OpenPhase

An open-source phase-field library

User Manual

Version 0.23, August 2015

Core development:
Interdisciplinary Centre for Advanced Materials Simulation (ICAMS)
Ruhr University Bochum, Germany

ICAMS
INTERDISCIPLINARY CENTRE FOR ADVANCED MATERIALS SIMULATION

Authors

The development of OpenPhase began in 2009 at the Interdisciplinary Centre for Advanced Materials Simulation (ICAMS), Ruhr University Bochum at the chair of Prof. Ingo Steinbach. Over the years many authors contributed to parts of the library. In alphabetical order:

Efim Borukhovich, Philipp S. Engels (PE), Svyatoslav Gladkov, Reza Darvishi Kamachali, Dmitry Medvedev, Oleg Shchyglo, Robert Spatschek, Ingo Steinbach, Marvin Tegeler (MT), Fathollah Varnik, Mingming Zeng.

In case of questions and/or requests please contact Oleg Shchyglo.

Resources

More informations and a download link can be found at www.openphase.de.

Documentation history

December 2013 First version of User Manual (PE)

April 2014 Added modules (PE)

September 2014 Fixed plot Eshelby test (PE)

June 2015 Added single grain example (PE)

August 2015 Added AdvectionHR (MT)

Contents

1 Introduction

The manual shall provide a guide line for new users of OpenPhase and phase-field simulation in general, and it provides the basic definitions which are needed to understand the basics of the multi-phase field method, get acquainted with the basic conventions in the module package and provide a reference for users and programmers. Section (2.1) will provide the theoretical basis OpenPhase is grounded on. Section (**??**) will define the most important parameters and conventions of the code. Section (**??**) describes how to construct and compile an executable for an individuals simulation. Section (**??**) describes introductory example simulations.

2 Phase-field models

2.1 Theoretical background

2.1.1 Traveling wave solution for the double obstacle potential

The Free Energy functional using a double obstacle potential is defined as

$$F^{dual} = \int_{\Omega} dx f^{dual} \tag{2.1}$$

$$f^{dual} = \frac{1}{2}\epsilon|\nabla\phi|^2 + \frac{1}{2}\gamma DO(\phi) + h(\phi)\Delta g \tag{2.2}$$

$$DO = \begin{cases} \phi(1-\phi)(\phi) & \text{for } \ 0 \leq \phi \leq 1 \\ \infty & \text{else} \end{cases}$$

$h(\phi)$ is a coupling function between 0 and 1 monotonous in ϕ in the range between 0 and 1 chosen in order to ensure a traveling wave solution (see below).

$$h(\phi) = \frac{1}{\pi}[(4\phi - 2)\sqrt{\phi(1-\phi)} + arcsin(2\phi - 1)] \tag{2.3}$$

$$\frac{\partial}{\partial\phi}h(\phi) = \frac{8}{\pi}\sqrt{\phi(1-\phi)} \tag{2.4}$$

The phase-field equation is derived

$$\begin{aligned} \tau\dot{\phi} &= -\frac{\delta}{\delta\phi}F^{dual} \\ &= (\nabla\frac{\partial}{\partial\nabla\phi} - \frac{\partial}{\partial\phi})f^{dual} \\ &= \epsilon\nabla^2\phi + \gamma(\phi - \frac{1}{2}) + \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g \end{aligned} \tag{2.5}$$

The solution of 2.5 in 1D along the x axis, where the phase $\phi = 1$ is arbitrarily positioned in the left, is

$$\phi(x,t) = \begin{cases} 1 & \text{for} \quad x < v_n t - \frac{\eta}{2} \\ \frac{1}{2} - \frac{1}{2} sin(\frac{\pi}{\eta}(x - v_n t)) & \text{for} \quad v_n t - \frac{\eta}{2} \leq x < v_n t + \frac{\eta}{2} \\ 0 & \text{for} \quad x \geq v_n t + \frac{\eta}{2} \end{cases} \tag{2.6}$$

with the velocity v_n of the wave traveling in positive x direction as defined below. The spacial derivatives normal to the front are:

$$\frac{\partial}{\partial x}\phi = -\frac{\pi}{\eta}\sqrt{\phi(1-\phi)} \tag{2.7}$$

$$\frac{\partial^2}{\partial x^2}\phi = \frac{\pi^2}{\eta^2}(\frac{1}{2} - \phi) \tag{2.8}$$

We prove this solution by inserting (2.8) into the 1D version of (2.5)

$$\begin{aligned} \tau\dot{\phi} &= \epsilon_{DO}\frac{\partial^2}{\partial x^2}\phi + \gamma(\phi - \frac{1}{2}) + \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g & (2.9) \\ &= \epsilon\frac{\pi^2}{\eta^2}(\frac{1}{2} - \phi) + \gamma(\phi - \frac{1}{2}) + \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g & (2.10) \\ &= [\epsilon\frac{\pi^2}{\eta^2} - \gamma](\frac{1}{2} - \phi) + \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g & (2.11) \end{aligned}$$

A steady state solution is found for $\Delta g = 0$, i.e. equilibrium between the phases

$$\begin{aligned} 0 &= [\epsilon\frac{\pi^2}{\eta^2} - \gamma](\frac{1}{2} - \phi) & (2.12) \\ &= [\epsilon\frac{\pi^2}{\eta^2} - \gamma] & (2.13) \\ \eta &= \sqrt{\frac{\epsilon\pi^2}{\gamma}} & (2.14) \end{aligned}$$

This shows that the ratio between ϵ and γ determines the interface width. The velocity v_n of the traveling wave solution (2.6) can be related to the model parameters τ,η and m:

$$\dot{\phi} = \dot{x}\frac{\partial}{\partial x}\phi = v_n\frac{\partial}{\partial x}\phi = -v_n\frac{\pi}{\eta}\sqrt{\phi(1-\phi)} \tag{2.15}$$

Here we have transformed $\dot{\phi}$ into a coordinate system traveling with velocity v_n and used the first spacial derivative of the traveling wave solution (2.7). In 1D the interface contribution proportional to ϵ and γ cancel, as shown above, and the phase field equation

(2.5) becomes

$$\tau\dot{\phi} = \epsilon\frac{\partial^2}{\partial x^2}\phi + \gamma(\phi - \frac{1}{2}) + \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g = \frac{8}{\pi}\sqrt{\phi(1-\phi)}\Delta g \tag{2.16}$$

By comparison of (2.15) and (2.16) we find the velocity

$$v_n = -\frac{8\eta}{\pi^2\tau}\Delta g \tag{2.17}$$

The sign means, that if the phase $\phi = 1$, which is placed on the left, is thermodynamically favorable, i.e. $\Delta g < 0$, the velocity is positif and the interface is traveling to the right, which means growth of phase $\phi = 1$.
It shall be noted that the coupling function $h(\phi)$ is constructed such that $\frac{\partial}{\partial\phi}h(\phi) = \frac{\partial}{\partial x}\phi$ and therefore the profile of the traveling wave is independent of ϕ and x. There is, however, no generalization of such a solution in junctions between more than 2 phase-fields (see below).
The interface energy $\sigma[\frac{J}{cm^2}]$ is

$$\begin{aligned}\sigma &= \int_{-\infty}^{\infty} dx[\frac{\epsilon}{2}(\nabla\phi)^2 + \frac{\gamma}{2}\phi(1-\phi)] \\ &= \int_{-\infty}^{\infty} dx[\frac{\epsilon}{2}\frac{\pi^2}{\eta^2} + \frac{\gamma}{2}]\phi(1-\phi) \\ &= \int_{-\infty}^{\infty} dx\gamma\phi(1-\phi)\end{aligned} \tag{2.18}$$

In the last equation the result (2.12) was used. To solve the integral we substitute dx by $d\phi\frac{dx}{d\phi}$

$$\begin{aligned}\sigma &= \int_0^1 d\phi\frac{dx}{d\phi}\gamma\phi(1-\phi) \\ &= \gamma\frac{\eta}{\pi}\int_0^1 d\phi\sqrt{\phi(1-\phi)} \\ &= \frac{\gamma\eta}{8} = \frac{\epsilon\pi^2}{8\eta}\end{aligned} \tag{2.19}$$

Finally we fix the time scale comparing the traveling wave equation (2.17) to the Gibbs Tomson equation with the interface mobility μ

$$\begin{aligned}v_n &= \mu\Delta G = \frac{m\eta}{\pi\tau} = \frac{8\Delta G\eta}{\pi^2\tau} \\ \mu &= \frac{8\eta}{\pi^2\tau}\end{aligned} \tag{2.20}$$

The relations between the model parameters τ, ϵ and γ and the physical parameters μ,

σ and η can be summarized:

$$\epsilon = \frac{8\sigma\eta}{\pi^2}, \qquad \gamma = 8\frac{\sigma}{\eta}, \qquad \tau = \frac{8\eta}{\pi^2\mu} \tag{2.21}$$

In the physical units the free energy density and phase field equation read:

$$f = \frac{4\sigma}{\eta}[(\frac{\eta}{\pi}\nabla\phi)^2 + \phi(1-\phi)] - h(\phi)\Delta g \tag{2.22}$$

$$\dot{\phi} = \mu \left[\sigma(\nabla^2\phi + \frac{\pi^2}{\eta^2}(\phi - \frac{1}{2})) + \frac{\pi\sqrt{\phi(1-\phi)}}{\eta}\Delta g \right] \tag{2.23}$$

2.1.2 The multi-phase-field model

We start out from a general free energy description separating different physical phenomena, interfacial f^{intf}, chemical f^{chem}. Later we will compare the multi-phase model to the 2 phase model from the last sections in the case that only 2 phases are present.

$$F = \int_\Omega f^{intf} + f^{chem} \tag{2.24}$$

other contributions like elastic, magnetic and electric energy will be added later.

$$f^{intf} = \sum_{\alpha,\beta=1..N,\alpha>\beta} \frac{4\sigma_{\alpha\beta}}{\eta} \left\{ -\frac{\eta^2}{\pi^2}\nabla\phi_\alpha \cdot \nabla\phi_\beta + \phi_\alpha\phi_\beta \right\} \tag{2.25}$$

where we have set the interface width η equal for all pairs of phases.

$$f^{chem} = \sum_{\alpha=1..N} \phi_\alpha f_\alpha(c_\alpha^i) + \tilde{\mu}^i(c^i - \sum_{\alpha=1..N} \phi_\alpha c_\alpha^i) \tag{2.26}$$

We use the sum convention over double indices of the components i. $N = N(x)$ is the local number of phases and we have the sum constraint

$$\sum_{\alpha=1..N} \phi_\alpha = 1 \tag{2.27}$$

$\sigma_{\alpha\beta}$ is the energy of the interface between phase – or grain – α and β. It may be anisotropic with respect to the relative orientation between the phases. $\eta_{\alpha\beta}$ is the interface width that will be treated equal for all interfaces in the following. The chemical free energy is built from the bulk free energies of the individual phases $f_\alpha(\vec{c}_\alpha)$ which depend on the phase concentrations c_α^i. $\tilde{\mu}^i$ is the generalized chemical potential or diffusion potential of component i introduced as a Lagrange multiplier to conserve the mass balance between the phases $c^i = \sum_{\alpha=1..N} \phi_\alpha c_\alpha^i$.
For $N = 2$, $\alpha = 1$, $\beta = 2$ we check for consistency

$$\begin{aligned} f^{intf-dual} &= \frac{4\sigma_{12}}{\eta}\left\{ -\frac{\eta^2}{\pi^2}\nabla\phi_\alpha\cdot\nabla\phi_\beta + \phi_\alpha\phi_\beta \right\} \\ &= \frac{4\sigma_{12}}{\eta}\left\{ \frac{\eta^2}{\pi^2}(\nabla\phi_\alpha)^2 + \phi_\alpha(1-\phi_\alpha) \right\} \end{aligned} \tag{2.28}$$

which is identical to (2.22) with $\phi_\beta = 1-\phi_\alpha$ in the dual case. The chemical free energy in the multi-phase case uses the identity $h(\phi)=\phi$ for the coupling function

$$\begin{aligned} f^{chem-dual} &= \phi_\alpha f_\alpha(c_\alpha^i) + \phi_\beta f_\beta(c_\beta^i) + \tilde{\mu}^i(c^i - \phi_\alpha c_\alpha^i - \phi_\beta c_\beta^i) \\ &= \phi_\alpha(f_\alpha(c_\alpha^i) - f_\beta(c_\beta^i) - \tilde{\mu}^i(c_\alpha^i - c_\beta^i) + f_\beta(c_\beta^i)) + \tilde{\mu}^i(c^i - c_\beta^i) \\ &= \phi_\alpha \Delta g_{\alpha\beta}^{dual} + f_0(c^i) \end{aligned} \tag{2.29}$$

$$\Delta g_{\alpha\beta}^{dual} = f_\alpha(c_\alpha^i) - f_\beta(c_\beta^i) - \tilde{\mu}^i(c_\alpha^i - c_\beta^i) \tag{2.30}$$

