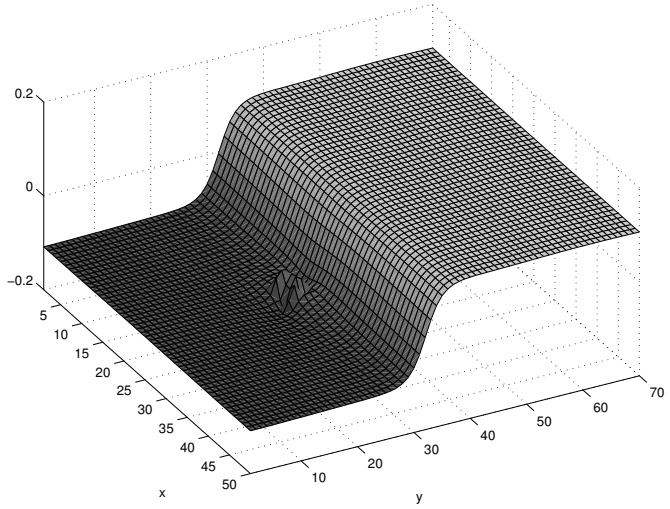

Figure 5.21: Traveling wave, shown is E_{12}. Time step 2600.

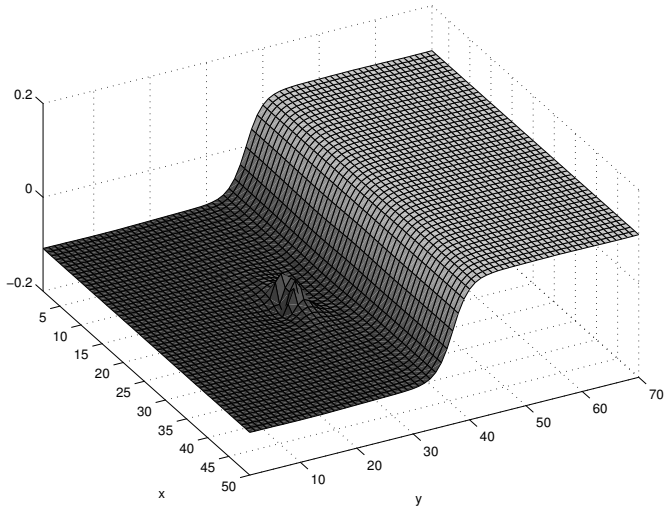

Figure 5.22: Traveling wave, shown is E_{12}. Time step 2800.

a set of four non transforming elements, a different energy function which was simply

$$W_{\text{obstacle}} = 20(E_{11}^2 + E_{22}^2 + E_{12}^2)$$

with only one minimizer at the unstrained reference configuration was used. To simulate a large traveling wave, the program code was modified to allow periodic boundary conditions at the $x = x_{\min}$ and $x = x_{\max}$ boundaries. This is done by connecting the elements at $x = x_{\max}$ with those on $x = x_{\min}$ again and thus working on a cylinder. Then we chose simply supported boundary conditions at $y = y_{\min}$ and on the remaining $y = y_{\max}$ boundary.

The domain Ω was chosen to be a 50×200 grid of elements each of size 0.5. The four non transforming elements were around the element coordinates 25×25. This basically results in non reflective boundary conditions at $y = y_{\max}$, because this way traveling sound waves would be eliminated by the viscosity before they could get back into the important region around the obstacle.

The initial conditions were $u_1 = 0.1y$ and $u_2 = 0$. One can easily see that this sets the whole system in the $E_{12} = +0.1$ minimum. After a relaxation time so that the system could adapt to the different energy function on the non transforming elements, we started to lift the boundary for u_1 at $y = y_{\min}$ upwards with a speed of 0.05, so that a traveling phase boundary was created through that hard loading. The time difference between two time steps t_2 and t_1 was 0.05. This loading results in the traveling wave seen in Figures 5.19 through 5.22.

The outcome might at first seem surprising. The traveling phase boundary does almost not change its shape when it hits the obstacle—as opposed to sound waves which would be dispersed. However, previous results (see [6]) already

indicate such a behavior even though this highly nonlinear process is essentially not understood at all. The simulation parameters were

Grid	50×200,
Element Size	0.5×0.5,
Boundary Conditions	Simply supported at $x = \text{const}$ boundaries, periodic at $y = \text{const}$,
Initial Conditions	$u_1 = 0.1y$, $u_2 = 0$
Strain Energy	See text,
Capillarity	$\gamma = 1$,
Viscosity	$\beta = 0.05$.

Appendix A

Notation

1. Derivatives
The partial derivative with respect to the variable x of a function u is denoted by u_x. The second derivative is denoted u_{xx} etc. To distinguish indices denoting differentiation from other indices a comma is used to separate them. For example, $u_{1,x}$ is the derivative of u_1 with respect to x.

The divergence of a matrix $A \in \mathrm{Mat(n,n)}$ is defined as the column vector the bundles the divergencies of the rows of A. To distinguish that from the "regular" divergence we write Div for the divergence of a matrix instead of div. We thus have $\mathrm{Div}A = \left(\sum_{k=1}^{n} \frac{\partial a_{jk}}{\partial x_k}\right)_{j=1,\ldots,n}$.

A dot above a function as usual denotes the total time derivative of the function.

2. Function Spaces
For spaces of functions defined on a domain Ω we use the standard notation. So, for example, $C^n(\Omega)$ denotes the space of n-times continuously differentiable functions defined on Ω. An index 0 is used if the functions vanish on the boundary. Other function spaces used are described in Appendix B.

Appendix B

Hilbert and Sobolev Spaces

1. Banach Spaces
A normed vector space $(X, ||\cdot||_X)$ is said to be a Banach space if it is *complete*. Complete means that every Cauchy sequence in the space X converges to an element of X. All n ($n \in \mathbb{N}$) dimensional Euclidean vector spaces are Banach spaces with the Euclidean norm, but usually the notion of Banach space is only used in the infinite dimensional setting, typically for function spaces.

2. Hilbert Spaces
A Hilbert space X is a vector space with an inner product $< \cdot, \cdot >$ such that the norm defined as $||f|| := \sqrt{< f, f >}$ for $f \in X$ turns the space into a complete metric space. Therefore, all Hilbert spaces are Banach spaces.

3. The space $L^2(\Omega)$
The vector space of all square integratable functions f on Ω is called $L^2(\Omega)$ and the L^2 norm $||\cdot||_{L^2}$ is defined by

$$||f||_{L^2} := \int_\Omega f^2 \mathrm{d}x.$$

For the domains under consideration, it can be shown that this space is a Hilbert space. See [10] for further information on L^p spaces.

4. Sobolev Spaces and the Weak Derivative
Suppose we want to solve a k-th order partial differential equation together with appropriate initial and boundary conditions on a suitable domain Ω. When trying to find a solution for such an equation, the classical way is to search in the space $C^k(\Omega)$, the space of k-times continuously differentiable functions defined on Ω. However, for some partial differential equations of this form this is not enough. Shock waves occurring in gas dynamics, for example, cannot be described by continuously differential functions. Therefore one has to find a different notion for differentiability and one can use a formal partial integration to achieve that. We will say that a function $f \in L^2$ defined on Ω is *weakly*

differentiable with respect to x if for every test function $\varphi \in C_0^\infty(\Omega)$, the space of smooth functions with compact support on Ω, one has

$$\int_\Omega f\varphi_x \mathrm{d}x = -\int_\Omega g\varphi \mathrm{d}x$$

with a function $g \in L^2$. The function g will be called the *weak derivative* of f.

The Sobolev space $H^1(\Omega), \Omega \subset \mathbb{R}$, is the space of all once partially weakly differentiable functions $f(x)$ endowed with the norm

$$||f||_{H^1} = ||f||_{L^2} + ||f_x||_{L^2}.$$

For $\Omega \subset \mathbb{R}^n$ the Sobolev space $H^1(\Omega)$ is defined analogously, with the norm including the sum over the L^2 norms of the higher derivatives. The Sobolev space H^n is accordingly the space of all functions that have partial derivatives up to order n in L^2. One can show that a Sobolev space modeled on L^2 is a Hilbert space. The character of the norm of the Sobolev spaces to include the L^2 norms of all derivatives is used in the proof in Section 3.5. Much more information on Sobolev spaces can be found in [10].

