A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

http://wrap.warwick.ac.uk/148427

Copyright and reuse:
This thesis is made available online and is protected by original copyright.
Please scroll down to view the document itself.
Please refer to the repository record for this item for information to help you to cite it.
Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk
Automatic Phase Field Regularisation of Interfacial Problems

by

Matthew Thomas Collins

Thesis
Submitted to the University of Warwick
for the degree of
Doctor of Philosophy

Warwick Mathematics Institute
December 2019
Contents

List of Tables vi
List of Figures vii
Abbreviations ix
Declarations x
Abstract xi

Chapter 1 Introduction 1
1.1 Background ................................................. 1
1.1.1 Gradient flow of the perimeter ......................... 5
1.1.2 Stationary solutions ................................. 7
1.1.3 Asymptotic and rigorous convergence .................. 9
1.2 Symbolic computation and software ...................... 10
1.3 Thesis contributions and structure ...................... 12
1.3.1 Mathematical framework ............................. 12
1.3.2 Software .............................................. 12
1.3.3 Fracture model ...................................... 13

Chapter 2 Sharp Model Framework 14
2.1 Sharp evolutions ........................................... 14
2.1.1 Evolutions of regular configurations .................. 16
2.1.2 Notation .............................................. 18
2.1.3 Balance laws ......................................... 22
2.1.4 Energy, variation and gradient ......................... 24
2.1.5 Gradient flow ......................................... 29
2.2 Examples .................................................. 33
2.2.1 Mean curvature flow .................................. 38
Chapter 3  From Sharp Model to Phase Field Model

3.1 Phase field evolutions .......................... 63
  3.1.1 Balance laws ................................ 64
  3.1.2 Energy, variation and gradient ............... 65
  3.1.3 Gradient flow ................................ 68
  3.1.4 Rewriting the gradient flow .................. 70

3.2 Linking sharp and phase field evolutions .......... 71
  3.2.1 Regularisation of energies .................... 71
  3.2.2 Regularisation of balance laws ............... 75
  3.2.3 Regularisation of convex set .................. 76
  3.2.4 Complete regularisation of gradient flow ....... 76

3.3 Semi-discretisation ............................ 77

3.4 Code and constructing the PhaseModel class ...... 79
  3.4.1 The PhaseModel Class ........................ 80
  3.4.2 UFL form of discretised equations ............. 80
  3.4.3 Well ....................................... 81
  3.4.4 Interpolation ................................ 82
  3.4.5 Balance degeneracy .......................... 83
  3.4.6 Summary of smoothing functions and default values .... 83

3.5 Examples ....................................... 84
  3.5.1 Mean curvature flow ......................... 84
  3.5.2 Mullins-Sekerka .............................. 85
  3.5.3 Dendritic growth ............................. 87
  3.5.4 Tumour growth ................................ 89
  3.5.5 Fracture .................................... 90
  3.5.6 Multi-phase curvature flow .................. 93

Chapter 4  Time Stepping .......................... 95

4.1 Time discretisation ................................ 95
  4.1.1 Balance laws ................................ 96
  4.1.2 Smoothing function in balance laws .......... 97
  4.1.3 Potential well ................................ 97
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.4</td>
<td>Interpolation function $I$</td>
<td>97</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Bulk density $\hat{a}$</td>
<td>98</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Energy written in terms of an orientation angle</td>
<td>98</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples</td>
<td>99</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Dendritic growth</td>
<td>99</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Tumour growth</td>
<td>100</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Fracture</td>
<td>101</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Multi-phase curvature flow</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>Structure of Software</td>
<td>105</td>
</tr>
<tr>
<td>5.2</td>
<td>The <code>PhaseModel</code> class</td>
<td>107</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Constructor</td>
<td>108</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Additional methods</td>
<td>108</td>
</tr>
<tr>
<td>5.3</td>
<td>The <code>PhaseStepper</code> class</td>
<td>109</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Constructor</td>
<td>109</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Additional methods and variables</td>
<td>111</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Dimension reduction</td>
<td>114</td>
</tr>
<tr>
<td>5.4</td>
<td>The <code>InspectSharp</code> class</td>
<td>115</td>
</tr>
<tr>
<td>5.5</td>
<td>The <code>External</code> class</td>
<td>116</td>
</tr>
<tr>
<td>6.1</td>
<td>Mean curvature flow</td>
<td>118</td>
</tr>
<tr>
<td>6.2</td>
<td>Mullins-Sekerka</td>
<td>118</td>
</tr>
<tr>
<td>6.3</td>
<td>Dendritic growth</td>
<td>118</td>
</tr>
<tr>
<td>6.4</td>
<td>Tumour growth</td>
<td>122</td>
</tr>
<tr>
<td>6.5</td>
<td>Fracture</td>
<td>123</td>
</tr>
<tr>
<td>6.6</td>
<td>Multi-phase curvature flow</td>
<td>124</td>
</tr>
<tr>
<td>7.1</td>
<td>Extensions</td>
<td>129</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Linearisation for anisotropic energy</td>
<td>129</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Complete bindings for TNNMG solver</td>
<td>129</td>
</tr>
<tr>
<td>7.1.3</td>
<td>N-phase energies</td>
<td>130</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Time discretisation of $E$</td>
<td>130</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Geometric constraints</td>
<td>130</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Surface quantities</td>
<td>131</td>
</tr>
<tr>
<td>7.2</td>
<td>Conclusion</td>
<td>132</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Summary of functions and parameters to specify a sharp model. Orange indicates default values. ........................................ 34
2.2 Summary of functions and parameters for mean curvature flow. ................................................................. 40
2.3 Difference in notation for mean curvature flow. ............................................................. 40
2.4 Summary of functions and parameters for Mullins-Sekerka type problem. .................................................. 43
2.5 Difference in notation for Mullins-Sekerka type problem. ............................................................. 43
2.6 Summary of functions and parameters for dendritic growth. ................................................................. 47
2.7 Difference in notation for dendritic growth. ............................................................. 49
2.8 Summary of functions and parameters for tumour growth. ................................................................. 52
2.9 Summary of functions and parameters for fracture. ............................................................. 57
2.10 Summary of functions and parameters for multi-phase mean curvature flow. .................................................. 60

3.1 Smoothing functions and parameters for $\mathcal{P}^\varepsilon$ gradient flow. ............................................... 84
3.2 Smoothing functions and parameters for mean curvature $\mathcal{P}^\varepsilon$ gradient flow. ............................................... 85
3.3 Smoothing functions and parameters for Mullins-Sekerka $\mathcal{P}^\varepsilon$ gradient flow. ............................................... 87
3.4 Smoothing functions and parameters for dendritic growth $\mathcal{P}^\varepsilon$ gradient flow. ............................................... 89
3.5 Smoothing functions and parameters for tumour growth $\mathcal{P}^\varepsilon$ gradient flow. ............................................... 91
3.6 Smoothing functions and parameters for fracture $\mathcal{P}^\varepsilon$ gradient flow. ................................. 92

5.1 Solver options. ......................................................... 110
## List of Figures

1.1 Diffuse vs sharp interface ................................................. 2
1.2 Relationships in phase field calculus ............................... 4
1.3 Double well varying with $\epsilon$ ...................................... 5
1.4 Fundamental phase field profile .................................... 8
1.5 Stationary phase field profile ....................................... 9
1.6 Expansion regions ...................................................... 10
1.7 Unified Modeling Language (UML) diagram with the main components of PHASEFIELD. ........................................... 12

2.1 Some possible regular configurations. ............................... 15
2.2 For fixed $G$ visualisation of (d) hypersurfaces, $d = 2, 1, 0$. ....... 17
2.3 Evolution of $G(t)$ in Example 2.1.6. ................................. 19
2.4 Triple junctions ......................................................... 33
2.5 Crack propagation, geometric setup ................................ 55

5.1 UML diagram with the main components of PHASEFIELD. ....... 106

6.1 Mean curvature flow with implicit well ............................... 119
6.2 Mean curvature flow with concave convex well ...................... 119
6.3 Mean curvature flow with obstacle well ............................. 119
6.4 Mean curvature flow with different potential wells, the legends show the radius for different values of $\epsilon$ ........................................ 119
6.5 Mullins-Sekerka volume tracking .................................... 120
6.6 Mullins-Sekerka simulation, $\varphi_1$ displayed at times $t = 0, 5, 10, \epsilon = 0.025$. 120
6.7 Dendritic growth simulation ........................................... 121
6.8 Tumour growth, $\varphi_1$ displayed at times $t = 5, 10, 20$. ........ 122
6.9 Fracture start time ...................................................... 125
6.10 Multi-phase consistency .............................................. 127
6.11 Multi-phase equilibrium shape ...................................... 127

vii
6.12 Multi-phase unstable initial conditions ........................................ 127
6.13 Multi-phase simulation with constantly transported profile ............. 128

7.1 Nanowire growth at times $t = 0, 10, 20.$ ...................................... 131
Abbreviations

**DGF** Dune Grid File.

**FEM** Finite Element.

**LHS** Left Hand Side.

**PDE** Partial Differential Equation.

**RHS** Right Hand Side.

**TNNMG** Truncated Non-Smooth Newton Multi-Grid.

**UFL** Unified Form Language.

**UML** Unified Modeling Language.
Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.
Abstract

Phase field models are a useful approximation method for sharp interfacial problems. Sharp interfacial problems aim to model physical phenomena by describing the problem as different phases separated by infinitely thin hypersurfaces. Phase field models introduce a (phase field) variable which varies smoothly between phases and has the effect of smoothing or diffusing the interfaces. A small parameter $\epsilon$ is present in the phase field equations that can be shown to be proportional to the diffuse interfacial width; when this is sent to 0 the “sharp” interfacial problem is recovered. Often one formally derives a phase field problem from thermodynamic principles. Formal asymptotics are performed to recover the sharp interfacial problem. Carrying out these asymptotics can be laborious, we show that the limit problem is often obvious using a formal gradient flow structure. Utilising this structure we implement a software interface which allows users to go in the converse direction. That is, the user postulates a sharp interfacial problem, the software then automatically regularises the sharp equations to produce a phase field approximation. This is done symbolically using the Unified Form Language. Unified Form Language is a domain specific language used by a wide range of software packages, consequently the phase field equations can be exported and solved using other finite element packages. We demonstrate the effectiveness of this package using a number of models found in the literature, ranging from two and three phase mean curvature flow problems, to more complex problems involving partial differential equations defined away from the interface e.g. dendritic growth, tumour growth and crack propagation models.
Chapter 1

Introduction

1.1 Background

Many physical systems can be modelled as material phases separated by a moving hypersurface. In the phase field method, conditions at the interface are substituted for an auxiliary smooth phase field variable which takes distinct values depending on the phase. The phase field variable therefore has the effect of making the interface between the phases “fuzzy” or “blurred”, see Figure 1.1. The paradigm we adopt is that a phase field is an analytical and numerical tool to approximate a given interfacial problem.

Phase field equations give only an approximation to the original problem, but have the advantage that they contain many additional desirable features. When solving interfacial problems using traditional numerical methods, the explicit tracking of the interface is required. Furthermore, it is often not clear how to continue the evolutions where interfaces intersect, or to more generally take into account topological change. The phase field method does not have such difficulties.

We have created a module PHASEFIELD in the programming language PYTHON. This module allows users to enter an interfacial problem that is then manipulated symbolically to recover a phase field approximation. A significant portion of this task is to classify the set of interfacial problems that we are able to regularise.

This thesis contributes the following. We define a framework which characterises a large class of interfacial problems on which a consistent regularisation can be performed. This is used to demonstrate that this classification can be used to create software that automates the procedure of producing a phase field approximation. We do this by describing examples which together show the versatility of the framework. We also investigate a fracture model where we demonstrate a natural way to
enforce the irreversibility of the crack surface based on a variational inequality.

The phase field methodology makes use of structures from applied and pure mathematics, as well as from computer science. To understand the aim of this thesis it is required to see how these different topics fit together and can be thought of in a unified framework. The structure that we take to link these areas together is the variational or energetic approach. We therefore briefly describe:

- $$\Gamma$$-convergence.
- Gradient flows.
- Formal and rigorous asymptotics.

The roots of the phase field approximation come from $$\Gamma$$-convergence [19]. This is an appropriate notion of convergence for functionals and ensures minimisers of the phase field energy converge to minimisers of the sharp energy. Under the right scaling the Ginzburg-Landau energy, the Left Hand Side (LHS) of equation Eq. (1.1), $$\Gamma$$-converges to the perimeter functional the Right Hand Side (RHS) of Eq. (1.1) [58] [68],

$$\int_{\Omega} \frac{\epsilon}{2} |\nabla \hat{\phi}|^2 + \frac{1}{\epsilon} W(\hat{\phi}) \xrightarrow{\epsilon \to 0} \int_{\Gamma}. \tag{1.1}$$

Here $$\Omega$$ is a 2 dimensional domain, $$\Gamma$$ is a 1 dimensional hypersurface, $$\epsilon$$ is a small parameter proportional to the width of the interface and $$W$$ is a double potential well with global minima at 0 and 1.

An appropriate notion of $$\hat{\phi}$$ converging to $$\Gamma$$ has to be made. In [61] this is taken to mean that the Hausdorff distance between $$\Gamma$$ and an appropriate level set of $$\hat{\phi}$$ converges.
The terms in Eq. (1.1) can be thought of as the following. The gradient term $|\nabla \hat{\phi}|^2$ has a smoothing effect on $\hat{\phi}$. Conversely, the double well term ensures that values not equal to 0 or 1 are penalised. The balance between these terms is what gives the effect of a “blurry” interface. Figure 1.3 shows the effect of varying $\epsilon$ on the shape of a smooth double well. Decreasing $\epsilon$ has a proportional effect on the strength of the well and makes it more energetically favourable to take values at the global minima. A closely related concept is to what is known as an obstacle potential [12] [13]. Obstacle potentials ensure that the value of $\hat{\phi}$ is identically $\pm 1$ in the bulk domains $\Omega^1, \Omega^2$. An example of an obstacle type potential is given by

$$W(r) = \begin{cases} \frac{1}{4}r(1-r) & \text{if } 0 < r < 1, \\ \infty & \text{otherwise.} \end{cases}$$

$\Gamma$ convergence was the beginning of a rigorous justification of using phase fields as a regularisation method. If one takes the variation with respect to the arc length of the perimeter functional and sets this equal to the normal velocity of the interface, mean curvature flow is recovered. A natural question asked by De Giorgi [30] was if gradient flows are taken of two functionals that $\Gamma$-converge to each other, does this imply the flows converge to each other? The answer in general is no, and there are well known finite dimensional counter-examples [65]. Despite this, there are many cases where this does hold. Furthermore, it is possible to identify conditions where if true, the $\Gamma$-convergence of the underlying functional implies the $\Gamma$-convergence of the gradient flow [65]. There are a number of examples where these conditions are shown to hold, the canonical example being the Allen-Cahn equation [25]. This is the $L^2$ gradient flow of the Ginzburg-Landau energy. With a smooth potential this was shown to converge to mean curvature flow via formal asymptotics [23] [64] and rigorously [31] [70]. Convergence with a obstacle potential was shown in [27]. The Cahn-Hilliard equation [24] which is known to approximate the Mullins-Sekerka type problems in the limit has a similar gradient flow structure and can be described as an gradient $H^{-1}$ flow. It is for this reason that we assume that the class of interface problems to be approximated must have a gradient flow structure.

Gradient flows of phase field functionals known to $\Gamma$-converge are used as a foundation to approximate more complicated problems. We think of the phase field functional as “generating” the regularised interface. The gradient flow is coupled to balance laws which hold off the interface. The balance laws are generally a variant of a diffusion equation. For example, in dendritic growth representing the diffusion of heat [41], tumour growth the diffusion of nutrients [43] and crack propagation the
Figure 1.2: Relationships between different equations in a consistent phase field calculus.

Gradient flows coupled with balance laws and their regularisations are best understood within the framework of phase field calculus [36]. This describes the way in which operations on the sharp equations have an analogue for the phase field equations, meaning they invert back to correct sharp equation in the limit $\epsilon \to 0$. In [36] only variations of energies were considered. In this thesis, we extend this to include coupling to bulk equations, in addition to the gradient flow of the energy, as opposed to just the stationary problem. Our extension of the phase field calculus can be seen in Figure 1.2. Previously in [36] only the left four boxes were considered. $\mathcal{F}^\epsilon$ is a phase field functional while $\mathcal{F}^s$ is its sharp analogue. The figure shows that if a phase field functional $\mathcal{F}^\epsilon$ converges to a sharp energy $\mathcal{F}^s$, we may expect the gradients to converge. Going further right we may expect gradient flows to also converge where $\vec{V}_t$ is a time dependent velocity field on $\Omega$. Finally, on the far right the flow of the energy are coupled to abstract balance laws $J$ supposed to hold off the interface for a field $\tilde{u}$. $\tilde{\chi}_G$ is a characteristic function of the splitting of $\Omega$ into different phases, see Eq. (2.1) for a precise definition. Notice when transferring to the phase field setting we formally replace $\tilde{\chi}_G$ with $\hat{\phi}$. This is a notion that is made more precise in Algorithm 2 in Section 3.2.2.

In addition to the Ginzburg Landau energy we also base our approximations on a variation of the Mumford-Shah functional introduced in [14]. This is similar to the Ginzburg-Landau energy, but uses a single instead of double well. It also contains a bulk term which is a function of $\nabla \tilde{u}$. The functional is given by

$$
\mathcal{F}(\Gamma, \nabla \tilde{u}) = \int_{\Omega \setminus \Gamma} a_1(\nabla \tilde{u}) + \int_{\Gamma} ,
$$  

(1.2)
and its elliptic approximation shown to Γ converge to [7] [14]

\[ \mathcal{F}^\epsilon(\hat{\phi}, \tilde{u}) = \int_\Omega (\hat{\phi}^2 + k_\epsilon a_1(\nabla \tilde{u}) + \frac{1}{2\epsilon}(1 - \hat{\phi})^2 + \epsilon |\nabla \hat{\phi}|^2. \]  

(1.3)

The small parameter \( k_\epsilon \) is of higher order in \( \epsilon \). Functionals of this type originate from image recognition, we use it as a regularisation of crack propagation. We build our classification of phase field models drawing on functionals whose Γ-convergence can be shown for the respective phase field functional.

### 1.1.1 Gradient flow of the perimeter

The canonical example of a gradient flow that fits into our framework is the convergence of the Ginzburg-Landau energy to the perimeter functions Eq. (1.1). This translates to the convergence of the Allen-Cahn equation to mean curvature flow [26, 31, 40, 11]. In the following we show the variations of each respective energy and derive the resulting gradient flow equations. We first calculate the variation of the
Ginzburg-Landau energy in the direction $\eta \in C^\infty_0(\Omega)$.

$$\left. \frac{d}{d\xi} F^\epsilon(\hat{\phi} + \xi \eta) \right|_{\xi=0} = \frac{d}{d\xi} \int_\Omega \frac{\epsilon}{2} |\nabla (\hat{\phi} + \xi \eta)|^2 + \frac{1}{\epsilon} W(\hat{\phi} + \xi \eta) \bigg|_{\xi=0}$$

$$= \int_\Omega \epsilon \nabla (\hat{\phi} + \xi \eta) \cdot \nabla \eta + \frac{1}{\epsilon} W'(\hat{\phi} + \xi \eta) \eta \bigg|_{\xi=0}$$

$$= \int_\Omega \left[ -\epsilon \Delta \hat{\phi} + \frac{1}{\epsilon} W'(\hat{\phi}) \right] \eta,$$

$$\left. \frac{d}{d\xi} \right|_{\xi=0} = \int_\Omega \nabla F^\epsilon(\hat{\phi} + \xi \eta) \cdot \eta = \left( \frac{d}{d\xi} F^\epsilon, \eta \right)_{L^2(\Omega)}.$$  (1.4)

Here $(\cdot, \cdot)_{L^2(\Omega)}$ is the usual $L^2$ inner produce with weight $\epsilon$. Taking the dual element under this inner produce gives the gradient $\nabla F^\epsilon$. Setting $\nabla F^\epsilon$ equal to $\partial_t \hat{\phi}$, we get the Allen-Cahn equation with time scaling $\epsilon$

$$\partial_t \hat{\phi} = -\nabla F^\epsilon,$$

or equivalently

$$\epsilon \partial_t \hat{\phi} = \epsilon \Delta \hat{\phi} - \frac{1}{\epsilon} W'(\hat{\phi}).$$  (1.5)

**Remark 1.1.1.** When an obstacle potential is used Eq. (1.6) will instead be parabolic variational inequality [12] [13].

We let our energy $F^s$ be given by the perimeter functional where $\Gamma$ is a closed 1 dimensional surface in a domain $\Omega$ that does not intersect the boundary $\partial \Omega$.

$$F^s = \int_\Gamma.$$

Using transport identities see e.g. [32] it can be shown the variation with respect to arc length is given by

$$\frac{d}{d\xi} F^s = \int_\Gamma \vec{\kappa} \cdot \eta,$$

$$\left. \frac{d}{d\xi} \right|_{\xi=0} = \left( \frac{d}{d\xi} F^s, \eta \right)_{L^2(\Gamma)}.$$

where $(\cdot, \cdot)_{L^2(\Gamma)}$ is the standard $L^2$ product on $\Gamma$. The gradient $\nabla F^s$ is the dual element under this inner product. Setting the normal velocity $\vec{V}$ equal to the gradient,
we recover mean curvature flow
\[
\vec{V} = -\nabla F^s,
\]
\[
= -\vec{\kappa}.
\]

Our convention for \(\vec{\kappa}\) is that it is positive for the sphere and \(\vec{\kappa} = \kappa \vec{\nu}\) where \(\vec{\nu}\) is the outward pointing unit normal. Notice that in each case we needed a inner product to define the gradient. In the first a weighted \(L^2\) product on \(\Omega\) and in the second case the standard \(L^2\) product on \(\Gamma\). We say that the \(L^2(\Omega)\) is the correct phase field inner product to take in connection with the \(L^2(\Gamma)\) inner product in the sharp setting. If we had chosen different inner products we would have produced different evolutions that may not converge. Other common gradient flows are Mullins-Sekerka type problems and their phase field approximation, the Cahn-Hilliard equation. The sharp equations are given by

\[
\Delta u = 0, \quad \text{in } \Omega \setminus \Gamma, \tag{1.7}
\]
\[
u = k, \quad \text{on } \Gamma,
\]
\[
[u]_2^1 = 0, \quad \text{on } \Gamma.
\]

Square brackets \([\cdot]\) denote the jump of a quantity over the interface. See Eq. (2.9) for a precise definition. If one instead considers a \(H^{-1}\) inner product, Eq. (1.7) is in fact a gradient flow of the perimeter function. Similarly, a \(H^{-1}\) flow of the Ginzburg-Landau energy will produce a convergent phase field approximation.

Some systems are naturally written under different inner products however from a classification perspective all gradient flows considered in this thesis will be \(L^2\).

1.1.2 Stationary solutions

Two Phase

An insight into the behaviour of \(\dot{\phi}\) can be discovered by studying the stationary solutions of Eq. (1.6). Given a 1 dimensional domain \(\Omega\) and \(W(\dot{\phi}) = 9\dot{\phi}^2(1 - \dot{\phi})^2\), we get the stationary problem

\[
\epsilon \phi''(x) - \frac{9}{\epsilon} \phi(x)(1 - \phi(x))(1 - 2(\phi(x))) = 0. \tag{1.8}
\]
A solution of Eq. (1.8) is given by

$$\phi(x) = \frac{1}{2} \left[ \tanh \left( \frac{x}{\sqrt{2} \epsilon} \right) + 1 \right].$$

(1.9)

This is known as the fundamental solution of the Allen-Cahn equation which is plotted in Figure 1.4. In the regions away from the interface, $\phi$ is constant and at $x = 0$, there is a transition region of order $\epsilon$, where one phase smoothly transitions into the other.

**One Phase**

While we also consider multi-phase energies, the profiles between two neighbouring phases will always look of the form of Figure 1.4. We next discuss the stationary profile of a one-phase energy which is used in material science to model crack propagation. For the energy Eq. (1.2), negating elastic effects for the 1 dimensional stationary profile we have [54]

$$2\epsilon \phi''(x) - \frac{1}{2\epsilon} (\phi - 1) = 0.$$  

(1.10)

For a bar of length $2L$, for $-L \leq x \leq L$, the solution with the crack in the centre at $x = 0$ is given by

$$\phi(x) = \begin{cases} 1 - \cosh \left( \frac{x}{2L} \right) + \coth \left( \frac{L}{2\epsilon} \right) \sinh \left( \frac{x}{2\epsilon} \right) & x \geq 0, \\ 1 - \cosh \left( \frac{x}{2L} \right) - \coth \left( \frac{L}{2\epsilon} \right) \sinh \left( \frac{x}{2\epsilon} \right) & x < 0. \end{cases}$$

(1.11)
This is shown in Figure 1.5. In the limit $\epsilon \to 0$ $\phi$ becomes discontinuous. We can recover the crack energy by inserting Eq. (1.11) into the free energy

$$F^\epsilon = \int_{-L}^{L} \left( \frac{1 - \phi(x)}{4\epsilon} + \epsilon (\phi'(x))^2 \right) dx,$$

$$= \frac{1}{2} \left( \coth \left( \frac{L}{2\epsilon} \right) + \coth \left( \frac{L}{2\epsilon} \right) \right) \xrightarrow{\epsilon \to 0} 1.$$

### 1.1.3 Asymptotic and rigorous convergence

As previously stated knowing two functionals $\Gamma$ converge to one another is not enough to conclude that the respective flows also converge. The first steps to identifying the nature of convergence is usually done using formal asymptotics. This approach is formal in the sense that on its own it is not enough to conclude that the flows converge. The power in formal asymptotics lies in being able to identify the limiting problem and the order at which terms affect the limit equation.

It is assumed that the solution can be split up into two regimes. One close to the interface (inner solution) and one far away from the interface (outer solution). The intuition is that far from the interface the solution should be independent of $\epsilon$, while close there should be a lot more variability. Figure 1.4 shows where the various expansions are deemed to be valid; in a middle region both are expected to be valid and matching is done between terms to identify the limiting problem.
1.2 Symbolic computation and software

In this thesis when referring to a piece of software we typeset it in lower capitals for example PHASEFIELD. When presenting code or referring to an identifier contained within PHASEFIELD we typeset in a verbatim environment. For example for the function Implicit which is part of the software PHASEFIELD.

A major contribution of this thesis is the PYTHON module PHASEFIELD. We discuss the aims and problems from a scientific computing perspective when dealing with Partial Differential Equation (PDE) software.

The aim of any software or computer program generically is to take a set of tasks the user would like to accomplish and automate them. Programming languages do this by identifying particular common tasks that need to be accomplished and frame them in a way that makes it easy for the human mind to understand. This enables users to minimise code repetition and to accomplish tasks that previously may have needed many times the amount of effort and code.

For PDE software an important question is how the user inputs an equation. This should be made as straightforward as possible. As an example take a standard elliptic problem. In principle, after defining the continuous weak form, and finite element space, the fully discrete form is completely fixed. A software interface should be motivated by this. To attack the problem of allowing users in different packages to enter PDEs in a unified and consistent way the Unified Form Language (UFL) was created[4] [55].

UFL is a domain specific language that closely resembles the weak forms that one may write down on paper. It is noted that modern programming techniques naturally allow PDEs to be symbolically represented; while a computer can never truly store a continuous object. The problem may be approached from a different

\[ \Omega_1(t) \quad \Gamma(t) \quad \Omega_2(t) \]

\[ \Omega_1(t) \quad \Omega_2(t) \]

\[ 2^\Omega \quad e^{2\pi} \]

Figure 1.6: Regions in \( \Omega \) where inner and outer expansions are valid.
angle by defining which operations are allowed to be done on types. In this sense, while PDEs cannot be directly stored, symbolic expressions representing separate parts of PDEs may be identified with types. An algebra on these symbols may consistently be defined by overloading operators and functions.

We have created a Python module named PHASEFIELD. A phase field model can be used to approximate some given problem in the limit. Often this procedure of going from the sharp-model to the phase field one is mechanical and can/should be done by an algorithm. This is what we provide. Finite-element modules give users who are not experts in finite-elements the capability to compute solutions using a finite element approximation. We provide users who are not familiar with the phase field methodology the opportunity to compute and compare solutions using a phase field approximation. Like any software, while we aim to make it easy to use, we also offer enough functionality to experts wishing to implement advanced features.

One of the first tasks when building PHASEFIELD was to classify the class of equations that it was able to solve. This classification is not a trivial task and mathematically requires looking for a “good” generic structure behind the phase field equations. For this, we have chosen a type of gradient flow coupled to a distributional equation.

This thesis is structured in a way that mirrors the structure of PHASEFIELD. In Figure 1.7 is a UML diagram of PHASEFIELD, we describe Figure 1.7 in relation to the structure of the thesis. In Chapter 2 we describe an abstract framework for interfacial problems that defines the way in which function and parameters are passed into PHASEFIELD. These interfacial problems in PHASEFIELD are completely defined in what we call a Sharp class. In Chapter 3 we describe the phase field methodology and how to translate the previously defined problems into phase field equations. In PHASEFIELD this is done by creating a PhaseModel class which takes the previously defined Sharp class and produces the regularised phase field equations. By default, the UFL form produced is a backward Euler scheme. The time stepping in Chapter 4 describes the ways in which to modify the PhaseModel class to implement different time discretisation. Up to and including Chapter 4 all the code presented is completely independent of any backend finite element package. This is to ensure that, if one wishes to use another finite element package for the computation, while retaining the front end and manipulation of sharp equations to phase field, this is indeed possible. In Chapter 5 we describe the PhaseStepper class which relies on a small number of methods from an external Finite Element (FEM) backend which are clearly identified. Finally, in Chapter 6, we perform some numerical experiments using the Distributed Unified Numerics Environment (DUNE) as a FEM backend to
verify that PhaseField works as intended.

1.3 Thesis contributions and structure

1.3.1 Mathematical framework

We classify a large number of phase field models in a single unified framework. At its heart, this is an $L^2$ gradient flow of functionals that $\Gamma$ converge to each other. This framework makes it clear the steps needed to go from sharp to phase field equations. We present a number of examples in this direction to showcase the versatility and generality of the framework. These flows are coupled to distributional equations which model the physical phenomena in the bulk domain.

1.3.2 Software

A Python module PhaseField has been created. PhaseField can be used for two purposes. The first is the manipulation of sharp interfacial problems into phase field equation. We remark that numerical analysis is still necessary to establish the
relationship between the discretisation parameters to get a solution that approximates the interfacial problem well. The second is the solution of these phase field equations using DUNEas a back end. Substantial effort has gone into making the abstract framework and PHASEFIELD interface look as similar as possible.