$\Delta g_{\alpha\beta}^{dual}$ is the equivalent to Δg in (2.22) under the assumption that the phase compositions c_α^i obey a parallel tangent construction with the same chemical potential $\tilde{\mu}^i$. $f_0(c^i)$ is an offset independent of ϕ.
The multi-phase-field equations are derived

$$\dot{\phi}_\alpha = -\sum_{\beta=1..N} \frac{\pi^2}{4\eta N}\mu_{\alpha\beta}\left(\frac{\delta F}{\delta\phi_\alpha} - \frac{\delta F}{\delta\phi_\beta}\right) \tag{2.31}$$

$\mu_{\alpha\beta}$ is defined individually for each pair of phases. Inserting the free energy (2.24) to (2.31) we calculate explicitly

$$\dot{\phi}_\alpha = \sum_{\beta=1..N} \frac{\mu_{\alpha\beta}}{N}\left[\left\{ \sigma_{\alpha\beta}(I_\alpha - I_\beta) + \sum_{\gamma=1..N,\gamma\neq\alpha,\gamma\neq\beta}(\sigma_{\beta\gamma}-\sigma_{\alpha\gamma})I_\gamma \right\} + \frac{\pi^2}{4\eta}\Delta g_{\alpha\beta}\right] \tag{2.32}$$

$$I_\alpha = \nabla^2\phi_\alpha + \frac{\pi^2}{\eta^2}\phi_\alpha \tag{2.33}$$

$$\Delta g_{\alpha\beta} = f_\alpha(c_\alpha^i) - f_\beta(c_\beta^i) - \tilde{\mu}^i(c_\alpha^i - c_\beta^i) \tag{2.34}$$

Again we compare the model for consistency with the dual phase model. For $N=2$ we have:

$$
\begin{aligned}
\dot{\phi}_1 &= -\frac{\pi^2}{8\eta}\mu_{12}(\frac{\delta F}{\delta\phi_1} - \frac{\delta F}{\delta\phi_2}) \\
&= \frac{\mu_{12}}{2}\{[\sigma_{12}(I_1 - I_2) + \frac{\pi^2}{4\eta}\Delta g_{12}\} \\
&= \mu_{12}\{[\sigma_{12}(\nabla^2\phi_1 + \frac{\pi^2}{\eta^2}\frac{\phi_1 - \phi_2}{2}) + \frac{\pi^2}{8\eta}\Delta g_{12}\} \\
&= \mu_{12}\{[\sigma_{12}(\nabla^2\phi_1 + \frac{\pi^2}{\eta^2}(\phi_1 - \frac{1}{2}) + \frac{\pi^2}{8\eta}\Delta g_{12}\} \qquad (2.35) \\
\dot{\phi}_2 &= -\dot{\phi}_1 \qquad (2.36)
\end{aligned}
$$

The interface contribution is identical to (2.22), the non equilibrium part, however, violates the traveling wave solution as the identity for the coupling function is used for thermodynamic consistency. We may check that the integral of the rate change of ϕ is the same in both cases (taking $\mu_{12} = \mu$ and $\Delta g_{12} = \Delta g$

$$
\begin{aligned}
\int_0^1 d\phi \frac{\pi\sqrt{\phi(1-\phi)}}{\eta}\Delta g &= \frac{\pi\Delta g}{\eta}\int_0^1 d\phi\sqrt{\phi(1-\phi)} = \frac{\pi^2\Delta g}{8\eta} \qquad (2.37) \\
\int_0^1 dx \frac{\pi^2}{8\eta}\Delta g_{12} &= \frac{\pi^2\Delta g}{8\eta} \qquad (2.38)
\end{aligned}
$$

This shows, that for practical use in a phase-field simulation we may approximate in dual interfaces

$$\Delta g \approx \frac{\pi}{8}\sqrt{\phi(1-\phi)}\Delta g \qquad (2.39)$$

Resigning from full thermodynamic consistency we will use in the following the approximation

$$
\begin{aligned}
\dot{\phi}_\alpha = \sum_{\beta=1..N}\frac{\mu_{\alpha\beta}}{N}\Big\{\Big[\,\sigma_{\alpha\beta}(I_\alpha - I_\beta) + \sum_{\gamma=1..N,\gamma\neq\alpha,\gamma\neq\beta}(\sigma_{\beta\gamma} - \sigma_{\alpha\gamma})I_\gamma\,\Big] \\
+\frac{2\pi}{\eta}\sqrt{\phi(1-\phi)}\Delta g_{\alpha\beta}\Big\} \qquad (2.40)
\end{aligned}
$$

3 Modules

OpenPhase provides a set of additional solvers for all types of partial differential equations.

3.1 Settings

Module in brief...

What it does Reads and stores the most general calculation informations, mostly given in the input file.

Header File Settings.h

Requires Nothing

Input file ProjectInput.opi[1]

Examples /examples/HeatEquationSolver, ...

Input parameters and input file The settings module reads in the calculation parameters from an input file, typically ProjectInput/ProjectInput.opi. The following list provides an explanation of the required input parameters. Mandatory entries are underlined.

$Nx, $Ny, $Nz Sets the number of grid points in the x, y and z direction, respectively. The minimum dimension size is one, the maximum is typically limited by the amount of memory. Can be accessed via Settings.Nx.

$nSteps Number of time steps. Accessible via Settings.nSteps. The use of this parameter can be convenient in a typical simulation setup such as

```
1 for(int tStep = 0; tStep < Settings.nSteps; tStep++)
2 {
3         // increment operations
4 }
```

$FTime, $FDisk Sets parameters useful for the amount of data output to disk and screen. They can be accessed via the variables Settings.tFileWrite and Settings.tScreenWrite. Usefully, the

```
1 if (!(tStep%Settings.tFileWrite))
2 {
3     PhaseField.WriteVTK(tStep);
4 }
```

[1] All following input file names in the 'Module in brief' boxes can be altered by the user.

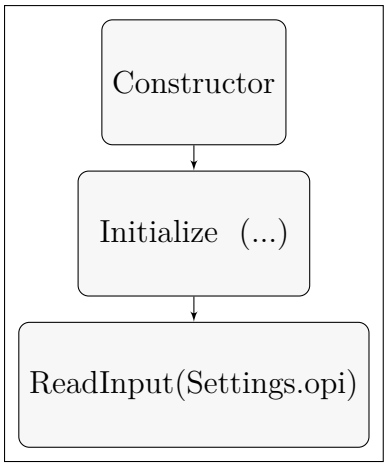

Figure 3.1: Call graph for module Settings

$dx Sets the distance between two calculation points of the regular grid. Note, that this value is the same for all spacial directions. Can be accessed via Settings.dx.

$IWidth Definition of the interface with in grid points which is stored as Settings.eta. Note that the small strain elasto-plastic framework of OpenPhase also works in the sharp interface limit, i.e. Settings.eta $= 0$.

$dt Sets the initial time step. Can be accessed via Settings.dt. Note that some modules might use an internal time stepping scheme.

$nOMP Sets the maximum number of used threads for parallelization with OpenMPI [1]. The domain is then splitted along the x-axis, which should be considered when non-cubic computation grids are created. In addition, some initializations are specifically optimized for this domain decomposition.

$nPhses Sets the number of thermodynamic phases (but not the number of grains/phase-fields) and can be accessed via Settings.nPhases. Typically, material parameters have to be given for Settings.nPhases in further input files such as ElasticProperties.opi.

3.2 BoundaryConditions

Module in brief...

What it does Sets boundary conditions for scalar and vectorial calculation variables.

Requires Settings

Header file BoundaryConditions.h

Input file BoundaryConditions.opi

Examples /examples/HeatEquationSolver

Boundary Conditions can be set individually for every side in the boundary condition input file (typically BoundaryConditions.opi) which is read by ReadInput() . The keywords $BC0X , $BCNX , $BC0Y , $BCNY , $BC0Z and $BCNZ define the six individual sides of the calculation domain, see Listing 3.1

```
1 $BC0X   X axis beginning boundary condition  : Periodic
2 $BCNX   X axis far end boundary condition    : Periodic
3
4 $BC0Y   Y axis beginning boundary condition  : NoFlux
5 $BCNY   Y axis far end boundary condition    : NoFlux
6
7 $BC0Z   Z axis beginning boundary condition  : Periodic
8 $BCNZ   Z axis far end boundary condition    : Periodic
```

Listing 3.1: Example of BoundaryConditions.opi input file for BoundaryConditions

The available types for a scalar or vectorial variable ξ

Periodic $\xi_1 = \xi_N$

Free ξ_1, ξ_N

NoFlux $(\nabla\xi_1)\mathbf{n} = (\nabla\xi_N)\mathbf{n} = 0$

Fixed ξ_1, ξ_N

Internally, a boundary condition object containing the boundary definitions is passed to the indivual modules, such as Temperature or PhaseField.

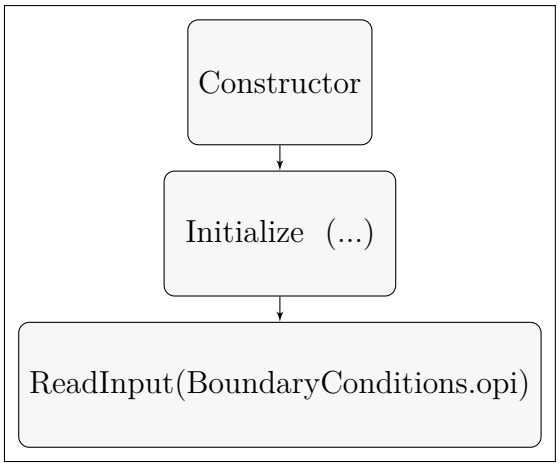

Figure 3.2: Call graph for module BoundaryConditions

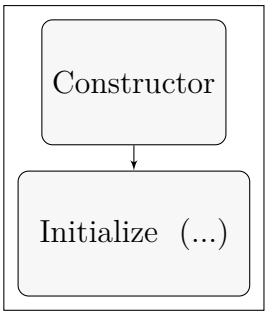

Figure 3.3: Call graph for module PhaseField

3.3 PhaseField

Module in brief...

What it does Stores and manages phase-fields.

Requires Settings, BoundaryConditions

Header File PhaseField.h

Input file Several important parameters are given in the main input file ProjectInput.opi

Examples

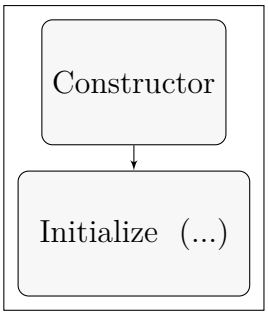

Figure 3.4: Call graph for module InterfaceField

3.4 InterfaceField

Module in brief...

What it does Calculates the pairwise phase-field interactions

Requires Settings, BoundaryConditions

Input file None

Examples benchmarks/SingleGrain

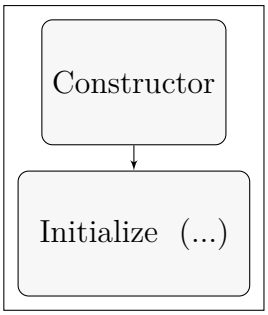

Figure 3.5: Call graph for module InterfaceEnergy

3.5 InterfaceEnergy

Module in brief...

What it does Calculates the interfacial energy

Requires Settings, BoundaryConditions

Header file InterfaceEnergy.h

Input file None

Examples benchmarks/SingleGrain

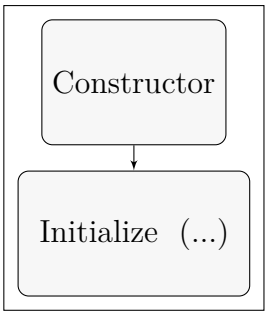

Figure 3.6: Call graph for module InterfaceMobility

3.6 InterfaceMobility

Module in brief...

What it does Calculates the interfacial mobility

Requires Settings, BoundaryConditions

Header file InterfaceMobility.h

Input file None

Examples benchmarks/SingleGrain

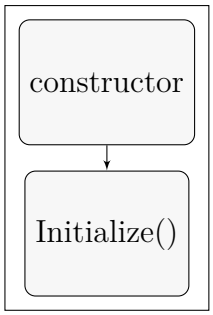

Figure 3.7: In and output for module Velocities

3.7 Velocities

Module in brief...

What it does Stores velocity field of phases and calculates average.

Requires Settings, PhaseField, BoundaryConditions

Header file Velocities.h

Input file None

Examples /examples/LDPlasticity, ...

Background and usage The main purpose of the Velocities module is the storage of a velocity field, here called $\mathrm{V}(i, j, k)$
The temperature field can be read and write to disk using Write(int tStep) and Read(tStep). VTK output can be generated via WriteAverageVTK(int tStep) for the phase averaged quantity.

3.8 Orientations

Module in brief...

What it does Reads, outputs and stores local and global, absolute and incremental rotation informations. Provides methods to calculate Euler angles.

Requires Settings

Header File Orientations.h

Input file Orientations.opi

Examples /benchmarks/EshelbyTestWithPlasticity, ...

Purpose and structure The Orientations module is intended to store local and global orientations, provide output routines and manages the calculation from rotation matrices to Euler angles or quaternions. This class is required

- if rotated grains should be considered in a calculation using elasticity. In this case, an orientations object has to be passed to functions ElasticProperties .SetGrainsProperties() , ElasticityModule. SetEffectiveEigenstrains () and ElasticityModule.SetEffectiveElasticConsta .
- if plasticity should be used.