Appendix C

The MATLAB Program "shape"

code/shape

1. Main Program.

This is the implementation of the finite element algorithm using rectangular C^1 bicubic elements for the simulation of a martensite phase transformation with surface energy. Also includeed in this code is the program to display the computed results, *showview.m*, which reuses many of the initialisation routines for the program. The main program is divided into the following parts:

```
clear ; % remove any leftover variables from the workspace.
```

⟨ Specify Initial Conditions 2 ⟩
⟨ Initialise FEM Parameters 4 ⟩
⟨ Set Initial and Boundary Conditions 7 ⟩
⟨ Compute Element Matrix 12 ⟩
⟨ Prepare Energy Function 13 ⟩
⟨ Assemble System Matrix 14 ⟩
⟨ Time Integration 15 ⟩

2. Initial Conditions.
In this program section we specify the initial conditions for the partial differential equation.
Variables introduced:

u1_0	function for the initial value of u_1
u1_t_0	function for the initial value of $u_{1,t}$
u1_x_0	function for the initial value of $u_{1,x}$

and so on.

The element we use allows us to specify the spatial derivatives and the function values at each node independently (four degrees of freedom for every node: u, u_x, u_y and u_{xy}) and we chose to set them by introducing four functions like *u1_0* for the actual values at those nodes. The functions are evaluated on the coordinates of every node and should be specified on $\Omega = \left[\frac{-1}{2}Specimen\_sizex, \frac{1}{2}Specimen\_sizex\right] \times \left[\frac{-1}{2}Specimen\_sizey, \frac{1}{2}Specimen\_sizey\right]$
Here, we set the functions specifying the spatial derivatives to be the actual derivatives of the function values specified in *u1_0* and *u2_0* by using the symbolic derivative computed by *diff*.
Our second order differential equation also needs to be given the time derivatives of all these functions as initial conditions.

⟨ Specify Initial Conditions 2 ⟩ ≡

```
syms x y
u1_0 = 1/π∗sin(2∗π∗x/12.5);
u1_t_0 = sym(0);
u2_0 = sym(0);
u2_t_0 = sym(0);
u1_x_0 = diff(u1_0, x);
u1_y_0 = diff(u1_0, y);
u1_xy_0 = diff(diff(u1_0, x), y);
u1_tx_0 = diff(u1_t_0, x);
u1_ty_0 = diff(u1_t_0, y);
u1_txy_0 = diff(diff(u1_t_0, x), y);
u2_x_0 = diff(u2_0, x);
u2_y_0 = diff(u2_0, y);
u2_xy_0 = diff(diff(u2_0, x), y);
u2_tx_0 = diff(u2_t_0, x);
u2_ty_0 = diff(u2_t_0, y);
u2_txy_0 = diff(diff(u2_t_0, x), y);
```

This code is used in section 1.

3. Boundary Conditions.
In this program section we specify the boundary conditions for the differential equation. The vector *bc* specifies for which sides of the specimen the displacement u is constrained. If the first entry is nonzero, then the lower ($y = 0$) edge has this constraint, the second is for the right edge and so on. The variable *bc_n* specifies for which boundaries the normal derivative $\nabla u \cdot n$ should be constrained.
The symbolic functions describe the change of the boundary conditions with time t. They can be functions of t and ξ, where ξ is the coordinate along the boundary, going from $-\frac{1}{2}Specimen\_sizex$ to $\frac{1}{2}Specimen\_sizex$ (or y for boundarys $x = const$). The variable *bcu1* is for u_1, *bcu2* is for u_2. Concatenated is the number of the boundary, starting with one from the lower ($x = 0$) and counting counter clockwise. The functions with *_n* at the end specify the change in the boundary conditions for clamped boundaries, that is $\nabla u \cdot n$. The functions with *_t* are for the derivative along the boundary, this is obviously the derivative with respect to ξ and can be computed symbolically with *diff*.

⟨ Set Restrictions 3 ⟩ ≡

```
bc = [1 1 1 1];
bc_n = [0 0 0 0];

syms xi t

bcu1_1 = t;
bcu1_2 = t;
bcu1_3 = t;
bcu1_4 = t;

bcu1_1_n = t;
bcu1_2_n = t;
bcu1_3_n = t;
bcu1_4_n = t;

bcu2_1 = t;
bcu2_2 = t;
bcu2_3 = t;
bcu2_4 = t;

bcu2_1_n = t;
bcu2_2_n = t;
bcu2_3_n = t;
bcu2_4_n = t;
```

$bcu1\_1\_t = \mathit{diff}\,(bcu1\_1\,,\,xi)$;
$bcu1\_2\_t = \mathit{diff}\,(bcu1\_2\,,\,xi)$;
$bcu1\_3\_t = \mathit{diff}\,(bcu1\_3\,,\,xi)$;
$bcu1\_4\_t = \mathit{diff}\,(bcu1\_4\,,\,xi)$;
$bcu2\_1\_t = \mathit{diff}\,(bcu2\_1\,,\,xi)$;
$bcu2\_2\_t = \mathit{diff}\,(bcu2\_2\,,\,xi)$;
$bcu2\_3\_t = \mathit{diff}\,(bcu2\_3\,,\,xi)$;
$bcu2\_4\_t = \mathit{diff}\,(bcu2\_4\,,\,xi)$;

This code is used in section 7.

4. Initialisation.
Some basic parameters for the finite element method are set and others, like the nodal connection, are computed.

Variables introduced in this program section:

Specimen_Sizex	Size of the simulated object in x-direction
Specimen_Sizey	Size of the simulated object in y-direction
N_NDx,*N_NDy*	Number of nodes in x and y direction
N_ND	Total number of nodes in the system
N_NDpEL	Number of nodes per element (see element description in Section 5.4)
N_ELx, *N_ELy*	Number of elements in x and y direction
N_EL	Total number of elements
N_DOFpND	Number of degrees of freedom per node (see element description)
N_DOFpEL	Number of degrees of freedom per element (see element description)
N_DOF	Total number of degrees of freedom
Delta_t	Timestep
STime	Starting time
FTime	Final time
N_Time	Number of timesteps (*fix* rounds towards zero)
Capillarity_Factor	The capillarity γ of the system
Potential_Factor	The potential energy is multiplied by this factor
Viscosity_Factor	The factor β for the viscous stress
file1, *file2*	File names where *u1* and *u2* are saved to
Initial_File1, *Initial_File2*	Files to read the initial conditions from (if set)
Initial_File_index	At what timestep to start in the initial conditions file

⟨ Initialise FEM Parameters 4 ⟩ ≡

```
Specimen_Sizex = 25;
Specimen_Sizey = 25;
N_NDx = 51;
N_NDy = 51;
N_ND = N_NDx*N_NDy;
N_NDpEL = 4;
N_ELx = N_NDx − 1;
N_ELy = N_NDy − 1;
```

```
N_EL = N_ELx*N_ELy;
N_DOFpND = 4;
N_DOFpEL = N_NDpEL*N_DOFpND;
N_DOF = N_ND*N_DOFpND;

Delta_t = 0.0005;
STime = 0.0;
FTime = 500;
N_Time = fix((FTime - STime)/Delta_t);
Capillarity_Factor = 1;
Potential_Factor = 1;
Viscosity_Factor = 0.01;

file1 = '/home/pwd/u1.fem';
file2 = '/home/pwd/u2.fem';

Initial_File1 = '';
Initial_File2 = '';

Initial_File_index = 0;
⟨ Calculate Coordinates 5 ⟩
⟨ Compute Element-Node Connections 6 ⟩
```

This code is used in sections 1 and 24.

5. In this loop we calculate the x- and y-coordinate for every node.

Variables introduced:

$ncoord(nodeindex, 1)$	x-coordinate for node $nodeindex$
$ncoord(nodeindex, 2)$	y-coordinate for node $nodeindex$
$xlen$ and $ylen$	the length of one element in each direction

The nodes are arranged in N_NDy rows of N_NDx nodes, so when counting nodes with the indices $iNDx$ and $iNDy$ in x- and y-direction, $(iNDy-1)*N\_NDx$ gives us the number of nodes in the rows below the currently processed line; adding $iNDx$ we have the number of the node at ($iNDx$,$iNDy$). The x and y coordinate are then found by normalising $iNDx$ and $iNDy$ to run from $-\frac{1}{2}$ $Specimen\_Size$ to $\frac{1}{2}$ $Specimen\_Size$. Here we also have to take into account that the indices start from 1 and the coordinates from 0. Then we can set the $ncoord(nodeindex, 1)$ to the computed x-coordinate and $ncoord(nodeindex, 2)$ to the y-coordinate. For the indexing of the nodes see Figure 5.1 (N and M there are N_NDx and N_NDy here).
Since all elements have the same length, we compute this right now, taking the difference in the coordinates of the nodes in the first element. (See Figure 5.1.)

```
⟨ Calculate Coordinates 5 ⟩ ≡
  for iNDy = 1 : N_NDy
    for iNDx = 1 : N_NDx
      ncoord((iNDy − 1)*N_NDx + iNDx,
          1) = (iNDx − 1)*Specimen_Sizex/(N_NDx − 1) − Specimen_Sizex/2;
      ncoord((iNDy − 1)*N_NDx + iNDx,
          2) = (iNDy − 1)*Specimen_Sizey/(N_NDy − 1) − Specimen_Sizey/2;
    end
  end
  xlen = ncoord(2, 1) − ncoord(1, 1);
  ylen = ncoord(N_NDx + 1, 2) − ncoord(1, 2);
```

This code is used in section 4.