PHASEFIELD offers significant flexibility when solving the equations. Adaptivity is offered by default. This is extremely useful when solving phase field equations as much of the important behaviour happens in a small band around the interface. Different time discretisations are also easy to implement for both balance and phase field equations.

1.3.3 Fracture model

We show how a fracture model fits into our framework. While the asymptotics have been computed for this model, we are unaware of any work that explicitly realises the relationship between the phase field and the sharp interface models as corresponding $L^2$ flows in the sense of phase field calculus shown in Figure 1.2. We give a simple proof of the variation of the sharp functional by using integration by parts and transport theorems from the fracture literature.

When solving these equations numerically, often an ad-hoc or a posterior method has to be used to enforce the irreversibility conditions [54]. We solve the equations instead with a Truncated Non-Smooth Newton Multi-Grid (TNNMG) [49] solver which is contained within the framework. Additionally, this same solver can be used to solve the Allen-Cahn equation with obstacle potential.
Chapter 2

Sharp Model Framework

In this chapter we define a class of sharp interfacial problems. This is done by introducing a formal gradient flow structure. We do this classification not for its own sake but with the goal to identify the class of problems that PhaseField is able to regularise. We are often vague with smoothness assumptions as we are primarily interested with the symbolic representation of the PDE system, and not in well-posedness issues.

We start in Section 2.1 by describing the geometric setup for the class of problems by defining what we call a regular configuration. To produce an evolution, the regular configuration is deformed by a time dependent vector field described in Section 2.1.1. We define balance laws in Section 2.1.3, these are PDEs that must hold in the bulk domains. In Section 2.1.4 we introduce an energy and calculate the variation. This allows us to define evolutions laws for the regular configuration. The full gradient flow is then defined in Section 2.1.5. Finally, we give examples that fit into the framework in Section 2.2. These examples are taken from a number of areas from within applied mathematics. Code snippets for how to specify the examples in PhaseField are provided.

2.1 Sharp evolutions

Evolutions are built upon what we call a regular configuration. This is a partition of a domain in a specific sense which we deform over time to produce an evolution. In the following $\Omega$ denotes a bounded square domain $\Omega \subset \mathbb{R}^2$, and $\partial\nu_{\partial\Omega}$ denote the outward pointing unit normal to $\Omega$.

**Definition 2.1.1.** A regular configuration $G := \{\Omega^i : 1 \leq i \leq m\}$ is a set such that the following hold:
• \( \Omega \) is a partition of 2 dimensional subsets \( \Omega^i \subset \mathbb{R}^2 \) such that \( \bigcup_{i=1}^m \Omega^i = \bar{\Omega} \) where \( m \in \mathbb{N} \).

• The subsets are non-overlapping \( \Omega^i \cap \Omega^j = \emptyset, \ 1 \leq i < j \leq m \).

• The boundaries \( \partial \Omega^i, \ 1 \leq i \leq m \) are Lipschitz and piecewise differentiable.

A bar above \( \Omega^i \) denotes the closure of the set while \( \partial \Omega^i \) is the boundary.

We denote by \( \mathcal{G} \) the set of all regular configurations. A vector valued characteristic function \( \hat{\chi}_G \) can be used to specify a regular configuration which is defined as

\[
\chi_{\Omega^i}(\vec{x}) = \begin{cases} 
1 & \text{if } \vec{x} \in \Omega^i, \\
0 & \text{otherwise.}
\end{cases}
\tag{2.1}
\]

(\( \hat{\chi}_G \))\(_i := \chi_{\Omega^i}, \ 1 \leq i \leq m \).

We build a hierarchy of hypersurfaces from a regular configuration as follows. Given a regular configuration \( G \in \mathcal{G} \) for \( d \in [0, 1] \),

\[
\mathcal{M}^d_G := G = \{ \Omega^i : 1 \leq i \leq m \},
\]

\[
m^d_G := \{ \partial \omega^1 \cap \partial \omega^2 : \omega^1, \omega^2 \in \mathcal{M}^{d+1}_G, \omega^1 \neq \omega^2, \mathcal{H}^d(\partial \omega^1 \cap \partial \omega^2) > 0 \},
\]

\[
\mathcal{M}^d_G := m^d_G \cup \{ \partial \omega \setminus (\partial \Omega \cup \bigcup_{k \in m^d_G} k) : \omega \in \mathcal{M}^{d+1}_G, \mathcal{H}^d(\partial \omega \setminus (\partial \Omega \cup \bigcup_{k \in m^d_G} k)) > 0 \}.
\]

Here and in the following we denote the \( d \) dimensional Hausdorff and Lebesgue measures respectively as \( \mathcal{H}^d \) and \( \mathcal{L}^d \). The set \( m^d_G \) consists of all intersecting boundaries.
that have positive $d$ dimensional Hausdorff measure. The set $M^d_G$, additionally includes the boundaries of subsets which are contained in the interior of the subset; see for example the $m = 1$ case in Figure 2.1.

**Definition 2.1.2.** We call an element of $M^d_G$ a $(d)$ hypersurface.

The $(1)$ hypersurface that separates $\Omega^i$ and $\Omega^j$ is denoted

$$
\Gamma^{ij} = \left\{ \partial \Omega^i \cap \partial \Omega^j : \Omega^i, \Omega^j \in G, \Omega^i \neq \Omega^j, \mathcal{H}^1(\partial \Omega^i \cap \partial \Omega^j) > 0 \right\} \in m^d_G.
$$

Each $\Gamma^{ij}$ comes with a unit normal $\vec{\nu}_{ij}$ that is oriented to be outward pointing from $\Omega^i$ into $\Omega^j$. It also comes with an outer co normal $\vec{\mu}_{ij}$ of $\Gamma^{ij}$ on $\partial \Gamma^{ij}$; i.e. $\vec{\mu}_{ij}$ is tangential to $\Gamma^{ij}$ and normal to $\partial \Gamma^{ij}$. We also let $\Gamma^{ii} \in M^1_G \setminus m^1_G$ which also comes with a normal field $\vec{\nu}_{ii}$.

$$
\Gamma^{ii} = \left\{ \partial \Omega^i \setminus (\partial \Omega \cup \bigcup_{k \in m^d_G} k) : \mathcal{H}^1(\partial \Omega^i \setminus (\partial \Omega \cup \bigcup_{k \in m^1_G} k)) > 0 \right\}.
$$

Often we deal with what we call “cracks” which are $(1)$ hypersurfaces contained on the interior of one of the $(2)$ hypersurfaces. We formalise this with the following definition.

**Definition 2.1.3.** A crack is a $(1)$ hypersurface $\Gamma^{ii}$.

A selection of regular configurations can be seen in Figure 2.1. In the first image we have a standard setup for fracture which contains a $(1)$ hypersurface (see Definition 2.1.2). Here, $m^1_G$ is empty as the only hypersurface is on the interior of $\Omega^1$. The second image is standard for a number of problems in which there is some mass in the middle of the domain which is evolving. For example, dendritic and tumour growth. There is a closed $(1)$ hypersurface in $m^1_G$ which partitions the domains $\Omega^1$ and $\Omega^2$. In the third image there is a triple junction that is seen in multi-phase simulations. There are three $(1)$ hypersurfaces all contained in $m^1_G$. A further more complicated example of $M^d_G$ for a given regular configuration can be see in Figure 2.2.

### 2.1.1 Evolutions of regular configurations

Time-dependent vector fields (i.e. velocities) are used to define evolving configurations. The latter are used later to define gradient flows of energies defined on such configurations. In general, the outcomes of such gradient flows are laws for the velocities of points on some parts of the evolving configuration. Whether the obtained
set of equations is a well-posed problem is a separate analytical question that we do not address in this work. For the interfacial problems that we derive within this framework there exist some analytical results, see Section 2.2 for further details.

Remark 2.1.4. Let us remark here that the velocity field is not associated with any transport of material. For instance, in the tumour grown model that we derive (see Section 2.2.4) the boundary between different types of tissue move, but there is no transport of nutrition in the adjacent domains other than by diffusion.

In future work, fluid flow in the bulk domains may be considered, then the notion of material transport with a velocity field on the whole domain $\Omega$ will not be abstract. Instead, one expects to obtain PDEs for the material transport from some gradient flow or otherwise different dynamics [37].

Given a $G \in \mathcal{G}$ we evolve $G$ by deforming the sets $\Omega^i \in G$ by smooth vector fields defined on $\Omega$ that do not deform the boundary $\partial \Omega$.

\[ D := \{ \vec{F} : \vec{F} \in C^\infty(\Omega, \mathbb{R}^2), \vec{F} \cdot \nu_{\partial \Omega} = 0 \}. \]  \hspace{1cm} (2.4)

Let $\vec{V} : [0,T] \rightarrow D$ be a smoothly evolving vector field. For $\vec{x}_I \in \Omega$ let $\vec{x}(t, \vec{x}_I, \vec{V})$, be the solution of

\[ \frac{d\vec{x}(t)}{dt} = \vec{V}(\vec{x}(t)), \quad \vec{x}(0) = \vec{x}_I. \] \hspace{1cm} (2.5)

This equation describes the evolution of a point $\vec{x}_I \in \Omega$ under a time-dependent vector field on $\Omega$. Given an initial regular configuration $G := \{ \Omega^i, i = 1, \ldots, m \} \in \mathcal{G}$ The sets $\Omega^i(t)$ are defined as

\[ \Omega^i(t) = \{ \vec{y} \in \Omega : \exists \vec{x}_I \in \Omega^i s.t. \vec{y} = \vec{x}(t, \vec{x}_I, \vec{V}) \}. \]

Figure 2.2: For fixed $G$ visualisation of (d) hypersurfaces, $d = 2, 1, 0$. 

}\[ \Omega^2 \quad \Omega^3 \]

\[ \mathcal{M}^2_G. \quad \mathcal{M}^1_G. \quad \mathcal{M}^0_G. \]
As \( \vec{V} \) is smooth the solution of Eq. (2.5) defines smoothly evolving points [34]. The set \( \{ \Omega^i(t), i = 1, \ldots, m \} \) defines an evolving regular configuration which we denote \( G(t, \vec{V}) := \{ \Omega^i(t), i = 1, \ldots, m \} \). Given \( \vec{V} \) and \( G \) we often write as shorthand

\[
G(t) := G(t, \vec{V}). \tag{2.6}
\]

We say that Eq. (2.5) describes the abstract evolution

\[
\partial_t G(t) = \vec{V}, \text{ where } \vec{V} \in \mathcal{D}. \tag{2.7}
\]

\( \partial_t G(t) \) is the velocity \( \partial_t \vec{x}(t) \) of points \( \vec{x}(t) \in \omega(t) \in M^d_G(t), d = 0, 1, 2 \).

**Remark 2.1.5.** It can be shown that for short times given a sufficiently smooth \( \vec{V} \) and \( G \) the evolution of \( \Gamma(t) \) is also smooth [66].

**Example 2.1.6.** The following example demonstrates an evolution which is a shrinking circle travelling inwards. We let \( \Omega \) be the domain \([-1, 1] \times [-1, 1] \). Let \( \theta \in [0, 2\pi] \) be the angle counterclockwise from the positive \( x \)-axis and let \( r \) be the distance from the origin.

\[
\begin{align*}
\Omega &:= [-1, 1] \times [-1, 1], \\
\Omega^1_1 &:= \{ x, y \in \Omega : r < 1 \}, \\
\Omega^2_1 &:= \{ x, y \in \Omega : r > 1 \}, \\
G &:= \{ \Omega^1, \Omega^2 \}, \\
f(r) &:= \begin{cases} e^{r^{-1}} & \text{if } r \leq 1, \\
0 & \text{otherwise}, \end{cases} \\
\vec{V}(\theta, r) &:= (\pi + \theta, f(r)).
\end{align*}
\]

Figure 2.3 shows a graphical illustration of an evolving regular configuration under these dynamics. The vector field \( \vec{V} \) is constant over time and displayed in blue. The sets \( \Omega^i(t) \) are also displayed.

### 2.1.2 Notation

Before proceeding further we introduce notation for dealing with tensor expressions. Let \( \vec{x} = (\vec{x}_j)_{j=1,2}, \vec{y} = (\vec{y}_j)_{j=1,2} \) be points in the physical domain \( \mathbb{R}^2 \). Let \( \vec{f} = (\vec{f}_i)_{i=1,\ldots,m}, \vec{g} = (\vec{g}_i)_{i=1,\ldots,m} \) be points in \( \mathbb{R}^m \). Let \( X = (X_{ij})_{i=1,\ldots,m,j=1,\ldots,2}, Y = \)
(Y_{ij})_{i=1,...,m; j=1,2} be m \times n matrices. Then we use the notation

$$X_i = (X_{ij})_{j=1,2} \in \mathbb{R}^2, \quad i = 1, \ldots, m,$$

$$\bar{x} \cdot \bar{y} = \sum_{j=1}^{2} \bar{x}_j \bar{y}_j \in \mathbb{R},$$

$$\hat{f} \cdot \hat{g} = \sum_{i=1}^{2} \hat{f}_i \hat{g}_i \in \mathbb{R},$$

$$X \bar{y} = \left( \sum_{j=1}^{2} X_{ij} \bar{y}_j \right)_{i=1,\ldots,m} \in \mathbb{R}^m,$$

$$X : Y = \sum_{i,j} X_{ij} Y_{ij} \in \mathbb{R}.$$

By $\nabla \hat{f}$ we denote the spatial Jacobian of a function $\hat{f}$. The divergence $\nabla \cdot A$ of a matrix valued function $A : \Omega \rightarrow \mathbb{R}^{m \times 2}$ is defined as the divergence of each line giving an $m$-vector as a result, i.e. $\nabla \cdot A = \left( \sum_{j=1}^{2} \partial_j A_{ij} \right)_{i=1,\ldots,m}$. A dash i.e. $p'$ denotes the derivative of a function $p$ with respect to its argument.

Given an evolving regular configuration $\{G(t, \bar{V})\}_t$. Let $\partial_t^\bullet$ denote the material derivative of a scalar function defined on $\{\Gamma^{ij}(t)\}_t$. Denote the normal time derivative $\partial_t^\perp$. This is the material derivative, where only the normal contributions of the velocity are taken into account $\partial_t^\perp \eta = \partial_t \eta + (\bar{V}_t \cdot \bar{\nu}_{ij}) \frac{\partial \eta}{\partial \bar{\nu}_{ij}}$. Let $\nabla_{\Gamma}$ be the surface divergence. As a consequence of splitting of $\bar{V}$ into normal and tangential parts, we get the relation $\partial_t^\bullet \eta = \partial_t^\perp \eta + (\bar{V}_t \cdot \bar{\mu}_{ij}) \bar{\mu}_{ij} \cdot \nabla_{\Gamma} \eta$. The mean curvature of $\Gamma^{ij}$ defined by $\kappa_{ij} = -\nabla_{\Gamma} \cdot \bar{\nu}_{ij}$ and its vector valued analogue $\bar{\kappa}_{ij} := \bar{\nu}_{ij} \kappa_{ij}$.

Letting $\delta_{\alpha\beta}$ be the usual Kronecker delta, we denote the basis functions

$$e_\beta = (\delta_{\alpha\beta})_{\alpha=1}^m \in \mathbb{R}^m, \quad (2.8)$$
which is a vector of length \( m \) with a 1 in position \( \beta \) and 0 otherwise.

In fracture mechanics we often have a bulk field which is singular on \( \partial \Gamma^{ii} \). The following \( \Xi \in \mathbb{R}^+ \)-dependent control volume, called the tip disc, is used to isolate these points,

\[
D_\Xi(\vec{z}) = \{ \vec{x} \in \Omega : |\vec{x} - \vec{z}| \leq \Xi \}.
\]

When \( D_\Xi \) is integrated over and \( \Xi \) sent to zero, we call the resulting expression a tip integral. For a scalar field \( \Phi : \Omega \to \mathbb{R} \) we define

\[
\oint_{\text{tip}(\vec{z})} \Phi \vec{v}_{\text{tip}} \ d\mathcal{H}^1 := \lim_{\Xi \to 0} \int_{\partial D_\Xi(\vec{z})} \Phi \vec{v}_{\partial D_\Xi(\vec{z})} \ d\mathcal{H}^1.
\]

Denote the jump across an (1) hypersurface \( \Gamma^{ij} \) as

\[
[\Phi]^{ij}_{ij} := \lim_{\Xi \to 0} \Phi(\vec{x} + \Xi \vec{v}_{ij}) - \Phi(\vec{x} - \Xi \vec{v}_{ij}).
\] (2.9)

We make use of the following transport formula for computing variations. When dealing with evolving hypersurfaces \( \{\Gamma^{ij}(t)\}_t \) we will often omit the dependence on \( t \), since it is clear from the context whether we are dealing with an evolving hypersurface or hypersurface at a specific time.

**Lemma 2.1.7** (Lemma 2.1 [39]). Let \( \{G(t, \vec{V})\}_t \subset \mathcal{G} \) be an evolving regular configuration. Let \( \Phi \) be a scalar field on \( \{\Gamma^{ij}(t)\}_t \) such that the following integrals exist then

\[
\frac{d}{dt} \int_{\Gamma^{ij}} \Phi \ d\mathcal{H}^1 = \int_{\Gamma^{ij}} (\partial_t^* \Phi + \Phi \nabla \Gamma \cdot \vec{V}_t) \ d\mathcal{H}^1, \tag{2.10}
\]

\[
= \int_{\Gamma^{ij}} (\partial_t^\rho \Phi - \Phi(\vec{V}_t \cdot \vec{v}_{ij})\kappa_{ij}) \ d\mathcal{H}^1 + \int_{\Gamma^{ij}} \nabla \Gamma \cdot (\Phi(\vec{V}_t \cdot \vec{v}_{ij})(\vec{\mu}_{ij})) \ d\mathcal{H}^1,
\]

\[
= \int_{\Gamma^{ij}} (\partial_t^\rho \Phi - \Phi(\vec{V}_t \cdot \vec{r}_{ij})) \ d\mathcal{H}^1 + \int_{\partial\Gamma^{ij}} \Phi \vec{V}_t \cdot \vec{\mu}_{ij} \ d\mathcal{H}^0. \tag{2.11}
\]

Splitting the velocity into tangential and normal parts gives

\[
\frac{d}{dt} \int_{\Gamma^{ij}} \Phi \ d\mathcal{H}^1 = \int_{\Gamma^{ij}} (\partial_t^* \Phi + \vec{V}_t \cdot \vec{v}_{ij} \frac{\partial \Phi}{\partial \vec{v}_{ij}} - \Phi \vec{V}_t \cdot \vec{r}_{ij}) \ d\mathcal{H}^1 + \int_{\partial\Gamma^{ij}} \Phi \vec{V}_t \cdot \vec{\mu}_{ij} \ d\mathcal{H}^0. \tag{2.12}
\]
Similarly, the formula for partial integration of surfaces is
\[
\int_{\Gamma^{ij}} \nabla \Phi \, d\mathcal{H}^1 = -\int_{\Gamma^{ij}} \Phi \kappa_{ij} \vec{\nu}_{ij} \, d\mathcal{H}^1 + \int_{\partial \Gamma^{ij}} \Phi \vec{\nu}_{ij} \, d\mathcal{H}^0.
\] (2.13)

In the following when we write \(\int_{\Omega^i} \Phi \, d\mathcal{H}^2\) and \(\int_{\Gamma^{ii}} \Phi^i_{ii} \vec{\nu}_{ii} \, d\mathcal{H}^1\). By an abuse of notation this is taken to mean
\[
\int_{\Omega^i} \Phi \, d\mathcal{H}^2 \equiv \lim_{\Xi \to 0} \int_{\Omega^i \setminus \bigcup_{z \in \partial \Gamma^{ii}} D_{\Xi}(z)} \Phi \, d\mathcal{H}^2, \quad \text{(2.14)}
\]
\[
\int_{\Gamma^{ii}} \Phi^i_{ii} \vec{\nu}_{ii} \, d\mathcal{H}^1 \equiv \lim_{\Xi \to 0} \int_{\Gamma^{ii} \setminus \bigcup_{z \in \partial \Gamma^{ii}} D_{\Xi}(z)} \Phi^i_{ii} \vec{\nu}_{ii} \, d\mathcal{H}^1. \quad \text{(2.15)}
\]

The following is a generalised Leibniz formula used when computing the variation in the presence of cracks, proven in [52].

**Lemma 2.1.8.** Let \(\{G(t, \vec{V})\}_t \subset \mathcal{G}\) be an evolving regular configuration. Let \(\Phi\) a scalar field on \(\{\Omega^i(t)\}_t\) such that the following integrals exist then
\[
\frac{d}{dt} \left( \int_{\Omega^i} \Phi \, d\mathcal{H}^2 \right) = \int_{\Omega^i} \partial_t \Phi \, d\mathcal{H}^2 + \int_{\partial \Omega^i \setminus \Gamma^{ii}} \Phi \vec{V}_t \cdot \vec{\nu}_{\partial \Omega}\, d\mathcal{H}^1
- \sum_{z \in \partial \Omega^i} \Phi(\vec{V}_t \cdot \vec{\nu}_{\text{tip}(z)}) \, d\mathcal{H}^1,
\]
\[
\int_{\Omega^i} \nabla \Phi \, d\mathcal{H}^2 = \int_{\partial \Omega^i \setminus \Gamma^{ii}} \Phi \vec{\nu}_{\partial \Omega} \, d\mathcal{H}^1 - \int_{\Gamma^{ii}} \Phi^i_{ii} \vec{\nu}_{ii} \, d\mathcal{H}^1 - \sum_{z \in \partial \Gamma^{ii}} \Phi \vec{\nu}_{\text{tip}(z)} \, d\mathcal{H}^1.
\] (2.16)

We next state some definitions for a field \(\Phi\) defined on a regular configuration. We shall later utilise these to ensure we have the appropriate regularity to compute variations.

**Definition 2.1.9.** Given a regular configuration \(G \in \mathcal{G}\). We refer to a field on \(\Omega^i\) as smooth away from the tip if \(1 \leq i \leq j \leq m\)

- \(\Phi\) is smooth in \(\Omega^i\).
- If \(i \neq j\), \(\Phi\) and its derivatives exist and have limits up to \(\Gamma^{ij}\) from either side and is continuous across \(\Gamma^{ij}\).
- Away from \(z \in \partial \Gamma^{ii}\), \(\Phi\) and its derivatives exist and have limits up to \(\Gamma^{ii}\) from either side.
When we wish to apply transport and gradient theorems to domains where we have isolated singularities with tip discs, we require extra regularity which we state.

**Definition 2.1.10.** \( \Phi \) is referred to as *admissible* with respect to \( G \in \mathcal{G} \) if in addition to being smooth away from the tip for \( 1 \leq i \leq m \).

- \( \Phi \) is integrable on \( \Omega \).
- \( [\Phi]_i \nu_i \) is integrable on \( \Gamma^i \).
- \( \oint_{\text{tip}} (\vec{z}) \Phi \nu_{\text{tip}} \) exists \( \forall \vec{z} \in \partial \Gamma^i \).

### 2.1.3 Balance laws

In applications, the evolution of configurations are coupled with fields that are present in the bulk domains \( \Omega^i(t) \). These fields are subject to partial differential equations and boundary conditions that typically emerge from balances for energy and masses of components. These laws can be written in a distributional form which is beneficial when transferring the model using the phase field methodology.

These fields are denoted \( \tilde{u} : \Omega \times [0, T] \to \mathbb{R}^r \) and are assumed to be admissible in the sense of Definition 2.1.10 with respect to an evolving regular configuration \( \{G(t)\}_t \). The field \( \tilde{u} \) represents quantities such as the temperature in dendritic growth, deformation in elasticity or nutrient in tumour growth.

For \( i = 1, \ldots, m \) let \( E^i = (E^i_1, \ldots, E^i_r) \in C^\infty(\mathbb{R}^r, \mathbb{R}^r) \) denote a set of \( r \) scalar fields. Similarly for \( Q \) and \( F \), by \( Q^i = (Q^i_1, \ldots, Q^i_r) \in C^\infty(\mathbb{R}^r \times \mathbb{R}^{r \times 2}, \mathbb{R}^r \times 2) \) we denote the corresponding fluxes and by \( F^i = (F^i_1, \ldots, F^i_r) \in C^\infty(\mathbb{R}^r \times \Omega, \mathbb{R}^r) \) the source terms. We write

\[
E = (E^1, \ldots, E^m) \in C^\infty(\mathbb{R}^r, \mathbb{R}^{r \times m}), \tag{2.18}
\]

to clarify the relation between \( E \) and \( E^i \). Similarly for \( Q \) and \( F \)

\[
Q = (Q^1, \ldots, Q^m) \in C^\infty(\mathbb{R}^r \times \mathbb{R}^{r \times 2}, \mathbb{R}^{r \times 2 \times m}), \tag{2.19}
\]
\[
F = (F^1, \ldots, F^m) \in C^\infty(\mathbb{R}^r \times \Omega, \mathbb{R}^{r \times m}). \tag{2.20}
\]
Given these functions we define the following distributions.

\[ E_{G(t)}(\tilde{u}(x, t), \bar{x}, t) := \sum_{i=1}^{m} E(\tilde{u}(x, t))^i \chi_{\Omega(t)}(\bar{x}), \quad (2.21) \]

\[ Q_{G(t)}(\tilde{u}(x, t), \nabla \tilde{u}(x, t), \bar{x}, t) := \sum_{i=1}^{m} Q(\tilde{u}(x, t), \nabla \tilde{u}(x, t))^i \chi_{\Omega(t)}(\bar{x}), \quad (2.22) \]

\[ F_{G(t)}(\tilde{u}(x, t), \bar{x}, t) := \sum_{i=1}^{m} F(\tilde{u}(x, t), \bar{x})^i \chi_{\Omega(t)}(\bar{x}), \quad (2.23) \]

We write a balance law in terms of \( E, Q, F \) in both distributional Eq. (2.24) and strong form Eqs. (2.25) to (2.27). The link between the two forms is given by the following theorem, which is similar to [5][Theorem 2.4]. For ease of presentation we drop the arugments of \( E, Q \) and \( F \).

**Theorem 2.1.11.** Let \( \{G(t, \bar{V})\}_{t \in [0, T]} \subset \mathcal{G} \) be an evolving regular configuration. Let \( E, Q, F \) be as in Eqs. (2.18) to (2.20). The following are equivalent:

1. Distributional formulation: \( \forall \tilde{\zeta} \in C^\infty_0(\Omega \times [0, T])^r \),

\[ 0 = \int_0^T \int_{\Omega} E_{G(t)} \cdot \partial_t \tilde{\zeta} + Q_{G(t)} \cdot \nabla \tilde{\zeta} + F_{G(t)} \cdot \tilde{\zeta} \, d\mathcal{H}^2 \, d\mathcal{L}^1, \quad (2.24) \]

2. Strong formulation, the following hold a.e \( t \in [0, T] \), we have dropped the arugments in the \( E, Q \) and \( F \),

\[ \partial_t E_i^j + \nabla \cdot Q_i^j = F_i^j \quad \text{on} \ \Omega(t) \in \mathcal{M}^2_G(t), \quad (2.25) \]

\[ [Q_i^j]_i \cdot \bar{v}_{ij} - [E_i^j]_i (\bar{v}_{ij} \cdot \bar{V}_i) = 0 \quad \text{on} \ \Gamma^{ij}(t) \in \mathcal{M}^1_G(t), \quad (2.26) \]

\[ \int_{\partial \Gamma_{ij}} (Q_i^j - E_i^j \cdot \bar{V}_i) \cdot \bar{v}_{ijp} \, d\mathcal{H}^1 = 0 \quad \text{on} \ \partial \Gamma^{ij}(t) \in \mathcal{M}^0_G(t). \quad (2.27) \]

**Proof.** We show the result for the case \( r = 1 \) as the multi-component case follows directly from this. Rewrite Eq. (2.24) using Eq. (2.1)

\[ 0 = \sum_{i=1}^{m} \int_0^T \int_{\Omega} E^i \chi_{\Omega^i} \cdot \partial_t \zeta + Q^i \chi_{\Omega^i} \cdot \nabla \zeta + F^i \chi_{\Omega^i} \zeta \, d\mathcal{H}^2 \, d\mathcal{L}^1, \]

\[ = \sum_{i=1}^{m} \int_0^T \int_{\Omega^i} E^i \partial_t \zeta + Q^i \cdot \nabla \zeta + F^i \zeta \, d\mathcal{H}^2 \, d\mathcal{L}^1. \quad (2.28) \]
Apply the transport theorem Eq. (2.16) to the first term in Eq. (2.28)

\[ \sum_{i=1}^{m} \int_{0}^{T} \int_{\Omega} E(\bar{u})^i \zeta \, d\mathcal{H}^2 \, d\mathcal{L}^1 \]  

(2.29)

\[ - \sum_{i=1}^{m} \int_{0}^{T} \int_{\Omega} \partial_t E^i \zeta + Q^i \cdot \nabla \zeta + F^i \zeta \, d\mathcal{H}^2 \, d\mathcal{L}^1 \]  

(2.30)

\[ - \sum_{i,j=1}^{m} \int_{0}^{T} \int_{\Gamma} E^i \bar{V}_t \cdot \bar{v}_{ij} \zeta \, d\mathcal{H}^1 \, d\mathcal{L}^1 \]

\[ - \sum_{i=1}^{m} \int_{0}^{T} \sum_{\vec{z} \in \partial \Gamma_i} \oint_{\text{tip}(\vec{z})} E^i \bar{V}_t \cdot \bar{v}_{\text{tip}} \zeta \, d\mathcal{H}^1 \, d\mathcal{L}^1. \]

Eq. (2.29) is zero because of the boundary conditions on \( \zeta \). Apply the integration by parts formula Eq. (2.13) to Eq. (2.30).

\[ = \sum_{i=1}^{m} \int_{0}^{T} \int_{\Omega} \left[ -\partial_t E^i - \nabla \cdot Q^i + F^i \right] \zeta \, d\mathcal{H}^1 \, d\mathcal{L}^1 \]

\[ + \sum_{i,j=1}^{m} \int_{0}^{T} \int_{\Gamma} \left[ -E^i \bar{V}_t \cdot \bar{v}_{ij} + Q^i \cdot \bar{v}_{ij} \right] \zeta \, d\mathcal{H}^1 \, d\mathcal{L}^1 \]

\[ + \sum_{i=1}^{m} \int_{0}^{T} \sum_{\vec{z} \in \partial \Omega_i} \oint_{\text{tip}(\vec{z})} \left[ -E^i \bar{V}_t \cdot \bar{v}_{\text{tip}} + Q^i \cdot \bar{v}_{\text{tip}} \right] \zeta \, d\mathcal{H}^1 \, d\mathcal{L}^1. \]

Localising gives the result. The reverse implication is done by multiplying conditions Eq. (2.25) by the functions \( \tilde{\zeta} \), integrating over the respective domains, and summing.