This class is **not** required

- if Eigenstrain and stiffness tensors are given in a rotated representation in the input file.
- if no elasticity is used. Still, grain orientations can be set (see below) which allow anisotropic phase-field and diffusion calculation. FIXED CONVENTION USED!

The local orientation of a material point is stored as a 3x3 rotation matrix, that rotates lattice properties from the fixed origin x-y-z coordinate system into the local one[2]. Three different contributions are considered

$$\mathbf{R}(\mathbf{x}) = \mathbf{R}^{glob}\mathbf{R}^{grain}\mathbf{R}^{elastic}(\mathbf{x})$$

Here, $\mathbf{R}^{glob}$ a global rotation matrix that is applied to each material point with the same magnitude. It can be given in the input-file (see below) or changed throughout the calculation. Furthermore, $\mathbf{R}^{grain}$ is the grain rotation with respect to the reference frame ro-

[2]The rotation from of a point in a fixed coordinate system is called 'active'

tated by $\mathbf{R}^{glob}$. In opposite to $\mathbf{R}^{glob}$ and $\mathbf{R}^{elastic}$, $\mathbf{R}^{grain}$ is stored as a phase-field parameter and can be accessed via PhaseField.FieldsStatistics[alpha].Orientation (returns an object of type Angles) and changed via PhaseField.FieldsStatistics[alpha].Orientation[GrainIndex].set(Q1, Q2, Q3, 'Con1', 'Con2', 'Con3'). The latter three parameters (see description below). The calculation and integration of $\mathbf{R}^{elastic}$, is described in SpectralElasticSolver.

$$R_{\hat{n}}(\alpha) = \begin{pmatrix} n_1^2\,(1-\cos\alpha)+\cos\alpha & n_1 n_2\,(1-\cos\alpha)-n_3\sin\alpha & n_1 n_3\,(1-\cos\alpha)+n_2\sin\alpha \\ n_2 n_1\,(1-\cos\alpha)+n_3\sin\alpha & n_2^2\,(1-\cos\alpha)+\cos\alpha & n_2 n_3\,(1-\cos\alpha)-n_1\sin\alpha \\ n_3 n_1\,(1-\cos\alpha)-n_2\sin\alpha & n_3 n_2\,(1-\cos\alpha)+n_1\sin\alpha & n_3^2\,(1-\cos\alpha)+\cos\alpha \end{pmatrix}$$

Input parameters and input file The orientations module reads in the calculation parameters from an input file, typically ProjectInput/Orientations.opi. The following list provides an explanation of the required input parameters. Mandatory entries are underlined. **Note: All input angles have to be given in radian.**

$rotationflag If option 'Yes' is chosen, the rotations stemming from the deformation field - calculated in SpectralElasticSolver - are integrated. Otherwise, the storage Orientations.Rotations will store

$$\mathbf{R}^{t+1} = \mathbf{R}^{t} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

throughout the calculation.

$globalconvention The conversion from Euler angles to rotation matrices requires the definition of the . The following purely intrinsic rotations are allowed: XYX, XYZ, XZX, XZY, YXY, YXZ, YZX, YZY, ZXY, ZXZ, ZYX, ZYZ. Note: Two succeeding rotations around the same axis can inherently not lead to all possible orientation states. In addition, the following purely rotations are allowed: xyx, xyz, xzx, xzy, yxy, yxz, yzx, yzy, zxy, zxz, zyx, zyz.

$globalQ1, $globalQ2, $globalQ3 This set of angles will set the global rotation matrix $\mathbf{R}^{glob}$.

$$X_1Z_2X_3 = \begin{bmatrix} c_2 & -c_3s_2 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$$

$$X_1Z_2Y_3 = \begin{bmatrix} c_2c_3 & -s_2 & c_2s_3 \\ s_1s_3 + c_1c_3s_2 & c_1c_2 & c_1s_2s_3 - c_3s_1 \\ c_3s_1s_2 - c_1s_3 & c_2s_1 & c_1c_3 + s_1s_2s_3 \end{bmatrix}$$

$$X_1Y_2X_3 = \begin{bmatrix} c_2 & s_2s_3 & c_3s_2 \\ s_1s_2 & c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 \\ -c_1s_2 & c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$$

$$X_1Y_2Z_3 = \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 & -c_2s_1 \\ s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$$

$$Y_1X_2Y_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & s_1s_2 & c_1s_3 + c_2c_3s_1 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_3s_1 - c_1c_2s_3 & c_1s_2 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$$

$$Y_1X_2Z_3 = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 & c_1c_2 \end{bmatrix}$$

$$Y_1Z_2Y_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_3s_1 + c_1c_2s_3 \\ c_3s_2 & c_2 & s_2s_3 \\ -c_1s_3 - c_2c_3s_1 & s_1s_2 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$$

$$Y_1Z_2X_3 = \begin{bmatrix} c_1c_2 & s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 \\ s_2 & c_2c_3 & -c_2s_3 \\ -c_2s_1 & c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 \end{bmatrix}$$

$$Z_1Y_2Z_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 & c_1s_2 \\ c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 & s_1s_2 \\ -c_3s_2 & s_2s_3 & c_2 \end{bmatrix}$$

$$Z_1Y_2X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 \\ c_2s_1 & c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 \\ -s_2 & c_2s_3 & c_2c_3 \end{bmatrix}$$

$$Z_1X_2Z_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix}$$

$$Z_1X_2Y_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix}$$

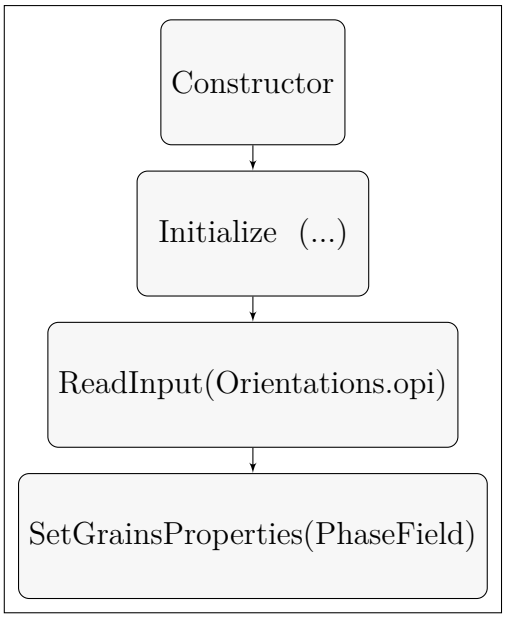

Figure 3.8: Call graph for module Orientations

3.9 ElasticProperties

Module in brief...

What it does Defines and manages storages for mechanical calculation

Requires Settings

Header file Mechanics/Storages/ElasticProperties.h

Input file ElasticityInput.opi

Examples /benchmarks/EshelbyTest /examples/LD-J2Plasticity

Input parameters and input file Several parameters can be defined in the input file for the ElasticProperties module and read via ReadInput() . An example is given in Listing 3.2.

0 - Pressure relaxation mode This boundary condition is only supported in SpectralElasticSolver and SpectralElastoPlasticSolver.

1 - Applied strain mode This boundary condition is supported by all solvers.

2 - Applied stress mode This boundary condition is supported by all solvers.

3 - Mixed stress strain mode This boundary condition is supported by SpectralElasticSolverBS and SpectralElasticSolverAL.

```
$MechState  Mechanical State: 1

$globorientation_1 Global orientation 1: 0.0
$globorientation_2 Global orientation 2: 0.0
$globorientation_3 Global orientation 3: 0.0

$Comp_0      Component name                : NN

$CREF_NN_0                Reference concentration          : 0

$Phase 0
$C11 280e9   0
$C22 280e9   0
$C33 280e9   0
$C12 120e9   0
$C13 120e9   0
$C23 120e9   0
$C44 80e9    0
$C55 80e9    0
$C66 80e9    0

$E1 0    0
$E2 0    0
$E3 0    0
$E4 0    0
$E5 0    0
$E6 0    0
```

Listing 3.2: Example of ElasticProperties.opi input file for ElasticProperties

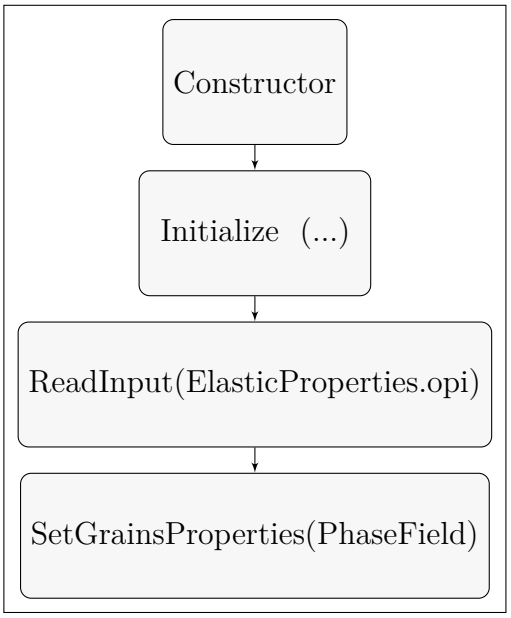

Figure 3.9: Call graph for module ElasticProperties

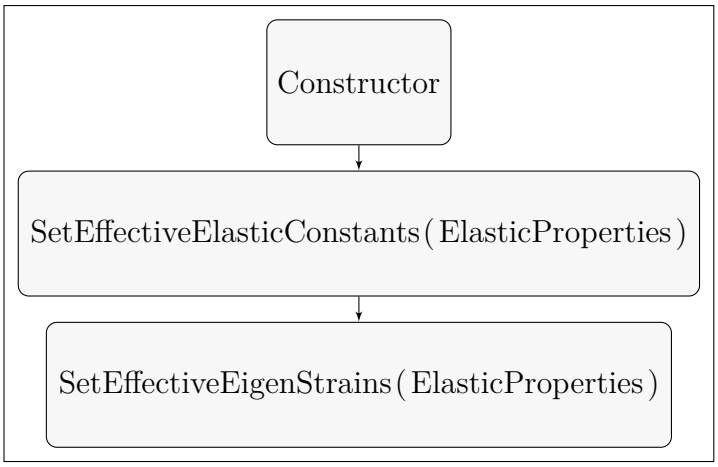

Figure 3.10: Call graph for module ElasticityReuss

3.10 ElasticityReuss

Module in brief...

What it does Calculates homogenized elastic properties in the interface; calculates elastic driving force.

Requires Settings, ElasticProperties

Header file Mechanics/ElasticityModels/ElasticityReuss.h

Input file None

Examples /examples/MultiComp-Elastic

The EffectiveElasticConstants are calculated by calling the method SetEffectiveElasticConstants(EP) as

$$\mathbf{C} = \left(\sum_{\alpha} \mathbf{C}_{\alpha}^{-1} \phi_{\alpha} \right)^{-1} \tag{3.1}$$

The EffectiveEigenStrains are calculated by calling the method SetEffectiveEigenStrains(EP)

$$\varepsilon^{*} = \sum_{\alpha} \phi_{\alpha} \boldsymbol{\varepsilon}_{\alpha}^{*} \tag{3.2}$$

The driving force $\Delta G_{\alpha\beta}^{el}$ becomes

$$\Delta G_{\alpha\beta}^{el} = \frac{1}{2} \boldsymbol{\varepsilon}^{el} \mathbf{C} \left(\mathbf{C}_{\alpha}^{-1} - \mathbf{C}_{\beta}^{-1} \right) \mathbf{C} \boldsymbol{\varepsilon}^{el} + \varepsilon^{el} \mathbf{C} \left(\boldsymbol{\varepsilon}_{\alpha}^{*} - \boldsymbol{\varepsilon}_{\beta}^{*} \right) \tag{3.3}$$

$$= \frac{1}{2} \boldsymbol{\sigma} \left(\mathbf{C}_{\alpha} - \mathbf{C}_{\beta} \right) \boldsymbol{\sigma} - \boldsymbol{\sigma} \left(\boldsymbol{\varepsilon}_{\alpha}^{*} - \boldsymbol{\varepsilon}_{\beta}^{*} \right) \tag{3.4}$$

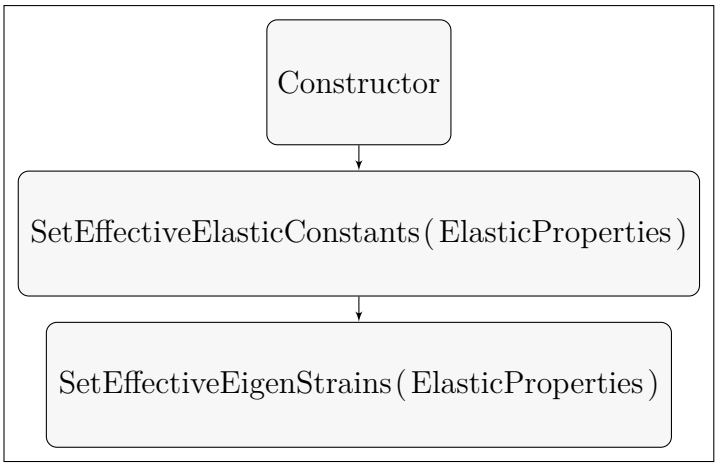

Figure 3.11: Call graph for module **??**

3.11 ElasticityKhachaturyan

Module in brief...

What it does Calculates homogenized elastic properties in the interface; calculates elastic driving force.