6. Here we determine which nodes belong to a certain element:

Variables introduced:

$nodes(elementindex, 1)$	First node belonging to element *elementindex*
$nodes(elementindex, 2)$	Second node belonging to element *elementindex*
$nodes(elementindex, 3)$	Third node belonging to element *elementindex*
$nodes(elementindex, 4)$	Fourth node belonging to element *elementindex*

The element index here is determined in the same fashion as the node index in the previous program section (only by counting elements instead of nodes), then the corresponding nodes are found counterclockwise around the element starting in the lower left corner. See Figure 5.1.

This code is of course only usable for a rectangular element with four nodes.

```
⟨ Compute Element-Node Connections 6 ⟩ ≡
  for iELy = 1 : N_ELy
    for iELx = 1 : N_ELx
      nodes((iELy − 1)∗N_ELx + iELx, 1) = (iELy − 1)∗N_NDx + iELx;
      nodes((iELy − 1)∗N_ELx + iELx, 2) = (iELy − 1)∗N_NDx + iELx + 1;
      nodes((iELy − 1)∗N_ELx + iELx, 3) = (iELy)∗N_NDx + iELx + 1;
      nodes((iELy − 1)∗N_ELx + iELx, 4) = (iELy)∗N_NDx + iELx;
    end
  end
```

This code is used in section 4.

7. Boundary- and Initial Conditions.

For the initial conditions we can either start from scratch with the functions given in program section 1 or we continue the computations from a file with old data.

⟨ Set Initial and Boundary Conditions 7 ⟩ ≡

```
if isempty(Initial_File1)
  ⟨ Set Initial Conditions According to Given Functions 8 ⟩
else
  ⟨ Read Initial Conditions from File 9 ⟩
end
⟨ Set Restrictions 3 ⟩
⟨ Compute Boundary Conditions 10 ⟩
```

This code is used in section 1.

8. Set Initial Conditions

t1u1	the solution vector for u_1 at time 0
t1u2	the solution vector for u_2 at time 0
t2u1, *t2u2*	the corresponding vectors at the time Δt

In each of these vectors we have $N\_DOFpND = 4$ degrees of freedom (dof) per node, each of which has its own symbolic function that specifies one initial condition for this degree of freedom. We set the dofs by evaluating these functions for the initial conditions at the corrdinates x and y for each node. The value obtained is stored in the corresponding vector for the initial condition at the index of the corresponding degree of freedom: For each node we have four entries in the vector, the first at $iND*4-3$ for the function value, the second at $iND*4-2$ for the x-derivative at this point, the third at $iND*4-1$ for the y-derivative and finally the fourth at iND for the xy-derivative. The solution vectors for the time Δt are computed by adding the initial conditions for the time derivatives multiplied by the timestep. The degrees of freedom addressing spatial derivatives have to be multiplied by the corresponding element lengths because the actual derivative of the function on an element is the derivative of the shape function times the value of the corresponding degree of freedom and the derivatives of the shape functions obviously are inversely proportional to the element length (See description of the shape functions in program section 12).

The symbolic functions for the initial conditions are no longer needed afterwards and thus are cleared from memory.

⟨ Set Initial Conditions According to Given Functions 8 ⟩ ≡

```
t1u1 = zeros(N_DOF, 1);
t2u1 = zeros(N_DOF, 1);
t1u2 = zeros(N_DOF, 1);
t2u2 = zeros(N_DOF, 1);
for iND = 1 : N_ND
  x = ncoord(iND, 1);
  y = ncoord(iND, 2);
  t1u1(iND*4 − 3) = eval(u1_0);
  t1u1(iND*4 − 2) = eval(u1_x_0)*xlen;
  t1u1(iND*4 − 1) = eval(u1_y_0)*ylen;
  t1u1(iND*4) = eval(u1_xy_0)*xlen*ylen;
  t2u1(iND*4 − 3) = t1u1(iND*4 − 3) + Delta_t*eval(u1_t_0);
  t2u1(iND*4 − 2) = t1u1(iND*4 − 2) + Delta_t*eval(u1_tx_0)*xlen;
  t2u1(iND*4 − 1) = t1u1(iND*4 − 1) + Delta_t*eval(u1_ty_0)*ylen;
  t2u1(iND*4) = t1u1(iND*4) + Delta_t*eval(u1_txy_0)*xlen*ylen;
  t1u2(iND*4 − 3) = eval(u2_0);
  t1u2(iND*4 − 2) = eval(u2_x_0)*xlen;
  t1u2(iND*4 − 1) = eval(u2_y_0)*ylen;
  t1u2(iND*4) = eval(u2_xy_0)*xlen*ylen;
  t2u2(iND*4 − 3) = t1u2(iND*4 − 3) + Delta_t*eval(u2_t_0);
```

```
    t2u2(iND*4 - 2) = t1u2(iND*4 - 2) + Delta_t*eval(u2_tx_0)*xlen;
    t2u2(iND*4 - 1) = t1u2(iND*4 - 1) + Delta_t*eval(u2_ty_0)*ylen;
    t2u2(iND*4) = t1u2(iND*4) + Delta_t*eval(u2_txy_0)*xlen*ylen;
  end
  clear u1_0 u1_x_0 u1_y_0 u1_xy_0 u1_t_0 u1_tx_0 u1_ty_0 u1_txy_0
  clear u2_0 u2_x_0 u2_y_0 u2_xy_0 u2_t_0 u2_tx_0 u2_ty_0 u2_txy_0
  clear x y
  pack
```

This code is used in section 7.

9. Read Initial Conditions from File.

We open the files and read one inital data set. Then, in the loop we assign the current data to the variables for the data from one timestep ago and read one more data set until we get to the index in *Initial_File_index*.

⟨ Read Initial Conditions from File 9 ⟩ ≡

```
  fid1 = fopen(Initial_File1);
  fid2 = fopen(Initial_File2);
  t2u1 = fread(fid1, N_DOF, 'double');
  t2u2 = fread(fid2, N_DOF, 'double');
  for it = 2 : Initial_File_index
    it
    t1u1 = t2u1;
    t1u2 = t2u2;
    t2u1 = fread(fid1, N_DOF, 'double');
    t2u2 = fread(fid2, N_DOF, 'double');
  end
```

This code is used in section 7.

10. In this program section we determine which boundary conditions have to be set and what their initial values have to be.
Variables introduced

$bcdof\,(1)$	Index of first constrained degree of freedom for u_1 and u_2
$bcval1\_0\,(1)$, $bcval2\_0\,(n)$	Corresponding initial values for u_1 and u_2
⋮	⋮
$bcdof\,(n)$	Index of last constrained degree of freedom for u_1 and u_2
$bcval1\_0\,(n)$, $bcval2\_0\,(n)$	Corresponding initial values for u_1 and u_2

First, in the "Find boundary" program section we find out which degrees of freedom belong to a certain boundary.

After initialising the index counter cc, we go through a **for** loop for every boundary condition that needs to be set. The first four are for u itself, the next four are for the tangential derivative which is set when u is set, the last four are for the normal derivative of u, which is controlled by the bc_n variable. The four loops are one for every side of the specimen.

These loops basically all do the same: They check every entry in the vector that is storing the information about the indices of the constrained degrees of freedom. The counter cc is incremented so that a new entry can be written into $bcdof$ and $bcval$. The respective index number is written into $bcdof$ and the values in $bcval1$ and $bcval2$ are set to be the values of the inital conditions on these degrees of freedom.