**Boundary Conditions**

As the test function \( \tilde{\zeta} \) is zero on the boundary, we must in addition define conditions for \( \bar{u} \) on the boundary. Unless specified, we assume the no-flux boundary conditions,

\[ Q^i \cdot \bar{n}\partial \Omega = 0 \text{ on } \partial \Omega, \quad 1 \leq i \leq m. \]  

(2.31)

### 2.1.4 Energy, variation and gradient

We would like to define an evolving configuration as a gradient flow of an energy functional. As well as a regular configuration \( G \in \mathcal{G} \), the energy will also depend on the previously introduced \( r \) dimensional field \( \bar{u} \).
Energy

The energy is built from bulk and surface densities. The presence of bulk fields will be modelled by a smooth bulk free energy density. This is denoted as

\[ \hat{a} \in C^\infty(\mathbb{R}^r \times \mathbb{R}^{r \times 2}, \mathbb{R}^r), \]

and the \( i \)th component as \( \hat{a}_i \).

We next introduce the relevant functions which we use to represent surface densities. A \( m \times m \) matrix of anisotropic surface tensions is denoted \( (\gamma_{ij})_{i,j=1}^m \). The element \( \gamma_{ij} \) is the surface tension of the \((1)\) hypersurface \( \Gamma_{ij} \) see Eq. (2.2).

\[ \gamma_{ij} : \mathbb{R}^2 \to \mathbb{R}. \]

**Definition 2.1.12.** If \( \gamma_{ij}(\vec{x}) \) is constant \( \forall i, j \in [1, m] \) and \( \forall \vec{x} \in \mathbb{R}^2 \) we say that \( \gamma \) is isotropic.

We assume that the following natural symmetry condition holds

**Assumption 1.**

\[ \gamma_{ij} = \gamma_{ji}, \quad \forall 1 \leq i < j \leq m. \]

(2.33)

When \( \gamma \) is anisotropic we additionally make the following homogeneity assumption.

**Assumption 2.** \( (\gamma)_{ij} \) is positively homogeneous of degree one, i.e.

\[ \gamma_{ij}(\alpha \vec{p}) = \alpha \gamma_{ij}(\vec{p}), \quad \forall \alpha > 0, \vec{p} \in \mathbb{R}^2. \]

(2.34)

Assuming \( \gamma_{ij} \) is anisotropic we collect together some facts [35]

\[ \partial_{\vec{u}_i} \gamma_{ij}(\vec{v}_{ij}) = \partial_{\vec{v}_{ij}} \vec{v}_{ij} \cdot \gamma'_{ij}(\vec{v}_{ij}) = 0 \cdot \gamma'_{ij}(\vec{v}_{ij}) = 0, \]

(2.35)

\[ \vec{v}_{ij} \cdot \gamma'_{ij}(\vec{v}_{ij}) = \gamma_{ij}(\vec{v}_{ij}). \]

(2.36)

The free energy is given by

\[ \mathcal{F}(G, \vec{u}) = \sum_{i=1}^{m} \int_{\Omega_i} \hat{a}_i(\vec{u}, \nabla \vec{u}) \, dH^2 + \]

\[ \sum_{i \leq j} \int_{\Gamma_{ij}} \gamma_{ij}(\vec{v}_{ij}) \, dH^1, \]

(2.37)
We now utilise these formulas and geometric setup to compute derivatives with respect to geometric quantities. Given an evolving regular configuration \( \{G(t, \vec{V})\} \),

**Lemma 2.1.13.** The derivative of the anisotropic surface tension at time \( t \) is

\[
\partial_t \gamma_{ij}(\vec{v}_{ij}) = -\frac{\partial \gamma_{ij}(\vec{v}_{ij})}{\partial \vec{v}_{ij}} \cdot \nabla \Gamma(\vec{V} \cdot \vec{v}_{ij}).
\]

**Remark 2.1.14.** When we write \( \vec{v}_{ij} \) on the left hand side, strictly speaking \( \vec{v}_{ij} \) is dependent on \( G(t) \). For notation purposes this is understood implicitly.

**Proof.** Applying [35][Theorem 2.4] to \( \gamma_{ij}(\vec{v}_{ij}) \) gives

\[
\partial_t \gamma_{ij}(\vec{v}_{ij}) = \frac{\partial \gamma_{ij}(\vec{v}_{ij})}{\partial \vec{v}_{ij}} \cdot \partial_t \vec{v}_{ij}. \tag{2.38}
\]

Furthermore, [35][Lemma 3.1] if \( \vec{V} \) and \( \Gamma^{ij} \) are sufficiently smooth this implies

\[
\partial_t \vec{v}_{ij} = -\nabla \Gamma(\vec{V} \cdot \vec{v}_{ij}). \tag{2.39}
\]

Substituting Eq. (2.39) into Eq. (2.38) gives the result. \( \square \)

In the following we define the variation of a functional \( F^* \) in the direction \( \vec{\eta} \in \mathcal{D} \). Given \( \vec{\eta} \in \mathcal{D} \) and \( G \in \mathcal{G} \) we fix an evolving velocity field as \( \vec{\eta}_\xi = \vec{\eta}, \forall \xi \in [0, \xi] \) where \( \xi \in \mathbb{R}^+ \) is sufficiently small so that \( \{G(\xi, \vec{\eta})\}_{\xi \in [0, \xi]} \) is an evolving regular configuration with the usual abbreviation \( G(\xi) \). Let \( \vec{u} \) be \( r \) dimensional field on \( \Omega \) such that \( a_i(\vec{u}, \nabla \vec{u}) \) and \( (\nabla \vec{u})^T \partial \nabla a_i(\vec{u}, \nabla \vec{u}) \) are admissible for \( i = 1, \ldots m \). Define \( \vec{u}_\xi \) as

\[
\vec{u}_\xi(\vec{x}) := \vec{u}(\vec{y}(\xi, \vec{x}, -\vec{\eta}))
\]

where \( \vec{y} \) solves Eq. (2.5) with \(-\vec{\eta}\) as the RHS. Therefore \( \vec{u}_\xi \) is constant along trajectories and \( a_i(\vec{u}_\xi, \nabla \vec{u}_\xi) \) and \( (\nabla \vec{u}_\xi)^T \partial \nabla a_i(\vec{u}_\xi, \nabla \vec{u}_\xi) \) are admissible with respect to \( G(\xi) \).

**Definition 2.1.15** (Variation of an energy functional \( F \)). Given the previous setup the variation of \( F^* \) in the direction \( \vec{\eta} \in \mathcal{D} \) is

\[
\langle \delta F^*(G, \vec{u}), \vec{\eta} \rangle := \frac{d}{d\xi} F^*(G(\xi), \vec{u}_\xi) \bigg|_{\xi = 0}. \tag{2.40}
\]

The following calculations are all done where \( F^* \) is assumed to be anisotropic. The isotropic case follows similarly.
Theorem 2.1.16. The variation of $\mathcal{F}^s$ in the direction $\tilde{\eta} \in \mathcal{D}$ is

$$
\langle \delta \mathcal{F}^s(G, \tilde{u}), \tilde{\eta} \rangle = \sum_{i=1}^{m} \int_{\Omega_i} -\partial_i \hat{a}_i(\tilde{u}, \nabla \tilde{u}) \cdot ((\nabla \tilde{u})\tilde{\eta}) - [\nabla \cdot \nabla u \hat{a}_i(\tilde{u}, \nabla \tilde{u})] \cdot ((\nabla \tilde{u})\tilde{\eta}) \ dH^2,
$$

(2.41)

+ \sum_{i \neq j} \int_{\Gamma_{ij}} [\hat{a}_i(\tilde{u}, \nabla \tilde{u}) - (\nabla \tilde{u})^T \partial \nabla \hat{a}_i(\tilde{u}, \nabla \tilde{u})] \tilde{\nu}_{ij} \cdot \tilde{\eta} \ dH^1

- \sum_{i \leq j} \int_{\Gamma_{ij}} \gamma_{ij}(\tilde{v}_{ij}) \tilde{\nu}_{ij} \cdot \tilde{\eta} \ dH^1

+ \sum_{i \leq j} \int_{\partial \Gamma_{ij}} \gamma_{ij}(\tilde{v}_{ij}) \tilde{\nu}_{ij} - (\gamma_{ij}'(\tilde{v}_{ij}) \cdot \tilde{\nu}_{ij}) \tilde{\nu}_{ij} \cdot \tilde{\eta} \ dH^0

+ \sum_{i=1}^{m} \int_{\tilde{z} \in \partial \Gamma_{ii}} \oint_{\mathcal{L}_{tip}(\tilde{z})} [((\nabla \tilde{u})^T \partial \nabla \hat{a}_i(\tilde{u}, \nabla \tilde{u}) - \hat{a}_i(\tilde{u}, \nabla \tilde{u})) \tilde{\nu}_{tip} \ dH^1 \cdot \tilde{\eta} \ dH^0.

Proof. As the energy is linear, we apply the transport identities to the bulk term $\hat{a}(\tilde{u}, \nabla \tilde{u})$, and surface terms $\gamma$ separately. Fix $i \in [1, m]$. Using the definition of variation in Eq. (2.40) to the energy Eq. (2.37) and applying the transport identity in Eq. (2.16)

$$
\frac{d}{d\xi} \int_{\Omega_i} \hat{a}_i(\tilde{u}_\xi, \nabla \tilde{u}_\xi) \ dH^2\bigg|_{\xi=0} = \int_{\Omega_i} \frac{d}{d\xi} \hat{a}_i(\tilde{u}_\xi, \nabla \tilde{u}_\xi) \ dH^2\bigg|_{\xi=0}

+ \sum_{j \neq i} \int_{\Gamma_{ij}} \hat{a}_i(\tilde{u}, \nabla \tilde{u})(\tilde{\eta} \cdot \tilde{\nu}_{ij}) \ dH^1

- \sum_{\tilde{z} \in \partial \Gamma_{ii}} \oint_{\mathcal{L}_{tip}(\tilde{z})} \hat{a}_i(\tilde{u}, \nabla \tilde{u})(\tilde{\eta} \cdot \tilde{\nu}_{tip}) \ dH^1.
$$

(2.42)

Expanding the first term gives

$$
= \int_{\Omega_i} \partial_\xi \hat{a}_i(\tilde{u}_\xi, \nabla \tilde{u}_\xi) \cdot \partial_\xi \tilde{u}_\xi + \partial \nabla \tilde{u} \hat{a}_i(\tilde{u}_\xi, \nabla \tilde{u}_\xi) : \partial_\xi \nabla \tilde{u}_\xi \ dH^2\bigg|_{\xi=0}

+ \sum_{j \neq i} \int_{\Gamma_{ij}} \hat{a}_i(\tilde{u}, \nabla \tilde{u})(\tilde{\eta} \cdot \tilde{\nu}_{ij}) \ dH^1

- \sum_{\tilde{z} \in \partial \Gamma_{ii}} \oint_{\mathcal{L}_{tip}(\tilde{z})} \hat{a}_i(\tilde{u}, \nabla \tilde{u})(\tilde{\eta} \cdot \tilde{\nu}_{tip}) \ dH^1.
$$

27
Applying the identity in Eq. (2.5).

\[
\int_\Omega \partial_\alpha \hat{a}_i(\bar{u}, \nabla \bar{u}) \cdot ((\nabla \bar{u})\bar{\eta}) - \partial_{\nabla \bar{u}}\hat{a}_i(\bar{u}, \nabla \bar{u}) : \nabla ((\nabla \bar{u})\bar{\eta}) \, d\mathcal{H}^2
\]

(2.43)

\[
+ \sum_{j \neq i} \int_{\Gamma_{ij}} \hat{a}_i(\bar{u}, \nabla \bar{u})(\bar{\eta} \cdot \bar{\nu}_{ij}) \, d\mathcal{H}^1
\]

\[
- \sum_{\bar{x} \in \partial \Omega} \int_{\Gamma_{tip}(\bar{x})} \hat{a}_i(\bar{u}, \nabla \bar{u})(\bar{\eta} \cdot \bar{\nu}_{tip}) \, d\mathcal{H}^1.
\]

Applying the transport theorem in Eq. (2.10) to the surface terms in Eq. (2.45).

\[
\frac{d}{d\xi} \int_{\Gamma_{ij}} \gamma_{ij}(\bar{\nu}_{ij}) \, d\mathcal{H}^1 \bigg|_{\xi=0}
\]

(2.45)

\[
= \int_{\Gamma_{ij}} \partial_\xi \gamma_{ij}(\bar{\nu}_{ij}) \, d\mathcal{H}^1 \bigg|_{\xi=0}
\]

(2.46)

\[
+ \int_{\Gamma_{ij}} \left[ \partial_\xi (\gamma_{ij}(\bar{\nu}_{ij}) - \gamma_{ij}(\bar{\nu}_{ij})) \right] \bar{\nu}_{ij} \cdot \bar{\eta} \kappa_{ij} \, d\mathcal{H}^1
\]

\[
+ \int_{\partial \Gamma_{ij}} \gamma_{ij}(\bar{\nu}_{ij}) \bar{\mu}_{ij} \cdot \bar{\eta} \, d\mathcal{H}^0.
\]

Applying the transport theorem in Eq. (2.10) to the surface terms in Eq. (2.45).

Using the homogeneity of \( \gamma_{ij} \) stated in Eq. (2.35),

\[
= \int_{\Gamma_{ij}} \partial_\xi \gamma_{ij}(\bar{\nu}_{ij}) \, d\mathcal{H}^1 \bigg|_{\xi=0}
\]

(2.47)

\[
- \int_{\Gamma_{ij}} \gamma_{ij}(\bar{\nu}_{ij}) \bar{\nu}_{ij} \cdot \bar{\eta} \kappa_{ij} \, d\mathcal{H}^1
\]

\[
+ \int_{\partial \Gamma_{ij}} \gamma_{ij}(\bar{\nu}_{ij}) \bar{\mu}_{ij} \cdot \bar{\eta} \, d\mathcal{H}^1.
\]
Using the equation for the derivative in Eq. (2.39) and applying to Eq. (2.47),

\[
\begin{align*}
&= \int_{\Gamma_{ij}} \gamma'_{ij}(\vec{v}_{ij}) \cdot \nabla \Gamma(\vec{\eta} \cdot \vec{v}_{ij}) - \gamma_{ij}(\vec{v}_{ij})(\vec{\eta} \cdot \vec{v}_{ij}) \kappa_{ij} \, dH^1 \\
&\quad + \int_{\partial\Gamma_{ij}} \gamma_{ij}(\vec{v}_{ij})(\vec{\eta} \cdot \vec{\mu}_{ij}) \, dH^0. 
\end{align*}
\]

(2.48)

Applying the integration by parts formula in Eq. (2.13),

\[
\begin{align*}
&= \int_{\Gamma_{ij}} \left[ -\nabla \Gamma \cdot \gamma'_{ij}(\vec{v}_{ij}) + \kappa_{ij}(\gamma'_{ij}(\vec{v}_{ij}) \cdot \vec{v}_{ij}) \right] (\vec{\eta} \cdot \vec{v}_{ij}) \, dH^1 \\
&\quad - \int_{\Gamma_{ij}} \gamma_{ij}(\vec{v}_{ij})(\vec{\eta} \cdot \vec{v}_{ij}) \kappa_{ij} \, dH^1 \\
&\quad + \int_{\partial\Gamma_{ij}} \gamma_{ij}(\vec{v}_{ij})(\vec{\eta} \cdot \vec{\mu}_{ij}) \, dH^0 \\
&\quad - \int_{\partial\Gamma_{ij}} (\gamma'_{ij}(\vec{v}_{ij}) \cdot \vec{\mu}_{ij})(\vec{\eta} \cdot \vec{v}_{ij}) \, dH^0. \\
&= \int_{\Gamma_{ij}} -\nabla \Gamma \cdot \gamma'_{ij}(\vec{v}_{ij})\vec{v}_{ij} \cdot \vec{\eta} \, dH^1 \\
&\quad + \int_{\partial\Gamma_{ij}} \gamma'_{ij}(\vec{v}_{ij})\vec{\mu}_{ij} \cdot \vec{\eta} - (\gamma_{ij}(\vec{v}_{ij}) \cdot \vec{\mu}_{ij})\vec{v}_{ij} \cdot \vec{\eta} \, dH^0. 
\end{align*}
\]

(2.49)

Finally use the one homogeneity Eq. (2.35) of \( \gamma \) Eq. (2.49). Adding and summing over \( i \) we get the result.

\[ \square \]

2.1.5 Gradient flow

Having defined the variation of a sharp energy and also balance laws, we use these concepts to define a weighted gradient flow for a regular configuration coupled to balance laws.

We have in mind evolutions where only the value of the velocity field on the \((1)\), and \((0)\) hypersurfaces are important. We do not wish to enforce anything in the bulk. For this reason we consider an evolution dependent on the restriction of \( \delta F^a \) to the \((1)\) and \((0)\) hypersurfaces. We do this by taking the previously defined variation and subtracting the terms that are present in the bulk domains. See [69] for an alternative approach where smooth cut-off functions are used to restrict the variation to the lower dimensional hypersurfaces.

**Definition 2.1.17** (Restricted variation). Let the variation of \( F^a \) be given by Eq. (2.40)
we define the restricted variation $\mathcal{P}(G, \tilde{u})$ in the direction $\tilde{\eta} \in D$ as
\[
(\mathcal{P}(G, \tilde{u}), \tilde{\eta}) := \langle \delta \mathcal{F}^\eta(G, \tilde{u}), \tilde{\eta} \rangle + \sum_{i=1}^{m} \int_{\Omega} \partial_{\tilde{a}} \mathcal{F}(\tilde{\eta}) \cdot (\nabla \tilde{u}) \tilde{\eta} + [\nabla \cdot \partial_{\tilde{a}} \mathcal{F}(\tilde{\eta}) \cdot (\nabla \tilde{u}) \tilde{\eta}] d\mathcal{H}^2.
\]

We introduce a mobility function $\tau_G : \Omega \rightarrow \mathbb{R}$ for each $G$ in $\mathcal{G}$. We have two forms depending on whether the energy is isotropic or anisotropic.

**Definition 2.1.18.** $\tau_G$ has the following form. For $\bar{x} \in \omega \in \mathcal{M}_G$

\[
\tau_G(\bar{x}) = \begin{cases} 
\tau_0 & \text{if } \gamma \text{ is anisotropic,} \\
\tau_0 & \text{if } \gamma \text{ is isotropic.}
\end{cases}
\]

and for $\bar{x} \in \omega \in m^0_G \cup \mathcal{G}$, $\tau_G(\bar{x}) = 0$. The last condition sets the mobility to zero at triple and higher order junctions and is needed to recover angle conditions.

We define a weighted (degenerate) inner product on $D$, $\forall \tilde{f}, \tilde{g} \in D$

\[
(f, g)_{\tau_G} = \sum_{i \leq j} \int_{\Gamma_{ij}} \tau_G(\tilde{f}, \tilde{g}, \tilde{\nu}_{ij}) (\tilde{f} \cdot \tilde{\nu}_{ij}) d\mathcal{H}^1 + \sum_{i \leq j} \int_{\partial \Gamma_{ij}} \tau_G(\tilde{f}, \tilde{g}, \tilde{\nu}_{ij}) d\mathcal{H}^0.
\]

We will write our evolution using this inner product and the restricted variation in Definition 2.1.17. We require that the velocity of the evolutions is constrained within a subset $\mathcal{K}(G) \subset D$. This is to ensure that the configuration only moves in physically reasonable directions. We choose $\mathcal{K}(G)$ to enforce the irreversibility of the cracks.

\[
\mathcal{K}(G) = \{ \tilde{F} \in D : \tilde{F} \cdot \tilde{\mu}_{ii} \in [0, \infty] \text{ on } \partial \Gamma_{ii}, \tilde{F} \cdot \tilde{\nu}_{ii} = 0 \text{ on } \Gamma_{ii}, 1 \leq i \leq m \}.
\]

To ensure that the evolution stays in this set we propose a variational inequality for the evolution. After defining the evolution we show that the velocity field takes the desired values on the $(1)$ and $(0)$ hypersurfaces.

**Definition 2.1.19 (P Gradient flow).** Given an initial regular configuration $G$, initial condition $\tilde{u}_I$, functions for the balance laws $E, Q, F$, bulk densities $\hat{a}$, surface tension $\gamma$, mobility $\tau_0$ and final time $T$. A $\mathcal{P}$ gradient flow is the family of regular configurations $\{G(t)\}_{t=0}^T$ along with the family of functions $\{\tilde{u}(\cdot, t)\}_{t=0}^T$ such that
∀\vec{\eta} ∈ \mathcal{K}(G(t)) and ∀t ∈ [0, T]

\begin{equation}
(\partial_t G(t), \vec{\eta} - \partial_t G(t))_{G(t),\tau_{G(t)}} \geq -\langle \mathcal{P}(G(t), \tilde{u}(\cdot, t)), \vec{\eta} - \partial_t G(t) \rangle,
\end{equation}

such that ∀\zeta ∈ C_0^\infty(\Omega × (0, T), \mathbb{R}^r)

\begin{equation}
0 = \int_0^T \int_\Omega E_{G(t)} \partial_t \zeta + Q_{G(t)} : \nabla \zeta + F_{G(t)} : \zeta \ d\mathcal{H}^2 \ d\mathcal{L}^1,
\end{equation}

and

\begin{equation}
G(0) = G, \quad \tilde{u}(\cdot, 0) = \tilde{u}_I(\cdot).
\end{equation}

in addition to boundary conditions for \tilde{u}, see Eq. (2.31).

We refer to Eq. (2.53) as the interface equation and Eq. (2.54) as the balance laws. The inner product was defined in Eq. (2.51). We collect some implications of a \mathcal{P} gradient flow.

**Remark 2.1.20.** While we call Definition 2.1.19 a gradient flow, the energy decrease of the system will be model dependent and is not a-priori implied by the formulation. This is because the variation of the energy has been restricted and the balance laws in general lead to changes to the free energy.

**Remark 2.1.21.** In general Eq. (2.53) does not provide enough information to define a vector field on the whole of \Omega. Typically only the velocity of lower dimensional objects can be learned, and it remains to analyse whether the thus obtained set of equations is well-posed. To keep PHASEFIELD flexible to future developments this question is left to the user.

**Evolution in non-crack case**

Assume we have a regular configuration and \mathcal{M}_G = m_G i.e there are no cracks, in addition assume that \alpha is not dependent on \nabla \tilde{u} i.e \partial_{\nabla \tilde{u}} \alpha(\tilde{u}, \nabla \tilde{u}) = 0. Some examples for these regular configurations are displayed in Figure 2.1 in the \textit{m} = 2 and \textit{m} = 3 case. Referring to Eq. (2.52) we therefore have \mathcal{K}(G) = \mathcal{D} and the variational inequality Eq. (2.53) becomes an equality.

\begin{equation}
(\partial_t G(t), \vec{\eta})_{G(t),\tau_{G(t)}} = -\langle \mathcal{P}(G(t), \tilde{u}(\cdot, t)), \vec{\eta} \rangle,
\end{equation}
We localise $\vec{\eta}$ to the (1) hypersurfaces to recover the following values of $\partial_t G(t)$ on $\Gamma^{ij}$

$$\tau_{G(t)} \partial_t G(t) \cdot \vec{v}_{ij} = \nabla_{\Gamma} \cdot \gamma'_{ij}(\vec{v}_{ij}) - [\hat{a}(\hat{u}, \nabla \hat{u})]_{ij}$$  \hspace{1cm} (2.56)

and on the (0) hypersurfaces we get for fixed $\omega \in m^0_{G(t)}$

$$\sum_{\partial \Gamma^{ij} \cap \omega \neq \emptyset} \gamma_{ij}(\vec{v}_{ij}) \vec{\mu}_{ij} - (\gamma'_{ij}(\vec{v}_{ij}) \cdot \vec{\mu}_{ij}) \vec{\nu}_{ij} = 0,$$  \hspace{1cm} (2.57)

which is an angle condition at the junctions where 3 or more phases meet. Similarly if the energy is isotropic we get on $\Gamma^{ij}$

$$\tau_{G(t)} \partial_t G(t) \cdot \vec{v}_{ij} = -\gamma_{ij} \kappa_{ij} - [\hat{a}(\hat{u}, \nabla \hat{u})]_{ij},$$  \hspace{1cm} (2.58)

and for fixed $\omega \in m^0_{G(t)}$

$$\sum_{\partial \Gamma^{ij} \cap \omega \neq \emptyset} \gamma_{ij}(\vec{v}_{ij}) \vec{\mu}_{ij} = 0.$$  \hspace{1cm} (2.59)

**Evolution in crack case**

Assume we have an initial regular configuration in $\mathbb{R}^2$ as in Figure 2.1 the first image in the $m = 1$ case and isotropic energy. We localise Eq. (2.53) to the crack tip $\partial \Gamma^{11}$ to deduce that on $\Gamma^{11}$

$$\partial_t G(t) \cdot \vec{v}_{11} = 0,$$

and on $\partial \Gamma^{11}$

$$\tau_0 \partial_t G(t)(\vec{\eta} - \partial_t G(t)) \geq$$

$$- \left[ \int_{\text{tip}} [(\nabla \hat{u})^T \partial_{\nabla \hat{u}} \hat{a}(\hat{u}, \nabla \hat{u}) - \hat{a}(\hat{u}, \nabla \hat{u})] \vec{v}_{\text{tip}} d\mathcal{H}^1 + \gamma_{11} \vec{\mu}_{11} \right] (\vec{\eta} - \partial_t G(t)).$$

A short calculation then shows that this implies on $\partial \Gamma^{11}$

$$\tau_{G(t)} \partial_t G(t) \cdot \vec{\mu}_{11} =$$

$$\max \left( \vec{\mu}_{11} \cdot \int_{\text{tip}} [\hat{a}(\hat{u}, \nabla \hat{u}) - (\nabla \hat{u})^T \partial_{\nabla \hat{u}} \hat{a}(\hat{u}, \nabla \hat{u})] \vec{v}_{\text{tip}} \, d\mathcal{H}^1 - \gamma_{11}, 0 \right).$$  \hspace{1cm} (2.60)
**Example 2.1.22.** From Eq. (2.59) at the triple junctions we recover
\[
\sum_{1 \leq i < j \leq 3} (\gamma_{ij}(\vec{v}_{ij})\vec{\mu}_{ij} - (\gamma'_{ij}(\vec{v}_{ij}) \cdot \vec{\mu}_{ij})\vec{v}_{ij}) = 0.
\]

This matches the conditions derived in [45]. As a special case of this, given a triple junction in \( \mathbb{R}^2 \) such as in Figure 2.4 and isotropic energies Eq. (2.59) is equivalent to
\[
\sigma_{12}\vec{v}_{12} + \sigma_{13}\vec{v}_{13} + \sigma_{23}\vec{v}_{23} = 0.
\]

A standard calculation shows this is equivalent to Young’s Law where \( \theta_1, \theta_2, \theta_3 \) are shown in Figure 2.4.

\[
\frac{\sin(\theta_1)}{\gamma_{23}} = \frac{\sin(\theta_2)}{\gamma_{13}} = \frac{\sin(\theta_3)}{\gamma_{12}} = 0. \quad (2.61)
\]

### 2.2 Examples

We present examples and show how they can be written as a \( \mathcal{P} \) gradient flow. These examples contain an interface equation in addition to balance laws stated in the strong form Eq. (2.25). These equations are manipulated into a form where the respective functions are easily recognised as part of a \( \mathcal{P} \) gradient flow. The functions and parameters later are specified using code snippets in a Python class. A class is an abstraction used in programming to group together related functions or variables; the functions that make up a class are called methods. Together these two groups completely specify the \( \mathcal{P} \) gradient flow, these are shown in Table 2.1 When cells are highlighted in orange this means they are the default options and therefore do not need to be provided in the class. As in the last section, \( r \) is the number of
Boundary Conditions

Initial Conditions

<table>
<thead>
<tr>
<th>Energy</th>
<th>Balance Laws</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>$\ddot{a}$</td>
</tr>
<tr>
<td>$0^{n \times m}$</td>
<td>$0^m$</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of functions and parameters to specify a sharp model. Orange indicates default values.

components of a field $\tilde{u}$ and $m$ is the number of phases.

The initial regular configuration $G$ is set by specifying the characteristic function $\hat{\chi}_G$. We do not go into the solving of these models in this chapter but focus on how the interface to PHASEFIELD is defined. The notation between the framework presented and the code in PHASEFIELD is kept as similar as possible to avoid confusion.

Each example is described in the following order.

1. Background on the model.
2. Interface equation is identified.
3. Balance laws are identified.
4. Parameter values and initial conditions are stated.
5. Boundary conditions are specified if present.

Providing the sharp class

The code is as similar as possible to the mathematical formulation however there are some specifics that must be dealt with when defining a number of the functions and variables. We go through how to define every function and variable in Table 2.1.

- $\Omega$ - This is a square domain. There are two methods to specify the grid. First by providing a DUNE cartesianDomain. The arguments are the corners and the last argument is the number of cells. In Listing 1 we create a $4 \times 4$ grid with centre at the origin. Strictly speaking the number of cells are not needed in the continuum description however this is unavoidable in the code specification.

The second method of providing the grid is a tuple. The first argument is a
string to a Dune Grid File (DGF) file and the second argument the dimension of the grid. This can be seen in line two in Listing 1.

```python
omega = cartesianDomain([-2, -2], [2, 2], [3, 3])
omega = ('my_mesh_file.dgf', 2)
```

Listing 1: Providing $\Omega = [-2, 2] \times [-2, 2]$ to the Sharp class.

- $\gamma$ - The surface tension matrix in the energy Eq. (2.37). This is implemented by defining a member function `def gamma(nu)` which is a function of $\vec{\nu}$. Whether it is dependent on this nu gives a different evolution. This is because the mobility of the evolution Eq. (2.50) depends on this choice. Listing 2 contains an example of an isotropic energy where $\gamma_{12} = 2$.

```python
def gamma(nu):
    return [[0, 2], [2, 0]]
```

Listing 2: Providing the isotropic surface tension $\gamma_{12} = 2$ to the Sharp class.

Remark 2.2.1. The code is able to deduce whether the energy is anisotropic or isotropic. The user must remember that if an anisotropic energy is provided it must be one-homogeneous and the mobility function $\tau$ is slightly different, see Eq. (2.50).