Requires Settings, ElasticProperties

Header file Mechanics/ElasticityModels/ElasticityKhachaturyan.h

Input file None

Examples /examples/MultiComp-Elastic

The EffectiveElasticConstants are calculated by calling the method SetEffectiveElasticConstants(EP) as

$$\mathbf{C} = \sum_{\alpha} \mathbf{C}_{\alpha} \phi_{\alpha} \tag{3.5}$$

The EffectiveEigenStrains are calculated by calling the method SetEffectiveEigenStrains(EP)

$$\varepsilon^{*} = \sum_{\alpha} \phi_{\alpha} \boldsymbol{\varepsilon}^{*}_{\alpha} \tag{3.6}$$

The driving force $\Delta G^{el}_{\alpha\beta}$ becomes

$$\Delta G^{el}_{\alpha\beta} = \frac{1}{2} \boldsymbol{\varepsilon}^{el} \left(\mathbf{C}_{\beta} - \mathbf{C}_{\alpha} \right) \boldsymbol{\varepsilon}^{el} + \varepsilon^{el} \mathbf{C} \left(\boldsymbol{\varepsilon}^{*}_{\beta} - \boldsymbol{\varepsilon}^{*}_{\alpha} \right) \tag{3.7}$$

$$= \frac{1}{2} \boldsymbol{\varepsilon}^{el} \left(\mathbf{C}_{\beta} - \mathbf{C}_{\alpha} \right) \boldsymbol{\varepsilon}^{el} - \boldsymbol{\sigma} \left(\boldsymbol{\varepsilon}^{*}_{\beta} - \boldsymbol{\varepsilon}^{*}_{\alpha} \right) \tag{3.8}$$

3.12 SpectralElasticSolver

Module in brief...

What it does Calculates the mechanical equilibrium $\nabla \cdot \sigma = 0$.

Requires Settings, PhaseField, BoundaryConditions, ElasticProperties

Input file ElasticProperties.opi

Examples /examples/EshelbyTest

Background This module implements the scheme published by Hu and Chen [2]. Starting from splitting the average strain

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \bar{\boldsymbol{\varepsilon}} + \tilde{\boldsymbol{\varepsilon}}(\mathbf{x}) \tag{3.9}$$

with the classical engineering strain

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \frac{1}{2}\left(\nabla \mathbf{u}(\mathbf{x}) + (\nabla \mathbf{u}(\mathbf{x}))^{\mathrm{T}}\right). \tag{3.10}$$

Note, that this linearized strain measure with respect to the undeformed reference configuration. However for applications where the elastic strains are small ($\varepsilon < 0.1$) this assumption is valid.

$$\mathbf{C}^0 \frac{\partial^2 \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}^2} = (\mathbf{C}^0 \boldsymbol{\varepsilon}^0 - \tilde{\mathbf{C}}(\mathbf{x})\bar{\boldsymbol{\varepsilon}}) \frac{\partial X}{\partial \mathbf{x}} + \tilde{\mathbf{C}}(\mathbf{x}) \boldsymbol{\varepsilon}^0 \frac{\partial^2 X}{\partial \mathbf{x}^2} - \tilde{\mathbf{C}}(\mathbf{x}) \frac{\partial}{\partial \mathbf{x}} \left(X \frac{\partial \mathbf{u}^{n-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \tag{3.11}$$

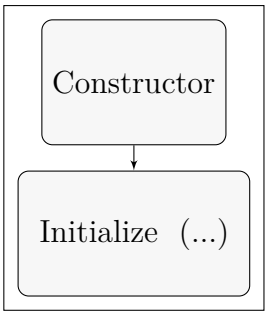

Figure 3.12: Call graph for module SpectralElastoPlasticSolver

3.13 SpectralElastoPlasticSolver

Module in brief...

What it does Calculates the mechanical equilibrium $\nabla \cdot \sigma = 0$.

Requires Settings, PhaseField, BoundaryConditions, ElasticProperties, ??, ??

Input file ElasticProperties.opi

Examples /examples/EshelbyTest

Background This module extends SpectralElasticSolver for consideration of plastic deformations.

3.14 SpectralElasticSolverBS

Module in brief...

What it does Calculates the mechanical equilibrium $\nabla \cdot \boldsymbol{\sigma} = 0$.

Requires Settings, PhaseField, BoundaryConditions, ElasticProperties

Input file ElasticProperties.opi

Examples /examples/EshelbyTest

Background The SpectralElasticSolverBS module represents an alternative to the SpectralElasticSolver and the SpectralElasticSolverAL module. It implements the scheme as described by Moulinec and Suquet [3] using an enhanced mathematical formulation, strong phase contrast can be treated with improved convergence behavior, see [4] for a comparison. However, the basic scheme uses, like the AL solver, a strain related and hence can not be used in combination with the Large Deformations module.

Starting from the mechanical equilibrium,

$$\nabla \cdot \boldsymbol{\sigma}(\mathbf{x}) = 0, \tag{3.12}$$

where $\boldsymbol{\sigma} = \mathbf{C}(\mathbf{x}) : \boldsymbol{\varepsilon}$ is the symmetric Cauchy stress tensor using Hooke's law assuming a linear stress-strain relationship. The principle idea is to split the total strain $\boldsymbol{\varepsilon}(\mathbf{x})$ into an homogeneous and an heterogeneous part

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \mathbf{E} + \tilde{\boldsymbol{\varepsilon}}(\mathbf{x}) \tag{3.13}$$

where the latter one is assumed to be periodic. It follows, that the stress

$$\boldsymbol{\sigma}(\mathbf{x}) = \mathbf{C}(\mathbf{x}) : (\mathbf{E} + \tilde{\boldsymbol{\varepsilon}}(\mathbf{x})) \tag{3.14}$$

If one introduces a reference stiffness[3],

$$\mathbf{C}^0 = \frac{1}{2}\left(\max_\phi(\tilde{\mathbf{C}}) - \min_\phi(\tilde{\mathbf{C}}_\phi)\right), \tag{3.15}$$

with $\tilde{\mathbf{C}}$ being the rotated initial stiffness tensors,
By transferring (3.13) to Fourier space, the convolution becomes a simple multiplication

[3]

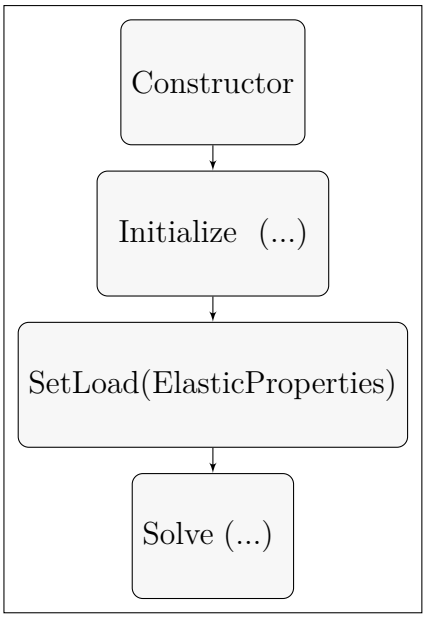

Figure 3.13: Call graph for module SpectralElasticSolverBS

$$\boldsymbol{\varepsilon}^{i+1}(\mathbf{x}) = \Gamma * \left(\mathbf{C}(\mathbf{x}) - \mathbf{C}^0\right) \boldsymbol{\varepsilon}^i + \mathbf{E} \tag{3.16}$$

The Green operator for anisotropic materials reads in Fourier space

$$\hat{\Gamma}^0_{ijkl} = -k_j k_l \left(C^0_{ijkl} k_l k_j\right)^{-1} \tag{3.17}$$

Note that the choice of the reference stiffness $\mathbf{C}^0$ has a severe impact on the convergence performance.

Usage The solver is called via Solve(ElasticProperties , **double** equilibrium, **int** MAXIterations) , where equilibrium is the convergence criterium following (3.20) and MAXIterations the maximum allowed number of iterations. Before the solver is called, the load is internally finalized with SetLoad(ElasticProperties) . Check wether that SetGrainsProperties (...) , SetEffectiveElasticConstants () and SetEffectiveEigenstrains(...) of ElasticProperties and ElasticityModel have been called.

3.15 SpectralElasticSolverAL

Module in brief...

What it does Calculates the mechanical equilibrium $\nabla \cdot \sigma = 0$.

Requires Settings, PhaseField, BoundaryConditions, ElasticProperties

Input file ElasticProperties.opi

Examples /examples/EshelbyTest

Background The SpectralElasticSolverAL module represents an alternative to the SpectralElasticSolver and the SpectralElasticSolverBS. It implements the scheme as described by [5] using an enhanced mathematical formulation (augmented lagrangian or AL), strong phase contrast can be treated with improved convergence behavior, see [4] for a comparison. However, the AL scheme uses, like the BS solver, a strain related and hence can not be used in combination with the Large Deformations module.

$$\mathbf{e}^i = \boldsymbol{\varepsilon} + (\mathbf{C}^0 + \mathbf{C})^{-1} \left(\lambda^{-1} - \mathbf{C} : \boldsymbol{\varepsilon} \right) \tag{3.18}$$

The Green operator for anisotropic materials reads in Fourier space

$$\hat{\Gamma}^0_{ijkl} = -k_j k_l \left(C^0_{ijkl} k_l k_j \right)^{-1} \tag{3.19}$$

The iteration procedere is stopped, when

$$\max \left(\frac{\|\boldsymbol{\varepsilon}^i - \mathbf{e}^i\|}{\|\mathbf{E}\|}, \frac{\|\lambda^i - \frac{\partial w}{\partial \varepsilon}(\varepsilon^i)\|}{\|\langle \frac{\partial w}{\partial \varepsilon}(\mathbf{E}) \rangle\|} \right) \leq \eta \tag{3.20}$$

Usage The solver is called via Solve(ElasticProperties , **double** equilibrium, **int** MAXIterations) , where equilibrium is the convergence criterium following (3.20). From our experience values of $\eta = 10^{-5}$ or $\eta = 10^{-6}$ yield good accuracy. MAXIterations sets the maximum allowed number of iterations, which inherently depends severly on the stiffness contrast of the used materials. Before the solver is called, the load is internally finalized with SetLoad(ElasticProperties) .Check wether SetGrainsProperties (...) , SetEffectiveElasticConstants () and SetEffectiveEigenstrains(...) of ElasticProperties and ElasticityModel have been called.

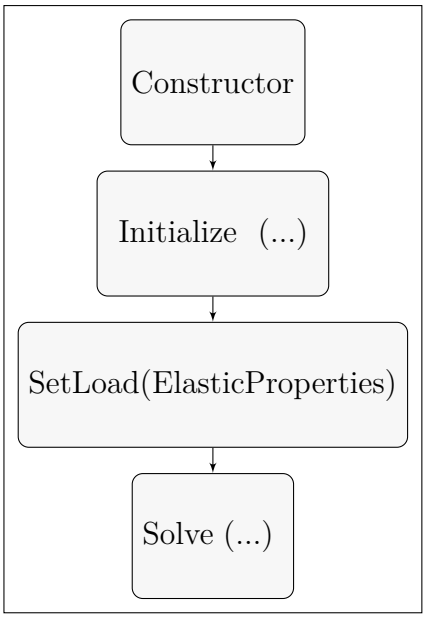

Figure 3.14: Call graph for module SpectralElasticSolverAL

3.16 Large Deformations

Module in brief...

What it does Calculates the mechanics in the case of large deformations

Requires Settings, BoundaryConditions, PhaseField, ElasticProperties

Header file Mechanics/LargeDeformations/LargeDeformationsElastic.h, Mechanics/LargeDeformations/LargeDeformationsElastoPlastic.h

Input file LDInput.opi

Examples /examples/LD-J2Plasticity

The plasticity modules use OpenPhase's large deformation framework following an Eulerian approach with a fixed grid. The total deformation increment due to external strain or changes in the eigenstrain field is evaluated and splitted to increments which are *small* and can be passed to the spectral linear elastic solver. Once the BVP is solved for the given load increment, the resulting velocity field updates the geometry (phase-field), the stresses are integrated and the next load increment is applied.

Usage The solver is called by the single command

```
1 $verbose Severe information output (Yes/No)          : Yes
2 $MaxSolverIterations  Max. num of solver iterations  : 200
3 $SolverPrec1    Solver precision strain              : 10.0e-6
4 $SolverPrec2    Solver precision pressure            : 100.0
5 $MaxStrainInc  Max. allowed strain increment         : 0.01
6 $nSubMultiplicator  Increment reduction factor       : 2.0
7 $maxRefinementNumber                                 : 5
```

Listing 3.3: Example of LDInput.opi input file for LargeDeformations module

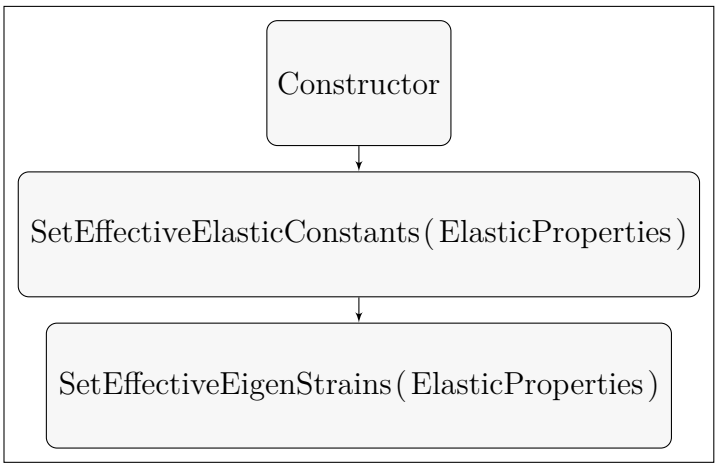

Figure 3.15: Call graph for module Large Deformations

3.17 Plasticity modules

Module in brief...