⟨ Compute Boundary Conditions 10 ⟩ ≡

```
⟨ Find Boundary 11 ⟩
cc = 0;
if (bc(1) ≠ 0)
  for iND = 1 : length (boundarydof1_u)
    cc = cc + 1;
    bcdof(cc) = boundarydof1_u(1, iND);
    bcval1_0(cc) = t1u1(boundarydof1_u(1, iND));
    bcval2_0(cc) = t1u2(boundarydof1_u(1, iND));
  end
end
if (bc(2) ≠ 0)
  for iND = 1 : length (boundarydof2_u)
    cc = cc + 1;
    bcdof(cc) = boundarydof2_u(1, iND);
    bcval1_0(cc) = t1u1(boundarydof2_u(1, iND));
    bcval2_0(cc) = t1u2(boundarydof2_u(1, iND));
```

```
  end
end
if (bc(3) ≠ 0)
  for iND = 1 : length (boundarydof3_u)
    cc = cc + 1;
    bcdof(cc) = boundarydof3_u(1, iND);
    bcval1_0(cc) = t1u1(boundarydof3_u(1, iND));
    bcval2_0(cc) = t1u2(boundarydof3_u(1, iND));
  end
end
if (bc(4) ≠ 0)
  for iND = 1 : length (boundarydof4_u)
    cc = cc + 1;
    bcdof(cc) = boundarydof4_u(1, iND);
    bcval1_0(cc) = t1u1(boundarydof4_u(1, iND));
    bcval2_0(cc) = t1u2(boundarydof4_u(1, iND));
  end
end
if (bc(1) ≠ 0)
  for iND = 1 : length (boundarydof1_u_t)
    cc = cc + 1;
    bcdof(cc) = boundarydof1_u_t(1, iND);
    bcval1_0(cc) = t1u1(boundarydof1_u_t(1, iND));
    bcval2_0(cc) = t1u2(boundarydof1_u_t(1, iND));
  end
end
if (bc(2) ≠ 0)
  for iND = 1 : length (boundarydof2_u_t)
    cc = cc + 1;
    bcdof(cc) = boundarydof2_u_t(1, iND);
    bcval1_0(cc) = t1u1(boundarydof2_u_t(1, iND));
    bcval2_0(cc) = t1u2(boundarydof2_u_t(1, iND));
  end
end
if (bc(3) ≠ 0)
  for iND = 1 : length (boundarydof3_u_t)
    cc = cc + 1;
    bcdof(cc) = boundarydof3_u_t(1, iND);
    bcval1_0(cc) = t1u1(boundarydof3_u_t(1, iND));
    bcval2_0(cc) = t1u2(boundarydof3_u_t(1, iND));
  end
end
if (bc(4) ≠ 0)
  for iND = 1 : length (boundarydof4_u_t)
    cc = cc + 1;
```

```
      bcdof(cc) = boundarydof4_u_t(1, iND);
      bcval1_0(cc) = t1u1(boundarydof4_u_t(1, iND));
      bcval2_0(cc) = t1u2(boundarydof4_u_t(1, iND));
    end
  end
  if (bc_n(1) ≠ 0)
    for iND = 1 : length (boundarydof1_u_n)
      cc = cc + 1;
      bcdof(cc) = boundarydof1_u_n(1, iND);
      bcval1_0(cc) = t1u1(boundarydof1_u_n(1, iND));
      bcval2_0(cc) = t1u2(boundarydof1_u_n(1, iND));
    end
  end
  if (bc_n(2) ≠ 0)
    for iND = 1 : length (boundarydof2_u_n)
      cc = cc + 1;
      bcdof(cc) = boundarydof2_u_n(1, iND);
      bcval1_0(cc) = t1u1(boundarydof2_u_n(1, iND));
      bcval2_0(cc) = t1u2(boundarydof2_u_n(1, iND));
    end
  end
  if (bc_n(3) ≠ 0)
    for iND = 1 : length (boundarydof3_u_n)
      cc = cc + 1;
      bcdof(cc) = boundarydof3_u_n(1, iND);
      bcval1_0(cc) = t1u1(boundarydof3_u_n(1, iND));
      bcval2_0(cc) = t1u2(boundarydof3_u_n(1, iND));
    end
  end
  if (bc_n(4) ≠ 0)
    for iND = 1 : length (boundarydof4_u_n)
      cc = cc + 1;
      bcdof(cc) = boundarydof4_u_n(1, iND);
      bcval1_0(cc) = t1u1(boundarydof4_u_n(1, iND));
      bcval2_0(cc) = t1u2(boundarydof4_u_n(1, iND));
    end
  end
```

This code is used in section 7.

11. In this program sections we determine which degrees of freedom belong to the boundary.
First we find the nodes on every side of the specimen. This information is saved in *boundarynodes1* for the first (lower, $y = y_{\min}$) side of the specimen, and continuing to count counter-clockwise. Then we find the degrees of freedom corresponding to the various boundary conditions. Since there are four degrees of freedom (u, u_x, u_y and u_{xy}) for every node their indices are found by multiplying the index of the node minus 1 with 4 and then adding 1 for u, 2 for u_y and so on. In the second row of the *boundarydof* vectors we save the coordinate along the boundary of the node the dof belongs to. The first 4 loops find the indices of the degrees of freedom for u, where only one—the starting—corner node is used, therefore we loop only to $N\_NDx - 1$ or $N\_NDy - 1$. The next 4 loops are for the tangential derivative, so the first of them finds the indices of the x-derivative, the second the y-derivative, the third x again and the last of them finds the y derivative. The last 4 loops are for the normal derivative, they find the indices of the respectively other derivatives.

⟨ Find Boundary 11 ⟩ ≡

```
boundarynodes1 = (1 : N_NDx);
boundarynodes2 = N_NDx*(1 : N_NDy);
boundarynodes3 = N_NDx*N_NDy − ((1 : N_NDx) − 1);
boundarynodes4 = (1 + N_NDx*(N_NDy − 1)) − (N_NDx*((1 : N_NDy) − 1));
for iND = 1 : N_NDx
  boundarydof1_u(1, iND) = (boundarynodes1(iND) − 1)*4 + 1;
  boundarydof1_u(2, iND) = ncoord(boundarynodes1(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof2_u(1, iND) = (boundarynodes2(iND) − 1)*4 + 1;
  boundarydof2_u(2, iND) = ncoord(boundarynodes2(iND), 2);
end
for iND = 1 : N_NDx
  boundarydof3_u(1, iND) = (boundarynodes3(iND) − 1)*4 + 1;
  boundarydof3_u(2, iND) = ncoord(boundarynodes3(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof4_u(1, iND) = (boundarynodes4(iND) − 1)*4 + 1;
  boundarydof4_u(2, iND) = ncoord(boundarynodes4(iND), 2);
end
for iND = 1 : N_NDx
  boundarydof1_u_t(1, iND) = (boundarynodes1(iND) − 1)*4 + 2;
  boundarydof1_u_t(2, iND) = ncoord(boundarynodes1(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof2_u_t(1, iND) = (boundarynodes2(iND) − 1)*4 + 3;
  boundarydof2_u_t(2, iND) = ncoord(boundarynodes2(iND), 2);
```

```
end
for iND = 1 : N_NDx
  boundarydof3_u_t(1, iND) = (boundarynodes3(iND) − 1)∗4 + 2;
  boundarydof3_u_t(2, iND) = ncoord(boundarynodes3(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof4_u_t(1, iND) = (boundarynodes4(iND) − 1)∗4 + 3;
  boundarydof4_u_t(2, iND) = ncoord(boundarynodes4(iND), 2);
end
for iND = 1 : N_NDx
  boundarydof1_u_n(1, iND) = (boundarynodes1(iND) − 1)∗4 + 3;
  boundarydof1_u_n(2, iND) = ncoord(boundarynodes1(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof2_u_n(1, iND) = (boundarynodes2(iND) − 1)∗4 + 2;
  boundarydof2_u_n(2, iND) = ncoord(boundarynodes2(iND), 2);
end
for iND = 1 : N_NDx
  boundarydof3_u_n(1, iND) = (boundarynodes3(iND) − 1)∗4 + 3;
  boundarydof3_u_n(2, iND) = ncoord(boundarynodes3(iND), 1);
end
for iND = 1 : N_NDy
  boundarydof4_u_n(1, iND) = (boundarynodes4(iND) − 1)∗4 + 2;
  boundarydof4_u_n(2, iND) = ncoord(boundarynodes4(iND), 2);
end
```

This code is used in section 10.

12. Element Matrix.
To compute the element matrix we use MATLAB's symbolic math package, an interface to the MAPLE kernel.
Variables introduced:

f,*g*	Vectors with the cubic 1D shape functions, *f* for *zeta* direction, *g* for *eta*
phi	Vector with the 2D shape functions, tensor products of the 1D shape functions
lapphi	Vector that contains the functions $\triangle N$
Ke	Element stiffness matrix
Me	Element mass matrix

First we set *eta* and *zeta* to be symbolic variables, so that all functions containing these variables will be stored as symbolic functions. Then we specify the 1D shape functions in F and G, where we already normalize to the element length, and we create the 2D shape functions in N as tensor products of the 1D functions (See Section 5.4). We immediately take the divergence of each of these functions in N and store that in the vector *lapN* because this is the only thing we use in the element matrix.
Then we calculate the element stiffness matrix *Ke* and the element mass matrix *Me*:

$$K_{i,j}^{(e)} = \int_{\Omega_e} \triangle\varphi_i \triangle\varphi_j \,\mathrm{d}V \tag{C.1}$$

$$M_{i,j}^{(e)} = \int_{\Omega_e} \varphi_i \varphi_j \,\mathrm{d}V \tag{C.2}$$

where Ω is the element domain and φ_i are the shape functions.
To save time we make use of the fact that *Ke* and *Me* are symmetric matrices. After that, the variables f, g and *lapphi* are no longer needed and we remove them from memory.