- $\gamma^\theta(\theta)$ - We additionally let entry of the surface tension be written in terms of a zero-homogeneous function in the case $m = 2$. This function must be written in terms of the orientation angle $\theta$ which is the angle from the positive x-axis. The function is defined as

$$\gamma_{12}(\vec{v}_{12}) := \gamma^\theta(\theta(\vec{v}_{12}))|\vec{v}_{12}|$$

and assumed to be smooth away from 0. In the code this is defined by the function `gammaTheta`.

- $E, Q, F$ - The are the balance law densities in Eq. (2.54). These are defined by the member functions `distE`, `distQ` and `distF` respectively. The transport
term $\text{distE}$ takes as an argument the discrete UFL function $\tilde{u}$. The source term $\text{distF}$ takes as arguments the UFL discrete function $\tilde{u}$ and the UFL spatial co-ordinate $\vec{x}$. The function $\text{distQ}$ takes as an argument the discrete UFL function $\tilde{u}$. For input of $\nabla \tilde{u}$ in $\text{distQ}$ the UFL function $\text{grad}$ can be used. In the function definitions $\tilde{u}$ should always be indexed even when $r = 1$.

One has to ensure that the definitions are consistent and each return a list of the correct dimensions according to Eqs. (2.18) to (2.20). A requirement in the class is that $\text{distE}$, $\text{distQ}$ and $\text{distF}$ must all be specified if one wishes to define a balance law. An example of specifying $E, Q$ and $F$ is given in Listing 3 with $r = 1, m = 2$. The following functions are defined

$$E(\tilde{u}(\vec{x}, t)) = \begin{pmatrix} 1 & 0 \end{pmatrix},$$  
(2.63)

$$Q(\tilde{u}(\vec{x}, t), \nabla \tilde{u}(\vec{x}, t)) = \begin{pmatrix} \nabla \tilde{u}(\vec{x}, t) & \nabla \tilde{u}(\vec{x}, t) \end{pmatrix},$$  
(2.64)

$$F(\tilde{u}(\vec{x}, t), \vec{x}) = \begin{pmatrix} 2\tilde{u}(\vec{x}, t) & \tilde{u}(\vec{x}, t) + \vec{x} \end{pmatrix}.$$  
(2.65)

Listing 3: Defining balance law density $E, Q$ and $F$ in Eqs. (2.63) to (2.65) in the Sharp class.

Remark 2.2.2. The indexing in Python starts from 0 e.g. if $r = 2$ the first component $\tilde{u}_1$ is $u[0]$, and the second component $\tilde{u}_2$ is $u[1]$.

- $a(\tilde{u}, \nabla \tilde{u})$ - The bulk density in the energy Eq. (2.37). This is a function of $\tilde{u}$ and $\nabla \tilde{u}$ and therefore in PHASEFIELD is a member function that takes exactly these arguments. In Listing 4, is an example where $m = 2$ and $a(\tilde{u}, \nabla \tilde{u})$ has value $\nabla \tilde{u} : \nabla \tilde{u}$ in $\Omega^1$ and $4\tilde{u}^3$ in $\Omega^2$. We have made use of the UFL expression $\text{inner}$ which is a sum of all the component wise products.

- $G$, $\tilde{u}_I$ - The initial conditions for the gradient flow Eq. (2.55). This is a
Listing 4: Providing the bulk energy $a(\tilde{u}, \nabla \tilde{u}) = (\nabla \tilde{u} : \nabla \tilde{u} - 4\tilde{u}^3)$ to the Sharp class.

member function with one argument $\vec{x}$. The function must return a list with two elements. The first element must be a list which has $m$ elements each specifying the value of the characteristic function of $G$ (Eq. (2.1)) at the point $\vec{x}$. The second element in the list is also a list with $r$ elements, specifying the value of each component of $\tilde{u}_I$ at that point.

- $T$ and $\tau_0$ according to Eq. (2.50) are specified by defining the variables endTime and mobility. A self-explanatory example is shown in Listing 5.

Listing 5: Providing the end time $T = 0.2$ and mobility constant $\tau_0 = 10^{-2}$ to the Sharp class.

Boundary conditions - Dirichlet boundary conditions are provided with a class method def dirichlet(t,x). Markers for the edge of the domain must be specified in a DGF file provided. The function dirichlet must return a dictionary. The key specifies the boundary identifier, the corresponding value must a list of values of $\tilde{u}$ on that boundary. In Listing 6 can be seen an example where $\tilde{u}_2 = -1$ on $(\partial \Omega)_1$, $\tilde{u}_2 = -2$ on $(\partial \Omega)_2$.

$(\partial \Omega)_1$ here denotes the boundary of $\partial \Omega$ with identifier 1, not the boundary of the set $\Omega^1$.

Boundary conditions - Neumann boundary conditions are provided with a class method neumann. This takes two arguments, $t$ and $\vec{x}$ respectively. It must return the flux $Q$ on the boundary. For an example see Section 2.2.5.
```python
def dirichlet(t, x):
    return {1:[None, -1], 2:[None, -2]}
```

Listing 6: Providing the Dirichlet boundary conditions $\tilde{u}_2 = -1$ on $(\partial \Omega)_1$, $\tilde{u}_2 = -2$ on $(\partial \Omega)_2$ to the Sharp class.

### 2.2.1 Mean curvature flow

We start with the simplest example that fits into our framework and has a interface equation; this is mean curvature flow \(^1\). Mean curvature flow states that the velocity of each point on a (1) hypersurface moves with normal velocity equal to the mean curvature of the surface at that point. We roughly follow the presentation in [32] where the evolution is written as

$$
\vec{V}_t \cdot \vec{n}_{12} = -\beta \kappa_{12}, \text{ on } \Gamma^{12}(t),
$$

$$
G = \{\Omega_1^1, \Omega_2^2\}.
$$

The dimensional parameters are given by $m = 2$ and $r = 0$. The only variable in this equation is $\beta$ which is the surface tension. $G$ has two phases. $\Omega_1^1$ is in the centre of the domain and $\partial \Omega_1^1$ does not intersect the boundary $\partial \Omega$. See $m = 2$ in Figure 2.1.

To translate this into a $\mathcal{P}$ gradient flow (Definition 2.1.19) we write the dimensional parameters as well as the initial conditions.

**Interface equation**

Eq. (2.66) can immediately be recognised as a interface equation as it has a curvature term present. We aim to find $\gamma, \hat{a}$ and $\tau_0$ such that Eq. (2.66) matches Eq. (2.53) on the (1) hypersurface $\Gamma^{12}(t)$. As we do not have any cracks $K(G(t))$ is the full space $\mathcal{D}$. Additionally as we don’t have $\nabla \gamma$ present the energy is likely to be isotropic. Comparing $\beta \vec{n}$ to Eq. (2.58) we see that we need to match

$$
\beta \kappa_{12} = \gamma_{12} k_{12} \text{ on } \Gamma^{12}(t)
$$

We let $\gamma_{12} = \beta$ and have an isotropic energy and leave $\hat{a}(\tilde{u}, \nabla \tilde{u})$ as its default zero value. We try to find the correct value of the mobility function in Eq. (2.50). Looking

---

\(^1\)The simplest non-trivial example that fits into our framework that does **not** have a interface equation would be Poisson’s equation $\Delta u = 0$ in $\Omega^1$ where $m = 1$. 

38
at the isotropic form of the mobility in Eq. (2.50) we see taking $\tau_0 = 1$ gives the correct evolution.

**Balance laws**

Clearly Eq. (2.54) is not present in Eqs. (2.66) and (2.67) as we have already accounted for all the equations. As such we can leave these to be set as their default options as seen in Table 2.1.

**Computational parameters**

For the domain we choose $\Omega = [-2, 2]^2$. The constants of the problem are chosen as

$$\beta = 1, \quad T = 0.125.$$  

Comparing Eq. (2.67) to Eq. (2.1) we use the characteristic function of the configuration to specify the initial conditions, which in this case are

$$\chi_{\Omega_1}(\vec{x}) = \begin{cases} 1 & \text{if } |\vec{x}| < 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

$$\chi_{\Omega_2}(\vec{x}) = \begin{cases} 1 & \text{if } |\vec{x}| > 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

The implementation of this function can be seen in Listing 7. We define the appropriate functions so that Eqs. (2.66) and (2.67) can be written as $\mathcal{P}$ gradient flow Definition 2.1.19.

**Summary and code**

We summarise the discussion above in Table 2.2 which states what every function and parameter must be to define Eqs. (2.66) and (2.67) as a $\mathcal{P}$ gradient flow Definition 2.1.19.

In Listing 7 can be seen how to specify the model in PHASEFIELD. The model class we call `class Mcf`. We point out how concise and similar to the mathematical definition of the problem this is. In Table 2.3 we state how Eqs. (2.66) and (2.67) are different in notation to [32][Eq 1.2]. Additionally, in [32] the initial configuration is described by stating the position of the initial (1) hypersurface $\Gamma(t)$ as opposed to the characteristic functions.
**Boundary Conditions**

<table>
<thead>
<tr>
<th>( \Omega )</th>
<th>( T )</th>
<th>( \tau_0 )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>([-2, -2], [2, 2])</td>
<td>0.5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Initial Conditions**

<table>
<thead>
<tr>
<th>( \Omega_1^I )</th>
<th>( \Omega_2^I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { \vec{x} \in \Omega :</td>
<td>\vec{x}</td>
</tr>
</tbody>
</table>

**Energy Balance Laws**

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( \bar{a} )</th>
<th>( E )</th>
<th>( Q )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \begin{pmatrix} 0 &amp; \beta \ \beta &amp; 0 \end{pmatrix} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Summary of functions and parameters for mean curvature flow.

```python
class Mcf:
    omega = cartesianDomain([-2, -2], [2, 2], [3, 3])
    endTime = 0.125
    mobility = 1

    def gamma(nu):
        return [[0, 1], [1, 0]]

    def initial(x):
        return [[conditional(x[0]*x[0]+x[1]*x[1] < 0.5, 1, 0),
            conditional(x[0]*x[0]+x[1]*x[1] > 0.5, 1, 0)]]
```

Listing 7: Mean curvature flow class definition.

**Notation [32]**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Our Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma(t) )</td>
<td>( \Gamma^{12}(t) )</td>
</tr>
<tr>
<td>( \nu )</td>
<td>( \nu_{12} )</td>
</tr>
<tr>
<td>( V )</td>
<td>( \nu_{12} )</td>
</tr>
<tr>
<td>( H )</td>
<td>( \kappa_{12} )</td>
</tr>
</tbody>
</table>

Table 2.3: Difference in notation for mean curvature flow.
2.2.2 Mullins-Sekerka

In the Mullins-Sekerka problem a particle grows during phase transformation, this growth is regulated by the diffusion of material or the flow of heat. Two principal assumptions made for this approximation are the neglect of crystallographic factors, such as elastic strain energy or anisotropic interface properties, and the description of thermal or diffusion fields by Laplace’s equation. The following general model is presented in [59] while the specific parameter values are from [41].

\[ \Delta \tilde{u} = 0, \quad \text{in } \Omega^1(t) \cup \Omega^2(t), \quad (2.68) \]
\[ \alpha [\nabla \tilde{u}]_2 \cdot \bar{v}_{12} = \bar{V}_t \cdot \bar{v}_{12}, \quad \text{on } \Gamma^{12}(t), \quad (2.69) \]
\[ 2\tilde{u} = \beta \kappa_{12}, \quad \text{on } \Gamma^{12}(t), \quad (2.70) \]
\[ G = \{ \Omega_1^I, \Omega_2^I \}, \quad (2.71) \]
\[ \tilde{u}(\cdot, 0) = \tilde{u}_I(\cdot). \quad (2.72) \]

The dimensional parameters are given by \( m = 2 \) and \( r = 0 \). The surface tension is \( \beta \in \mathbb{R}^+ \), while \( \alpha \in \mathbb{R}^+ \) is a parameter related to the latent heat at the interface. The geometric setup is topologically the same as Section 2.2.1. There is one set \( \Omega^1 \) in the middle of a 2d domain, the boundary \( \partial \Omega^1 \) is closed and does not intersect \( \partial \Omega \), see Figure 2.1.

Interface equation

As the curvature appears, we recognise the interface equation as Eq. (2.70) and the task is therefore to find \( \tau_0, \gamma, \hat{a} \) such that Eq. (2.70) is equivalent to Eq. (2.53) on \( \Gamma^{12}(t) \). The first thing we do is move the left hand term to the right hand side so we can compare. As we do not have any cracks \( \mathcal{K}(G(t)) \) is the full space \( \mathcal{D} \). We compare Eq. (2.70) to Eq. (2.58)

\[ 0 = -\beta \kappa_{12} + 2\tilde{u}, \quad \text{on } \Gamma^{12}(t) \]
\[ \tau_{G(t)} \bar{V}_t \cdot \bar{v}_{12} = -\gamma_{12} \kappa_{12} - [\hat{a}(\tilde{u}, \nabla \tilde{u})]_j^i, \quad \text{on } \Gamma^{12}(t). \]

Comparing the two we see that unlike Eq. (2.58), Eq. (2.70) has no term with the velocity and therefore we must have \( \tau_0 = 0 \). The first term is the same as in Section 2.2.1 and therefore we must have \( \gamma_{12} = \beta \).

Looking at Eq. (2.58), as \( \partial \Omega^1 \) is closed the \( \partial \Gamma^{12} \) terms must be zero and we
must choose $\hat{a}(\tilde{u}, \nabla \tilde{u})$ such that

$$\beta \kappa_{12} - [\hat{a}(\tilde{u}, \nabla \tilde{u})]_2^1 = \beta \kappa_{12} - 2\tilde{u}$$

and on $\Gamma_{12}$ must have

$$\hat{a}_1(\tilde{u}, \nabla \tilde{u}) - \hat{a}_2(\tilde{u}, \nabla \tilde{u}) = -2\tilde{u}, \quad \text{on } \Gamma_{12}(t).$$

We make the choice

$$\hat{a}_1(\tilde{u}, \nabla \tilde{u}) = 0, \quad \hat{a}_2(\tilde{u}, \nabla \tilde{u}) = 2\tilde{u}.$$ 

An example for how to define these in PHASEFIELD can be seen in Listing 8.

**Balance law**

By choosing the energy $\mathcal{F}^s$ and initial conditions we have described Eqs. (2.70) to (2.72) however we must still account for Eqs. (2.68) and (2.69). We try to describe them as balance laws by Eq. (2.54) and for this we use the strong form of this equation Eq. (2.25). Comparing the condition on $\Gamma_{12}(t)$ in Eq. (2.25) to Eq. (2.69) and the condition on $\Omega^1$ and $\Omega^2$ in Eq. (2.25) to Eq. (2.68), we can clearly see that the correct equations to take are

$$E(\tilde{u}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$Q(\tilde{u}, \nabla \tilde{u}) = \begin{bmatrix} -\alpha \nabla \tilde{u} \\ -\alpha \nabla \tilde{u} \end{bmatrix},$$

$$F(\tilde{u}, \vec{x}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$ 

**Computational parameters**

$$\alpha = 1, \quad \beta = \frac{0.8}{9}, \quad T = 10. \quad (2.73)$$

Let $\Omega = [0, 2]^2$. The initial conditions for the regular configuration $G$ are specified in the same way as in Section 2.2.1 and given by a perturbed circle in the centre of the domain. Let $r_\theta(\vec{x}) = 1 + \cos(2 \text{atan2}(\vec{x}_1, \vec{x}_0))$

$$\chi_{\Omega^1}(\vec{x}) = \begin{cases} 1 & \text{if } |\vec{x}| < r_\theta, \\
0 & \text{otherwise.} \end{cases}$$

$$\chi_{\Omega^2}(\vec{x}) = 1 - \chi_{\Omega^1}(\vec{x}).$$
### Boundary Conditions

<table>
<thead>
<tr>
<th>$\Omega$</th>
<th>$T$</th>
<th>$\tau_0$</th>
<th>No-flux</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-2,-2],[2,2]$</td>
<td>0.2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Initial Conditions

| $\Omega_1^I = \{ \vec{x} \in \Omega : |\vec{x}| < r_\theta \}$, $\Omega_2^I = \{ \vec{x} \in \Omega : |\vec{x}| > r_\theta \}$, $\tilde{u}_I(\cdot) = 1$ |

### Energy Balance Laws

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>$\tilde{u}$</th>
<th>$E$</th>
<th>$Q$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{pmatrix} 0 &amp; \beta \ \beta &amp; 0 \end{pmatrix}$</td>
<td>$\begin{pmatrix} 0 \ 2\tilde{u} \end{pmatrix}$</td>
<td>$\begin{pmatrix} 1 &amp; 0 \end{pmatrix}$</td>
<td>$\begin{pmatrix} -\alpha \nabla \tilde{u} &amp; -\alpha \nabla \tilde{u} \end{pmatrix}$</td>
<td>$\begin{pmatrix} 0 &amp; 0 \end{pmatrix}$</td>
</tr>
</tbody>
</table>

Table 2.4: Summary of functions and parameters for Mullins-Sekerka type problem.

Because we also have a field $u$, `def initial(x)` must return a $1 \times 3$ list. The third position is the value of $\tilde{u}$ which we set to 1 everywhere.

#### Summary and code

Table 2.4 shows a summary so that Eqs. (2.68) to (2.72) can be written as a $\mathcal{P}$ gradient flow (Definition 2.1.19). The functions and variables summarised in Table 2.4 are taken and implemented in the class Listing 8. We name the class $\texttt{Ms}$. In Table 2.5 we state how Eqs. (2.68) to (2.72) are different in notation to [41][Eq 89].

<table>
<thead>
<tr>
<th>Notation [41]</th>
<th>Our Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma$</td>
<td>$\Gamma^{12}(t)$</td>
</tr>
<tr>
<td>$\mathcal{D}_+$</td>
<td>$\Omega^1$</td>
</tr>
<tr>
<td>$\mathcal{D}_-$</td>
<td>$\Omega^2$</td>
</tr>
<tr>
<td>$\tilde{u}$</td>
<td>$\tilde{u}$</td>
</tr>
<tr>
<td>$[\partial_r \tilde{u}]$</td>
<td>$[\nabla \tilde{u}]_{12}$</td>
</tr>
<tr>
<td>$\tilde{v}$</td>
<td>$\tilde{V}_{12}$</td>
</tr>
<tr>
<td>$\tilde{\sigma}$</td>
<td>$\gamma_{12}$</td>
</tr>
<tr>
<td>$\ell_\gamma$</td>
<td>$\alpha$</td>
</tr>
</tbody>
</table>

Table 2.5: Difference in notation for Mullins-Sekerka type problem.

### 2.2.3 Dendritic growth

We take the following model from [8]. The specific form of bulk density and anisotropic energy is taken from [71]. The model is analysed there in the context of pattern formation which appears in many physical situations such as snowflake growth, solidi-
Class Ms:

```python
class Ms:
    omega = cartesianDomain([0, 0], [2, 2], [3, 3])
    endTime = 10
    mobility = 0.

    def initial(x):
        r = sqrt(dot(x, x))
        theta = atan_2(x[1], x[0])
        rTheta = 1 + 0.4 * cos(2*theta)
        return [conditional(r > rTheta, 0, 1), conditional(r < rTheta, 0, 1)], [0]

    def gamma(nu):
        surface = 0.8/9
        return [[0, 0.8/9], [0.8/9, 0]]

    def a(u, nabla_u):
        return [0, 2*u[0]]

    def distE(u):
        return [[1, 0]]

    def distQ(u):
        return [[-grad(u[0]), -grad(u[0])]]

    def distF(u, x):
        return [[0, 0]]
```

Listing 8: Sharp interface class for the Mullins-Sekerka problem.

The solidification of metals and Ostwald ripening in alloys. Roughly speaking patterns appear as the competition between interfacial energy and diffusion. Interfacial energy is preferred to be small which reduces the energy. In opposition to this is diffusion that drives the system into irregular shapes with large surface area. In solidification this is so that latent heat can diffuse away from the interface easier. The equations are that of two-phase boundary motion and we have a diffusion equation in the bulk (Eq. (2.74)) coupled to the Stefan condition on the free boundary (Eq. (2.75)). To close the system an extra condition stating the local thermodynamical equilibrium on the interface is needed Eq. (2.76). In the case $f(u) = u$ Eq. (2.76) is known.
as the Gibbs-Thomson relation with kinetic under cooling.

\[ \tilde{u}_t = \alpha \Delta \tilde{u} \quad \text{in } \Omega_1(t) \cup \Omega_2(t), \quad (2.74) \]
\[ \alpha [\nabla \tilde{u}]_{12} \cdot \vec{v}_{12} = \lambda \tilde{V}_t \cdot \vec{v}_{12} \quad \text{on } \Gamma^{12}(t), \quad (2.75) \]
\[ \frac{\tau_0}{\gamma(\vec{v}_{12})}(\tilde{V}_t \cdot \vec{v}_{12}) = \nabla \Gamma \cdot \gamma'(\vec{v}_{12}) + f(\tilde{u}) \quad \text{on } \Gamma^{12}(t), \quad (2.76) \]
\[ G(0) = G = \{ \Omega_1^0, \Omega_2^0 \}, \quad (2.77) \]
\[ \tilde{u}(\cdot, 0) = \tilde{u}_I(\cdot). \quad (2.78) \]

The dimensional parameters are given by \( m = 2 \) and \( r = 1 \). The initial geometric setup is the same as in Sections 2.2.1 and 2.2.2. For existence results in the isotropic case and \( f(u) = u \) see [28] for local in time smooth solutions and [56] for global in time weak solutions. Weak in time solutions in the anisotropic settings are derived in [46].

**Interface equation**

Unlike in Section 2.2.2 we make use of the parameter \( m \) in the method \texttt{def gamma}. As we are working in 2d, the \( y \) coordinate of the unit normal is accessed with \( m[1] \) and the \( x \)-component as \( m[0] \). There are no cracks in \( G \) so

\[ \mathcal{K}(G(t)) = \mathcal{D}. \]

In this model we identify Eq. (2.76) as the interface equation and compare this with Eq. (2.56) to get

\[ \nabla \Gamma \cdot \gamma'_1(\vec{v}_{12}) - [\hat{a}(\tilde{u}, \nabla \tilde{u})]_1 = \nabla \Gamma \cdot \gamma'(\vec{v}_{12}) + f(\tilde{u}). \]

It is therefore clear that we must take \( \gamma'(\vec{v}_{12}) \) to be the surface energy and also have

\[ -f(\tilde{u}) = -\hat{a}_1(\tilde{u}, \nabla \tilde{u}) + \hat{a}_2(\tilde{u}, \nabla \tilde{u}). \]

Similarly to Section 2.2.2 there are numerous choices for \( \hat{a} \). We choose \( \hat{a}_1(\tilde{u}, \nabla \tilde{u}) = f(\tilde{u}), \hat{a}_2(\tilde{u}, \nabla \tilde{u}) = 0 \). Finally comparing the left hand side of Eq. (2.76) to the left hand side of Eq. (2.56). We dot both with the unit normal \( \vec{v}_{12} \) applying Eq. (2.50) gives

\[ \frac{\tau_0}{\gamma(\vec{v}_{12})}(\tilde{V}_t \cdot \vec{v}_{12}) = \frac{\tau_0}{\gamma'(\vec{v}_{12})}(\tilde{V}_t \cdot \vec{v}_{12}). \]
Therefore, as $\vec{V}_t$ is the velocity everywhere it must agree at the points on $\Gamma^{12}$. We get that $\tau_0$ is exactly the same as the $\tau_0$ in Eq. (2.50).

**Balance law**

Having described Eq. (2.76) and Eq. (2.77) we wish to write Eqs. (2.74) and (2.75) in the form of Eq. (2.54). We look at the strong form Eq. (2.25) of Eq. (2.54) and compare them to Eq. (2.74) and Eq. (2.75). Comparing each term we see that the following are the appropriate terms to take.

$$E(\tilde{u}) = \begin{bmatrix} \tilde{u} - \lambda \tilde{u} \end{bmatrix},$$

$$Q(\tilde{u}, \nabla \tilde{u}) = \begin{bmatrix} -\alpha \nabla \tilde{u} & -\alpha \nabla \tilde{u} \end{bmatrix},$$

$$F(\tilde{u}, \vec{x}) = \begin{bmatrix} 0 & 0 \end{bmatrix}.$$

**Computational parameters**

Let the domain $\Omega = [-2, 2]^2$. The energy and bulk densities are taken from [71] and given by

$$T = 0.08, \quad N = 6, \quad c = 0.02, \quad \kappa_1 = 0.9, \quad \kappa_2 = 20,$$

$$\tau_0 = 24, \quad \lambda = 1, \quad \alpha = 2.25,$$

$$\gamma^\theta(\theta) = \sqrt{18}(1 + c\beta(\theta)),$$

$$\beta(\theta) = \frac{1 - [\phi(\theta)]^2}{1 + [\phi(\theta)]^2},$$

$$\phi(\theta) = \tan\left(\frac{N}{2\psi(\theta)}\right),$$

$$\psi(\theta) = \frac{\pi}{8} + \theta,$$

$$f(\tilde{u}) = 400\frac{\kappa_1}{\pi} \arctan(\kappa_2 \tilde{u}).$$

Here $\gamma_{12}$ is provided as a zero-homogeneous function $\gamma^\theta$ in terms or an orientation angle as shown in Eq. (2.62). The initial conditions are given by

$$\chi_{\Omega_1^I}(\vec{x}) = \begin{cases} 1 & \text{if } |\vec{x}| < 0.3, \\ 0 & \text{otherwise}. \end{cases}$$

$$\chi_{\Omega_2^I}(\vec{x}) = \begin{cases} 1 & \text{if } |\vec{x}| > 0.3, \\ 0 & \text{otherwise}. \end{cases}$$
<table>
<thead>
<tr>
<th>$\Omega$</th>
<th>$T$</th>
<th>$\tau_0$</th>
<th>Boundary Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-2, -2], [2, 2]$</td>
<td>0.08</td>
<td>24</td>
<td>No-flux</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy</th>
<th>Initial Conditions</th>
<th>Balance Laws</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma'\theta$</td>
<td>$\Omega^t_I = { \bar{x} \in \Omega : |\bar{x} - \left(\begin{smallmatrix} 6 \ 6 \end{smallmatrix}\right)| &lt; 0.3}$,</td>
<td>$E$</td>
</tr>
<tr>
<td>$\sqrt{18(1 + c\beta(\theta))}$</td>
<td>$\Omega^t_I = { \bar{x} \in \Omega : |\bar{x} - \left(\begin{smallmatrix} 6 \ 6 \end{smallmatrix}\right)| &gt; 0.3}$,</td>
<td>$Q$</td>
</tr>
<tr>
<td></td>
<td>$\ddot{u}_I(\cdot) = -0.5$</td>
<td>$F$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(\ddot{u} - \lambda \ddot{u})$</td>
</tr>
</tbody>
</table>

Table 2.6: Summary of functions and parameters for dendritic growth.

**Summary and code**

All the functions defined above are summarised in Table 2.6. The class for implementing this model can be seen in Listing 9. We name the class `Crystal`. In Table 2.7 we show how our equations Eqs. (2.74) to (2.78) are different in notation to [8]|6.1].
class Crystal:
    omega = cartesianDomain([4, 4], [8, 8], [3, 3])
    endTime = 0.08
    mobility = 24

    def initial(x):
        r = sqrt( dot( x-as_vector([6,6]), x-as_vector([6,6]) ) )
        return [[conditional(r>0.3, 0, 1), conditional(r<0.3, 0, 1)], [-0.5]]

    def gammaTheta(theta):
        c = 0.02
        N = 6.
        psi = pi/8.0 + theta
        Phi = tan(N / 2.0 * psi)
        beta = (1.0 - Phi*Phi) / (1.0 + Phi*Phi)
        dbeta_dPhi = -2.0 * N * Phi / (1.0 + Phi * Phi)
        return sqrt(18) * (1.0 + c * beta)

    def a(u,nablau):
        kappa1 = 0.9
        kappa2 = 20
        gamma = kappa1/pi
        return [400 * gamma * atan(kappa2 * u[0]), 0 ]

    def distE(u):
        #Here K is the latent heat at the interface
        return [[u[0]-1, u[0]]]

    def distQ(u):
        alpha = 2.25
        return [[-alpha*grad(u[0]), -alpha*grad(u[0])]]

    def distF(u,x):
        return [[0, 0]]

Listing 9: Sharp class for dendritic growth.
Table 2.7: Difference in notation for dendritic growth.

2.2.4 Tumour growth

We present a two-phase tumour growth model. A phase field model was derived via thermodynamic arguments in [43]. The formal asymptotics were also done there and the sharp equations derived are the ones we present.

Our presentation differs from [43], as there the model has a canonical parameter \( \tilde{u} \) that is discontinuous across the interface. We have applied a linear transformation to ensure that here \( \tilde{u} \) is continuous across \( \Gamma^{12} \).

Letting \( n = 2 \), \( \Omega^1 \) can be identified with the inside of the tumour while \( \Omega^2 \) is the healthy tissue. The model is defined by Eqs. (2.79) and (2.81) to (2.84).

\[
\begin{align*}
-\Delta \tilde{u}_1 & = P(\tilde{u}_2 + \lambda) - A, & \text{in } \Omega^1(t), \\
\Delta \tilde{u}_2 & = C(\tilde{u}_2 + \lambda) & \text{in } \Omega^1(t), \\
\Delta \tilde{u}_2 & = 0 & \text{in } \Omega^2(t), \\
0 & = (\nabla \tilde{u}_2 + D \nabla \tilde{u}_2) \cdot \tilde{v}_{12} & \text{in } \Gamma^{12}(t), \\
- \nabla \tilde{u}_1 \cdot \tilde{v}_{12} & = \tilde{V}_t \cdot \tilde{v}_{12} & \text{in } \Gamma^{12}(t), \\
2\alpha \tilde{u}_2 + 2b \tilde{u}_1 & = \beta \kappa_{12} & \text{in } \Gamma^{12}(t), \\
\tilde{u}_2 & = 1 & \text{on } \partial \Omega, \\
G(0) & = G = \{ \Omega^1, \Omega^2 \}, \\
\tilde{u}(\cdot, 0) & = \tilde{u}_I(\cdot).
\end{align*}
\]

The dimensional parameters of the problem are \( m = 2 \) and \( r = 2 \). The setup is a circle like tumour in the middle of a domain identified with the set \( \Omega^1 \). This set \( \partial \Omega^1 \) is closed and does not intersect \( \partial \Omega \). This tumour is “fed” by a nutrient which diffuses through the domain and encourages the tumour to grow. \( \tilde{u}_1 \) transports a nutrient to the exterior of the tumour which encourages the tumour to grow. \( \tilde{u}_2 \) is
a chemical potential. \( P, A, C \) are the proliferation, apoptosis and consumption rate respectively and \( \sigma \) the surface tension. \( \beta \) is a parameter for transport mechanisms such as chemotaxis and active transport. Tumour growth/proliferation is represented by \( P u_1 \). Apoptosis is \( A \). \( C u_1 \) is consumption of nutrient which only happens in the presence of tumour cells. \( \lambda \) is a density parameter for active transport proteins.