What it does

Requires Settings, PhaseField, BoundaryConditions

Input file PlasticPropertiesJ2.opi,PlasticPropertiesCPphenom.opi, etc.

Examples

Structure

The plasticity functionality of OpenPhase is splitted to classes storing and managing the corresponding storages (PlasticProperties , PlasticPropertiesJ2 , ...) and classes to calculate the plastic strain (PlasticityModel, PlasticityModelJ2, ...). In a typical elasto-plastic calculation the properties and the model classes have to be invoked and are passed to the SpectralElastoPlasticSolver.

Homogenization

Plastic strains and plastic properties (hardening parameters, dislocation densities, etc.) are defined for each individual phase-field γ. Using

$$\boldsymbol{\varepsilon}^{\mathrm{pl}}_{\mathrm{eff}} = \sum_{\gamma}^{N_\gamma} \boldsymbol{\varepsilon}^{\mathrm{pl}}_{\gamma} \phi_\gamma \tag{3.21}$$

the complete homogenization (Reuss $\boldsymbol{\sigma} = \boldsymbol{\sigma}_\alpha = \boldsymbol{\sigma}_\beta$) becomes

$$\boldsymbol{\varepsilon}_{\mathrm{eff}} = \left(\sum_{\gamma}^{N_\gamma} \phi_\gamma \mathbf{C}^{-1}_\gamma \right) \boldsymbol{\sigma} + \underbrace{\sum_{\gamma}^{N_\gamma} \boldsymbol{\varepsilon}^{*}_{\gamma}}_{\boldsymbol{\varepsilon}^{*}_{\mathrm{eff}}} + \underbrace{\sum_{\gamma}^{N_\gamma} \boldsymbol{\varepsilon}^{\mathrm{pl}}_{\gamma}}_{\boldsymbol{\varepsilon}^{\mathrm{pl}}_{\mathrm{eff}}} . \tag{3.22}$$

The effective values are handed over to the spectral elastic solver.

J2 plasticity model

The J2 model represent the most simple plasticity model implemented in OpenPhase. It is an isotropic plasticity model with linear and kinematic hardening. Yielding occurs, if

$$f(\boldsymbol{\sigma}, \mathbf{q}) = \|\boldsymbol{\eta}\| - \sqrt{\frac{2}{3}} (\sigma_y + \theta \bar{H} \alpha)$$

where

$$\boldsymbol{\eta} \hat{=} \mathrm{dev}(\boldsymbol{\sigma}) - \boldsymbol{\beta}.$$

The stress deviator is defined as

$$\mathrm{dev}(\boldsymbol{\sigma}) = \boldsymbol{\sigma} - \frac{1}{3} \mathrm{trace}(\boldsymbol{\sigma})$$

Furthermore, σ_y is the yield strength obtained by unidirectional tensile tests, θ and $\bar{H}$

$$\begin{aligned} \boldsymbol{\eta} &\hat{=} && \mathrm{dev}(\boldsymbol{\sigma}) - \boldsymbol{\beta} \\ \dot{\boldsymbol{\varepsilon}}^{p} &= && \gamma \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} \\ \dot{\alpha} &= && \gamma \sqrt{\tfrac{2}{3}} \\ \dot{\boldsymbol{\beta}} &= && \gamma \tfrac{2}{3} (1 - \theta) \bar{H}(\alpha) \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} \end{aligned}$$

Hill plasticity model

In opposite to the J2 model, the Hill model allows an anisotropic yield surface. Currently no hardening model is implemented for this model. Starting with

$$\dot{\varepsilon}^p = \dot{\lambda}\frac{\partial f}{\partial \boldsymbol{\sigma}}$$

a yield surface is defined as

$$f = \alpha_{12}(\boldsymbol{\sigma}_{11}-\boldsymbol{\sigma}_{22})^2+\alpha_{23}(\boldsymbol{\sigma}_{22}-\boldsymbol{\sigma}_{33})^2+\alpha_{31}(\boldsymbol{\sigma}_{33}-\sigma_{11})^2+6\alpha_{44}\boldsymbol{\sigma}_{12}^2+6\alpha_{55}\boldsymbol{\sigma}_{23}^2+6\alpha_{66}\boldsymbol{\sigma}_{31}^2-\bar{\sigma}^2.$$

α_{12}, α_{23}, α_{31}, α_{44}, α_{55},α_{66} are the anisotropic parameters with respect to a reference yield strenght $\bar{\sigma}$. Briefly, one could also write

$$f = \frac{3}{2}\boldsymbol{\sigma}^T\mathbf{P}\boldsymbol{\sigma} - \bar{\sigma}^2$$

where

$$\mathbf{P} = \begin{pmatrix} \frac{1}{3}(\alpha_{12}+\alpha_{31}) & -\frac{1}{3}\alpha_{12} & -\frac{1}{3}\alpha_{31} & 0 & 0 & 0 \\ -\frac{1}{3}\alpha_{12} & \frac{1}{3}(\alpha_{23}+\alpha_{12}) & -\frac{1}{3}\alpha_{23} & 0 & 0 & 0 \\ -\frac{1}{3}\alpha_{31} & -\frac{1}{3}\alpha_{23} & \frac{1}{3}(\alpha_{31}+\alpha_{23}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\alpha_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\alpha_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\alpha_{66} \end{pmatrix}$$

The Hill model will fall back to the J2 model for $\alpha_{12} = \alpha_{23} = \alpha_{31} = \alpha_{44} = \alpha_{55} = \alpha_{66} = 1$. More details can be found in [6].

Crystal plasticity model

The FCC crystal plasticity model starts from

$$\mathbf{D}_\gamma^{\mathrm{pl}} = \sum_{s=1}^{N_s}\dot{\gamma}_\gamma^s\frac{1}{2}\left(\mathbf{m}_\gamma^s\otimes\mathbf{n}_\gamma^s + \mathbf{n}_\gamma^s\otimes\mathbf{m}_\gamma^s\right) = \sum_{s=1}^{N_s}\dot{\gamma}_\gamma^s\mathbf{P}^s \tag{3.23}$$

$$\mathbf{W}_\gamma^{\mathrm{pl}} = \sum_{s=1}\dot{\gamma}_\gamma^s\frac{1}{2}\left(\mathbf{m}_\gamma^s\otimes\mathbf{n}_\gamma^s - \mathbf{n}_\gamma^s\otimes\mathbf{m}_\gamma^s\right) = \sum_{s=1}^{N_s}\dot{\gamma}_\gamma^s\mathbf{M}^s \tag{3.24}$$

where

$\dot{\gamma}_\gamma^s$: shear rate on glide system s
$\mathbf{D}_\gamma^{\mathrm{pl}}$: plastic strain rate in phase γ
$\mathbf{W}_\gamma^{\mathrm{pl}}$: plastic spin in phase γ
$\mathbf{m}_\gamma^s$: vector of the slip direction
$\mathbf{n}_\gamma^s$: glide system's normal direction

GS	$n1$	$n2$	$n3$	$d1$	$d2$	$d3$
1	-1.0	1.0	1.0	0.0	1.0	-1.0
2	-1.0	1.0	1.0	1.0	0.0	1.0
3	-1.0	1.0	1.0	1.0	1.0	0.0
4	1.0	1.0	1.0	0.0	1.0	-1.0
5	1.0	1.0	1.0	1.0	0.0	-1.0
6	1.0	1.0	1.0	1.0	-1.0	0.0
7	1.0	1.0	-1.0	0.0	1.0	1.0
8	1.0	1.0	-1.0	1.0	0.0	1.0
9	1.0	1.0	-1.0	1.0	-1.0	0.0
10	1.0	-1.0	1.0	0.0	1.0	1.0
11	1.0	-1.0	1.0	1.0	0.0	-1.0
12	1.0	-1.0	1.0	1.0	1.0	0.0

Table 3.1: Glide systems in FCC material

The phenomenological crystal plasticity model uses the following model instead

$$\dot{\gamma}_\gamma^s = \gamma_0 \left(\frac{|\boldsymbol{\sigma}_\gamma : \mathbf{M}_\gamma^s|}{\tau_{c,\gamma}^s} \right)^{m_\gamma} \operatorname{sgn}\left(\boldsymbol{\sigma}_\gamma : \mathbf{M}_\gamma^s \right), \tag{3.25}$$

where

$$\begin{aligned} \gamma_0 &: \text{referential shear rate} \\ \boldsymbol{\sigma}_\gamma : \mathbf{M}_\gamma^s &: \text{resolved shear stress on glide system } s \\ \tau_{c,\gamma}^s &: \text{critical resolved shear stress on glide system } s \\ m_\gamma &: \text{hardening exponent} \end{aligned}$$

The evolution of the critical resolved shear stress is assumed to be

$$\dot{\tau}_{c,\alpha} = q_{\alpha\beta} h_0 \left(1 - \frac{\tau_\beta}{\tau_s} \right)^a |\dot{\gamma}_\beta| \tag{3.26}$$

Herein, h_0 is a referential hardening parameters, τ_s is the saturation stress and a the hardening exponent. Finally, $q_{\alpha\beta}$ is a 12×12 matrix 1.0 for self-hardening terms and 1.4 for co-planar hardening. In FCC materials, the glide systems are defined by <111> glide plane normals and [100] glide directions, see Table 3.1

For an dislocation based approach, the slip rate is defined as

$$\dot{\gamma}_\gamma^s = \rho_\gamma^s b_\gamma v_{0,\gamma} \left(\frac{|\boldsymbol{\sigma}_\gamma : \mathbf{M}_\gamma^s|}{\tau_{c,\gamma}^s} \right)^{m_\gamma} \operatorname{sgn}\left(\boldsymbol{\sigma}_\gamma : \mathbf{M}_\gamma^s \right), \tag{3.27}$$

where

$v_{0,\gamma}$: referential dislocation velocity
b_γ : length of the burgers vector
$\boldsymbol{\sigma}_\gamma : \mathbf{M}^s_\gamma$: resolved shear stress on glide system s
$\tau^s_{c,\gamma}$: critical resolved shear stress on glide system s
m_γ : hardening exponent

The evolution of statistically stored dislocations (SSD) due to micro-mechanical processes is governed by the Kocks-Mecking evolution law

$$\dot{\rho}^s_\gamma = \left(c_1\sqrt{\rho^s_\gamma} - c_2\rho^s_\gamma\right)|\dot{\gamma}^s_\gamma| \tag{3.28}$$

Integration

The velocity gradient $\mathbf{L}$ is returned from Fourier space as

$$\widetilde{\mathbf{L}}(\boldsymbol{\xi}) = i\boldsymbol{\xi} \otimes \widetilde{\mathbf{u}}(\boldsymbol{\xi}). \tag{3.29}$$

and can be decomposed additively to strain rate and spin

$$\mathbf{L} = \mathbf{D} + \mathbf{W} = \mathbf{D}^{\mathrm{el}} + \mathbf{W}^{\mathrm{el}} + \mathbf{D}^{\mathrm{pl}} + \mathbf{W}^{\mathrm{pl}} \tag{3.30}$$

Following the integration scheme as introduced by [Peirce, Asaro, Needleman, Acta metall. 1983], one gets

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n + \left(\boldsymbol{\sigma}^\nabla_{n+1} + \mathbf{W}_{n+1}\boldsymbol{\sigma}_n - \boldsymbol{\sigma}_n\mathbf{W}_{n+1}\right)\delta t. \tag{3.31}$$

where the Jaumann rate of Kirchhoff stress reads

$$\boldsymbol{\sigma}^\nabla_{n+1} = \mathbf{C}_n\mathbf{D}_{n+1} - \sum_{s=1}^{N_s}\mathbf{R}_n\dot{\gamma}_s - \boldsymbol{\sigma}_n\mathrm{tr}(\mathbf{D}_{n+1}) \tag{3.32}$$

and

$$\mathbf{R}_s = \mathbf{C}\mathbf{P}_s + \mathbf{W}\boldsymbol{\sigma} - \boldsymbol{\sigma}\mathbf{W}_s \tag{3.33}$$

All local tensors (stiffness, eigenstrains, diffusion parameter, ...) are rotated from the initial configuration to the current orientation by $\mathbf{O}_{n+1}$. Only the elastic spin $\mathbf{W}^{\mathrm{el}} = \mathbf{W} - \mathbf{W}^{\mathrm{pl}}$ rotates the crystal, hence

$$\mathbf{O}_{n+1} = \exp\left(\mathbf{W}^{\mathrm{el}}\delta t\right)\mathbf{O}_n \tag{3.34}$$

where

$$\exp\left(\mathbf{W}^{\mathrm{el}}\delta t\right) = \mathbf{I} + \frac{\sin\left(\|\delta t\mathbf{W}^{\mathrm{el}}\|\right)}{\|\mathbf{W}^{\mathrm{el}}\|} + \frac{1-\cos\left(\|\delta t\mathbf{W}^{\mathrm{el}}\|\right)}{\|\mathbf{W}^{\mathrm{el}}\|^2}\mathbf{W}^{\mathrm{el}}\mathbf{W}^{\mathrm{el}} \tag{3.35}$$

3.18 Temperature

Module in brief...