⟨ Compute Element Matrix 12 ⟩ ≡

```
syms eta zeta;
f = [(1 - zeta/xlen)^2*(1 + 2*zeta/xlen);
    (zeta/xlen)*(1 - zeta/xlen)^2;
    (zeta/xlen)^2*(3 - 2*zeta/xlen);
    -(zeta/xlen)^2*(1 - zeta/xlen)];
g = [(1 - eta/ylen)^2*(1 + 2*eta/ylen);
    (eta/ylen)*(1 - eta/ylen)^2;
    (eta/ylen)^2*(3 - 2*eta/ylen);
    -(eta/ylen)^2*(1 - eta/ylen)];
phi(1) = f(1)*g(1);  phi(2) = f(2)*g(1);  phi(3) = f(1)*g(2);
phi(4) = f(2)*g(2);
phi(5) = f(3)*g(1);  phi(6) = f(4)*g(1);  phi(7) = f(3)*g(2);
phi(8) = f(4)*g(2);
```

```
phi(13) = f(1)*g(3);  phi(14) = f(2)*g(3);  phi(15) = f(1)*g(4);
phi(16) = f(2)*g(4);
phi(9) = f(3)*g(3);  phi(10) = f(4)*g(3);  phi(11) = f(3)*g(4);
phi(12) = f(4)*g(4);
for i = 1:16
  lapphi(i) = diff(diff(phi(i), zeta), zeta) + diff(diff(phi(i), eta), eta);
end
for i = 1:16
  for j = i:16
    Ke(i, j) = eval(int(int(lapphi(i)*lapphi(j), eta, 0, ylen), zeta, 0,
        xlen));
    Ke(j, i) = Ke(i, j);
    Me(i, j) = eval(int(int(phi(i)*phi(j), eta, 0, ylen), zeta, 0, xlen));
    Me(j, i) = Me(i, j);
  end
end
clear f g lapphi
```

This code is used in sections 1 and 24.

13. Prepare Energy Function.
Variables introduced:

phi_x(*i*), *phi_y*(*i*)	Spatial derivatives of the shape functions (symbolic)
Shape_Int_x, *Shape_Int_y*	Integrals of the spatial derivatives of the shape functions

In this program section we prepare the integration of the potential. First we specify the strain tensor F depending on the spatial derivatives of the displacement, then we compute the Cauchy-Green strain tensor $C = F^t F$. The potential energy can then be expressed symbolically in terms of the entries of the Cauchy-Green strain tensor. The stress tensor $\sigma = \frac{\partial \Phi(F)}{\partial F}$ can then be computed symbolically with MATLAB's *diff* function. For the final integration we will need the integrals of the spatial derivatives of the shape functions over the element domain and so we precalculate them symbolically but evaluate the actual value of the integral and store it in the *Shape_Int_x* and *Shape_Int_y* vector.

⟨ Prepare Energy Function 13 ⟩ ≡

```
syms u1_x u1_y u2_x u2_y
F = [u1_x + 1, u1_y;  u2_x, u2_y + 1];
C = transpose(F)*(F);
Phi = (C(1, 1) - 1)^2 + (C(2, 2) - 1)^2 + (C(1, 2)^2 - 0.1)^2;
sigma(1, 1) = diff(Phi, u1_x);
sigma(1, 2) = diff(Phi, u1_y);
sigma(2, 1) = diff(Phi, u2_x);
sigma(2, 2) = diff(Phi, u2_y);
Shape_Int_x = zeros(N_DOFpEL, 1);
Shape_Int_y = zeros(N_DOFpEL, 1);
for iDOF = 1 : N_DOFpEL
  phi_x(iDOF) = diff(phi(iDOF), zeta);
  Shape_Int_x(iDOF) = eval(int(int(phi_x(iDOF), eta, 0, xlen), zeta, 0,
      ylen));
  phi_y(iDOF) = diff(phi(iDOF), eta);
  Shape_Int_y(iDOF) = eval(int(int(phi_y(iDOF), eta, 0, xlen), zeta, 0,
      ylen));
end
```

This code is used in sections 1 and 24.

14. Assemble System Matrix.

f1, *f2*	the system vectors for u_1 and u_2
M	the system mass matrx
K	the system stiffness matrix
u1, *u2*	the solution vectors
$index(1)$	first degree of freedom for the currently processed element
$index(2)$	second dof
$\vdots$	$\vdots$
$index(N\_DOFpEL)$	last degree of freedom for the element

For every element we extract the corresponding nodes from the *nodes* matrix, then we find the degrees of freedom for these nodes by using the function *elementdof* and store them into *index*. The function *assemble* adds the entries of the element matrix *Ke* to the entries in K described by *index* (Same for the mass matrix). Finally, the stiffness matrix is multiplied by the factor for the capillarity. The functions *elementdof* and *assemble* have been taken from [35].

```
⟨ Assemble System Matrix 14 ⟩ ≡
  f1 = zeros(N_DOF, 1);
  f2 = zeros(N_DOF, 1);
  K = sparse(N_DOF, N_DOF);
  M = sparse(N_DOF, N_DOF);
  u1 = zeros(N_DOF, 1);
  u2 = zeros(N_DOF, 1);
  index = zeros(N_NDpEL*N_DOFpND, 1);
  for iEL = 1 : N_EL
    for i = 1 : N_NDpEL
      nd(i) = nodes(iEL, i);
    end
    index = elementdof(nd, N_NDpEL, N_DOFpND);
    K = assemble(K, Ke, index);
    M = assemble(M, Me, index);
  end
  K = K*Capillarity_Factor;
```

This code is used in sections 1 and 24.

15. Time Integration.
We want to discretise the following equation for both *f1* & *u1* and *f2* & *u2*:

$$M\ddot{u} + Ku = f \tag{C.3}$$

so we say

$$\ddot{u} = \frac{u^{(t+2\Delta t)} - 2u^{(t+\Delta t)} + u^{(t)}}{(\Delta t)^2} \tag{C.4}$$

and get:

$$\left(M + (\Delta t)^2 K\right) u^{(t+2\Delta t)} = M\left(2u^{(t+\Delta t)} - u^{(t)}\right) + (\Delta t)^2 f \tag{C.5}$$

We do this by storing the entire matrix on the left hand side into K and the entire right hand side into the effective system vectors *fn1* and *fn2*. Then we solve $Ku = f$. So first we prepare K and apply the boundary conditions to this matrix. Also, we can immediately compute the LU-factorization of K, because the matrix will remain the same throughout the time integration. This saves time when solving the system of linear equations. K itself is no longer needed after that and thus cleared from memory.
In the time integration loop we first integrate the potential energy into the force vectors *f1* and *f2*. Then we compute the effective force vectors *fn1* and *fn2*, as decribed above (right hand side). After applying the boundary conditions to the force vectors the linear equations can be solved and the results are saved into the files.

⟨ Time Integration 15 ⟩ ≡
```
K = M + Delta_t^2*K;
⟨ Apply Boundary Conditions to Matrix 16 ⟩
[L, U] = lu(K);
clear K
pack
for it = 1 : N_Time
  t = it*Delta_t
  ⟨ Integrate Potential 17 ⟩
  fn1 = M*(2*t2u1 - t1u1) + Delta_t^2*f1;
  fn2 = M*(2*t2u2 - t1u2) + Delta_t^2*f2;
  ⟨ Apply Boundary Conditions to System Vectors 18 ⟩
  ⟨ Solve System of Linear Equations 20 ⟩
  ⟨ Save Results 21 ⟩
  t1u1 = t2u1;
  t2u1 = u1;
  t1u2 = t2u2;
  t2u2 = u2;
end;
```
This code is used in section 1.

16. Apply Boundary Conditions to Matrix.

We want to make sure that in the system of linear equations certain elements in the resulting vector have a specified value. To achieve this we set the entire row in the matrix to be zero except for the index of the constrained degree of freedom where it is one. If we then later (program section 18) set the corresponding entry in the force vector to the desired value for this dof, this value will also be the result. We repeat this procedure for every constraint specified in the *bcdof* vector.