Active transport is meant in a biological sense. This indicates that a mechanism is required to maintain the transport. This is in contrast to passive transporters which are driven by the concentration gradient of the substance.

**Interface equation**

We identify the interface equation as Eq. (2.84) by the same procedure as in Section 2.2.2. We deduce that \( \tau_0 = 0 \) and \( \gamma_{12} = \beta \).

Comparing the right hand side of Eq. (2.84) with the result of the additional terms in Eq. (2.58) we find that

\[
-\hat{a}_1(\tilde{u}, \nabla \tilde{u}) + \hat{a}_2(\tilde{u}, \nabla \tilde{u}) = 2b\tilde{u}_1 + \alpha\tilde{u}_2,
\]

and therefore get

\[
\hat{a}_2(\tilde{u}, \nabla \tilde{u}) = 2b\tilde{u}_1 + 2\alpha\tilde{u}_2.
\]

**Balance law**

Having described the interface equation Eq. (2.84) and the initial conditions in Eqs. (2.86) and (2.87), we must describe Eqs. (2.79) to (2.83) with the balance law Eq. (2.54).

We pair up the equations for \( \tilde{u}_1 \) which are Eqs. (2.79) and (2.81), and equations for \( \tilde{u}_2 \) which are Eqs. (2.80) to (2.82). Next we can compare these equations for \( \tilde{u}_1 \) and \( \tilde{u}_2 \) individually with the strong forms of the balance laws in Eq. (2.25). Again we can ignore the terms in the last line of Eq. (2.25).

Applying Theorem 2.1.11, Eqs. (2.79) and (2.81) to (2.83) are equivalent to

\[
E(\tilde{u}) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},
\]

\[
Q(\tilde{u}, \nabla \tilde{u}) = \begin{bmatrix} -\nabla \tilde{u}_1 & 0 \\ \nabla \tilde{u}_2 & D \nabla \tilde{u}_2 \end{bmatrix},
\]

\[
F(\tilde{u}, \vec{x}) = \begin{bmatrix} P(\tilde{u}_2 + \lambda) - A & 0 \\ c(\tilde{u}_2 + \lambda) & 0 \end{bmatrix}.
\]
These are defined in Listing 10.

**Boundary conditions**

In addition to the equations for $\tilde{u}$ on the interior of $\Omega$, we have Dirichlet boundary conditions (Eq. (2.85)). As stated in Section 2.2 these are provided by the additional member function `dirichlet`. In the DGF file provided for $\Omega$ we use 1 as the identifier for the boundary. The first component of $\tilde{u}$ has no-flux boundary conditions and needs to be set to `None`.

**Computational parameters**

We let $\Omega = [-12.5, 12.5]^2$ with the following parameters

\[
\beta = 0.2 \cdot \pi, \quad \alpha = 1, \quad C = 1, \quad P = 0.2, \quad \lambda = 0, \\
A = 0, \quad b = 1, \quad D = 1, \quad T = 10.
\]

The initial conditions are given by a perturbed circle as

\[
\theta_r(x) = 2 + 0.2 \cdot \cos (3 \cdot \text{atan2}(x_1, x_0)).
\]

\[
\chi_{\Omega_1}^1(x) = \begin{cases} 
1 & \text{if } |x| < \theta_r, \\
0 & \text{otherwise}.
\end{cases}
\]

\[
\chi_{\Omega_2}^1(x) = \begin{cases} 
1 & \text{if } |x| > \theta_r, \\
0 & \text{otherwise}.
\end{cases}
\]

$\tilde{u}_2 = 1$ on $\partial \Omega$.

**Summary and code**

A summary of all the functions above can be seen in Table 2.8. We call the class `class Tumour` and can be see in Listing 10. The biggest challenge when entering this model into PHASEFIELD is that because of the multiple balance laws, one must be careful in ensuring that the lists returned from each function are of the correct dimension. In addition we have demonstrated how to enter Dirichlet boundary conditions.
Boundary Conditions

\[ \Omega = [-2, -2], [2, 2], \quad T = 0.1, \quad \tau_0 = 0, \quad \tilde{u}_2 = 1, \text{ on } \partial \Omega \]

Energy

<table>
<thead>
<tr>
<th>\gamma \quad \beta</th>
<th>\hat{a}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{pmatrix} 0 \ \beta \ 0 \end{pmatrix}</td>
<td>\begin{pmatrix} 0 \ 2b\tilde{u}_1 + 2\alpha\tilde{u}_2 \end{pmatrix}</td>
</tr>
</tbody>
</table>

Initial Conditions

\[
\theta_r(\bar{x}) = 2 + 0.2 \cdot \cos \left( 3 \cdot \arctan \left( \frac{x_1}{x_0} \right) \right). \\
\Omega_1^I = \{ \bar{x} \in \Omega : |\bar{x}| < \theta_r \}, \\
\Omega_2^I = \{ \bar{x} \in \Omega : |\bar{x}| > \theta_r \}, \\
\tilde{u}_I(\cdot) = 1
\]

Balance Laws

\[
\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -\nabla \tilde{u}_1 & 0 \\ \nabla \tilde{u}_2 & D \nabla \tilde{u}_2 \end{pmatrix} \begin{pmatrix} P(\tilde{u}_2 + \lambda) - A \\ c(\tilde{u}_2 + \lambda) \end{pmatrix} = 0
\]

Table 2.8: Summary of functions and parameters for tumour growth.
class Tumour:
    omega = cartesianDomain([-12.5, -12.5], [12.5, 12.5], [3, 3])
    endTime = 10.0
    mobility = 0

    def dirichlet(t, x):
        return 1:{None, 1}

    def initial(x):
        r = sqrt(dot(x, x))
        theta = atan(x[1]/x[0])
        rTheta = 2 + 0.1 * cos(3*theta)
        return [[conditional(r > rTheta, 0, 1), conditional(r < rTheta, 0, 1)],
                 [0, 1]]

    def gamma(nu):
        beta = 0.2*pi
        return [[0, beta], [beta, 0]]

    def a(u, gradu):
        alpha = 5
        b = 1
        return [0, 2*b*u[0] + 2*alpha*u[1]]

    def distE(u):
        return [[1, 0], [0, 0]]

    def distQ(u):
        D = 1
        return [[-grad(u[0]), zero(2)], [grad(u[1]), D*grad(u[1])]]

    def distF(u, x):
        lam = 0
        P = 0.1
        A = 0
        C = 1
        return [[P*(u[1] + lam) - A, 0], [C*(u[1] + lam), 0]]

Listing 10: PYTHON class for tumour growth.
2.2.5 Fracture

Fracture mechanics attempts to model the propagation of cracks through materials. It is different from the other models in that our interface evolution is defined on a (1) hypersurface contained on the interior of a bulk domain and not on its boundary as before.

Crack propagation can either be analysed in the quasi-static or dynamic setting. We present a dynamic model from [29]. There a phase field model was proposed from the so-called configurational force balance. The following sharp equations were derived by formal asymptotics in the limit. \( \tilde{u} \) is a two component displacement field in the \( x \) and \( y \) directions respectively. \( \mathcal{E}(\nabla \tilde{u}) \) is the infinitesimal strain tensor while \( \sigma(\nabla \tilde{u}) \) is the Cauchy stress tensor. For specifying the boundary conditions we label the upper, lower, left and right boundary of the domain \( \partial \Omega^U, \partial \Omega^D, \partial \Omega^L, \partial \Omega^R \) respectively.

\[
\begin{align*}
\tilde{0} &= \nabla \cdot \sigma(\nabla \tilde{u}) \quad \text{in } \Omega \setminus \Gamma^{11}, \\
\tilde{0} &= \left[ \sigma(\nabla \tilde{u}) \right]_1 \nu_{11} \quad \text{on } \Gamma^{11}, \\
\tilde{0} &= \oint_{\Gamma^{tip}} \sigma(\nabla \tilde{u}) \nu_{tip} \, dH^1, \quad \text{on } \partial \Gamma^{11},
\end{align*}
\]

\[
\begin{align*}
\vec{V}_t \cdot \vec{\mu}_{11} &= \left( \vec{\mu}_{11} \cdot \oint_{\Gamma^{tip}} \frac{1}{2} \sigma(\nabla \tilde{u}) : \mathcal{E}(\nabla \tilde{u}) - (\nabla \tilde{u})^T \sigma(\nabla \tilde{u}) \right) \nu_{tip} \, dH^1 - \beta, 0 \right)^+ \quad \text{on } \partial \Gamma^{11},
\end{align*}
\]

and the boundary conditions

\[
\begin{align*}
\sigma(\nabla \tilde{u}) \nu_{\partial \Omega} &= \begin{pmatrix} 0 \\ -t \cdot k \end{pmatrix}, \quad \text{on } \partial \Omega^U, \\
\sigma(\nabla \tilde{u}) \nu_{\partial \Omega} &= \begin{pmatrix} 0 \\ t \cdot k \end{pmatrix}, \quad \text{on } \partial \Omega^L,
\end{align*}
\]

\[
G(0) = G = \{ \Omega^I \}, \quad \tilde{u}(\cdot, 0) = \tilde{u}_I(\cdot).
\]

The dimensions of the problem are given by \( m = 1 \) and \( r = 2 \). The setup can be seen in Figure 2.5 where we have domain \( \Omega \) and evolving crack \( \Gamma^{11} \).
Interface equation

We first identify the interface equation in Eqs. (2.88) to (2.93). Unlike the other examples, there is no curvature present in any of equations. We notice that Eq. (2.91) has the term $\vec{V}$ in it so try and write Eq. (2.91) in the form of Eq. (2.60) as the initial regular configuration $G$ contains a crack.

We attempt to find $\gamma$ and $\hat{a}$ such that at the points $\vec{x} \in \Gamma^{11}$

$$\vec{\mu}_{11} \cdot \left[ \int_{\text{tip}} \hat{a} (\tilde{u}, \nabla \tilde{u}) - (\nabla \tilde{u})^T \partial \gamma \hat{a} (\tilde{u}, \nabla \tilde{u}) \right] \bar{\nu}_{\text{tip}} dH^1 - \gamma_{11} (\vec{v}_{11}) \vec{\mu}_{11} = (2.94)$$

$$\vec{\mu}_{11} \cdot \left[ \int_{\text{tip}} \left[ \frac{1}{2} \sigma (\nabla \tilde{u}) : \mathcal{E} (\nabla \tilde{u}) - (\nabla \tilde{u})^T \sigma (\nabla \tilde{u}) \right] \bar{\nu}_{\text{tip}} dH^1 - \beta. \right.$$  

It is immediate that $\gamma_{11} = \beta$ and $\hat{a} (\tilde{u}, \nabla \tilde{u}) = \frac{1}{2} \sigma (\nabla \tilde{u}) : \mathcal{E} (\nabla \tilde{u})$.

Balance laws

The strong conditions for the balance laws are Eqs. (2.88) to (2.90). Comparing these to Eq. (2.25) we see that taking $E, Q$ and $F$ as following appropriately defines the balance law Eq. (2.54)

$$E(\tilde{u}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (2.95)$$

$$Q(\tilde{u}, \nabla \tilde{u}) = \sigma (\nabla \tilde{u}), \quad (2.96)$$

$$F(\tilde{u}, \bar{x}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (2.97)$$
As $\tilde{u}$ has two components we must insert Eq. (2.96) one row at a time into $Q$. This is straightforward in Python as we can slice using the colon operator which returns all the columns given a row index.

**Boundary conditions**

In addition to the above we also have the boundary conditions Eq. (2.92) that we must impose. This is done as shown in Listing 11.

```python
def neuman(t,x):
    return as_vector([0, conditional(x[1]<-75+1e-8,t*6,0)+conditional(x[1]>75-1e-8,-t*6,0)])
```

Listing 11: Providing boundary conditions for fracture model.

**Computational parameters**

We let $\Omega = [0,100] \times [-75,75]$ and

$$
\mathcal{E}(\nabla \tilde{u}) = \frac{1}{2} (\nabla \tilde{u} + (\nabla \tilde{u})^T),
$$

$$
\sigma(\nabla \tilde{u}) = \lambda Tr(\mathcal{E}(\nabla \tilde{u})) I + 2\mu \mathcal{E}(\nabla \tilde{u}).
$$

$Tr$ is the trace of a matrix, $I$ is the identity matrix, $\lambda$ and $\mu$ are the Lamé constants.

The relevant parameters of the problem are given by

$$
\tau_0 = 2e - 3, \quad \beta = \frac{1}{2}, \quad T = 0.1, \quad \lambda = \mu = 22 \cdot 10^3.
$$

The initial conditions are that we have a straight crack of length 25 along the $x$-axis at the origin.

$$
\hat{\chi}_{\Omega^1} = \begin{cases} 
0 & \text{if } \bar{x}_1 \leq 25 \text{ and } \bar{x}_2 = 0, \\
1 & \text{otherwise}.
\end{cases}
$$

In Listing 12 we take a small band around this crack to ensure we have no sharp edges.
<table>
<thead>
<tr>
<th>$\Omega$</th>
<th>$T$</th>
<th>$\tau_0$</th>
<th>Boundary Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0, -75], [75, 100]$</td>
<td>0.1</td>
<td>0</td>
<td>$\sigma(\nabla \tilde{u}) \nu_{\partial \Omega} = \begin{pmatrix} 0 \ -t \cdot k \end{pmatrix}$, on $\partial \Omega^U$, $\sigma(\nabla \tilde{u}) \nu_{\partial \Omega} = \begin{pmatrix} 0 \ t \cdot k \end{pmatrix}$ on $\Omega^L$,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy</th>
<th>$\gamma$</th>
<th>$\tilde{a}$</th>
<th>$\frac{1}{2} \sigma(\nabla \tilde{u}) : \mathcal{E}(\nabla \tilde{u})$</th>
</tr>
</thead>
</table>

| Initial Conditions | $\Omega_f = \{ \tilde{x} \in \tilde{\Omega} : \tilde{x} \notin \{ \tilde{x} \in \tilde{\Omega} : x_1 < 25, x_2 = 0 \} \}$, $\tilde{u}_I(\cdot) = 1$ |

<table>
<thead>
<tr>
<th>Balance Laws</th>
<th>$E$</th>
<th>$Q$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{pmatrix} 0 \ 0 \end{pmatrix}$</td>
<td>$\sigma(\nabla \tilde{u})$</td>
<td>$\begin{pmatrix} 0 \ 0 \end{pmatrix}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.9: Summary of functions and parameters for fracture.

Summary and code

We create our Python class in Listing 12 and define the class as Fracture.
def eps(u):
    return 0.5*(grad(u)+transpose(grad(u)))

def sigma(u):
    lam = mu = 22000
    geodim = 2
    return lam*tr(eps(u))*Identity(geodim) + 2*mu*eps(u)

class Fracture:
    omega = cartesianDomain([0, -75], [100, 75], [3, 3])

    endTime = 0.1
    mobility = 2e-3

    tipCenter = [25,0]
    tipRadius = 0.01

    def neuman(t,x):
        return as_vector([0,
                          conditional(x[1]<-75+1e-8,t*6,0)+conditional(x[1]>75-1e-8,-t*6,0)])

    def initial(x):
        tipCenter = as_vector(Fracture.tipCenter)
        tipRadius = Fracture.tipRadius
        crack = 1 - conditional(x[1] > tipRadius, 0, 1) * conditional(x[0] > tipCenter[0], 0, 1)
        return [[crack], [0, 0]]

    def gamma(nu):
        return [[0.5]]

    def a(u, gradu):
        return [0.5*inner(sigma(u), eps(u))]

    def distE(u):
        return [[0], [0]]

    def distQ(u):
        return [[sigma(u)[0, :]], [sigma(u)[1, :]]]

    def distF(u, x):
        return [[0], [0]]
2.2.6 Multi-phase mean curvature flow

We extend the mean curvature examples in Section 2.2.1 to the multi-phase setting. Formal asymptotics were done in [45].

\[ \tau_0 \vec{V}_t \cdot \vec{v}_{12} = -\beta_{12} \kappa_{12}, \quad \text{on } \Gamma^{12}(t), \quad (2.98) \]
\[ \tau_0 \vec{V}_t \cdot \vec{v}_{13} = -\beta_{13} \kappa_{13}, \quad \text{on } \Gamma^{13}(t), \quad (2.99) \]
\[ \tau_0 \vec{V}_t \cdot \vec{v}_{23} = -\beta_{23} \kappa_{23}, \quad \text{on } \Gamma^{23}(t), \quad (2.100) \]
\[ 0 = \beta_{12} \tilde{v}_{12} + \beta_{13} \tilde{v}_{13} + \beta_{23} \tilde{v}_{23}, \quad \text{on } \Gamma^{12}(t) \cap \Gamma^{13}(t) \cap \Gamma^{23}(t), \quad (2.101) \]
\[ G = \{ \Omega_1, \Omega_2, \Omega_3 \}. \quad (2.102) \]

The dimensional parameters are given by \( m = 3 \) and \( r = 0 \). The variables in this equation \( \beta_{12}, \beta_{13}, \beta_{23} \) are the respective surface tensions. \( G \) has three phases with the setup, as in Figure 2.4.

Interface equation

Eqs. (2.98) to (2.100) can be recognised as an interface equation as it has the curvature present in it. We therefore aim to find \( \gamma, \hat{a} \) and \( \tau_0 \) such that Eq. (2.66) matches Eqs. (2.98) to (2.100) on the (1) hypersurfaces. We aim to find \( \mathcal{F}^s \) such that for \( 1 \leq i < j \leq 3 \)

\[ \tau_{G(t)} \vec{V}_t \cdot \vec{v}_{ij} = -\beta_{ij} \kappa_{ij} \quad \text{on } \Gamma^{ij}(t). \]

It follows similarly to Section 2.2.1 that

\[ \gamma = \begin{pmatrix} 0 & \beta_{12} & \beta_{23} \\ \beta_{12} & 0 & \beta_{23} \\ \beta_{13} & \beta_{23} & 0 \end{pmatrix} \]

and additionally the force balance at the triple junction Eq. (2.101) is equivalent to Eq. (2.59) as shown in Example 2.1.22.

Balance laws

Clearly Eq. (2.54) is not present in Eqs. (2.98) to (2.100) as we have already accounted for all the equations. We therefore leave \( E, Q \) and \( F \) to be set as their default options as per Table 2.1.
Boundary Conditions

\[ \Omega = [0, 0], [2, 2], T = 0.5, \tau_0 = 1 \text{ No-flux} \]

Initial Conditions

\[ \Omega^1_I = \{ \bar{x} \in \Omega : |\bar{x}|^2 < 0.8 \}, \]
\[ \Omega^2_I = \{ \bar{x} \in \Omega : |\bar{x}|^2 > 0.8 \} \]

Energy Balance Laws

\[
\begin{pmatrix}
0 & \beta_{12} & \beta_{13} \\
\beta_{21} & 0 & \beta_{23} \\
\beta_{13} & \beta_{23} & 0
\end{pmatrix}
\]

Table 2.10: Summary of functions and parameters for multi-phase mean curvature flow.

**Computational parameters**

For the domain we choose \( \Omega = [0, 2]^2 \) and the constants of the problem as

\[ \beta_{12} = \beta_{13} = \beta_{23} = 1, \quad T = 0.5. \]

Comparing Eq. (2.67) to Eq. (2.1) we use the characteristic function of the configuration to specify the initial conditions, which in this case are,

\[
\begin{align*}
\chi_{\Omega^1_I}(\bar{x}) &= \begin{cases} 
1 & \text{if } \bar{x}_1 > 1 \text{ and } \bar{x}_2 > 1, \\
0 & \text{otherwise.}
\end{cases} \\
\chi_{\Omega^2_I}(\bar{x}) &= \begin{cases} 
1 & \text{if } \bar{x}_1 > 1 \text{ and } \bar{x}_2 < 1, \\
0 & \text{otherwise.}
\end{cases} \\
\chi_{\Omega^3_I}(\bar{x}) &= 1 - \chi_{\Omega^1_I}(\bar{x}) - \chi_{\Omega^2_I}(\bar{x}).
\end{align*}
\]  

(2.103)

The implementation of this function can be seen in Listing 13.

**Summary and code**

We summarise the discussion above in Table 2.10 which states the functions and parameters to define a multi-phase mean curvature flow as a \( \mathcal{P} \) gradient flow.

The class is called **Mcf3p**. The specification of the model in **PHASEFIELD** can be seen in Listing 13.
class Mcf3p:
    omega = cartesianDomain([0,0],[2,2],[3,3])
    mobility = 1
    endTime = 0.5

    def gamma(nu):
        return [[0,1,1],[1,0,1],[1,1,0] ]

    def initial(x):
        phase1 = conditional(x[0]>1, 1, 0)*conditional(x[1]>1, 1, 0)
        phase2 = conditional(x[0]>1, 1, 0)*conditional(x[1]<1, 1, 0)
        phase3 = 1-phase1-phase2
        return [[phase1, phase2, phase3]]

Listing 13: Multi-phase mean curvature flow class definition.
Chapter 3

From Sharp Model to Phase Field Model

We use the phase field method as a tool to produce approximations of $\mathcal{P}$ gradient flows. We mirror the presentation of Chapter 2 to make the analogy between various objects in the sharp and phase field setting clear. We describe phase field evolution in their own right in Section 3.1 before making the link between the sharp and phase field evolutions in Section 3.2.

We introduce notation in Section 3.1 before defining balance laws in the phase field setting in Section 3.1.1. In Section 3.1.2 we introduce an energy $\mathcal{F}^\varepsilon$ and describe how to vary it. This allows us to define system of PDEs in Section 3.1.3 which we call a $\mathcal{P}^\varepsilon$ gradient flow. As discussed in Chapter 1, $\varepsilon$ is a regularisation parameter proportional to the width of the interface.

In Section 3.2 we describe the link between $\mathcal{P}$ and $\mathcal{P}^\varepsilon$ gradient flows. Algorithms are provided in pseudocode to transform a $\mathcal{P}$ into a $\mathcal{P}^\varepsilon$ gradient flow. As $\varepsilon \to 0$ we expect to recover a $\mathcal{P}$ gradient flow. This convergence is motivated by a number of analytical results including; $\Gamma$-convergence of the underlying functionals, formal asymptotics and rigorous convergence results of the PDE system.

Once this transformation is complete, the $\mathcal{P}^\varepsilon$ gradient flow is discretised in time to produce a set of semi-discrete phase field equations in UFL. By default, a backward Euler scheme is produced. We describe how to implement additional time discretisations in Chapter 4.

In Chapter 2 we presented code demonstrating how to define a $\mathcal{P}$ gradient flow in PHASEFIELD by creating a PYTHON class. In Section 3.4 we show how to transform this same class into a $\mathcal{P}^\varepsilon$ gradient flow in PHASEFIELD. This is done by passing the sharp class to what we call a PhaseModel class. The constructor to
the `PhaseModel` class takes $\epsilon$, the time-step $\Delta t$ and a potential well $W$. In Section 3.5 we produce and explain how to create the `PhaseModel` class for the same examples as in Section 2.2.

### 3.1 Phase field evolutions

As in Chapter 2 assume we have a bounded square domain $\Omega \subset \mathbb{R}^2$. In Chapter 2 we produced an evolution by considering how the points in each $(n)$ hypersurface $\Omega^i$, $i = 1, \ldots, m$ evolve, in this setting to describe the evolution we use a smooth vector valued order parameter $\hat{\varphi}^\epsilon : \Omega \times [0, \infty] \to \mathbb{R}^m$, where $m$ is the number of phases. The $i^{th}$ component of $\hat{\varphi}^\epsilon$ describes the local fraction of phase $i$. As a matter of convenience we write $\hat{\varphi}$, as the dependence on $\epsilon$ is always clear.

To describe physical phenomena in $\Omega$ we have fields $\tilde{u}^\epsilon : \Omega \to \mathbb{R}^r$ assumed to be sufficiently smooth in $\Omega$. Because of the structure of the problems considered in the $\epsilon$ limit we expect to recover the $\tilde{u}$ in Definition 2.1.19. Here again we drop the $\epsilon$ dependence and write $\tilde{u}$. The specific $\tilde{u}$ being considered is clear from the context.

When defining the appropriate function spaces for $\hat{\varphi}$, there are a number of conditions enforced to ensure that the evolution of $\hat{\varphi}$, approximates $\hat{\chi}_{G(t)}$ of an appropriate problem in the sharp setting. We define the following two spaces that enforce the condition that when we have more than one-phase ($m \geq 2$), the components of the phase field variables must sum to one.

$$
\Sigma := \{ \hat{p} \in \mathbb{R}^m : \sum_{i=1}^{m} \hat{p}_i = 1, \text{ if } m \geq 2 \},
$$

$$
T\Sigma := \{ \hat{p} \in \mathbb{R}^m : \sum_{i=1}^{m} \hat{p}_i = 0, \text{ if } m \geq 2 \}.
$$

The intersection of $\Sigma$ and the unit cube is know in the literature as the Gibbs Simplex. We introduce the following spaces for $\hat{\varphi}$

$$
G^\epsilon := H^1(\Omega, \Sigma),
$$

$$
T G^\epsilon := H^1(\Omega, T\Sigma).
$$

We will require that $\hat{\varphi}$ and $\partial_t \hat{\varphi}$ belong to $G^\epsilon$ and $T G^\epsilon$ respectively in addition to some further constrains specified in Algorithm 3.

We define the operator $P_{T\Sigma}[\cdot]$ to project a vector $\hat{p} \in \mathbb{R}^m$ or matrix $\hat{p} \in \mathbb{R}^{m \times m}$ onto $T\Sigma$ or $(T\Sigma)^m$ respectively. We define the projection onto the tangent space
when \( m \geq 2 \) as
\[
\mathbb{P}_{\Sigma}[\hat{p}] := \hat{p} - \frac{1}{m} \left( \sum_{i=1}^{m} \hat{p}_i \right) \mathbbm{1}_m,
\]
where \( \mathbbm{1}_m = (1, 1, \ldots, 1) \in \mathbb{R}^m. \) (3.3)

When only one phase is present \( (m = 1) \) the projection operator is defined
as \( \mathbb{P}_{\Sigma}[\hat{p}] = \hat{p}. \)

3.1.1 Balance laws

To fix how \( \tilde{u} \) behaves in \( \Omega \) we specify balance laws. As \( \tilde{u} \) in the phase field setting
is smooth, we do not get any extra conditions such as in Eq. (2.25) that must hold
on the interface when the equations are written in strong form. For \( i = 1, \ldots, r \) let
\( E^i \in C^\infty(\mathbb{R}^r, \mathbb{R}^r) \) denote the set of \( r \) scalar fields, by \( Q^i \in C^\infty(\mathbb{R}^r \times \mathbb{R}^{r \times 2}, \mathbb{R}^{r \times 2}) \) we
denote the corresponding fluxes and by \( F^i \in C^\infty(\mathbb{R}^r \times \Omega, \mathbb{R}^r) \) the source terms. We
write
\[
E = (E^1, \ldots, E^m) \in C^\infty(\mathbb{R}^r, \mathbb{R}^{r \times m}),
\]
\[
Q = (Q^1, \ldots, Q^m) \in C^\infty(\mathbb{R}^r \times \mathbb{R}^{r \times 2}, \mathbb{R}^{r \times 2 \times m}),
\]
\[
F = (F^1, \ldots, F^m) \in C^\infty(\mathbb{R}^r \times \Omega, \mathbb{R}^{r \times m}).
\]

Given these functions we define the following operators which are smoothed versions
of the distributions in Eqs. (2.21) to (2.23). Let \( f, q : \mathbb{R}^m \to \mathbb{R}^{r \times m} \), be functions
where for each \( j = 1, \ldots, r \), \( f_j, q_j : \mathbb{R}^m \to \mathbb{R}^m \) are phase field interpolation functions
in the sense of Definition 3.1.2. We use \( \odot \) to denote the Hadamard or entry-wise
product.
\[
E^\epsilon_{\hat{\phi}}(\tilde{u}(\vec{x}, t)) := \sum_{i=1}^{m} E(\tilde{u}(\vec{x}, t))^i \hat{\phi}_i(\vec{x}, t),
\]
\[
(Q^\epsilon_{\hat{\phi}}(\tilde{u}(\vec{x}, t), \nabla \tilde{u}(\vec{x}, t)))^j := \sum_{i=1}^{m} Q(\tilde{u}(\vec{x}, t), \nabla \tilde{u}(\vec{x}, t))^i_q(\hat{\phi}(\vec{x}, t))^i_j,
\]
\[
F^\epsilon_{\hat{\phi}}(\tilde{u}(\vec{x}, t), \vec{x}) := \sum_{i=1}^{m} (F(\tilde{u}(\vec{x}, t), \vec{x}) \odot f(\hat{\phi}(\vec{x}, t)))^i,
\]
In the following we drop the arguments of $E_{\hat{\phi}}, Q_{\hat{\phi}}, F_{\hat{\phi}}$ for ease of presentation. We define the following balance laws in $\Omega$, $\forall \tilde{\zeta} \in C_0^\infty(\Omega \times [0, T], \mathbb{R}^r)$

\[
0 = \int_0^T \int_{\Omega} E_{\hat{\phi}} \cdot \partial_t \tilde{\zeta} + Q_{\hat{\phi}} : \nabla \tilde{\zeta} + F_{\hat{\phi}} \cdot \tilde{\zeta} \, dL^3.
\] (3.10)

In addition to this, boundary conditions for $\tilde{u}$ must be specified which hold on $\partial \Omega$.