What it does Stores temperature field and provides several methods to control it during the simulation.

Requires Settings, PhaseField, BoundaryConditions

Header File Temperature.h

Input file Temperature.opi

Examples /examples/HeatEquation, ...

Background and usage The main purpose of the temperature module is the storage of a temperature field, here called $\mathrm{Tx}(i, j, k)$, as well as the temperature gradients used for the heat calculations. Since $\mathrm{Tx}(i, j, k)$ is public it can be accessed from the main.cpp in order to declare boundary conditions manually. However, the module provides various routines to set the temperature at the beginning or during the simulation.

SetInitial (BoundaryConditions& BC)
: Sets temperature $\mathrm{Tx}(i, j, k)$ to constant value T_0 as defined in the input file. For $\mathrm{dT/dx} \neq 0$, $\mathrm{dT/dy} \neq 0$ or $\mathrm{dT/dz} \neq 0$.

Set(BoundaryConditions& BC, **double** dt)
: Integrates the temperature by $\delta T = \mathrm{dT/dt}dt$, where dT/dt is given in the temperature input file.

The temperature field can be read and write to disk using Write(**int** tStep) and Read(**int** tStep). VTK output can be generated via WriteVTK(**int** tStep) for temperature and WriteTemperatureGradientVTK(**int** tStep) for gradients.

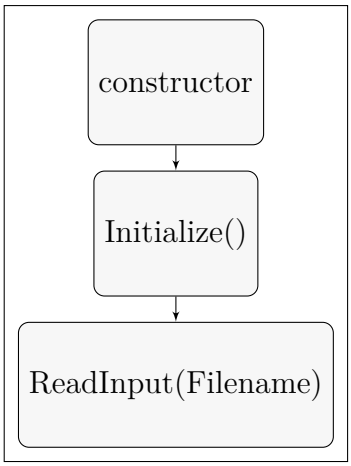

Figure 3.16: In and output for module Temperature

3.19 Heat

Module in brief...

What it does Calculates the heat equation $\frac{\partial u}{\partial t} = \Delta u + \dot{q}$

Requires Settings, PhaseField, BoundaryConditions, Temperature

Header file Heat.h

Input file Heat.opi

Examples /examples/HeatEquationSolver

Background The heat module solves the heat equation

$$\frac{\partial u(\mathbf{x})}{\partial t} = \alpha(\mathbf{x})\Delta u(\mathbf{x}) + \dot{q}(\mathbf{x}) \tag{3.36}$$

where $u(\mathbf{x})$ is the local temperature, $\dot{q}(\mathbf{x})$ is a heat source (or sink) and $\Delta = \sum_{k=1}^{3} \frac{\partial^2}{\partial x^2}$ is the Laplacian operator. The intergration of (3.36) using a Forward Euler scheme in time and a central difference stencil reads

$$\begin{aligned}(u^{n+1}(\mathbf{x}) - u^n(\mathbf{x}))\Delta t^{-1} = &\frac{u^n_{i+1,j,k} - 2u^n_{i,j,k} + u^n_{i-1,j,k}}{(\Delta x)^2} \\ &\frac{u^n_{i,j+1,k} - 2u^n_{i,j,k} + u^n_{i,j-1,k}}{(\Delta x)^2} \\ &\frac{u^n_{i,j,k+1} - 2u^n_{i,j,k} + u^n_{i,j,k-1}}{(\Delta x)^2} + \dot{q}(\mathbf{x}).\end{aligned} \tag{3.37}$$

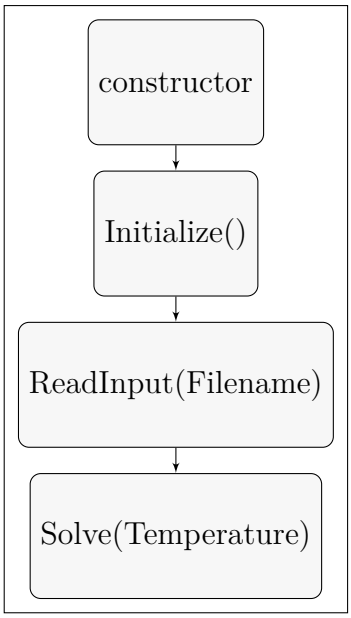

Figure 3.17: Call graph for module Heat

The scheme is stable for

$$\Delta t < 0.25 * (\Delta dx)^2/\alpha \tag{3.38}$$

Usage Following the initialization, settings are read from ProjectInput/Heat.opi. In every time step, the heat equation can be evaluated by Solve(Temperature) . Internally, this method sets boundary conditions, calculates the effective thermal diffusivity as well as the effective heat capacity, determines the Laplacian according to (3.37) and integrates the temperature.

3.20 Advection - High Resolution

Module in brief...

Header file AdvectionHR/AdvectionHR.h

What it does Advects data in a Storage3D according to a given velocity field.

Requires Settings, **??**, BoundaryConditions

Input file Advection.opi

Examples /benchmarks/AdvectionHR, ...

Background and usage Passive fields q like temperature, concentrations and density are advected with velocity v according to the advection equation

$$\partial_t q + \nabla \cdot (qv) = 0. \tag{3.39}$$

Advection in each grid axis is handled individually as this allows for larger time steps due to the CFL-condition. Therefore we can simplify the problem to the one-dimensional case. A Godunov-type scheme is utilized that considers the fluxes $f_{i+\frac{1}{2}}$ and $f_{i-\frac{1}{2}}$ over the border of a control volume

$$q_i^{n+1} = q_i^n - \frac{dt}{dx}\left[f_{i+\frac{1}{2}}(t^n, q^n) - f_{i-\frac{1}{2}}(t^n, q^n)\right], \tag{3.40}$$

using the high resolution fluxes

$$f_{i+\frac{1}{2}}(t,q) = v^+(x_{i+\frac{1}{2}}, t)\left[q_i + \phi\left(r_i\right)\left(q_{i+1} - q_i\right)\right] \tag{3.41}$$

$$+ v^-(x_{i+\frac{1}{2}}, t)\left[q_i + \phi\left(\frac{1}{r_{i+1}}\right)\left(q_i - q_{i+1}\right)\right], \tag{3.42}$$

with $v^+(x_{i+\frac{1}{2}}, t) = \max(\frac{v_{i+1}+v_i}{2}, 0)$ and $v^-(x_{i+\frac{1}{2}}, t) = \min(\frac{v_{i+1}+v_i}{2}, 0)$ and a flux limiter function ϕ. that with a proper choice allows second order accuracy in smooth regions and counteracts oscillations near discontinuities as observed with second-order accurate methods such as Lax-Wendroff or Beam-Warming, see [**?**]. The quotient r_i is determined by two consecutive slopes as

$$r_i = \frac{q_i - q_{i-1}}{q_{i+1} - q_i}. \tag{3.43}$$

In order to avoid a division by zero when $q_{i+1} = q_i$ we replace the limited slopes in 3.42 with

$$\phi(r_i)(q_{i+1} - q_i) = L(q_{i+1} - q_i, q_i - q_{i-1}), \tag{3.44}$$

$$\phi\left(\frac{1}{r_{i+1}}\right)(q_i - q_{i+1}) = -L(q_{i+2} - q_{i+1}, q_{i+1} - q_i), \tag{3.45}$$

as suggested in [**?**]. The limiters implemented in OpenPhase are

Minmod:	$L(a,b) := S(a,b)\min\left(\lvert a\rvert, \lvert b\rvert\right),$
Monotonized Central:	$L(a,b) := S(a,b)\min\left(\frac{\lvert a+b\rvert}{2}, 2\lvert a\rvert, 2\lvert b\rvert\right),$
Superbee:	$L(a,b) := S(a,b)\max\left(\min\left(2\lvert a\rvert, \lvert b\rvert\right), \min\left(\lvert a\rvert, 2\lvert b\rvert\right)\right),$

with

$$S(a,b) := \frac{\mathrm{sgn}(a) + \mathrm{sgn}(b)}{4}. \tag{3.46}$$

In the case of $L(a,b) = 0$ for all $a, b \in \mathbb{R}$ the scheme becomes a simple first-order upwind method. Finally we have

$$f_{i+\frac{1}{2}}(t,q) = v^+(x_{i+\frac{1}{2}}, t)\left[q_i + L(q_{i+1} - q_i, q_i - q_{i-1})\right] \tag{3.47}$$

$$+ v^-(x_{i+\frac{1}{2}}, t)\left[q_i - L(q_{i+2} - q_{i+1}, q_{i+1} - q_i)\right], \tag{3.48}$$

$$f_{i-\frac{1}{2}}(t,q) = v^+(x_{i-\frac{1}{2}}, t)\left[q_{i-1} + L(q_i - q_{i-1}, q_{i-1} - q_{i-2})\right] \tag{3.49}$$

$$+ v^-(x_{i-\frac{1}{2}}, t)\left[q_{i-1} - L(q_{i+1} - q_i, q_i - q_{i-1})\right]. \tag{3.50}$$

For multi-dimensional system those one dimensional-advection methods are used subsequently in each direction. The Strang splitting technique, which alternates order of execution in each time step, is second order accurate in time [**?**]. For $n \bmod 2 = 0$ the schemes becomes

$$q^{n+\frac{1}{3}} = q^n - dt(v_1 q^n)^h_x, \tag{3.51}$$

$$q^{n+\frac{2}{3}} = q^{n+\frac{1}{3}} - dt(v_2 q^{n+\frac{1}{3}})^h_y, \tag{3.52}$$

$$q^{n+1} = q^{n+\frac{2}{3}} - dt(v_3 q^{n+\frac{2}{3}})^h_z, \tag{3.53}$$

$$q^{n+1+\frac{1}{3}} = q^{n+1} - dt(v_3 q^{n+1})^h_z, \tag{3.54}$$

$$q^{n+1+\frac{2}{3}} = q^{n+1+\frac{1}{3}} - dt(v_2 q^{n+1+\frac{1}{3}})^h_y, \tag{3.55}$$

$$q^{n+2} = q^{n+1+\frac{2}{3}} - dt(v_1 q^{n+1+\frac{2}{3}})^h_x, \tag{3.56}$$

with the velocity $v = (v_1, v_2, v_3)^t$ and the numerical high resolution flux differences $(.)^h_x$, $(.)^h_y$, $(.)^h_z$.

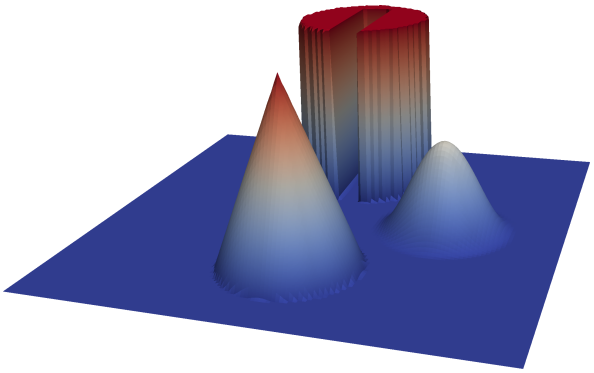

Figure 3.18: Initial conditions for the solid body rotation test in section 3.20.1.

3.20.1 Solid body rotation test

The implementation of the advection scheme is tested with the solid body rotation test mentioned in [?]. In the domain $\Omega = [0,1] \times [0,1]$ we have a velocity field $v(x,y) = (0.5 - y, x - 0.5)^t$ that describes a totation around the center $c = (0.5, 0.5)^t$. In the domain Ω are three bodies, a slotted cylinder given by

$$q_{\text{cyl}}(x,y,0) = \begin{cases} 1 & \text{if } (|x - x_0| \geq 0.025 \vee y \geq 0.85) \wedge r(x,y) \leq 1 \\ 0 & \text{else ,} \end{cases} \tag{3.57}$$

with $(x_0, y_0) = (0.5, 0.75)$ and $r(x,y) = \frac{1}{r_0}\sqrt{(x - x_0)^2 + (y - y_0)^2}$ and $r_0 = 0.15$, a cone at $(x_0, y_0 = (0.5, 0.25)$ given by

$$q_{\text{cone}}(x,y,0) = \begin{cases} 1 - r(x,y) & \text{if } r(x,y) \leq 1 \\ 0 & \text{else ,} \end{cases} \tag{3.58}$$

and a hump at $(x_0, y_0) = (0.25, 0.5)$ determined by

$$q_{\text{hump}} = \begin{cases} 0.25(1 - \cos(\pi \min\{r(x,y), 1\})) & \text{if } r(x,y) \leq 1 \\ 0 & \text{else .} \end{cases} \tag{3.59}$$

The initial conditions are $q(x,y,0) = \max\{q_{\text{cyl}}(x,y,0), q_{\text{cone}}(x,y,0), q_{\text{hump}}(x,y,0)\}$. The domain Ω is discretized by a regular grid with a 128×128 grid points and a time step of $dt = 10^{-3}$ is used. After one full rotation the solution is compared to the initial conditions.