⟨ Apply Boundary Conditions to Matrix 16 ⟩ ≡

```
for ibc = 1 : length (bcdof)
  bcindex = bcdof(ibc);
  K(bcindex, :) = 0;
  K(bcindex, bcindex) = 1;
end
```

This code is used in section 15.

17. Integrate Potential.

After initialising the force vectors, we loop for every element to do the integration. First we get the indices of the degrees of freedom for the current element, exactly as in the "Assemble System Matrix" program section. We then compute the strain, that is the spatial derivatives of the displacement,

$$u_x(x,y) = \sum_{i=1}^{N} u_i \varphi_{i,x}(x,y).$$

The same for the other derivatives, too. Here the approximation is made, that one considers the gardient, and therefore the derived quantities like the stress tensor, to be constant on one element. One can say that the element is linearized before integration. With this approximation one obtains

$$\begin{aligned} \int_{\Omega_E} u_x \mathrm{d}V &= \int_{\Omega_E} \sum_{i=1}^{N} u_i \varphi_{i,x}(x,y) \mathrm{d}V \\ &= \sum_{i=1}^{N} u_i \cdot \int_{\Omega_E} \varphi_{i,x}(x,y) \mathrm{d}V. \end{aligned}$$

Since we have already prepared the integrals over the shape functions, all we hav to do is to dot-product this vector with the vector containing the coefficients of the element in the right order. This is exactly *t2u1*(*index*) or *t2u2*(*index*). After that, we evaluate the stress tensor for the just computed strain. Since the strain tensor has already been specified as a symbolic function in the variables *u1_x*, *u1_y*, *u2_x* and *u2_y* the MATLAB function *eval* will automatically evaluate σ at these just computed variables. Then, we are ready to integrate this into the force vectors:
The force vector of the element is given by

$$f_i = \int_{\Omega_E} \varphi_i \cdot (\mathrm{Div}\sigma + \triangle u_t) \mathrm{d}V = -\int_{\Omega_e} \nabla\varphi_i \cdot (\sigma + \beta \nabla u_t) \mathrm{d}V.$$

where $\sigma = \frac{\partial \Phi}{\partial F}$ is the stress tensor and φ_i are the shape functions of the element. The boundary terms cancel on the inside of the whole domain Ω. The time derivatives of the gradients are computed by taking the difference of the gradients from two timesteps. They are used to compute the viscous stress.

Here, the integration of the stress tensor σ and the viscous stress is simply done by multiplying the precalculated integrals of the spatial derivatives of the shape functions with the stress. The result is added to the force vector of the system at the indices of the degrees of freedom of the element.

⟨ Integrate Potential 17 ⟩ ≡

```
f1 = zeros(N_DOF, 1);
f2 = zeros(N_DOF, 1);
for iEL = 1 : N_EL
```

```
for i = 1 : N_NDpEL
  nd(i) = nodes(iEL, i);
end
index = elementdof(nd, N_NDpEL, N_DOFpND);
u1_x = dot(t2u1(index), Shape_Int_x/(xlen*ylen));
u1_y = dot(t2u1(index), Shape_Int_y/(xlen*ylen));
u2_x = dot(t2u2(index), Shape_Int_x/(xlen*ylen));
u2_y = dot(t2u2(index), Shape_Int_y/(xlen*ylen));
u1_xt = (u1_x - dot(t1u1(index), Shape_Int_x/(xlen*ylen)))/Delta_t;
u1_yt = (u1_y - dot(t1u1(index), Shape_Int_y/(xlen*ylen)))/Delta_t;
u2_xt = (u2_x - dot(t1u2(index), Shape_Int_x/(xlen*ylen)))/Delta_t;
u2_yt = (u2_y - dot(t1u2(index), Shape_Int_y/(xlen*ylen)))/Delta_t;
esigma = eval(sigma)*Potential_Factor;
f1(index) = f1(index) - ((esigma(1,
      1) + Viscosity_Factor*u1_xt)*Shape_Int_x + (esigma(1,
      2) + Viscosity_Factor*u1_yt)*Shape_Int_y);
f2(index) = f2(index) - ((esigma(2,
      1) + Viscosity_Factor*u2_xt)*Shape_Int_x + (esigma(2,
      2) + Viscosity_Factor*u2_yt)*Shape_Int_y);
end
```

This code is used in section 15.

18. Apply Boundary Conditions to System Vectors.
First we compute the new boundary conditions for this timestep. Then, for every constrained degree of freedom, we set the effective force vectors at the constrained index to be the value to which they are constrained. (See program section 16)

⟨ Apply Boundary Conditions to System Vectors 18 ⟩ ≡

```
⟨ Compute new Boundary Conditions 19 ⟩
for ibc = 1 : length (bcdof)
  bcindex = bcdof(ibc);
  fn1(bcindex) = bcval1(ibc);
  fn2(bcindex) = bcval2(ibc);
end
```

This code is used in section 15.

19. In this program section we compute the new boundary conditions for this timestep.

This works much like setting the initial boundary conditions in program section 10. We first set the variable ξ that is used in the functions that describe the change in the boundary conditions as the coordinate along the boundary to the value stored for it in the *boundarydof* matrix, then we evaluate these functions and add their value to the respective initial boundary condition and save the result in the vector *bcval*.

⟨ Compute new Boundary Conditions 19 ⟩ ≡

```
cc = 0;
if (bc(1) ≠ 0)
  for iND = 1 : length (boundarydof1_u)
    cc = cc + 1;
    xi = boundarydof1_u(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_1);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_1);
  end
end
if (bc(2) ≠ 0)
  for iND = 1 : length (boundarydof2_u)
    cc = cc + 1;
    xi = boundarydof2_u(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_2);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_2);
  end
end
if (bc(3) ≠ 0)
  for iND = 1 : length (boundarydof3_u)
    cc = cc + 1;
    xi = boundarydof3_u(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_3);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_3);
  end
end
if (bc(4) ≠ 0)
  for iND = 1 : length (boundarydof4_u)
    cc = cc + 1;
    xi = boundarydof4_u(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_4);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_4);
  end
end
if (bc(1) ≠ 0)
```

```
  for iND = 1 : length (boundarydof1_u_t)
    cc = cc + 1;
    xi = boundarydof1_u_t(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_1_t);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_1_t);
  end
end
if (bc(2) ≠ 0)
  for iND = 1 : length (boundarydof2_u_t)
    cc = cc + 1;
    xi = boundarydof2_u_t(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_2_t);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_2_t);
  end
end
if (bc(3) ≠ 0)
  for iND = 1 : length (boundarydof3_u_t)
    cc = cc + 1;
    xi = boundarydof3_u_t(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_3_t);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_3_t);
  end
end
if (bc(4) ≠ 0)
  for iND = 1 : length (boundarydof4_u_t)
    cc = cc + 1;
    xi = boundarydof4_u_t(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_4_t);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_4_t);
  end
end
if (bc_n(1) ≠ 0)
  for iND = 1 : length (boundarydof1_u_n)
    cc = cc + 1;
    xi = boundarydof1_u_n(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_1_n);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_1_n);
  end
end
if (bc_n(2) ≠ 0)
  for iND = 1 : length (boundarydof2_u_n)
    cc = cc + 1;
    xi = boundarydof2_u_n(2, iND);
    bcval1(cc) = bcval1_0(cc) + eval(bcu1_2_n);
    bcval2(cc) = bcval2_0(cc) + eval(bcu2_2_n);
```

```
    end
  end
  if (bc_n(3) ≠ 0)
    for iND = 1 : length (boundarydof3_u_n)
      cc = cc + 1;
      xi = boundarydof3_u_n(2, iND);
      bcval1(cc) = bcval1_0(cc) + eval(bcu1_3_n);
      bcval2(cc) = bcval2_0(cc) + eval(bcu2_3_n);
    end
  end
  if (bc_n(4) ≠ 0)
    for iND = 1 : length (boundarydof4_u_n)
      cc = cc + 1;
      xi = boundarydof4_u_n(2, iND);
      bcval1(cc) = bcval1_0(cc) + eval(bcu1_4_n);
      bcval2(cc) = bcval2_0(cc) + eval(bcu2_4_n);
    end
  end
```

This code is used in section 18.

20. Solve Linear Equations.
We first solve the system $fn1 = L*y$ and then $y = U*u$. The matrices L and U are the results of the LU-factorization carried out earlier.

⟨ Solve System of Linear Equations 20 ⟩ ≡

```
y = L\fn1;
u1 = U\y;
y = L\fn2;
u2 = U\y;
```

This code is used in section 15.

21. Save.
The results are saved in two files, one for u_1 and one for u_2. The data in $u1$ and $u2$ we just computed is appended to the files.