The presence of $\hat{\phi}$ in Eq. (3.10) restricts the operators $E, Q$ and $F$ to the relevant part of $\Omega$ where $\hat{\phi}$ is non-zero. In the simplest case $r = 1$ we have

\[
q(\hat{\phi}) = f(\hat{\phi}) = \hat{\phi}.
\]

If one wants to kill off unwanted bulk behaviour more rapidly higher order terms can be taken such as

\[
q(\hat{\phi}) = f(\hat{\phi}) = (\hat{\phi}_1^2 \cdots \hat{\phi}_m^2).
\]

### 3.1.2 Energy, variation and gradient

We produce evolutions by considering the variation of a given functional $\mathcal{F}(\hat{\phi}, \tilde{u})$. This is in analogue to Section 2.1.4. The energy is assumed to be of Ginzburg Landau form and for $\hat{\phi} \in \mathcal{G}, \tilde{u} \in H^1(\Omega, \mathbb{R}^r)$ is defined as

\[
\mathcal{F}(\hat{\phi}, \tilde{u}) = \int_{\Omega} \epsilon \Lambda(\nabla \hat{\phi}) + \frac{1}{\epsilon} W(\hat{\phi}) + I(\hat{\phi}, \hat{a}(\tilde{u}, \nabla \tilde{u})) \, dL^2.
\] (3.11)

Here, $\Lambda$ is the gradient part of the energy that takes surface energy contributions into account. The potential free energy is given by

\[
\frac{1}{\epsilon} W(\hat{\phi}) + I(\hat{\phi}, \hat{a}(\tilde{u}, \nabla \tilde{u})),
\]

where $W$ is a smooth potential and $I$ an interpolation function. The well $W$ will often be a double or single potential well. These energies are chosen to ensure $\Gamma$-convergence of $\mathcal{F}^\epsilon$ to $\mathcal{F}^s$, where $\mathcal{F}^s$ is a given sharp energy. As in Chapter 2, let us give some further details on the energy contributions.

The gradient energy density $\Lambda : (T\mathcal{G})^n \to \mathbb{R}$ is assumed to be smooth away from 0, non-negative, and homogeneous degree two, i.e.,

\[
\Lambda(X) \geq 0, \quad \text{and} \quad \Lambda(\eta X) = \eta^2 \Lambda(X) \quad \forall X \in (T\mathcal{G})^n \text{ and } \forall \eta \in \mathbb{R}^+.
\]
An example for the gradient part of the energy [17] is
\[
\Lambda(\nabla \hat{\phi}) = \sum_{i=1}^{m} \frac{\epsilon}{2} |\nabla \hat{\phi}_i|^2,
\]
or the so-called “irreducible differences” [67] later developed in [47]
\[
\Lambda(\nabla \hat{\phi}) = \sum_{i<j} \gamma_{ij}(\hat{\phi}_i \nabla \hat{\phi}_j - \hat{\phi}_j \nabla \hat{\phi}_i)^2. \tag{3.12}
\]
The well \(W\) ensures that the phase field variables \(\hat{\phi}\) energetically prefer to stay in one of the pure states. Let \(\delta_{ij}\) be the normal Kronecker delta symbol. \(W : \mathbb{R}^m \to \mathbb{R}\) is assumed to be a smooth function with exactly \(m\) global minima at the points \(e_j = (\delta_{ij})_{i=1}^{m}\), \(1 \leq \beta \leq m\) with \(W(e_j) = 0\) i.e. for \(r \in \mathbb{R}^m\),
\[
W(r) \geq 0, \quad \text{and} \quad W(r) = 0 \iff r = e_j \text{ for some } j \in \{1, \cdots, m\}.
\]
Observe that \(e_j\) are the corners of the Gibbs simplex and global minima for the well \(W\). A common choice in the two phase case is [18]
\[
W(\hat{\phi}) = (\hat{\phi}_1)^2(\hat{\phi}_2)^2.
\]
Further examples are given in Section 3.4.3.

**Remark 3.1.1.** It is common in the literature for \(W\) to be chosen to have minima at \(-1\) and \(1\) such as \(W(\hat{\phi}) = (1 + \hat{\phi}_1)^2(1 - \hat{\phi}_2)^2\). To make the analogy with \(\tilde{\chi}_G\) clear we always assume that the minima lie at 0 and 1.

We will need the following functions

**Definition 3.1.2.** A phase field interpolation function \(h\) is a smooth function \(h : \mathbb{R}^m \to \mathbb{R}^m\) such that
\[
h(e_i) = e_i, \quad \frac{d}{dx} h(x e_i) \geq 0, \quad 0 \leq x \leq 1, \quad 1 \leq i \leq m.
\]

We also introduce an interpolation function \(I\) which interpolates smoothly between the bulk phases \(I : \mathcal{G}^\epsilon \times \mathbb{R}^m \to \mathbb{R}\).
\[
I(\hat{\phi}, \hat{a}(\tilde{u}, \nabla \tilde{u})) = \sum_{i=1}^{m} \epsilon_i(\hat{\phi}) \hat{a}_i(\tilde{u}, \nabla \tilde{u}). \tag{3.13}
\]
with phase field interpolation function $h$. This ensures that

$$ I(e_i, \bar{a}(\bar{u}, \nabla \bar{u})) = a_i(\bar{u}, \nabla \bar{u}). $$

Having defined the energy $F_\epsilon$ (Eq. (3.11)) we continue in the same manner as in Chapter 2. First by defining and computing a variation of the energy and secondly by using an appropriate inner product to define a gradient flow. We let $\xi \in \mathbb{R}$ be a small parameter. Similarly to Eq. (2.40) we wish to keep $\bar{u}$ fixed and as $\bar{u}$ is continuous in $\Omega$ we do not vary it at all with $\xi$.

We first compute the variation on the full space $H^1(\Omega, \mathbb{R}^m)$.

**Definition 3.1.3.** The variation of an energy $F_\epsilon(\hat{\phi}, \bar{u})$ in the direction $\hat{\eta} \in H^1(\Omega, \mathbb{R}^m)$ is

$$ \langle \delta F_\epsilon(\hat{\phi}, \bar{u}), \hat{\eta} \rangle := \frac{d}{d\xi} F_\epsilon(\hat{\phi} + \xi \hat{\eta}, \bar{u}) \bigg|_{\xi=0}. $$

We apply Definition 3.1.3 to $F_\epsilon$ Eq. (3.11), as is done in [45]. A standard computation shows that the variation of $F_\epsilon$ in the direction $\hat{\eta} \in H^1(\Omega, \mathbb{R}^m)$ is

$$ \langle \delta F_\epsilon(\hat{\phi}, \bar{u}), \hat{\eta} \rangle = \int_{\Omega} \epsilon \Lambda'(\nabla \hat{\phi}) : \nabla \hat{\eta} + \frac{1}{\epsilon} W'(\hat{\phi}) \cdot \hat{\eta} \; d\mathcal{L}^2 + I'(\hat{\phi}, \bar{a}(\bar{u}, \nabla \bar{u})) \cdot \hat{\eta} \; d\mathcal{L}^2. \quad (3.14) $$

We would like to consider variations in the direction $T\mathcal{G}_\epsilon$. If $\hat{\eta} \in H^1(\Omega, \mathbb{R}^m)$ then $\mathbb{P}_{T\Sigma}[\hat{\eta}]$ maps onto $T\mathcal{G}_\epsilon$. A calculation along the lines of [3] using Eq. (3.14) gives for $\hat{\eta} \in H^1(\Omega, \mathbb{R}^m)$

$$ \langle \delta F_\epsilon(\hat{\phi}, \bar{u}), \mathbb{P}_{T\Sigma}[\hat{\eta}] \rangle = \int_{\Omega} \epsilon \mathbb{P}_{T\Sigma}[\Lambda'(\nabla \hat{\phi})] : \nabla \hat{\eta} + \mathbb{P}_{T\Sigma}[\frac{1}{\epsilon} W'(\hat{\phi}) + I'(\hat{\phi}, \bar{a}(\bar{u}, \nabla \bar{u}))] \cdot \hat{\eta} \; d\mathcal{L}^2, $$

$$ = \int_{\Omega} -\epsilon \nabla \cdot \mathbb{P}_{T\Sigma}[\Lambda'(\nabla \hat{\phi})] \cdot \hat{\eta} + \mathbb{P}_{T\Sigma}[\frac{1}{\epsilon} W'(\hat{\phi}) + I'(\hat{\phi}, \bar{a}(\bar{u}, \nabla \bar{u}))] \cdot \hat{\eta} \; d\mathcal{L}^2. $$

To obtain the last line the boundary conditions

$$ \mathbb{P}_{T\Sigma}[\Lambda'(\nabla \hat{\phi})] \cdot \vec{n}_{\partial \Omega} = \bar{0}, \quad (3.16) $$

has been imposed. In analogy to the sharp inner product Eq. (2.51) we introduce an inner product on $T\mathcal{G}_\epsilon$. This link will be made more explicit in Section 3.2.

To take into account the correct time rescaling as well as mobility we intro-
duce the weighted $L^2$ product on $T\mathcal{G}^\epsilon$.

$$
(\hat{w}, \hat{v})_{\hat{\phi}, \tau_0} := \epsilon^{\min(1, m-1)} \tau_0 \int_\Omega \hat{w} \cdot \hat{v} \, dL^2, \quad \forall \hat{w}, \hat{v} \in T\mathcal{G}^\epsilon.
$$

(3.17)

The scaling done on the $\epsilon$ is important and is done to ensure that in the $\epsilon$ limit we recover the correct sharp equations. The $m = 1$ case evolutions happen at different time-scale (see Section 3.5.5) which is why we need this special choice. As the $\epsilon$ scaling is done automatically we do not indicate this in the notation.

**Remark 3.1.4.** The LHS of Eq. (3.17) contains $\hat{\phi}$ however there is no dependence on this on the RHS. This is done to make the inner product analogous to $(\cdot, \cdot)_{\tau_G, G}$ in the sharp setting, see Eq. (2.51).

Defining the gradient $\nabla F^\epsilon$ as the dual element under the inner product in Eq. (3.17) we get

$$
\nabla F^\epsilon(\hat{\phi}, \tilde{u}) =
$$

(3.18)

$$
\epsilon^{-\min(1, m-1)} \tau_0^{-1} \epsilon \left[ -\nabla \cdot P_{T\Sigma}[\Lambda'(\nabla \hat{\phi})] + P_{T\Sigma}[\frac{1}{\epsilon} W'(\hat{\phi}) + I'(\hat{\phi}, \hat{a}(\tilde{u}, \nabla \tilde{u}))] \right].
$$

We later use this definition of the gradient to define a gradient flow of $F^\epsilon$.

### 3.1.3 Gradient flow

We have introduced the relevant objects to define a weighted gradient flow in analogy to Definition 2.1.19 in the sharp setting. There are two different forms for our phase field equations depending on whether $m = 1$ or $m \geq 2$. We first define the case for $m \geq 2$ as the setting is more standard.

**Definition 3.1.5** ($P^\epsilon$ Gradient flow, $m > 1$). Given the initial condition $\tilde{u}_I$, initial phase field $\hat{\phi}_I$, functions $E_{\hat{\phi}}, Q_{\hat{\phi}}, F_{\hat{\phi}}$, energy $F^\epsilon$, mobility $\tau_0 \in \mathbb{R}$, final time $T$, set $\mathcal{K}^\epsilon$ and boundary conditions for $\tilde{u}$. A $P^\epsilon$ gradient flow when $m > 1$ is the family of functions $\varphi(\cdot, t), \tilde{u}(\cdot, t), \forall t \in [0, T]$, such that $\forall \tilde{\eta} \in \mathcal{K}^\epsilon$,

$$
(\partial_t \hat{\phi}, \hat{\phi} - \tilde{\eta})_{\hat{\phi}, \tau_0} \geq -(\nabla F^\epsilon(\hat{\phi}, \tilde{u}), \hat{\phi} - \tilde{\eta})_{\hat{\phi}, \tau_0},
$$

(3.19)

and such that $\forall \zeta \in C_0^\infty(\Omega \times [0, T], \mathbb{R}^r)$,

$$
0 = \int_0^T \int_\Omega E_{\hat{\phi}}^\epsilon \cdot \partial_t \zeta + Q_{\hat{\phi}}^\epsilon \cdot \nabla \zeta + F_{\hat{\phi}}^\epsilon \cdot \zeta \, dL^3,
$$

(3.20)
and
\[
\begin{align*}
\tilde{u}(\cdot, 0) &= \tilde{u}_I(\cdot), \\
\hat{\phi}(\cdot, 0) &= \hat{\phi}_I(\cdot).
\end{align*}
\] (3.21) (3.22)

In addition to boundary conditions for \(\tilde{u}\) and the boundary conditions for \(\hat{\phi}\) in Eq. (3.16)

\[
P_{\mathcal{T}_\Sigma}[\Lambda'(\nabla \hat{\phi})] : \mathbf{\tilde{v}}_{\partial \Omega} = \mathbf{0}.
\]

In the case \(m = 1\) we wish to ensure that the time derivative of \(\varphi m\) is negative to ensure the irreversibility of the crack. Rather than considering the evolution

\[
\partial_t \hat{\phi} = -\nabla F(\hat{\phi}, \tilde{u})
\] (3.23)

we project the right-hand side onto the appropriate space. We make use of the following projection theorem

**Theorem 3.1.6** (Theorem 2.1 [63]). Let \(K\) be a closed, non-empty, convex subset of a Hilbert space \(V\). For any \(w \in V\), there exists a unique \(u = P_K w \in K\), called the projection of \(w\) on \(K\), characterized by the following conditions:

\[
\begin{align*}
&u = P_K w \in K : \|u - w\| \leq \|v - w\|, \quad \forall v \in K, \\
&u = P_K w \in K : (u - w, v - u) \geq 0, \quad \forall v \in K.
\end{align*}
\] (3.24) (3.25)

Applying Theorem 3.1.6 we project the RHS of Eq. (3.23) onto \(T \mathcal{G}^\epsilon\) and get a variational inequality. The problem is then to find \(\hat{\phi}\) such that \(\forall \tilde{\eta} \in T \mathcal{G}^\epsilon\)

\[
(\partial_t \hat{\phi}, \tilde{\eta} - \partial_t \hat{\phi})_{\hat{\phi}, \tau_0} \geq - (\nabla F(\hat{\phi}, \tilde{u}), \tilde{\eta} - \partial_t \hat{\phi})_{\hat{\phi}, \tau_0}.
\] (3.26)

We now couple this to the balance laws in Section 3.1.1 to define a \(\mathcal{P}^\epsilon\) gradient flow in the case that \(m = 1\).

**Definition 3.1.7** (\(\mathcal{P}^\epsilon\) Gradient flow, \(m = 1\)). Given the initial condition \(\tilde{u}_I\), initial phase field \(\hat{\phi}_I\), functions \(E_{\hat{\phi}}, Q_{\hat{\phi}}, F_{\hat{\phi}}\), energy \(F^\epsilon\), mobility \(\tau_0 \in \mathbb{R}\), final time \(T\), set \(\mathcal{K}^\epsilon\) and boundary conditions for \(\tilde{u}\). A \(\mathcal{P}^\epsilon\) gradient flow is the family of functions \(\varphi(\cdot, t), \tilde{u}(\cdot, t), \forall t \in [0, T]\), such that \(\forall \tilde{\eta} \in T \mathcal{G}^\epsilon\),

\[
(\partial_t \hat{\phi}, \tilde{\eta} - \partial_t \hat{\phi})_{\hat{\phi}, \tau_0} \geq - (\nabla F(\hat{\phi}, \tilde{u}), \tilde{\eta} - \partial_t \hat{\phi})_{\hat{\phi}, \tau_0}.
\] (3.27)
and such that $\forall \tilde{\zeta} \in C_0^\infty(\Omega \times [0, T], \mathbb{R}^r)$,

$$0 = \int_0^T \int_\Omega E_{\tilde{\varphi}} \cdot \partial_t \tilde{\zeta} + Q_{\tilde{\varphi}} \cdot \nabla \tilde{\zeta} + F_{\tilde{\varphi}} \cdot \tilde{\zeta} \, dL^3,$$  \hspace{1cm} (3.28)

and

$$\tilde{u}(\cdot, 0) = \tilde{u}_I(\cdot), \hspace{1cm} (3.29)$$
$$\hat{\varphi}(\cdot, 0) = \hat{\varphi}_I(\cdot). \hspace{1cm} (3.30)$$

In addition to boundary conditions for $\tilde{u}$ and the boundary conditions for $\hat{\varphi}$ in Eq. (3.16)

$$\mathbb{P}_{T\Sigma}[\Lambda'(\nabla \tilde{\varphi})] \cdot \nu_{\partial \Omega} = \vec{0}.$$

**Remark 3.1.8.** If $K_\epsilon = TG^e$ then Eq. (3.27) becomes the equality Eq. (3.23).

### 3.1.4 Rewriting the gradient flow

We describe a reformulation of the $P^e$ gradient flows which will motivate the solving of the system in the time-discrete setting \[48\]. The balance laws Eqs. (3.19) and (3.20) are written in an abstract form with a functional $J_B^\epsilon$ defined as

$$J_B^\epsilon(\hat{\varphi}, \tilde{u}, \tilde{\zeta}) := \int_\Omega \partial_t E_{\hat{\varphi}} \cdot \hat{\zeta} - Q_{\hat{\varphi}} \cdot \nabla \hat{\zeta} - F_{\hat{\varphi}} \cdot \hat{\zeta} \, dL^3.$$  \hspace{1cm} (3.31)

Let $J_{PF}^\epsilon$ be a functional defined as

$$J_{PF}^\epsilon(\hat{w}, \tilde{u}, \hat{v}) := (\partial_t \hat{w} + \nabla F^\epsilon(\hat{w}, \tilde{u}), \hat{v})_{\hat{\varphi}, 0}.$$  \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} (3.31)

Rewriting Eqs. (3.27) and (3.28) if $m = 1$ and Eqs. (3.19) and (3.20) if $m > 1$ we get the constrained minimisation problem

**Case $m = 1$**

Find $\hat{\varphi}$ and $\tilde{u}$ such that $\forall t \in [0, T]$

$$J_{PF}^\epsilon(\hat{\varphi}, \tilde{u}, \hat{\eta}) \geq J_{PF}^\epsilon(\hat{\varphi}, \tilde{u}, \partial_t \hat{\varphi}), \hspace{1cm} \forall \hat{\eta} \in TG^e, \hspace{1cm} (3.32)$$

$$J_B^\epsilon(\hat{\varphi}, \tilde{u}, \tilde{\zeta}) = 0, \hspace{1cm} \forall \tilde{\zeta} \in C_0^\infty(\Omega, \mathbb{R}^r). \hspace{1cm} (3.33)$$

70
Case \( m > 1 \)

Find \( \hat{\varphi} \) and \( \tilde{u} \) such that \( \forall t \in [0, T] \)

\[
J_{PF}(\hat{\varphi}, \tilde{u}, \tilde{\eta}) \geq J_{PF}(\hat{\varphi}, \tilde{u}, \hat{\varphi}), \quad \forall \tilde{\eta} \in \mathcal{K}^\epsilon, \tag{3.34}
\]

\[
J_{B}(\hat{\varphi}, \tilde{u}, \tilde{\zeta}) = 0, \quad \forall \tilde{\zeta} \in C_0^\infty(\Omega, \mathbb{R}^r). \tag{3.35}
\]

### 3.2 Linking sharp and phase field evolutions

Having described the \( \mathcal{P}^\epsilon \) gradient flow we describe the algorithm for approximating a \( \mathcal{P} \) gradient flow with a \( \mathcal{P}^\epsilon \) gradient flow. We approach the problem in the same order it has previously been defined. First by regularising \( F^s \) in Section 3.2.1 to produce \( F^\epsilon \). We next regularise the balance laws in Section 3.2.2 and the convex set \( \mathcal{K}(G) \) in Section 3.2.3. We describe the entire algorithm in Section 3.2.4 which in addition takes into account boundary and initial conditions. Having derived the \( \mathcal{P}^\epsilon \) gradient flow we finish by discretising the equations in time in Section 3.3. This is to give the form of the semi-discrete equations solved by PHASEFIELD.

We do not discuss the problem of calibrating the energies \( F^\epsilon \). Appropriate constants have been chosen in the energies to ensure that the regularised phase field energy \( F^\epsilon \) give the correct value in the limit. The required calculations can be found in Appendix B.1.

### 3.2.1 Regularisation of energies

As described in Chapter 1 we derive a phase field energy \( F^\epsilon \) from a given sharp energy \( F^s \). This is done by building energies that agree with known \( \Gamma \)-convergence results in specific cases by choosing \( \Lambda, W \) and \( I \) appropriately.

We give four different models where \( \Lambda \) and \( W \) are chosen automatically depending on the number of phases and type of energy specified. In all the models we let \( I \) be defined as Eq. (3.13) with \( h(\hat{\varphi}) := \hat{\varphi} \), which gives

\[
I(\hat{\varphi}, \hat{a}(\tilde{u}, \nabla \tilde{u})) = \sum_{i=1}^{m} \hat{\varphi}_i \hat{a}_i(\tilde{u}, \nabla \tilde{u}). \tag{3.36}
\]

When writing the models we put the names in bold italics for clarity. For example when referring to the one phase model we shall write, **Model one-phase.**
**Model one-phase**

Let \( m = 1 \) and therefore only one phase is present, we choose

\[
\Lambda(\nabla \phi) = 2\gamma_{11}|\nabla \hat{\phi}|^2, \quad \text{(3.37)}
\]

\[
W(\hat{\phi}) = \frac{\gamma_{11}}{2} (1 - \hat{\phi})^2. \quad \text{(3.38)}
\]

Substituting Eq. (3.37) Eq. (3.38) in Eq. (3.11) gives the phase field energy

\[
F^\epsilon = \int_{\Omega} 2\gamma_{11}|\nabla \phi|^2 + \frac{1}{\epsilon} \frac{\gamma_{11}}{2} (1 - \hat{\phi})^2 + \hat{\phi} \cdot a(\bar{u}, \nabla \bar{u}) \, d\mathcal{L}^2.
\]

The relevant \( \Gamma \)-convergence result is found in [14] where it was shown that

\[
F^\epsilon = \int_{\Omega} \left( 2\epsilon |\nabla \hat{\phi}|^2 + \frac{(1 - \hat{\phi})^2}{2\epsilon} \right) + (\hat{\phi}^2 + k_\epsilon)|\nabla \bar{u}|^2 \, d\mathcal{L}^2, \quad \text{(3.39)}
\]

\( \Gamma \)-converges to

\[
F^s = \int_{\Gamma_1} d\mathcal{H}^1 + \int_{\Omega^1} |\nabla \bar{u}|^2 \, d\mathcal{H}^2. \quad \text{(3.40)}
\]

The Eq. (3.40) fits into our framework with, \( m = 1, r = 1, n = 2 \) and the following functions defined in the sharp energy \( F^s \) in Eq. (2.37)

\[
\gamma = \sigma,
\]

\[
\hat{a}(\bar{u}, \nabla \bar{u}) = |\nabla \bar{u}|^2.
\]

In addition to Eqs. (3.37) and (3.38), substituting the following interpolation function \( I \) into the energy \( F^\epsilon \) in Eq. (3.11) gives Eq. (3.39)

\[
I(\hat{\phi}, \hat{a}(\bar{u}, \nabla \bar{u})) = \hat{\phi}^2 + k_\epsilon. \quad \text{(3.41)}
\]

**Remark 3.2.1.** The Eq. (3.39) has an extra regularisation parameters \( k_\epsilon \) which is known as the residual stiffness of the crack. This can be modified into the energy by editing the smoothing parameter \( I \) see Eq. (3.41). To be consistent with the other models the default interpolation functions Eq. (3.36) is used.
Model multi-iso

We next deal with multi-phase isotropic energies. We take the energy for \((m = 2, 3)\) from [16],

\[
\Lambda(\nabla \hat{\varphi}) = \frac{1}{2} \left( \gamma_{12} \quad \gamma_{13} \quad \gamma_{23} \right) \left( \begin{array}{ccc}
1 & 1 & -1 \\
1 & -1 & 1 \\
-1 & 1 & 1 \\
\end{array} \right) \left( \begin{array}{c}
|\nabla \hat{\varphi}_1|^2 \\
|\nabla \hat{\varphi}_2|^2 \\
|\nabla \hat{\varphi}_3|^2 \\
\end{array} \right), \quad (3.42)
\]

\[
W(\hat{\varphi}) = \frac{9}{2} \left( \gamma_{12} \quad \gamma_{13} \quad \gamma_{23} \right) \left( \begin{array}{ccc}
1 & 1 & -1 \\
1 & -1 & 1 \\
-1 & 1 & 1 \\
\end{array} \right) \left( \begin{array}{c}
(\hat{\varphi}_1)^2(1 - \hat{\varphi}_1)^2 \\
(\hat{\varphi}_2)^2(1 - \hat{\varphi}_2)^2 \\
(\hat{\varphi}_3)^2(1 - \hat{\varphi}_3)^2 \\
\end{array} \right). \quad (3.43)
\]

Compared to other possible choices for the gradient term, for example Eq. (3.12), the choice Eq. (3.42) has the numerical advantage of having linear gradient terms. Taking Eqs. (3.42) and (3.43) in the energy Eq. (3.11) gives

\[
\mathcal{F} = \int_{\Omega} \left( \gamma_{12} \quad \gamma_{13} \quad \gamma_{23} \right) \left( \begin{array}{ccc}
1 & 1 & -1 \\
1 & -1 & 1 \\
-1 & 1 & 1 \\
\end{array} \right) \left[ \frac{\epsilon}{2} \left( \frac{1}{|\nabla \hat{\varphi}_1|^2} \right) + \frac{9}{2\epsilon} \left( \frac{1}{|\nabla \hat{\varphi}_1|^2} \right) \right] \, d\mathcal{L}^2. \quad (3.44)
\]

The energy defined by Eq. (3.44) was shown in [21] to \(\Gamma\) converge to the multi-phase perimeter function Eq. (2.37).

\[
\mathcal{F} = \int_{\Gamma_{12}} \gamma_{12} \, d\mathcal{H}^1 + \int_{\Gamma_{13}} \gamma_{13} \, d\mathcal{H}^1 + \int_{\Gamma_{23}} \gamma_{23} \, d\mathcal{H}^1.
\]

Remark 3.2.2. The convergence result proved in [21] was even more general and allowed for \(n\) – phase isotropic energies.

If there are only two phases present in the system taking \(\hat{\varphi}_3(\vec{x}) = 0, \forall \vec{x} \in \Omega\) and \(\gamma_{13} = \gamma_{23} = 0\) in Eq. (3.44) gives the two-phase Ginzburg-Landau energy

\[
\mathcal{F}(\hat{\varphi}) = \int_{\Omega} \gamma_{12} \frac{\epsilon}{2} \left[ |\nabla \hat{\varphi}_1|^2 + |\nabla \hat{\varphi}_2|^2 \right] + \gamma_{12} \frac{9}{2\epsilon} \left[ (\hat{\varphi}_1)^2(1 - \hat{\varphi}_1)^2 + (\hat{\varphi}_2)^2(1 - \hat{\varphi}_2)^2 \right] \, d\mathcal{L}^2.
\]
**Model aniso-nu**

When two-phases \((m = 2)\) are present and the energy is anisotropic, we choose the following two functions for the gradient density and potential well.

\[
\Lambda(\nabla \dot{\varphi}) = \gamma_{12}(\nabla \dot{\varphi}_1)^2, \tag{3.45}
\]
\[
W(\dot{\varphi}) = 9(\dot{\varphi}_1)^2(\dot{\varphi}_2)^2. \tag{3.46}
\]

Substituting Eq. (3.45) and Eq. (3.45) into Eq. (3.11) gives

\[
\mathcal{F}^e = \int_\Omega \epsilon \gamma_{12}(\nabla \dot{\varphi}_1)^2 + \frac{9}{\epsilon}(\dot{\varphi}_1)^2(\dot{\varphi}_2)^2 \, d\mathcal{L}^2.
\]

This was shown in [9] to \(\Gamma\) convergence to the anisotropic surface energy

\[
\mathcal{F}^s = \int_{\Gamma_{12}} \gamma_{12}(\vec{\nu}_{12}) \, d\mathcal{H}^1.
\]

**Model aniso-theta**

In the two-phase case \((m = 2)\) we allow the energy to be written in terms of an orientation angle \(\theta\) from the positive \(x\)-axis as described in Eq. (2.62) written as \(\gamma_{12}^\theta\). The reasons for this become clear in Section 4.1.6 when we discuss time discretisation. Let the gradient density and well be given by

\[
\Lambda(\nabla \dot{\varphi}) = \gamma_{12}^\theta(\text{atan2}((\nabla \dot{\varphi}_1)_1, (\nabla \dot{\varphi}_1)_2))^2|\nabla \dot{\varphi}_1|^2, \tag{3.47}
\]
\[
W(\dot{\varphi}) = 9(\dot{\varphi}_1)^2(\dot{\varphi}_2)^2. \tag{3.48}
\]

Substituting Eq. (3.47) and Eq. (3.48) into Eq. (3.11) gives

\[
\mathcal{F}^e = \int_\Omega \epsilon \gamma_{12}^\theta(\text{atan2}((\nabla \dot{\varphi}_1)_1, (\nabla \dot{\varphi}_1)_2))^2|\nabla \dot{\varphi}_1|^2 + \frac{9}{\epsilon}(\dot{\varphi}_1)^2(\dot{\varphi}_2)^2 \, d\mathcal{L}^2. \tag{3.49}
\]

As this is also an anisotropic two-phase energy, the same \(\Gamma\)-convergence results as stated in Section 3.2.1 holds. Eq. (3.49) \(\Gamma\)-converges to

\[
\mathcal{F}^s = \int_{\Gamma_{12}} \gamma_{12}^\theta(\theta) \, d\mathcal{H}^1.
\]

Depending on the form of \(\gamma\) given we take one of the models defined in Section 3.2.1.

The procedure for producing a phase field energy \(\mathcal{F}^e\) from a given sharp energy \(\mathcal{F}^s\) is summarised in Algorithm 1. The idea is we start with a given energy and replace the respective terms with their phase field analogue. We think of this
procedure symbolically and it is not mathematically rigorous. The replacement of an object $A$ with another object $B$ is denoted $A \leftarrow B$. In Algorithm 1 we have dropped indices on objects in the interest of presentation.