In table 3.2 we see that the superbee limiter generates the smallest error, while the first-order upwind method creates the largest error. However in figure 3.22 we see that the cone gets a bit distorted with the superbee limiter, whereas the MC-Limiter produces a much better result as seen in figure 3.21, also the top of the hump remains smooth. With the first-order Upwind method as well as the Minmod-Limiter we see

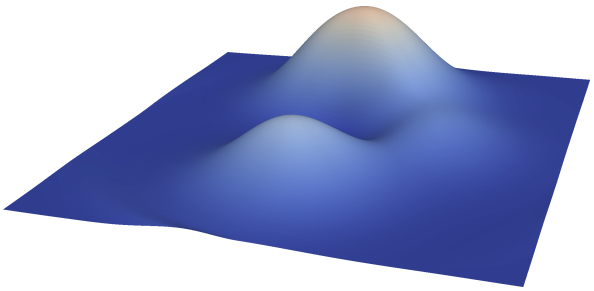

Figure 3.19: Result with the upwind method.

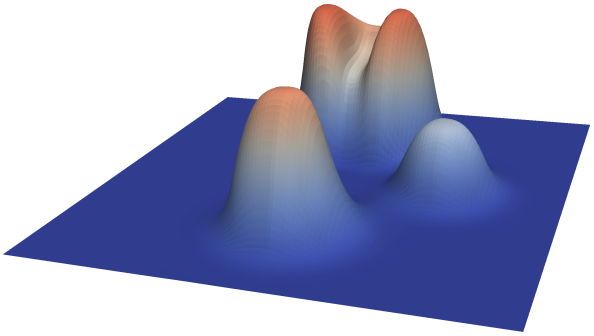

Figure 3.20: Result of the high resolution method with the minmod limiter.

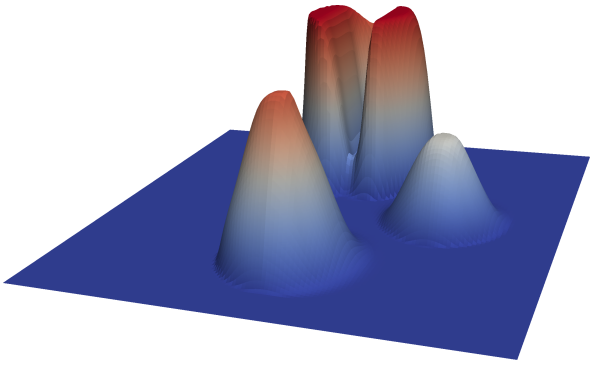

Figure 3.21: Result of the high resolution method with the MC-limiter.

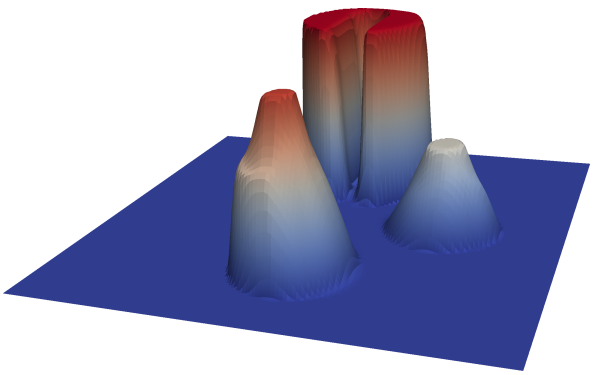

Figure 3.22: Result of the high resolution method with the superbee limiter.

Method	L_1 error	L_2 error
Upwind	0.0948	0.1798
Minmod	0.0405	0.1142
MC	0.0202	0.0768
Superbee	0.0134	0.0540

Table 3.2: Results of the solid body rotation test in section 3.20.1 with various limiters.

significant numerical diffusion in figures 3.19 and 3.20.

void Initialize (**const** Settings &Settings)
Standard initialize function.

void ReadInput (**const** std::string InputFileName)
Standard read input functions, selects the scheme via $scheme from Advection.opi. Options are Upwind, Minmod, MC and Superbee.

void Advect (Storage3D< T, rank > &Field, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Advects data stored in Storage3D if the type T is **double** , dVector3 , Quaternion or vStress according to the velocity field in Vel with the time step dt and the grid width dx . In order to use the Strang splitting the number of the timestep tStep has to be provided.

void Advect (PhaseField &Phase, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Specialization for the Phasefield class.

void Advect (Composition &Cx, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Specialization for the Composition class.

void Advect (Temperature &Tx, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Specialization for the Temperature class.

void Advect (ElasticProperties &EP, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

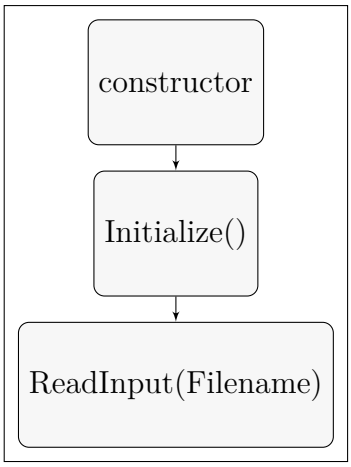

Figure 3.23: In and output for module Advection - High Resolution

Specialization for the ElasticProperties class.

void Advect (Orientations &OR, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Specialization for the Orientations class.

void Advect (Plasticity &PL, PhaseField &Phase, **const** Velocities &Vel, **const** BoundaryConditions &BC, **const double** dx, **const double** dt, **const int** tStep)

Specialization for the Plasticity class.

3.21 Tools

Module in brief...

What it does Provides additional methods which are not part of the principales modules and/or required for the execution of OpenPhase

Header File Tools.h

Requires Nothing

Input file None

Examples

Abaqus input file writer

Using the initialization methods of OpenPhase makes it very convenient to create representative volume elements for Simulia Abaqus calculations. The routine can be invoked after the initialization of the phase-field using Tools::WriteAbaqusInput(PhaseField) .

4 Examples and benchmarks

4.1 Grain growth/shrinkage of a spherical grain

4.1.1 Preliminaries

A spherical grain will shrink due to its curved interface in order to minimize the interface energy. Doing so, the grain retains its sperical shape while the radius decreases with time.

From theoretical considerations, an analytic formalism can be can be found which describes the kinetics of this shrinkage. This example simulation should reproduce this dependence. Starting with the phase-field equation for two phases

$$\dot{\phi} = \mu \left(\sigma \left(\nabla^2 \phi + \frac{\pi^2}{2\eta^2}(2\phi - 1) \right) + \Delta G \right) \tag{4.1}$$

with σ being the interfacial energy, μ the mobility parameter and ΔG any additional driving force, For the grain boundary evolution when particle radius $R \ll \eta$, an approximate theoretical time dependence of the grain radius can be obtained ($\Delta G = 0 J/m^3$). At time $t = 0$ the sphere radius is R_0. In the interface of width η, the phase-field is expressed in spherical coordinates

$$\phi(r) = \frac{1}{2} - \frac{1}{2} \sin \frac{\pi(r - R)}{\eta} \tag{4.2}$$

and noting that

$$v = -\frac{dr}{dt} = \dot{\phi} \frac{\partial r}{\partial \phi} \tag{4.3}$$

one can obtain the following growth relation

$$R^2 - R_0^2 = -4\sigma t. \tag{4.4}$$

4.1.2 Modules and parameters

The benchmark uses the three mandatory modules PhaseField, BoundaryConditions and Settings. Moreover, the modules InterfaceField, InterfaceEnergy and InterfaceMobility are required. A spherical grain of phase 1, initial radius R_0, surrounded by phase 0 is

Parameter	Value
$N_x = N_y = N_z$	65
Δx	1×10^{-6}m
η	$5 \cdot \Delta x$
μ	4×10^{-9}m^4/(Js^2)
σ	0.24J/m^2
Δt	1×10^{-5}s
R_0	$N_x/3$

Table 4.1: Setup parameters

considered. The grain is placed in the center of a cubic box with zero-gradient boundary conditions. No driving force is applied.

The box size is 64x64x64 grid points. The input parameters are listed in table 4.1.

4.1.3 Results

The output data in .vts format is written in the VTK directory and can be directly visualized using Paraview software (www.paraview.org). The R_2_graph.dat file contains the values of the time step and the particle radius calculated analytically and numerically. Figure **??** shows the results of R^2 over timesteps for different values of η. Obviously, a wider interface width improves result towards the analytic solution, yet at the price of higher calculation costs. In addition, simulation with constant driving forces $\Delta G_{\alpha\beta}$ (Matrix α, Phase β) are performed and compared against the solution shown above. Here, however, the solution for equation (4.1) can only be obtained numerically. Figure 4.2 compares the solution of OpenPhase with these solutions.
As a further example, the influence of a elastic driving forces should be benchmarked. Therefore, an eigenstrain of $\varepsilon_{11} = \varepsilon_{22} = \varepsilon_{33} = 0.005$ is defined for the inclusion. The elastic constants for inclusion and matrix are taken equally (see Eshelby example). Free stress boundary conditions are applied, $\bar{\boldsymbol{\sigma}} = 0$. The results are shown in figure 4.3. Obviously, the driving force remains constant throughout the shrinkage of the grain. Looking at the analytic solution of the stress field as derived by Eshelby, eq. (4.5), the maximum stress in the interface does not depend on the inclusion diameter, but only on the eigenstrain magnitude. Consequently, the elastic driving force (3.4) has to remain constant, as reflected by the analytical solution.

4.2 Eshelby Test

The Eshelby test benchmark is designed to validate the different mechanical solver that come along with OpenPhase, namely the SpectralElasticSolver by Hu and Chen [2], the

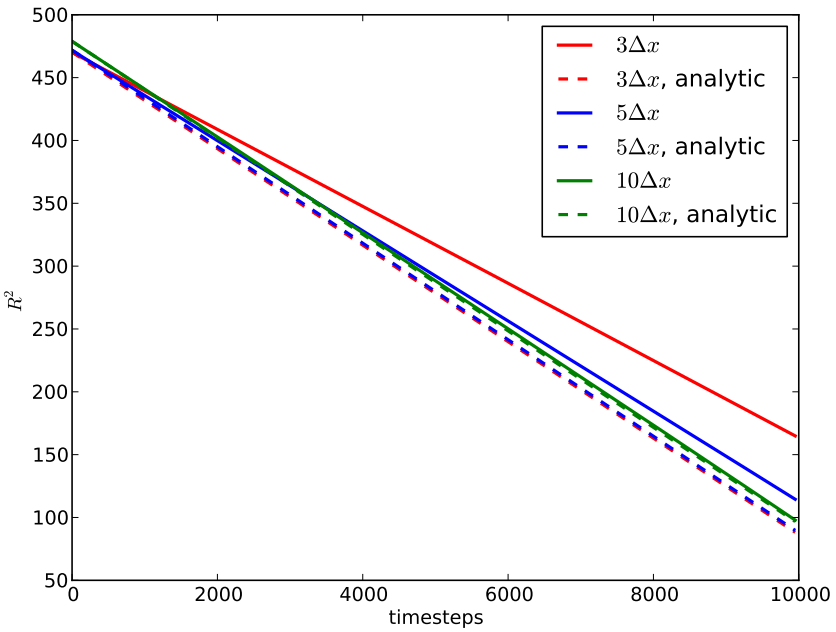

Figure 4.1: R^2 of sphere radius for different interfacial widths. Comparison with analytical solution (dashed lines).

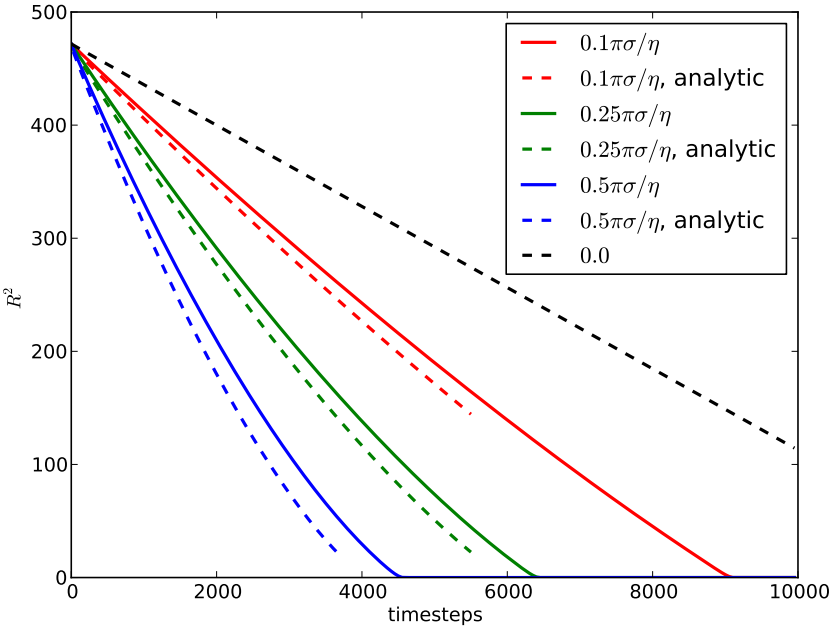

Figure 4.2: R^2 of sphere radius for different constant driving forces $\Delta G_{\alpha\beta}$. Comparison with analytical solution (dashed lines).