⟨ Save Results 21 ⟩ ≡

```
fid1 = fopen(file1, 'a');
fwrite(fid1, u1, 'double');
fclose(fid1);
fid2 = fopen(file2, 'a');
fwrite(fid2, u2, 'double');
fclose(fid2);
```

This code is used in section 15.

22. Assemble the Matrices.
This function assembles the entries of the element stiffness matrix or mass matrix into the system stiffness matrix or mass matrix. The parameters it is calles with are K, the actual system matrix, Ke, the element matrix, and *index*, the indices of the degrees of freedom for the element.

First, we find the number of degrees of freedom per element by looking at the lenght of the *index* vector. Then we go through all the entries in the element matrix and add them to the according indices in the system matrix. For the i-th element dof, $index(i)$ stores the index of the system dof. This function has been taken from [35].

⟨ `assemble.m` 22 ⟩ ≡

```
function [K] =assemble(K, Ke, index)
  N_DOF = length(index);
  for iDOF = 1 : N_DOF
    DOFindex_i = index(iDOF);
    for jDOF = 1 : N_DOF
      DOFindex_j = index(jDOF);
      K(DOFindex_i, DOFindex_j) = K(DOFindex_i,
          DOFindex_j) + Ke(iDOF, jDOF);
    end
  end
```

23. Find Degrees of Freedom for an Element.
This function is used to find the indices of the degrees of freedom for a given element. To do this, one first has to loop over the nodes in the element. Their indices can be found in the vector *nodes*. Then one can find the index of the first dof for this node. Since there are *N_DOFpEL* dofs for every element one has to skip *N_DOFpEL* times the node number minus one to get to the last index for the previous element. By adding 1 one has the first dof index for the node. Then one adds *N_DOFpND* entries to *index*, the indices of the dofs are in a row. This function has been taken from [35].

```
⟨ elementdof.m 23 ⟩ ≡
  function [index] =elementdof(nodes, N_NDpEL, N_DOFpND)
    k = 0;
    for iND = 1 : N_NDpEL
      start = (nodes(iND) − 1)∗N_DOFpND + 1;
      for iDOF = 1 : N_DOFpND
        k = k + 1;
        index(k) = start − 1 + iDOF;
      end
    end
```

24. Display.
This helper application called *showview.m* reads the data computed by the main program shape.m and displays it on the screen. To do this we need a lot of the parameters of the main program and so we reuse a few of the program sections that perform the initialisation in *shape.m*. The only new part here is the Display Loop that does the actual displaying.

⟨ `showview.m` 24 ⟩ ≡
```
clear ;
⟨ Initialise FEM Parameters 4 ⟩
⟨ Compute Element Matrix 12 ⟩
⟨ Prepare Energy Function 13 ⟩
⟨ Assemble System Matrix 14 ⟩
⟨ Display Loop 25 ⟩
```

25. Display Loop.

In this program section we read the computed data from the files and prepare it for displaying.

First we read one set of values into memory from the files, then we start the **while** loop. In the loop we first read a new set of values after setting the previously read values to be those of one timestep ago (*t1u1*, *t1u2*). If we encountered the end of the files while trying to read the files we break out of the loop.
Then we compute the displayable results. This part can be altered to compute specific order parameters of the system, too. If one wants to make surface plots of the variables computed here, one first has to assemble the result vectors into a suitable MATLAB matrix for display.
After we left the **while** loop we close the files.

```
⟨ Display Loop 25 ⟩ ≡
  it = 0;
  fid1 = fopen(file1, 'r');
  fid2 = fopen(file2, 'r');
  u1 = fread(fid1, N_DOF, 'double');
  u2 = fread(fid2, N_DOF, 'double');
  while (¬feof(fid1) ∧ ¬feof(fid2))
    it = it + 1
    t1u1 = u1;
    t1u2 = u2;
    u1 = fread(fid1, N_DOF, 'double');
    u2 = fread(fid2, N_DOF, 'double');
    if (feof(fid1) ∨ feof(fid2)), break, end
    ⟨ Compute Variables 26 ⟩
    end
    fclose(fid1);
    fclose(fid2);
```

This code is used in section 24.

26. Compute Variables.

First we prepare the time derivatives of the solutions and compute the total energy split into the kinetic part, the capillarity, and the potential energy for every timestep (after putting the time into the variable x),

$$E_{kin} = < u_t, \mathrm{M}u_t >, \tag{C.6}$$
$$E_{cap} = < u, \mathrm{K}u >, \tag{C.7}$$
$$E_{pot} = \sum_{\text{elements}} E_{\text{element}}, \tag{C.8}$$

as usual in the finite element method.

⟨ Compute Variables 26 ⟩ ≡

```
⟨ Compute Potential Energy Density 27 ⟩
u1_t = (u1 − t1u1)/Delta_t;
u2_t = (u2 − t1u2)/Delta_t;
x(it) = it∗Delta_t;
Ekin(it) = 0.5∗(dot(u1_t, M∗u1_t) + dot(u2_t, M∗u2_t));
Ecap(it) = 0.5∗(dot(u1, K∗u1) + dot(u2, K∗u2));
Epot(it) = sum(EL_Epot);
```

This code is used in section 25.

27. Potential Energy Density

First we compute the entries of the deformation gradient, exactly as in the "Integrate Potential" program section. Then the value of the potential energy for this deformation gradient is evaluated.

⟨ Compute Potential Energy Density 27 ⟩ ≡

```
for iEL = 1 : N_EL
  for i = 1 : N_NDpEL
    nd(i) = nodes(iEL, i);
  end
  index = elementdof(nd, N_NDpEL, N_DOFpND);
  u1_x = dot(u1(index), Shape_Int_x/(xlen∗ylen));
  u1_y = dot(u1(index), Shape_Int_y/(xlen∗ylen));
  u2_x = dot(u2(index), Shape_Int_x/(xlen∗ylen));
  u2_y = dot(u2(index), Shape_Int_y/(xlen∗ylen));
  EL_Epot(iEL) = eval(Phi);
end
EL_Epot = EL_Epot∗xlen∗ylen∗Potential_Factor;
```

This code is used in section 26.

Index of `code/shape`

List of Refinements in `code/shape`

Bibliography

[1] M. Achenbach. A model of shape memory alloys as a solid/solid phase transition. In K.-H. Hoffmann and J. Sprekels, editors, *Free boundary problems: theory and applications. Vol. II. Proceedings of the International Colloquium held in Irsee, June 11–20, 1987*, pages 796–801, Harlow, 1990. Longman Scientific & Technical.

[2] M. Achenbach and I. Müller. Creep and yield in martensitic transformations. *Ingenieur-Archiv*, 53(2):73–83, 1983.

[3] H. W. Alt, K.-H. Hoffmann, M. Niezgódka, and J. Sprekels. A numerical study of structural phase transitions in shape memory alloys. Technical Report 90, Institut für Mathematik, Universität Augsburg, 1985.

[4] J. M. Ball and R. D. James. Fine phase mixtures as minimizers of energy. *Arch. Rational Mech. Anal.*, 100(1):13–52, 1987.

[5] Pavel Bělík and Mitchell Luskin. A computational model for the indentation and phase transformation of a martensitic thin film. *J. Mech. Phys. Solids*, 50(9):1789–1815, 2002.

[6] Kaushik Bhattacharya. Phase boundary propagation in a heterogeneous body. *R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci.*, 455(1982):757–766, 1999.

[7] Kaushik Bhattacharya. *Martensitic Phase Transformations*. Oxford University Press, Oxford, 2003.

[8] M. I. G. Bloor and M. J. Wilson. Partial differential equations for shape generation in geometric modelling. In *Geometry and topology of submanifolds, III (Leeds, 1990)*, pages 32–48. World Sci. Publishing, River Edge, NJ, 1991.

[9] Dietrich Braess. *Finite elements*. Cambridge University Press, Cambridge, second edition, 2001. Theory, fast solvers, and applications in solid mechanics, Translated from the 1992 German edition by Larry L. Schumaker.

[10] Haïm Brezis. *Analyse fonctionnelle*. Collection Mathématiques Appliquées pour la Maîtrise. [Collection of Applied Mathematics for the Master's Degree]. Masson, Paris, 1983. Théorie et applications. [Theory and applications].

[11] Nikolaus Bubner, Jan Sokołowski, and Jürgen Sprekels. Optimal boundary control problems for shape memory alloys under state constraints for stress and temperature. *Numer. Funct. Anal. Optim.*, 19(5-6):489–498, 1998.

[12] Shu Ping Chen and Roberto Triggiani. Proof of two conjectures by G. Chen and D. L. Russell on structural damping for elastic systems. In *Approximation and optimization (Havana, 1987)*, volume 1354 of *Lecture Notes in Math.*, pages 234–256. Springer, Berlin, 1988.