**Algorithm 1** Regularisation of surface energy.

<table>
<thead>
<tr>
<th>Input: $\mathcal{F}^s$ of the form Eq. (2.37)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: $\mathcal{F}^e$ of the form Eq. (3.11)</td>
</tr>
<tr>
<td>$\sum_i \int_{\Omega_i} \hat{a}<em>i \leftarrow \int</em>{\Omega} \sum_i \hat{\varphi}_i \hat{a}_i$</td>
</tr>
<tr>
<td>if $\gamma_{ij}$ is isotropic, $\forall i &lt; j \leq m$ then</td>
</tr>
<tr>
<td>if $m = 1$ then</td>
</tr>
<tr>
<td><strong>Model one-phase</strong></td>
</tr>
<tr>
<td>$\int_{\Gamma} \gamma \leftarrow \int_{\Omega} \epsilon \Lambda + \frac{1}{\epsilon} W$ defined by Eq. (3.37) Eq. (3.38)</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td><strong>Model multi-iso</strong></td>
</tr>
<tr>
<td>$\sum \int_{\Gamma} \gamma \leftarrow \int_{\Omega} \epsilon \Lambda + \frac{1}{\epsilon} W$ defined by Eq. (3.42) Eq. (3.43)</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>if $\gamma^\theta$ is defined then</td>
</tr>
<tr>
<td><strong>Model aniso-theta</strong></td>
</tr>
<tr>
<td>$\sum \int_{\Gamma} \gamma^\theta \leftarrow \int_{\Omega} \epsilon \Lambda + \frac{1}{\epsilon} W$ defined by Eq. (3.47) Eq. (3.48)</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td><strong>Model aniso-nu</strong></td>
</tr>
<tr>
<td>$\sum \int_{\Gamma} \gamma \leftarrow \int_{\Omega} \epsilon \Lambda + \frac{1}{\epsilon} W$ defined by Eq. (3.45) Eq. (3.46)</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end if</td>
</tr>
</tbody>
</table>

### 3.2.2 Regularisation of balance laws

The regularisation of the balance law Eq. (2.54) is straightforward. We describe the steps beginning with the balance law Eq. (2.54) before summarising the algorithm in pseudocode. The balance law Eq. (2.54) is given by

$$0 = \int_0^T \int_{\Omega} E_{G(t)} \cdot \partial_t \tilde{\zeta} + Q_{G(t)} : \nabla \tilde{\zeta} + F_{G(t)} \cdot \dot{\tilde{\zeta}} d\mathcal{H}^2 d\mathcal{L}^1.$$

We then replace the distributions $E_{G(t)}, Q_{G(t)}, F_{G(t)}$ with phase field analogues which are $E^\epsilon_{\varphi}, Q^\epsilon_{\varphi}$ and $E^\epsilon_{\varphi}$ respectively, this then gives Eq. (3.10) which is

$$0 = \int_0^T \int_{\Omega} E^\epsilon_{\varphi} \cdot \partial_t \tilde{\zeta} + Q^\epsilon_{\varphi} : \nabla \tilde{\zeta} + F^\epsilon_{\varphi} \cdot \dot{\tilde{\zeta}} d\mathcal{L}^3.$$

This procedure is summarised in Algorithm 2.
Algorithm 2 Regularisation of balance laws.

Input: \(0 = \int_0^T \int_\Omega E_{G(t)} \cdot \partial_t \tilde{\zeta} + Q_{G(t)} : \nabla \tilde{\zeta} + F_{G(t)} \cdot \tilde{\zeta} \, dH^2 \, dL^1.\)

Output: \(0 = \int_0^T \int_\Omega E_{\hat{\phi}} \cdot \partial_t \hat{\zeta} + Q_{\hat{\phi}} : \nabla \hat{\zeta} + F_{\hat{\phi}} \cdot \hat{\zeta} \, dL^3.\)

\[
E_{G(t)} \leftarrow E_{\hat{\phi}} \quad Q_{G(t)} \leftarrow Q_{\hat{\phi}} \\
E_{G(t)} \leftarrow F_{\hat{\phi}}
\]

3.2.3 Regularisation of convex set

In the \(P\) gradient flow we constrained the evolution to lie in the set \(K(G(t))\) which given by Eq. (2.52).

\[
K(G(t)) = \{ \hat{F} \in \mathcal{D} : \hat{F} \cdot \hat{\mu}_{ii} \in [0, \infty], \quad \text{on } \partial \Gamma_{ii} \}
\]

This was chosen to enforce the irreversibility of cracks. When passing to the \(P\) gradient flow a new set \(K^c\) must be chosen to mimic this feature for when \(m = 1\). When \(m > 1\) \(K^c\) can be chosen to enforce the phase field variables to lie between 0 and 1. This is the usual obstacle potential and is usually introduced by setting the energy \(F^c\) to \(\infty\) if \(\hat{\phi} \notin G^c\). To choose to use an obstacle constraint the user needs to set a variable in the code \texttt{constrained=True}, where by default the value is \texttt{False}. The following procedure summarises how \(K^c\) is set in PHASEFIELD.

Algorithm 3 Regularisation of \(K(G)\).

Input: \(K(G),\) constrained

Output: \(K^c\)

if \(m = 1\) then
    \(K^c := \{ \hat{\nu} \in TG^c : \hat{\nu} \leq 0 \}.\)
else if \(\text{constrained} == \text{True}\) then
    \(K^c := \{ \hat{\nu} \in G^c : 0 \leq \hat{\nu}_i \leq 1, \forall i \in [1, m] \},\)
else
    \(K^c = G^c\)
end if

3.2.4 Complete regularisation of gradient flow

Utilising the previous algorithms for the energy and balance laws we regularise the \(P\) gradient flow Eq. (2.53). We state the steps and summarise them with pseudocode. The energy is the first object regularised by applying Algorithm 1. The balance
Algorithm 4 Regularisation of $\mathcal{P}$ gradient flow.

**Input:** $\mathcal{P}$ Gradient flow in Def (2.1.19).

**Output:** $\mathcal{P}^\epsilon$ Gradient flow in Def (3.1.5) or (3.1.7)

1. Apply Algorithm 1 to $\mathcal{F}^s$ to produce $\mathcal{F}^\epsilon$.
2. Apply Algorithm 2 to regularised balance laws.
3. Apply Algorithm 3 to $\mathcal{K}$ to produce $\mathcal{K}^\epsilon$.

In Eq. (2.53) use phase field inner product $\langle \cdot, \cdot \rangle_{G(t),\tau_G(t)} \leftarrow \langle \cdot, \cdot \rangle_{\hat{\varphi},\tau_0}$, $\vec{V}_t \leftarrow \partial_t \hat{\varphi}$ and $\mathcal{P} \leftarrow \nabla \mathcal{F}^\epsilon$ to produce Eq. (3.26).

Setup initial conditions, keep $\tilde{u}_I$ and set $\hat{\varphi}_I = \hat{\chi}_G$.

Keep boundary conditions for $\tilde{u}$.

laws are then regularised by applying Algorithm 2. We symbolically compute the derivative of the densities in the energy with respect to $\hat{\varphi}$ and project onto the tangent space $T\mathcal{G}^\epsilon$ with the appropriate inner product to give the gradient as Eq. (3.18). The phase field analogue of the velocity is the derivative of $\hat{\varphi}$ with respect to time. Finally for the initial conditions the initial phase field variable is set equal to the characteristic function from the sharp equations. In practice one may wish to do some initial smoothing of this characteristic function.

### 3.3 Semi-discretisation

The UFL form produced by the class *PhaseModel* is a time-discretised equation. This has the advantage that the user has extra flexibility when solving the equation to pick a time discretisation. We therefore need a procedure for automating the transition between a $\mathcal{P}^\epsilon$ gradient flow and a time discrete analogue. The default implementation is a backward Euler scheme with fixed step size which approximates the solution of the continuous problem. Different time discretisations are considered in Chapter 4. *PhaseField* can deal with variable step sizes (see Section 5.3.2) however for the presentation here we keep it fixed.

Rather than solving Eq. (3.20) directly we can instead solve the following equation at $\text{a.e } t \in [0,T]$

$$0 = \int_\Omega \partial_t E_\varphi^s \cdot \tilde{\zeta} \, d\mathcal{L}^2 - Q_\varphi^s \cdot \nabla \tilde{\zeta} - F_\varphi^s \cdot \tilde{\zeta} \, d\mathcal{L}^2. \tag{3.51}$$

Denote a fixed step size by $\Delta t = \frac{T}{N}$, $N \in \mathbb{N}$. We use the notation $\tilde{u}^k \approx \tilde{u}(\cdot, t_k)$, $\hat{\varphi}^k \approx \hat{\varphi}(\cdot, t_k)$ and $t_k = k\Delta t$, $k = 0, \cdots, N - 1$.

We give a definition of what we mean by a semi-discrete $\mathcal{P}^\epsilon$ gradient flow before showing how to transform a given $\mathcal{P}^\epsilon$ into its semi-discrete analogue.
**Definition 3.3.1** (Semi-discretised \( P^\epsilon \) Gradient flow). Given the initial condition \( \tilde{u}_I \), initial phase field \( \hat{\varphi}_I \), functions \( E^\epsilon_{\hat{\varphi}} \), \( Q^\epsilon_{\hat{\varphi}} \), \( F^\epsilon_{\hat{\varphi}} \), energy \( F^\epsilon \), mobility \( \tau_0 \in \mathbb{R} \), closed convex set \( \mathcal{K}^\epsilon_{\Delta}(\hat{\varphi}^k) \subset H^1(\Omega, \mathbb{R}^m) \), final time \( T \) and boundary conditions for \( \tilde{u} \). Let \( \Delta t = \frac{T}{N} \) be a fixed step size. A semi-discrete \( P^\epsilon \) gradient flow is the sequence of functions \( \hat{\varphi}^{k+1}_I \in \mathcal{K}^\epsilon_{\Delta}(\hat{\varphi}^k) \), \( \tilde{u}^{k+1} \in H^1(\Omega, \mathbb{R}^r) \) such that for \( k = 0, \ldots, N - 1 \), \( \forall \hat{\eta} \in \mathcal{K}^\epsilon_{\Delta}(\hat{\varphi}^k) \),

\[
\frac{\hat{\varphi}^{k+1} - \hat{\varphi}^k}{\Delta t}, \hat{\eta} - \hat{\varphi}^{k+1}, \hat{\varphi}, \tau_0 \geq - (\nabla F^\epsilon(\hat{\varphi}^{k+1}, \tilde{u}^{k+1}), \hat{\eta} - \hat{\varphi}^{k+1})_{\hat{\varphi}, \tau_0},
\]

and such that \( \forall \hat{\zeta} \in H^1(\Omega, \mathbb{R}^r) \)

\[
\int_{\Omega} \frac{E^\epsilon_{\hat{\varphi}^{k+1}}(\tilde{u}^{k+1}) - E^\epsilon_{\hat{\varphi}^k}(\tilde{u}^k)}{\Delta t} : \hat{\zeta} \, d\mathcal{L}^2 = \int_{\Omega} Q^\epsilon_{\hat{\varphi}^{k+1}}(\tilde{u}^{k+1}) : \nabla \hat{\zeta} + F^\epsilon_{\hat{\varphi}^{k+1}}(\tilde{u}^{k+1}) \cdot \hat{\zeta} \, d\mathcal{L}^2,
\]

and letting

\[
\tilde{u}^0 = \tilde{u}_I(\cdot), \quad \hat{\varphi}^0 = \hat{\varphi}_I(\cdot).
\]

In addition to boundary conditions for \( \tilde{u} \) and \( \hat{\varphi} \). We have again made use of the inner product \( (\cdot, \cdot)_{\hat{\varphi}, \tau_0} \) defined in Eq. (3.17).

Transforming a continuous equation to a fully-implicit scheme is often trivial mathematically however we describe the steps that must be undertaken from a symbolic point of view to better understand the ideas underpinning the structure of PhaseField.

Appearances of \( \hat{\varphi} \), \( \nabla \hat{\varphi} \), \( \tilde{u} \), \( \nabla \tilde{u} \) are replaced by \( \hat{\varphi}^{k+1} \), \( \tilde{u}^{k+1} \), \( \nabla \hat{\varphi}^{k+1} \), \( \nabla \tilde{u}^{k+1} \) respectively. A first order linear difference is taken for the time derivatives of \( \hat{\varphi} \) in the interface equation and \( E(\hat{\varphi}) \hat{\varphi} \) in the balance laws.

One of the most challenging tasks in this transformation is to ensure the tangent space of the discretised equations appropriately captures the features of \( \mathcal{K}^\epsilon \). We denote the semi-discrete analogue of \( \mathcal{K}^\epsilon \) as \( \mathcal{K}^\epsilon_{\Delta}(\hat{\varphi}^k) \). We discretise the tangent space \( \mathcal{K}^\epsilon \) defined in Algorithm 5. This is done either to enforce the obstacle constraint to keep the phase field variables between 1 and 0, or in the case of fracture to ensure that the crack is irreversible. The regularisation of the tangent space in Algorithm 5 is coupled with the notion of replacing the continuous functions with their discrete analogues which gives the procedure for the backward Euler discretisation seen in Algorithm 6.
Algorithm 5 Regularisation of the tangent space.

Input: $K^\epsilon$, constrained

Output: $K^\epsilon_\Delta(\hat{\varphi}^k)$

\[
\begin{array}{l}
\text{if} \quad m = 1 \quad \text{then} \\
K^\epsilon_\Delta(\hat{\varphi}^k) := \{ \hat{v} \in H^1(\Omega, \mathbb{R}^m) : \hat{v} \in [0, \hat{\varphi}^k] \}.
\end{array}
\]

\[
\begin{array}{l}
\text{else if} \quad \text{constrained=True} \quad \text{then} \\
K^\epsilon_\Delta(\hat{\varphi}^k) := \{ \hat{v} \in H^1(\Omega, \mathbb{R}^m) : \hat{v} \in [0, 1] \}
\end{array}
\]

\[
\begin{array}{l}
\text{else} \\
K^\epsilon_\Delta(\hat{\varphi}^k) = H^1(\Omega, \mathbb{R}^m),
\end{array}
\]

end if

Algorithm 6 Backward Euler discretisation of $\mathcal{P}^\epsilon$ gradient flow.

Input: $\mathcal{P}^\epsilon$ gradient flow

Output: $\mathcal{P}^\epsilon_\Delta$ gradient flow

Given phase field equation

$K^\epsilon \leftarrow K^\epsilon_\Delta(\hat{\varphi}^k)$

$\hat{\varphi} \leftarrow \hat{\varphi}^{k+1}$

$\tilde{u} \leftarrow \tilde{u}^{k+1}$

$\dot{\varphi} \leftarrow \frac{\tilde{\varphi}^{k+1} - \hat{\varphi}^k}{\Delta t}$

Given balance Laws

$\dot{\varphi} \leftarrow \frac{E(\tilde{\varphi}^{k+1}) - E(\hat{\varphi}^k)}{\Delta t}$

$Q\hat{\varphi} \leftarrow Q\varphi^{k+1}$

$F\hat{\varphi} \leftarrow F\varphi^{k+1}$

$\tilde{u} \leftarrow \tilde{u}^{k+1}$

3.4 Code and constructing the PhaseModel class

Having described the procedure for transforming a $\mathcal{P}$ gradient flow into a time discrete analogue we next describe how to accomplish this in PHASEFIELD. This is done by creating what we call a PhaseModel. The sharp classes defined in Section 3.5 are passed into the constructor of the PhaseModel class. This class also takes the interface width $\epsilon$, time step $\Delta t$ and well $W$ as arguments.

The phase field class contains a public method setupPhase which gives access to the phase field model. This is discussed in detail in Chapter 5. From a programming point of view the requirements on the sharp class are very minimal. It must contain either a non-zero balance law or an energy.

Remark 3.4.1. The classes present here additionally contain a domain $\Omega$ and end time $T$, strictly speaking, these are only required at the solving step and not to build the phase field model. The sharp model only needs to provide the space dimension.
since this can not be determined from the other methods. To this end the sharp class must either contain a description of the domain $\Omega$ or a property $\dimDomain$.

### 3.4.1 The PhaseModel Class

Listing 14 shows the arguments that the PhaseModel takes in its constructor. Firstly, it needs a class defining a $\mathcal{P}$ gradient flow; in this case called $\text{sharpCls}$. The next two arguments are both floating point numbers: $\epsilon$ the width of the diffuse interfaces and $\Delta t$ the step size. The fourth argument $\text{well}$ specifies the potential well that

```python
(sharpCls, epsilon=None, dt=None, well = Implicit)
```

Listing 14: Constructor of PhaseModel class.

is to be used in the simulations; it must return $W(\hat{\phi})$. We describe the possible options in Section 3.4.3; by default PhaseModel uses a well $W$ with all the terms taken implicitly.

We describe the different specifications of the well and also how to specify the additional smoothing parameters $I, q$ and $f$ (see Eqs. (3.8), (3.9) and (3.13)). We first specify exactly the form of the UFL equations that are returned from the class. A more technical specification of the class is given in Section 5.2.

#### 3.4.2 UFL form of discretised equations

We give the exact form of the UFL equations returned from the PhaseModel class when the method setupPhase is called. Eqs. (3.32) and (3.34) can be written as semi-discrete constrained minimisation problems. Let $J_{Pi}^F, J_{Bi}, i = 1, 2$ be given by

$$
\begin{align*}
J_{1i}^F(\hat{v}, \tilde{w}, \hat{\eta}) & := (\hat{v} - \hat{\phi}^k, \hat{\eta})_{\hat{\phi}, \tau_0}, \\
J_{2i}^F(\hat{v}, \tilde{w}, \hat{\eta}) & := (\nabla \mathcal{F}^\epsilon(\hat{v}, \tilde{w}), \hat{\eta})_{\hat{\phi}, \tau_0}, \\
J_{1i}^B(\tilde{v}, \tilde{w}, \tilde{\zeta}) & := \int_\Omega \left( \frac{E_{\tilde{w}^{k+1}}(\tilde{w}^{k+1}) - E_{\tilde{\phi}^k}(\tilde{w}^k)}{\Delta t} \cdot \tilde{\zeta} \right) d\mathcal{L}^2, \\
J_{2i}^B(\tilde{v}, \tilde{w}, \tilde{\zeta}) & := \int_\Omega Q_{\tilde{w}^{k+1}}(\tilde{w}^{k+1}) : \nabla \tilde{\zeta} + F_{\tilde{w}^{k+1}}(\tilde{w}^{k+1}) \cdot \tilde{\zeta} d\mathcal{L}^2.
\end{align*}
$$
The problem is to find \( \hat{\varphi}_{k+1} \in \mathcal{K}_{\Delta}(\hat{\varphi}^k) \) and \( \tilde{u}_{k+1} \in H^1(\Omega, \mathbb{R}^r) \) such that \( \forall \hat{\eta} \in \mathcal{K}_{\Delta}(\hat{\varphi}^k) \)

\[
\begin{align*}
J^1_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\eta}) - J^2_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\eta}) & \geq \\
J^1_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\varphi}^k) - J^2_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\varphi}^k),
\end{align*}
\]

and \( \forall \tilde{\zeta} \in H^1(\Omega, \mathbb{R}) \)

\[
J^1_{B}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \tilde{\zeta}) - J^2_{B}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \tilde{\zeta}) = 0.
\]

The UFL form returned from the \texttt{setupPhase} method is the following matrix.

\[
\begin{pmatrix}
J^1_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\eta}) & J^2_{PF}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \hat{\eta}) \\
J^1_{B}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \tilde{\zeta}) & J^2_{B}(\hat{\varphi}_{k+1}, \tilde{u}_{k+1}, \tilde{\zeta})
\end{pmatrix}
\]

(3.54)

### 3.4.3 Well

The well \( W \) used can have a large influence on the accuracy of the approximation. We focus on the continuum form while different time discretisations are discussed in Section 4.1.

\texttt{PHASEFIELD} allows the user to define a custom well if the default given by the models described in Section 3.2.1 are not adequate. If a custom well is defined one must define the correct constants as a pre-factor to ensure the correct limit equations are obtained. This has already been done for the wells defined in the models. These calibration calculations for both smooth and obstacle potentials can be seen in Appendix B.1 and Appendix B.2.

We describe the implementation of a two-phase quadratic well taken with an implicit discretisation. This is different from the well \texttt{Implicit} that is used by default which does extra checking on the number of phases and whether an isotropic or anisotropic energy is used.

The well passed to \texttt{PhaseModel} must take three arguments. The first is the instance of the \texttt{PhaseModel} class. The second and third are \( \hat{\varphi}_{k+1} \) and \( \hat{\varphi}^k \) respectively. We ignore the third argument as explicit time discretisations will be discussed in Section 5.2. The first argument provides all the information contained within the sharp class, this is held as a member variable \texttt{sharpClass}. An implementation is demonstrated in Listing 15 of the quadratic well \( W(\hat{\varphi}) = (\hat{\varphi}_1)^2(\hat{\varphi}_2)^2 \).

**Remark 3.4.2.** It is possible to access \( \hat{\varphi}_{k+1} \) and \( \hat{\varphi}^k \) directly through the \texttt{phaseModel} instance. They are passed as separate arguments as they must be symbolically differentiated.
def myWell(phaseModel, phi, phiK):
    return 9*phi[0]*phi[0]*phi[1]*phi[1]

Listing 15: The simplest implementation of a potential well.

3.4.4 Interpolation

We describe the implementation and setup of PhaseField for the interpolation function $I$. If no interpolation is set, Eq. (3.36) is used.

An interpolation function must take three arguments. The first is a vector of length $m$ that takes $\hat{\varphi}^{k+1}$, the second is $\hat{\varphi}^k$ which we discuss in Chapter 4, the third is a vector of length $m$ that are the values to be interpolated. The simplest way to define an interpolation function is by providing a function as seen in Listing 16. The following interpolation functions are provided

- **SimpleInterpolate**

\[ I(\hat{\varphi}^{k+1}, \hat{\varphi}^k, \hat{a}) = \sum_{i=1}^{m} \hat{\varphi}^{k+1}_i \cdot \hat{a}_i. \] (3.55)

- **SmoothStep**

\[ I(\hat{\varphi}^{k+1}, \hat{\varphi}^k, \hat{a}) = \hat{a}_2 + (\hat{a}_1 - \hat{a}_2)(3(\hat{\varphi}^{k+1})^2 - 2(\hat{\varphi}^{k+1})^3). \] (3.56)

**SmoothStep** can only be used when $m = 2$ and has the additional property that $I'(e_\alpha, \hat{a}(\hat{u}, \nabla \hat{u})) = 0, 1 \leq \alpha \leq m$. A dash denotes the derivative with respect to the first argument. This often gives better asymptotic properties at the expense of computations cost [73].

def interpolate(phi, phik, a):
    return phi[0]*a[0] + phi[1]*a[1]

Listing 16: Simplest interpolation function.
\[ I(\hat{\phi}^{k+1}, \hat{\phi}^k, \hat{a}) = (\hat{\phi}^{k+1})^2 \cdot \hat{a}. \quad (3.57) \]

**InterpolateOne** can only be used when \( m = 1 \).

### 3.4.5 Balance degeneracy

For regularising the balance laws the functions \( f \) and \( q \) in Eqs. (3.8) and (3.9) must be defined. This is done passing in or editing a dictionary `bulkSmoothdict` in the `PhaseModel` class. The interpolation function for \( Q \) or \( F \) is accessed with the first parameter of the dictionary while the second parameter is a number representing the balance law whose terms are to be edited. For instance `bulkSmoothdict['F', 1]` accesses the smoothing function for the second equation for the \( F \) term, \( f_2 \) from Section 3.1.1 (indexing starts from zero in PYTHON). By default the following functions are used

\[ q_i(\hat{\phi}) = f_i(\hat{\phi}) = (\hat{\phi}_1 \cdots \hat{\phi}_m), \quad 1 \leq i \leq r. \]

In Listing 17 we give an example for changing the smoothing term \( q \) to

\[ q(\hat{\phi}) = \left((\hat{\phi}_1)^2 \quad (\hat{\phi}_2)^2\right), \quad (3.58) \]

in the case \( m = 2, r = 1 \).

```python
phaseModel.bulkSmoothdict['Q', 0] = lambda phi, phiK: [phi[0]*phi[0], phi[1]*phi[1]]
```

Listing 17: Smooth dictionary implementing Eq. (3.58).

### 3.4.6 Summary of smoothing functions and default values

We finish this section by summarising the smoothing functions and their default implementations.

\[ q_i(\hat{\phi}) = f_i(\hat{\phi}) = (\hat{\phi}_1 \cdots \hat{\phi}_m), \quad 1 \leq i \leq r \]

\[ I(\hat{\phi}, \hat{a}) = \sum_{i=1}^{m} \hat{\phi}_i \hat{a}_i(\tilde{u}, \nabla \tilde{u}). \]

83
Table 3.1: Smoothing functions and parameters for $\mathcal{P}^\epsilon$ gradient flow.

<table>
<thead>
<tr>
<th>Model</th>
<th>$W(\hat{\phi})$</th>
<th>$I(\hat{\phi}, \hat{a}(\hat{u}, \nabla \hat{u}))$</th>
<th>$q_i(\hat{\phi})$</th>
<th>$f_i(\hat{\phi})$</th>
<th>$\epsilon$</th>
<th>$\Delta t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleInterpolate</td>
<td>$\hat{\phi}$</td>
<td>$\hat{\phi}$</td>
<td>(\hat{\phi})</td>
<td>(\epsilon)</td>
<td>(\Delta t)</td>
<td></td>
</tr>
</tbody>
</table>

3.5 Examples

Given the examples of $\mathcal{P}$ gradient flows given in Section 2.2 we show how these equations are regularised into $\mathcal{P}^\epsilon$ gradient flows in PHASEFIELD. The $\mathcal{P}^\epsilon$ gradient flow is further simplified by using the procedure described in Section 3.1.4. This is accompanied with code snippets showing how to construct the PhaseModel class in PHASEFIELD as well as set the additional functions needed for the phase field regularisation.

Each regularisation is summarised with a table of the form Table 3.1 that specifies all the parameters and functions that are needed in addition to the definition of a $\mathcal{P}$ gradient flow to define a $\mathcal{P}^\epsilon$ gradient flow. When these are chosen automatically by PHASEFIELD they are highlighted in orange. In each example we provide the functionals $J_{PF}^\epsilon$ and $J_B^\epsilon$ in Section 3.1.4.

3.5.1 Mean curvature flow

We apply our regularisation Algorithm 4 to the mean curvature equations given in Section 2.2.1 to give the regularised phase field equations written in terms of $\varphi$. We repeat Eq. (2.66) here which is the equation we regularise.

$$\vec{V}_t \cdot \vec{v}_{12} = -\beta \kappa_{12}, \quad \text{on } \Gamma_{12}(t). \quad (3.59)$$

The first step in Algorithm 4 is to regularise the energy. According to Algorithm 1, we get an energy of Model multi-iso type (see Section 3.2.1) which gives

$$\mathcal{F}^\epsilon := \int_{\Gamma_{12}} \beta \, d\mathcal{H}^1 \rightarrow \mathcal{F}^\epsilon := \int_{\Omega} \frac{\epsilon \beta}{2} \left[ |\nabla \varphi_1|^2 + |\nabla \varphi_2|^2 \right] +$$

$$\frac{9 \beta}{2\epsilon} \left[ (\hat{\varphi}_1)^2 (1 - \hat{\varphi}_1)^2 + (\hat{\varphi}_2)^2 (1 - \hat{\varphi}_2)^2 \right] \, d\mathcal{L}^2,$$

as $\beta = \gamma_{12}$. As stated in Section 2.2.1 there are no balance laws and therefore we do not need to apply Algorithm 2. An application of Algorithm 3 shows we also have the full space $\mathcal{K}^\epsilon = T\mathcal{G}^\epsilon$ for the evolution. Working through the rest of Algorithm 4
we get the final equations to solve in the form of Eq. (3.34) as

\[
J_{PF}^\epsilon(\hat{\varphi}, \tilde{u}, \hat{\eta}) = \int_\Omega \tau_0 \epsilon \partial_t \hat{\varphi} \cdot \hat{\eta} + \epsilon \beta \mathbb{P}_\Sigma[\nabla \hat{\varphi}] : \nabla \hat{\eta} \\
+ \frac{9 \beta}{\epsilon} \mathbb{P}_\Sigma[\hat{\varphi}_1(1 - \hat{\varphi}_1)(1 - 2 \hat{\varphi}_1) \hat{\varphi}_1(1 - \hat{\varphi}_2)(1 - 2 \hat{\varphi}_2) \cdot \hat{\eta} d\mathcal{L}^2 \\
= \int_\Omega \tau_0 \epsilon \partial_t \hat{\varphi} \cdot \hat{\eta} + \epsilon \beta \nabla \hat{\varphi} : \nabla \hat{\eta} \\
+ \frac{9 \beta}{\epsilon} \left( \hat{\varphi}_1(1 - \hat{\varphi}_1)(1 - 2 \hat{\varphi}_1) \hat{\varphi}_1(1 - \hat{\varphi}_2)(1 - 2 \hat{\varphi}_2) \right) \cdot \hat{\eta} d\mathcal{L}^2
\]

To construct the PhaseModel class we use Listing 18 where the only additional parameters set are \( \epsilon = 0.05 \) and \( \Delta t = 0.01 \) as the rest of the functions in this case are default. We remind the reader that the class which specified the \( \mathcal{P} \) gradient flow in this case was named \( \text{Mcf} \). A summary of the regularisation parameters can be seen

<table>
<thead>
<tr>
<th>Model</th>
<th>( W(\hat{\varphi}) )</th>
<th>( I(\hat{\varphi}, \hat{\alpha}(\tilde{u}, \nabla \tilde{u})) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model multi-iso</td>
<td>Model multi-iso</td>
<td></td>
</tr>
<tr>
<td>( q(\hat{\varphi}) )</td>
<td>( f(\hat{\varphi}) )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>-</td>
<td>0.05</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 3.2: Smoothing functions and parameters for mean curvature \( \mathcal{P}^\epsilon \) gradient flow.

\[\text{phaseField = PhaseModel(Mcf, epsilon=0.05, dt=0.001)}\]

Listing 18: PYTHON code for class \( \text{Mcf} \).

in Table 3.2

3.5.2 Mullins-Sekerka

We show the transformation of a Mullins-Sekerka \( \mathcal{P} \) gradient flow (see Section 2.2.2) to a \( \mathcal{P}^\epsilon \) gradient flow. Similarly to the mean curvature flow regularisation, this is an model multi-iso energy where in addition we have bulk contributions. For the Mullins-Sekerka problem we also need to regularise the balance law.
Applying Algorithm 1 to the sharp energy defined by
\[ \gamma = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix}, \]
\[ \hat{a}(\tilde{u}, \nabla \tilde{u}) = \begin{pmatrix} 0 & 2\tilde{u} \end{pmatrix}, \]
along with the default interpolation function SimpleInterpolate in Eq. (3.55) we get
\[ F^s := \int_{\Gamma} \beta \, d\mathcal{H}^1 \rightarrow F^c := \int_{\Omega} \frac{\epsilon \beta}{2} \left[ |\nabla \hat{\phi}_1|^2 + |\nabla \hat{\phi}_2|^2 \right] + \frac{9\beta}{2\epsilon} \sum_{i=1}^{2} \hat{\phi}_i^2 (1 - \hat{\phi}_i)^2 + \hat{\phi}_2 \tilde{u} \, d\mathcal{L}^2. \]

We next apply Algorithm 2 to the balance law defined by \( E, Q \) and \( F \) in Table 2.4, (see Eqs. (3.4) to (3.6) for a precise definition) we get
\[ E^\phi = \hat{\phi}_1, \]
\[ Q^\phi = -\alpha (\hat{\phi}_1 + \hat{\phi}_2) \nabla \tilde{u}, \]
\[ F^\phi = 0, \]

Similarly to Section 3.5.1 an application of Algorithm 3 have that \( K^\epsilon = T \mathcal{G}^\epsilon \). Applying the rest of Algorithm 4 we get the operators \( J_B^\phi \) and \( J_{PF}^\phi \) in Eq. (3.34) as
\[ J_B^\phi(\hat{\phi}, \tilde{u}, \hat{\zeta}) := \int_{\Omega} \partial_t \hat{\phi}_1 \cdot \hat{\zeta} + \alpha (\hat{\phi}_1 + \hat{\phi}_2) \nabla \tilde{u} \cdot \nabla \hat{\zeta} \, d\mathcal{L}^2. \]

\[ J_{PF}^\phi(\hat{\phi}, \tilde{u}, \hat{\eta}) \]
\[ := \int_{\Omega} \tau_0 \epsilon \partial_t \hat{\phi} \, d\mathcal{L}^2 + \epsilon \beta \mathbb{P}_{T\Sigma}[\nabla \hat{\phi}] : \nabla \hat{\eta} + \frac{9\beta}{\epsilon} \mathbb{P}_{T\Sigma}[\left( \hat{\phi}_2(1 - \hat{\phi}_2)(1 - 2\hat{\phi}_2) \right) \cdot \hat{\eta} + \mathbb{P}_{T\Sigma}[\left( \begin{pmatrix} 0 \\ 2\tilde{u} \end{pmatrix} \right) \cdot \hat{\eta}] \, d\mathcal{L}^2, \]
\[ = \int_{\Omega} \tau_0 \epsilon \partial_t \hat{\phi} \cdot \hat{\eta} \, d\mathcal{L}^2 + \epsilon \beta \nabla \hat{\phi} : \nabla \hat{\eta} + \frac{9\beta}{\epsilon} \left( \hat{\phi}_2(1 - \hat{\phi}_2)(1 - 2\hat{\phi}_2) \right) \cdot \hat{\eta} + \left( \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \tilde{u} \cdot \hat{\zeta} \, d\mathcal{L}^2. \]

The code for creating the \texttt{PhaseModel} class in \texttt{PHASEFIELD} can be seen in Listing 19. The variables and functions that need to be set are summarised in Table 3.3, the
Table 3.3: Smoothing functions and parameters for Mullins-Sekerka $P^l$ gradient flow.

only non-default values are $\epsilon$ and $\Delta t$.

```
1 phaseField = PhaseModel(Ms, epsilon=0.05, dt=1e-3)
```

Listing 19: PYTHON code for class Ms.