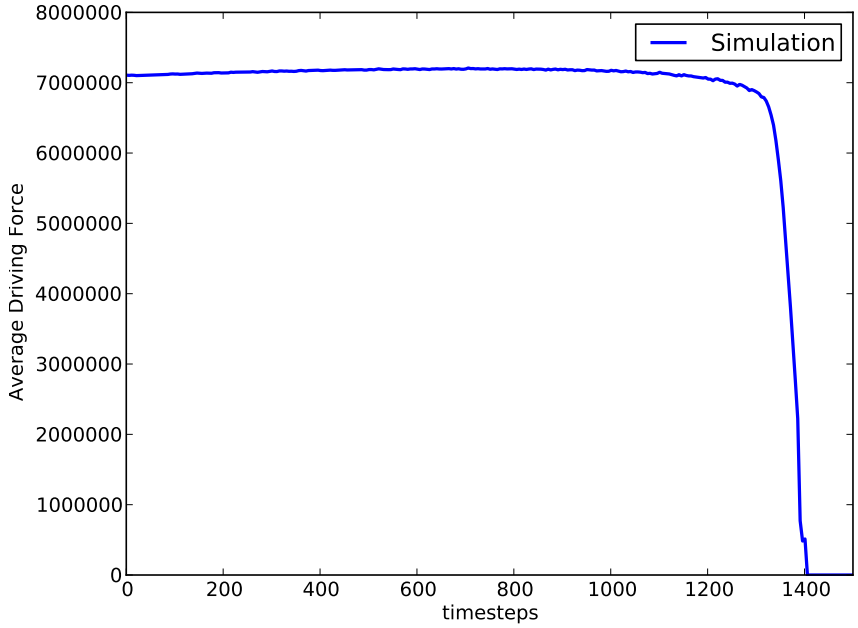

Figure 4.3: Average driving force in the interface during shrinkage of grain.

SpectralElasticSolverBS by Moulinec and Suquet [3], the SpectralElasticSolverAL by Michel et al. [5] as well as the SpectralElasticSolverAS by Eyre and Milton (soon).
A spherical inclusion is placed inside an infiniteley expanded elastic matrix. The sphere has an eigenstrain of $\varepsilon^*_{11} = \varepsilon^*_{22} = \varepsilon^*_{33} = 0.01$, both matrix and sphere have the same isotropic, elastic stiffness parameters. For such a setup, Eshelby [7] derived an analytical solution for stress and strain field, which can be used to validate the

4.2.1 Modules and parameters

The benchmark uses the three mandatory modules PhaseField, BoundaryConditions and Settings. The aforementioned elastic solvers (SpectralElasticSolver, SpectralElasticSolverBS and SpectralElasticSolverAL) can be chosen by changing the preprocessor flag. In order to perform the mechanical calculations it is necessary to load ElasticProperties as well one of the two homogenization modules ElasticityReuss or ElasticityKhachaturyan, respectively. Tables **??** and Table 4.3 summarize the most important parameters for the Eshelby test example. Note that in the numerical setup no infinitely large domain can be chosen.

Key	Value	Comment
$nSteps	0	Single step
$Nx	129	Note: In order to obtain appropriate results,
$Ny	129	a minimum resolution of 128 per dimension
$Nz	129	is suggested.
$Nphses	2	Two phases
$IWidth	0	No diffuse interface

Table 4.2: ProjectInput.opi for Eshelby test

Key	Value 1	Value 2
$Phase 0		
$C11	280,000	0
$C22	280,000	0
$C33	280,000	0
$C12	120,000	0
$C13	120,000	0
$C23	120,000	0
$C44	80,000	0
$C55	80,000	0
$C66	80,000	0

Table 4.3: ElasticityInput.opi for Eshelby test

4.2.2 Results

Comparisons can be obtained against the analytical solution of Eshelby [7].

$$\sigma_{ii} = \begin{cases} -\sigma_0 & \text{inside the particle; } x_1 < r_p \\ -\sigma_0 \left(\frac{r_p}{x_1}\right)^3 & \text{for } i = 1; x_1 > r_p \\ \frac{1}{2}\sigma_0 \left(\frac{r_p}{x_1}\right)^3 & \text{for } i \neq 1; x_1 > r_p \end{cases} \tag{4.5}$$

where

$$\sigma_0 = \frac{2}{3}\left(C_{11} + 2C_{12}\frac{1-2\nu}{1-\nu}\varepsilon^*\right) \tag{4.6}$$

4.3 Crystal plasticity UMAT

The CrystalPlasticityUMAT benchmark is designed to compare the CrystalPlasticity module of OpenPhase with results of CP codes such as DAMASK [8]. It evaluates the stresses and hardening variables at a single continuum point for a given strain increment. This approach is analogue to the evaluation of an integration point under a given deformation gradient (or strain) in finite element frameworks such as Abaqus (where the user-defined material model is described in a so called UMAT).

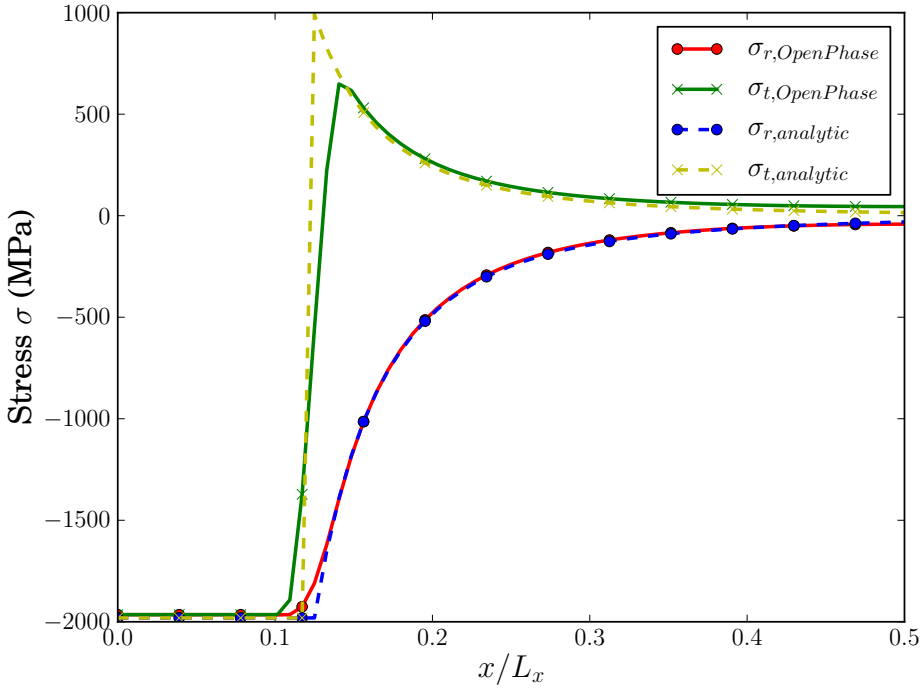

Figure 4.4: Comparison of analytic Eshelby solution with results calculated by OpenPhase.

4.3.1 Modules and Parameters

The benchmark uses the three mandatory modules PhaseField, BoundaryConditions and Settings. In order to perform the mechanical calculations it is necessary to load Orientations, ElasticProperties as well one of the two homogenization modules ElasticityReuss or ElasticityKhachaturyan, respectively. For the plasticity part PlasticPropertiesCPphenom and PlasticityModelCPphenom have to be loaded.
The benchmark sets up a one point phase field with a single phase/grain present. By defining an applied strain $\bar{\boldsymbol{\varepsilon}}$ the full stress tensor

$$\boldsymbol{\sigma} = \mathcal{C} : (\bar{\boldsymbol{\varepsilon}} - \boldsymbol{\varepsilon}_p) \tag{4.7}$$

the plastic strain increment

$$\varepsilon_p = f(\boldsymbol{\sigma}, \dot{\gamma}^{(\alpha=1..12)}, \tau_c^{(\alpha=1..12)}) \tag{4.8}$$

(see crystal plasticity model for more details) as well as the hardening

$$\tau_c^{(\alpha=1..12)} = f(\boldsymbol{\sigma}, \dot{\gamma}^{(\alpha=1..12)}) \tag{4.9}$$

for each glide system α is calculated. Since the plasticity model is of viscous type, parameter Δt is a crucial input parameter. The domain size, adjusted with parameter

dx can be chosen arbitrarily, since the length-scale will not enter the mechanical solution (at least for a local model). The calculation will return an error, if the plasticity could not converge for a given $\bar{\boldsymbol{\varepsilon}}$. The grain orientation can be adjusted by Orientations :: GrainEulerAngles[0].set() . For completeness, a set of important input parameters is given in table 4.4.

Key	Value	Comment
$nSteps	1	Single step
$Nx	1	
$Ny	1	Note: Only a single point is evaluated.
$Nz	1	
$Nphses	1	Single phase
$dt	1	
$dx	1	

Table 4.4: ProjectInput.opi for CrystalPlasticityUMAT

4.3.2 Results

All material parameters are given in mm, s, kg units.

Key	Value 1	Value 2
$Phase 0		
$C11	280,000	0
$C22	280,000	0
$C33	280,000	0
$C12	120,000	0
$C13	120,000	0
$C23	120,000	0
$C44	80,000	0
$C55	80,000	0
$C66	80,000	0

(a) Elasticity input for CrystalPlasticityUMAT benchmark

Key	Value
$explicitSolver	No
$plasticityflag_0	Yes
$nonlocalflag_0	No
$gamma0_0	0.001
$tauc_0	10.0
$taus_0	117.0
$h0_0	100.0
$flowPow_0	10.0
$hardPow_0	2.25

(b) Plasticity input for CrystalPlasticityUMAT benchmark

Table 4.5: Set of material parameters used for benchmark calculation.

References

[1] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[2] S. Y. Hu and L. Q. Chen. A phase-field model for evolving microstructures with strong elastic inhomogeneity. *Acta Materialia*, 49(11):1879–1890, 2001.

[3] H. Moulinec and P. Suquet. A numerical method for computing the overall response of nonlinear composites with complex microstructure. *Computer Methods in Applied Mechanics and Engineering*, 157(1-2):69–94, 1998.

[4] H. Moulinec and F. Silva. Comparison of three accelerated FFT-based schemes for computing the mechanical response of composite materials. *Int. J. Numer. Meth. Engng*, 2014.

[5] J. C. Michel, H. Moulinec, and P. Suquet. A computational method based on augmented Lagrangians and fast Fourier transforms for composites with high contrast. *Computer Modeling in Engineering and Science*, 1(2), 2000.

[6] René De Borst and Peter H. Feenstra. Studies in anisotropic plasticity with reference to the hill criterion. *Int. J. Numer. Meth. Engng.*, 29(2):315–336, 1990.

[7] J. D. Eshelby. The determination of the elastic field of an ellipsoidal inclusion, and related problems. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 241(1226):376–396, 1957.

[8] F. Roters, P. Eisenlohr, C. Kords, D.D. Tjahjanto, M. Diehl, and D. Raabe. DAMASK: the Düsseldorf Advanced MAterial Simulation Kit for studying crystal plasticity using an FE based or a spectral numerical solver. *Procedia IUTAM*, 3(0):3 – 10, 2012. IUTAM Symposium on Linking Scales in Computations: From Microstructure to Macro-scale Properties.

Publications

[9] E Borukhovich, P S Engels, T Böhlke, O Shchyglo, and I Steinbach. Large strain elasto-plasticity for diffuse interface models. *Modelling and Simulation in Materials Science and Engineering*, 22(3):034008, 2014.

[10] Efim Borukhovich, Philipp S. Engels, Jörn Mosler, Oleg Shchyglo, and Ingo Steinbach. Large deformation framework for phase-field simulations at the mesoscale. *Computational Materials Science*, 108, Part B:367 – 373, 2015. Selected Articles from Phase-field Method 2014 International Seminar.

[11] Reza Darvishi Kamachali and Ingo Steinbach. 3-d phase-field simulation of grain growth: Topological analysis versus mean-field approximations. *Acta Materialia*, 60(6–7):2719 – 2728, 2012.

[12] R. Darvishi Kamachali, E. Borukhovich, O. Shchyglo, and I. Steinbach. Solutal gradients in strained equilibrium. *Philosophical Magazine Letters*, 93(12):680–687, 2013.

[13] Reza Darvishi Kamachali, Efim Borukhovich, Nicholas Hatcher, and Ingo Steinbach. DFT-supported phase-field study on the effect of mechanically driven fluxes in Ni4Ti3 precipitation. *Modelling and Simulation in Materials Science and Engineering*, 22(3):034003, 2014.

[14] Reza Darvishi Kamachali, Se-Jong Kim, and Ingo Steinbach. Texture evolution in deformed AZ31 magnesium sheets: Experiments and phase-field study. *Computational Materials Science*, 104(0):193 – 199, 2015.

[15] Kumar Rajendran, Mohan, Shchyglo, Oleg, and Steinbach, Ingo. Large scale 3-d phase-field simulation of coarsening in ni-base superalloys. *MATEC Web of Conferences*, 14:11001, 2014.

[16] Dmitry Medvedev, Fathollah Varnik, and Ingo Steinbach. Simulating mobile dendrites in a flow. *Procedia Computer Science*, 18(0):2512 – 2520, 2013. 2013 International Conference on Computational Science.

Appendix

OpenPhase