[13] Zhi Ming Chen and Karl-Heinz Hoffmann. Asymptotic behaviors of Landau-Devonshire-Ginzburg model for structural phase transitions in shape memory alloys. *Adv. Math. Sci. Appl.*, 4(1):209–226, 1994.

[14] Zhiming Chen and K.-H. Hoffmann. On a one-dimensional nonlinear thermoviscoelastic model for structural phase transitions in shape memory alloys. *J. Differential Equations*, 112(2):325–350, 1994.

[15] Claude Chevalley. Invariants of finite groups generated by reflections. *Amer. J. Math.*, 77:778–782, 1955.

[16] Pierluigi Colli and Jürgen Sprekels. Global existence for a three-dimensional model for the thermomechanical evolution of shape memory alloys. *Nonlinear Anal.*, 18(9):873–888, 1992.

[17] Pierluigi Colli and Jürgen Sprekels. Global solution to the full one-dimensional Frémond model for shape memory alloys. *Math. Methods Appl. Sci.*, 18(5):371–385, 1995.

[18] Sergio Conti and Giovanni Zanzotto. Reconstructive phase transformations, maximal Ericksen-Pitteri neighborhoods, dislocations and plasticity in crystals. Technical Report 42/2002, Max-Planck Institut für Mathematik in den Naturwissenschaften, Leipzig, 2002.

[19] C. M. Dafermos. Global smooth solutions to the initial-boundary value problem for the equations of one-dimensional nonlinear thermoviscoelasticity. *SIAM J. Math. Anal.*, 13(3):397–408, 1982.

[20] J. L. Ericksen. On correlating two theories of twinning. *Arch. Ration. Mech. Anal.*, 153(4):261–289, 2000.

[21] Giuseppe Fadda, Lev Truskinovsky, and Giovanni Zanzotto. Unified Landau description of the tetragonal, orthorhombic, and monoclinic phases of zirconia. Submitted.

[22] F. Falk. Model free energy, mechanics, and thermodynamics of shape memory alloys. *Acta Metallurgica*, 28(12):1773–1780, 1980.

[23] I. M. Gelfand and S. V. Fomin. *Calculus of variations*. Revised English edition translated and edited by Richard A. Silverman. Prentice-Hall Inc., Englewood Cliffs, N.J., 1963.

[24] David Gilbarg and Neil S. Trudinger. *Elliptic partial differential equations of second order*. Classics in Mathematics. Springer-Verlag, Berlin, 2001. Reprint of the 1998 edition.

[25] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 2.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2001. `http://www.singular.uni-kl.de`.

[26] K.-H. Hoffmann and A. Żochowski. Existence of solutions of some nonlinear thermoelastic systems with viscosity. *Math. Methods Appl. Sci.*, 15(3):187–204, 1992.

[27] Toshio Honma. Types and mechanical characteristics of shape memory alloys. In *Shape Memory Alloys*, pages 61–115. Gordon and Breach Science Publishers, New York, London, Paris, Montreux, Tokyo, Melbourne, 1984.

[28] Song Jiang. Global large solutions to initial-boundary value problems in one-dimensional nonlinear thermoviscoelasticity. *Quart. Appl. Math.*, 51(4):731–744, 1993.

[29] David Kinderlehrer and Pablo Pedregal. Weak convergence of integrands and the Young measure representation. *SIAM J. Math. Anal.*, 23(1):1–19, 1992.

[30] P. Klouček. The computational modeling of nonequilibrium thermodynamics of the martensitic transformations. *Comput. Mech.*, 22(3):239–254, 1998.

[31] P. Klouček and M. Luskin. The computation of the dynamics of the martensitic transformation. *Contin. Mech. Thermodyn.*, 6(3):209–240, 1994.

[32] P. Klouček and M. Luskin. Computational modeling of the martensitic transformation with surface energy. *Math. Comput. Modelling*, 20(10-11):101–121, 1994. Theory and numerical methods for initial-boundary value problems.

[33] Donald E. Knuth and Silvio Levy. *The cweb System of Structured Documentation*. Addison-Wesley Professional, Boston, 2001.

[34] Robert V. Kohn and Stefan Müller. Surface energy and microstructure in coherent phase transitions. *Comm. Pure Appl. Math.*, 47(4):405–435, 1994.

[35] Young W. Kwon and Hyochoong Bang. *The Finite Element Method Using MATLAB (CRC Mechanical Engineering)*. CRC-Press, Boca Raton, FL, 2000.

[36] Peter D. Lax. *Functional Analysis*. Pure and Applied Mathematics. Wiley Interscience, New York, NY, 2002.

[37] I. Müller and K. Wilmański. A model for phase transition in pseudoelastic bodies. *Il Nuovo Cimento*, 57 B(2):283–318, 1980.

[38] Ingo Müller and Stefan Seelecke. Thermodynamic aspects of shape memory alloys. In G. Airoldi, editor, *Shape memory Alloys—From Microstructure to Macroscopic Properties*. Trans Tech Publications, 1996.

[39] Stefan Müller. Variational models for microstructure and phase transitions. In *Calculus of variations and geometric evolution problems (Cetraro, 1996)*, volume 1713 of *Lecture Notes in Math.*, pages 85–210. Springer, Berlin, 1999.

[40] K. Otsuka and C. M. Wayman. *Shape Memory Materials*. Cambridge Univ. Press, Cambridge, 2002.

[41] Wilfried H. Paus. Existence and uniqueness of local solutions in time for a three-dimensional model on shape memory alloys. Technical Report 301, SFB 256, Rheinische Friedrich-Willhelms-Universität Bonn, 1993.

[42] Irena Pawłow and Antoni Żochowski. Existence and uniqueness of solutions for a three-dimensional thermoelastic system. *Dissertationes Math. (Rozprawy Mat.)*, 406:46, 2002.

[43] Reinhard Racke and Songmu Zheng. Global existence and asymptotic behavior in nonlinear thermoviscoelasticity. *J. Differential Equations*, 134(1):46–67, 1997.

[44] A. C. E. Reid and R. J. Gooding. Pattern formation in a 2d elastic solid. *Physica A*, 239(1), 1997.

[45] T. Roubíček. Dissipative evolution of microstructure in shape memory alloys. In Hans-Joachim Bungartz, Ronald H. W. Hoppe, and Christoph Zenger, editors, *Lectures on applied mathematics (Munich, 1999)*, pages 45–63. Springer, Berlin, 2000.

[46] Hans-Rudolf Schwarz. *Methode der finiten Elemente*, volume 47 of *Leitfäden der Angewandten Mathematik und Mechanik [Guides to Applied Mathematics and Mechanics]*. B. G. Teubner, Stuttgart, third edition, 1991. Eine Einführung unter besonderer Berücksichtigung der Rechenpraxis. [An introduction with special reference to computational practice], Teubner Studienbücher Mathematik. [Teubner Mathematical Textbooks].

[47] N. Simha and L. Truskinovsky. Shear induced transformation toughening in ceramics. *Acta Metallurgica Et Materialia*, 42(11):3827–3836, November 1994.

[48] Ruben D. Spies. A state-space approach to a one-dimensional mathematical model for the dynamics of phase transitions in pseudoelastic materials. *J. Math. Anal. Appl.*, 190(1):58–100, 1995.

[49] Jürgen Sprekels and Song Mu Zheng. Global solutions to the equations of a Ginzburg-Landau theory for structural phase transitions in shape memory alloys. *Phys. D*, 39(1):59–76, 1989.

[50] Bernd Sturmfels. *Algorithms in invariant theory*. Springer-Verlag, Vienna, 1993.

[51] Pieter J. Swart and Philip J. Holmes. Energy minimization and the formation of microstructure in dynamic anti-plane shear. *Arch. Rational Mech. Anal.*, 121(1):37–85, 1992.

[52] L. Truskinovsky and G. Zanzotto. Elastic crystals with a triple point. *J. Mech. Phys. Solids*, 50(2):189–215, 2002.

[53] Stephen J. Watson. Unique global solvability for initial-boundary value problems in one-dimensional nonlinear thermoviscoelasticity. *Arch. Ration. Mech. Anal.*, 153(1):1–37, 2000.

[54] Hermann Weyl. *The classical groups*. Princeton University Press, Princeton, NJ, 1997. Their invariants and representations, Fifteenth printing, Princeton Paperbacks.

[55] Johannes Zimmer. Stored energy functions for phase transitions in crystals. Submitted.

[56] Johannes Zimmer. *Mathematische Modellierung und Analyse von Formgedächtnislegierungen in mehreren Raumdimensionen (Mathematical Modeling and Analysis of Shape Memory Alloys in Several Space Dimensions)*. PhD thesis, Technische Universität München, 2000.