### 3.5.3 Dendritic growth

We are regularising the energy defined by Table 2.6. This is a Model aniso-theta energy and is therefore regularised by default applying Algorithm 1 as

$$
F^\epsilon = \int_\Omega \epsilon \gamma(\theta(\nabla \hat{\phi}_1)) |\nabla \hat{\phi}_1|^2 + \frac{9}{2\epsilon} \sum_{i=1}^{2} \hat{\phi}_i^2 (1 - \hat{\phi}_i)^2 + \hat{\phi}_1 \frac{400\kappa_1}{\pi} \text{atan2}(\kappa_2 \tilde{u}) \, dL^2.
$$

We demonstrate using an interpolation function with the property

$$
I'(e_1, \hat{a}(\tilde{u}, \nabla \tilde{u})) = I'(e_2, \hat{a}(\tilde{u}, \nabla \tilde{u})) = 0,
$$

where the dash denotes the derivative with respect to the first argument. Functions with this property are known to give better asymptotic convergence however can be numerically more difficult to deal with. We make use of the SmoothStep function which in the two-phase is defined in Eq. (3.56). This is defined in the file python/auxfun.py and is named class SmoothStep. After creating the PhaseModel class the member variable interpolation can be edited and changed. Assuming we have done this the modified energy is

$$
F^\epsilon = \int_\Omega \epsilon \gamma(\theta(\nabla \hat{\phi}_1)) |\nabla \hat{\phi}_1|^2 + \frac{9}{2\epsilon} \sum_{i=1}^{2} \hat{\phi}_i^2 (1 - \hat{\phi}_i)^2 + (3\hat{\phi}_1^2 - 2\hat{\phi}_1^3) \frac{400\kappa_1}{\pi} \text{atan2}(\kappa_2 \tilde{u}) \, dL^2.
$$

87
Applying Algorithm 2 to the balance law defined by $E, Q$ and $F$ in Table 2.6, we get

\[ E^e_\phi = (\tilde{u} - \lambda) \hat{\phi}_1 + \tilde{u} \hat{\phi}_2, \quad \text{(3.61)} \]
\[ Q^e_\phi = -\alpha (\hat{\phi}_1 + \hat{\phi}_2) \nabla \tilde{u}, \quad \text{(3.62)} \]
\[ F^e_\phi = 0, \quad \text{(3.63)} \]

Applying Algorithm 3 we have the full space $K^e = T \mathcal{G}^e$. Applying the rest of Algorithm 4 and writing resulting equation in terms of the operators $J^e_{PF}$ and $J^e_B$ in Eq. (3.34) we get

\[ J^e_{PF}(\hat{\phi}, \tilde{u}, \tilde{\eta}) := \int_{\Omega} \tau_0 \partial_t \hat{\phi} \cdot \eta + \mathbb{P}_{T\Sigma} \left( \epsilon \frac{\partial}{\partial \nabla \hat{\phi}_1} \left[ \gamma (\theta(\nabla \hat{\phi}_1)) |\nabla \hat{\phi}_1|^2 \right] : \nabla \tilde{\eta} + \right. \]
\[ \left. \frac{9}{\epsilon} \mathbb{P}_{T\Sigma} \left[ \begin{pmatrix} \hat{\phi}_1(1 - \hat{\phi}_1)(1 - 2\hat{\phi}_1) \\ \hat{\phi}_2(1 - \hat{\phi}_2)(1 - 2\hat{\phi}_2) \end{pmatrix} \right] : \tilde{\eta} + \hat{a}(\tilde{u}, \nabla \tilde{u}) \mathbb{P}_{T\Sigma} \left( \begin{pmatrix} 6 \hat{\phi}_1(1 - \hat{\phi}_1) \\ 0 \end{pmatrix} \right) \cdot \tilde{\eta} d\mathcal{L}^2, \right. \]
\[ \left. \int_{\Omega} \partial_t [\tilde{u} - \lambda) \hat{\phi}_1 + \tilde{u} \hat{\phi}_2] \cdot \tilde{\zeta} + \alpha (\hat{\phi}_1 + \hat{\phi}_2) \nabla \tilde{u} \cdot \nabla \tilde{\zeta} d\mathcal{L}^2. \quad \text{(3.66)} \]

We have defined the complete set of phase field equations produced by the model. The code to do this can be seen in Listing 20. Here we must make sure after initialising the class that we set the member variable interpolation to the SmoothStep class. A summary of the smoothing functions can be seen in Table 3.4. The interface equation is known as the anisotropic Allen-Cahn equation, it was shown by formal asymptotics to recover anisotropic mean curvature flow in [10, 57, 72] and rigorous convergence was shown in [38] assuming a smooth evolution.
<table>
<thead>
<tr>
<th>Model</th>
<th>( W(\hat{\varphi}) )</th>
<th>( I(\hat{\varphi}, \hat{a}(\tilde{u}, \nabla \tilde{u})) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model aniso-theta</td>
<td>Model aniso-theta</td>
<td>SmoothStep</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|c|c|}
\hline
q(\hat{\varphi}) & f(\hat{\varphi}) & \epsilon & \Delta t \\
\hline
\hat{\varphi} & \hat{\varphi} & 0.015 & 5 \times 10^{-4} \\
\hline
\end{array}
\]

Table 3.4: Smoothing functions and parameters for dendritic growth \( \mathcal{P}^\epsilon \) gradient flow.

\[
\text{phaseField} = \text{PhaseModel(Crystal, epsilon = 0.015, dt = 0.0005, thetaSemi = True)}
\]

Listing 20: Python code for class `Crystal`.

### 3.5.4 Tumour growth

Regularising the energy \( \mathcal{E}^s \) defined by \( \hat{a}(\tilde{u}, \nabla \tilde{u}) \) and \( \gamma \) defined in Table 2.8, Algorithm 1 is applied to the following equation

\[
\gamma = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix}, \quad \hat{a}(\tilde{u}, \nabla \tilde{u}) = \begin{pmatrix} 0 & 2b\tilde{u}_1 + 2\alpha\tilde{u}_2 \end{pmatrix},
\]

which gives a **Model multi-iso** energy \( \mathcal{E}^\epsilon \)

\[
\mathcal{E}^s := \int_{\Gamma^1} \beta \, d\mathcal{H}^1 \to \mathcal{E}^\epsilon := \int_{\Omega} \frac{\epsilon \beta}{2} [ |\nabla \hat{\varphi}_1|^2 + |\nabla \hat{\varphi}_2|^2 ] + \frac{9\beta}{2\epsilon} \sum_{i=1}^{2} \hat{\varphi}_i^2(1 - \hat{\varphi}_i)^2 + \hat{\varphi}_2(2b\tilde{u}_1 + 2\alpha\tilde{u}_2) \, d\mathcal{L}^2.
\]

Here we choose more degenerate smoothing functions for the balance laws namely

\[
q_1(\hat{\varphi}) = \begin{pmatrix} (\hat{\varphi}_1)^2 \\ (\hat{\varphi}_2)^2 \end{pmatrix},
\]

(3.68)
with \( q_2 = f_1 = f_2 = \hat{\varphi} \). Applying Algorithm 2 to the balance law defined by \( E, Q \) and \( F \) in Table 2.8 along with the smoothing functions in Eq. (3.68), we get

\[
E_\varphi^\varepsilon = \begin{pmatrix} \hat{\varphi}_1 & 0 \\ 0 & 0 \end{pmatrix},
\]

\[
Q_\varphi^\varepsilon = \begin{pmatrix} (\hat{\varphi}_1)^2 \nabla \hat{u}_1 & \hat{\varphi}_1 \nabla \hat{u}_2 \\ -\hat{\varphi}_1 \nabla \hat{\varphi}_2 & \hat{\varphi}_2 D \nabla \hat{u}_2 \end{pmatrix},
\]

\[
F_\varphi^\varepsilon = \begin{pmatrix} \hat{\varphi}_1 (P(\hat{u}_2 + \lambda) - A) & 0 \\ \hat{\varphi}_1 (c \hat{u}_1 + \lambda) & 0 \end{pmatrix},
\]

Applying Algorithm 3 we have the full space \( K^\varepsilon = TG^\varepsilon \). Applying the rest of Algorithm 4 and writing in terms of \( J_{PF}^\varepsilon \) and \( J_B^\varepsilon \) in Eq. (3.34) we have

\[
J_{PF}^\varepsilon (\hat{\varphi}, \hat{\varphi}, \hat{\varphi}_1, \hat{\varphi}_2) := \int_\Omega \epsilon \beta \partial_{\hat{\varphi}_1} \left( \hat{\varphi}_1 \right) \cdot \nabla \hat{\varphi} + \frac{9 \beta}{\varepsilon} P \nabla \left( \hat{\varphi}_1 (1 - \hat{\varphi}_1) (1 - 2 \hat{\varphi}_1) \right) \cdot \hat{\varphi}
\]

\[
\quad \quad \quad + \frac{P \nabla \left( \hat{\varphi}_1 (1 - \hat{\varphi}_1) (1 - 2 \hat{\varphi}_1) \right)}{\varepsilon} \cdot \hat{\varphi}_2 D \nabla \hat{u}_2 \cdot \nabla \hat{\varphi}_2 \cdot \hat{\varphi}_1 (c \hat{u}_1 + \lambda) \cdot \hat{\varphi}
\]

\[
J_B^\varepsilon (\hat{\varphi}, \hat{\varphi}, \hat{\varphi}_1, \hat{\varphi}_2) := \int_\Omega \partial_t \left( \hat{\varphi}_1 \right) \cdot \hat{\varphi} + \left( \frac{-\hat{\varphi}_1 \nabla \hat{u}_1 \cdot \nabla \hat{\varphi}_1}{\hat{\varphi}_1 \nabla \hat{u}_2 + \hat{\varphi}_2 D \nabla \hat{u}_2} \cdot \nabla \hat{\varphi}_2 \right) \hat{\varphi}
\]

\[
\quad \quad \quad + \left( \hat{\varphi}_1 (P(\hat{u}_2 + \lambda) - A) \hat{\varphi}_1 (c \hat{u}_1 + \lambda) \right) \cdot \hat{\varphi}
\]

The regularisation variables and functions taken can be seen in Table 3.5 while the code for creating the \texttt{PhaseModel} class can be see in Listing 21. The well-posedness of the phase field equations can be found in [42] while the formal asymptotics were carried out in [43].

### 3.5.5 Fracture

The fracture propagation is a \textit{Model one-phase} type energy as it only has one phase. For the following presentation the asymptotics for the resulting system were...
\[
W(\hat{\phi}) = \begin{pmatrix} \hat{\phi}_1^2 & \hat{\phi}_2^2 \\ \hat{\phi}_1 & \hat{\phi}_2 \end{pmatrix}
\]

\[
I(\hat{\phi}, \hat{a}(\hat{u}, \nabla \hat{u})) = \begin{pmatrix} \hat{\phi}_1 & \hat{\phi}_2 \end{pmatrix}
\]

Table 3.5: Smoothing functions and parameters for tumour growth $P^\epsilon$ gradient flow.

<table>
<thead>
<tr>
<th>$q(\hat{\phi})$</th>
<th>$f(\hat{\phi})$</th>
<th>$\epsilon$</th>
<th>$\Delta t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\hat{\phi}_1^2, \hat{\phi}_2^2)$</td>
<td>$(\hat{\phi}_1, \hat{\phi}_2)$</td>
<td>0.05</td>
<td>$10^{-3}$</td>
</tr>
</tbody>
</table>

Listing 21: Python code for class Tumour.

```python
uf1Model = PhaseModel(Tumour, epsilon=0.05, dt=0.001)
uf1Model.bulkSmoothdict['Q',0] = lambda phi, phiK: as_vector([phi[0] * phi[0], phi[1] * phi[1]]
```
we therefore get the functions

\[
E^\epsilon_{\hat{\varphi}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},
\]

\[
Q^\epsilon_{\hat{\varphi}} = (\hat{\varphi})^2 \sigma(\nabla \tilde{u}),
\]

\[
F^\epsilon_{\hat{\varphi}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.
\]

Applying Algorithm 3 we have that

\[
K^\epsilon = \{ \hat{v} \in TG^\epsilon : \hat{v} \leq 0 \}
\]

Applying the rest of Algorithm 4, and making use of the fact that in the one phase case the projection operator is the identity Eq. (3.3). We get

\[
J^\epsilon_{PF}(\hat{\varphi}, \tilde{u}, \hat{\eta}) := \int_{\Omega} \tau_0 \partial_t \hat{\varphi} \eta + \epsilon \beta \nabla \hat{\varphi} \cdot \nabla \hat{\eta} + \frac{\beta}{\epsilon} (1 - \hat{\varphi}) \eta + \hat{\varphi} \sigma(\nabla \tilde{u}) : \mathcal{E}(\nabla \tilde{u}) \hat{\eta} \, d\mathcal{L}^2. \tag{3.71}
\]

and the operator \(J^\epsilon_B\) as

\[
J^\epsilon_B(\hat{\varphi}, \tilde{u}, \hat{\zeta}) := \int_{\Omega} \hat{\varphi}^2 \sigma(\nabla \tilde{u}) : \nabla \hat{\zeta} \, d\mathcal{L}^2. \tag{3.72}
\]

A summary of the regularisation parameters can be seen in Table 3.6. Listing 22 contains the relevant code.
```
ufllModel = PhaseModel(Fracture, epsilon=epsilon, dt=dt, constrained = True)
ufllModel.bulkSmoothdict['Q', 0] = lambda phi, phiK: as_vector([phi[0] * phi[0]])
ufllModel.bulkSmoothdict['Q', 1] = lambda phi, phiK: as_vector([phi[0] * phi[0]])
ufllModel.interpolate = InterpolateOne
```

Listing 22: Python code for class Fracture.

### 3.5.6 Multi-phase curvature flow

This follows similarly to the 2-phase mean curvature flow. An application of Algorithm 1 gives

\[ F^s := \sum_{i<j}^3 \int_{\mathcal{H}^1} \beta_{ij} \, dH^1 \rightarrow \]

\[ F^c := \int_{\Omega} \left( \gamma_{12} \, \gamma_{13} \, \gamma_{23} \right) \left( \begin{array}{ccc} 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{array} \right) \left( \begin{array}{c} |\nabla \phi_1|^2 \\ |\nabla \phi_2|^2 \\ |\nabla \phi_3|^2 \end{array} \right) + \]

\[ \frac{9}{2} \epsilon \left( \begin{array}{c} (\phi_1)^2(1-\phi_1)^2 \\ (\phi_2)^2(1-\phi_2)^2 \\ (\phi_3)^2(1-\phi_3)^2 \end{array} \right) \right] dL^2. \tag{3.74} \]

Applying Algorithm 4 we have

\[ J_{PF}(\phi, \bar{u}, \bar{\eta}) := \]

\[ \int_{\Omega} \tau_0 \beta_1 \phi \cdot \bar{\eta} - \left( \gamma_{12} \, \gamma_{13} \, \gamma_{23} \right) \left( \begin{array}{ccc} 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{array} \right) \left( \begin{array}{c} |\nabla \phi_1 \cdot \nabla \phi_1| \\ |\nabla \phi_2 \cdot \nabla \phi_2| \\ |\nabla \phi_3 \cdot \nabla \phi_3| \end{array} \right) dL^2 \]

\[- \frac{9}{2} \epsilon \left[ \left( \gamma_{12} \, \gamma_{13} \, \gamma_{23} \right) \left( \begin{array}{ccc} 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{array} \right) \left( \begin{array}{c} \phi_1(1-\phi_1)(1-2\phi_1) \\ \phi_2(1-\phi_2)(1-2\phi_2) \\ \phi_3(1-\phi_3)(1-2\phi_3) \end{array} \right) \right] \cdot \bar{\eta}. \tag{3.75} \]

We do not expand the projection operator in Eq. (3.75) as the algebra becomes messy however this is easily done symbolically in ufl. The code for implementing this can be seen in Listing 23.
phaseField = PhaseModel(Mcf3p, epsilon, dt=1e-3)

Listing 23: Python code for class Mcf3p.
Chapter 4

Time Stepping

4.1 Time discretisation

In Chapter 3 we discussed the automatic implicit discretisations of a $\mathcal{P}$ gradient flow to give a backward Euler scheme. In this chapter we discuss how to modify the class that defines the $\mathcal{P}$ gradient flow as well as the PhaseModel class to implement more complicated time discretisations. The time discretisation is done at the UFL level, we do not discuss consequences to the sharp limit or stability issues of the resulting numerical scheme.

We describe the functions that may be modified to include an explicit term. After introducing the modifications we again present the examples in Chapter 3 and make modifications to the time stepping schemes. This will then be the final version used in our numerical experiments.

In Section 4.1.1 we describe how to customise the time stepping in the balance laws. This is then followed by discretising the potential well $W$ in Section 4.1.3 and interpolation function $I$ in Section 4.1.4. Finally, in Section 4.1.6 we discuss a natural time discretisation that is provided when the energy is written in terms of the orientation angle $\theta$.

Implementing functions with explicit terms is done by including an extra parameter passed to the UFL classes. This is the explicit phase field $\hat{\varphi}^k$ or explicit bulk field $\tilde{u}^k$. As there is no inheritance structure on sharp model classes, deduction of how many arguments these functions take when explicit terms are presented is done within the InspectSharp class. This is further discussed in Chapter 5.
4.1.1 Balance laws

As defined in Eqs. (3.5) and (3.6) the functions $Q$ and $F$ in Eqs. (2.19) and (2.20) take as the first argument $\tilde{u}$. ($F$ has an extra “$\vec{x}$”). When regularised by Algorithm 2 and Algorithm 6 become $\tilde{u}^{k+1}$. Suppose the balance law we approximate is given by

$$\Delta \tilde{u} = \begin{cases} 2\tilde{u} & \text{in } \Omega^1, \\ 0 & \text{in } \Omega^2. \end{cases}$$

This would be entered in PHASEFIELD as in Listing 24. Applying Algorithm 6 the default time discretisation is

$$0 = \int_\Omega -\nabla \tilde{u}^{k+1} \cdot \nabla \zeta + \varphi^{k+1} 2\tilde{u}^{k+1} \zeta \, d\mathcal{L}^2. \quad (4.1)$$

We instead describe how to implement the discretisation Eq. (4.2)

$$0 = \int_\Omega -\nabla \tilde{u}^{k+1} \cdot \nabla \zeta + \varphi^{k+1} 2\tilde{u}^{k} \zeta \, d\mathcal{L}^2. \quad (4.2)$$

This is done by passing in an extra parameter as the second term in $Q$ and $F$ as seen in Listing 25. It is important that $\tilde{u}^{k}$ is always passed in as a parameter after

```
def distQ(u, uk)
    return [[grad(u), grad(u)]]

def distF(u, x, uk)
    return [[2*uk, 0]]
```

Listing 24: Providing $Q$ with an explicit term.

```
def distQ(u)
    return [[grad(u[0]), grad(u[0])]]

def distF(u, x)
    return [[2*u[0], 0]]
```

Listing 25: Providing $F$ and $Q$ with an explicit term.
the implicit term. The first is always $\bar{u}^{k+1}$ (implicit) and the second is always $\bar{u}^k$ (explicit); it is not dependent on the naming of the parameters. While changing the discretisation of $Q$ and $F$ is simple, it is currently not possible to enter a custom discretisation of the $E$ term without inheriting from and overloading the \texttt{PhaseModel} class directly.

4.1.2 Smoothing function in balance laws

In Section 3.4.5 we described how to customise the smoothing of the balance laws with custom interpolation functions with all terms taken implicitly. Like many of the functions it is also possible to take the terms explicitly instead of implicitly. An extra argument can be provided which is the explicit term $\hat{\varphi}^k$. Using this term instead of $\hat{\varphi}^{k+1}$ therefore gives a smoothing depending on the current phase. Take for example Eq. (4.2) which we instead discretise as

$$0 = \int_{\Omega} -\nabla \bar{u}^{k+1} \cdot \nabla \zeta + \hat{\varphi}^k 2\bar{u}^k \zeta \, dL^2.$$  

(4.3)

We do this by setting the member variable \texttt{bulkSmoothdict} on the object \texttt{uflModel}. We give an example of smoothing a more complicated equation in Section 4.2.2.

\begin{verbatim}
1 uflModel.bulkSmoothdict['F', 0] = lambda phi, phiK: as_vector([phiK[0]])
\end{verbatim}

4.1.3 Potential well

When defining a well (as described in Section 3.4.3) the \texttt{uflModel} is passed as the first parameter. We define a quadratic potential well with all the terms taken explicitly

$$W(\hat{\varphi}^{k+1}, \hat{\varphi}^k) = (\hat{\varphi}_1^k)^2 (\hat{\varphi}_2^k)^2.$$  

(4.4)

This is defined in Listing 26.

\textbf{Remark 4.1.1.} \textit{When defining a custom potential well, one has to be careful that it is calibrated properly.}

4.1.4 Interpolation function $I$

The interpolation function $I$ takes as its three arguments $\phi, \phi^k, a$. We discussed in Section 3.4.4 implementing the first and third arguments. The second argument
1. \texttt{def Explicit(phaseModel, \phi, \phiK):}
2. \hspace{1em} return (\phiK[0] \cdot \phiK[0] \cdot \phiK[1] \cdot \phiK[1])

Listing 26: Custom discretisation of the potential well.

1. \texttt{def a(u, \grad u, un, \grad un)}

Listing 27: Custom discretisation in $a(\tilde{u}, \nabla \tilde{u})$.

is the explicit term $\hat{\phi}^k$.

4.1.5 Bulk density $\hat{a}$

The bulk energy density $\hat{a}$ takes as its first two arguments $\tilde{u}$ and $\nabla \tilde{u}$. This can be extended up to four arguments where the third and fourth argument are $\tilde{u}^k$ and $\nabla \tilde{u}^k$ respectively as seen in Listing 27.

4.1.6 Energy written in terms of an orientation angle

In Section 3.2.1 a model was provided for the surface energy written in terms of an orientation angle $\theta$. The reason for the split of the general anisotropic energy and a 2d anisotropic energy written in terms of $\theta$ is the following. Assuming $n = 2$ let $\theta$ be the angle between from positive $x$-axis in the counter-clockwise direction. The surface energy $\gamma$ can then often be written as a zero homogeneous function of $\theta$ which is then extended. This is a \textit{Model aniso-theta} energy (see Section 3.2.1). The regularised energy is then given as

$$\int_{\Gamma_{12}} \frac{1}{2} \gamma^\theta(\theta(\tilde{\nu}_{12}))^2|\tilde{\nu}_{12}|^2 \, d\mathcal{H}^1.$$ 

The orientation angle is defined as $\theta(\tilde{\nu}) = \text{atan2}(\tilde{\nu}_2, \tilde{\nu}_1)$. Regularising this gives

$$\int_{\Omega} \frac{1}{2} \gamma(\theta(\nabla \hat{\phi}_1))^2|\nabla \hat{\phi}_1|^2 \, d\mathcal{L}^2.$$ 

98
As before we calculate the variation Eq. (3.15)

\[
\langle \delta F, \hat{\eta} \rangle = \int_{\Omega} \gamma(\theta)^2 \nabla \hat{\varphi}_1 \cdot \nabla \eta + \frac{1}{2} \frac{d}{d\nabla \hat{\varphi}_1} \left( \gamma(\theta(\nabla \hat{\varphi}_1))^2 \right) |\nabla \hat{\varphi}_1|^2 \cdot \nabla \eta \, d\mathcal{L}^2,
\]

\[
= \int_{\Omega} \gamma(\theta)^2 \nabla \hat{\varphi}_1 \cdot \nabla \eta + \gamma'(\theta) \frac{d(\theta(\nabla \hat{\varphi}_1))}{\nabla \hat{\varphi}_1} \gamma(\theta) |\nabla \hat{\varphi}_1|^2 \cdot \nabla \eta \, d\mathcal{L}^2. \tag{4.5}
\]

A short calculation shows that

\[
\frac{d}{d\nabla \hat{\varphi}_1} \arctan2((\nabla \hat{\varphi}_1)_1, (\nabla \hat{\varphi}_1)_2) = \frac{1}{|\nabla \hat{\varphi}_1|^2} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \nabla \hat{\varphi}_1.
\]

Substituting Eq. (4.6) into Eq. (4.5) we get

\[
= \int_{\Omega} \gamma(\theta) \left[ \gamma(\theta) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \gamma'(\theta) \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right] \nabla \hat{\varphi}_1 \cdot \nabla \eta \, d\mathcal{L}^2. \tag{4.6}
\]

In this form it is easy to see that a natural linearisation given by

\[
= \int_{\Omega} \gamma(\theta^k) \left[ \gamma(\theta^k) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \gamma'(\theta^k) \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right] \nabla \hat{\varphi}^{k+1}_1 \cdot \nabla \eta \, d\mathcal{L}^2,
\]

where \( \theta^k := \arctan2((\hat{\varphi}_1)^k)_y, (\hat{\varphi}_1)^k)_y \). In order to make use of this natural discretisation two things must be done. First, as described in Section 3.5.3, the surface energy must be defined in terms of a density named \texttt{gammaTheta}, to make sure the resulting discretisation is the one above. When creating the \texttt{PhaseModel} class the named argument \texttt{thetaSemi} must be set to true. See Listing 28 for an example.

\subsection{4.2 Examples}

We apply custom time discretisations to the examples from Section 3.5.

\subsection*{4.2.1 Dendritic growth}

As discussed in Section 3.5.3, as our energy is written in terms of an orientation angle \( \theta \), in 2d we can take advantage of a natural semi-discrete implementation provided by \texttt{PhaseField} and defined in Section 4.1.6. To do this we pass the named argument \texttt{thetaSemi = True} to the constructor of the phase field class. This can be seen in Listing 28. The resulting semi discrete equations have already been presented in Section 3.5.3.
4.2.2 Tumour growth

We mimic the semi-discretisation from [43] and take a number of the $\tilde{u}$ terms in the $F$ distribution as well as the bulk potential $\hat{a}(\tilde{u}, \nabla \tilde{u})$ explicitly. As the component $\tilde{u}_1$ is only present in one domain it is natural to want to use a more degenerate form for the bulk smoothing. Higher order terms however make the equation highly non-linear which is why we take this term explicitly

$$q_1(\hat{\varphi}^{k+1}, \varphi^k) = \left( (\varphi_1^k)^2 \quad (\varphi_2^k)^2 \right).$$ (4.7)

The resulting semi-discretisation equations are therefore in terms of the functions $J_{PF}^1, J_{PF}^2, J_B^1, J_B^2$ in Section 3.4.2.

$$J_B^1(\hat{\varphi}^{k+1}, \tilde{u}^{k+1}, \tilde{\zeta}) = \int_{\Omega} \frac{\tilde{\varphi}_1^{k+1} - \tilde{\varphi}_1^k}{\Delta t} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot \tilde{\zeta} \, d\mathcal{L}^2$$

$$J_B^2(\hat{\varphi}^{k+1}, \tilde{u}^{k+1}, \tilde{\zeta}) = \int_{\Omega} \left(-\begin{pmatrix} \varphi_1^k \nabla \tilde{u}_1^{k+1} \\ \varphi_1^{k+1} \nabla \tilde{u}_2^{k+1} \end{pmatrix} \cdot \nabla \tilde{\zeta} + \begin{pmatrix} 0 \\ D \nabla \tilde{u}_1^{k+1} \end{pmatrix} \cdot \nabla \tilde{\zeta} + \right.$$

$$\hat{\varphi}_1^{k+1} \begin{pmatrix} P \tilde{u}_2^{k+1} - A \\ C \tilde{u}_2^{k+1} \end{pmatrix} \cdot \tilde{\zeta} \, d\mathcal{L}^2,$$

and for the phase field equation

$$J_{PF}^1(\hat{\varphi}^{k+1}, \tilde{u}^{k+1}, \tilde{\zeta}) = 0,$$

$$J_{PF}^2(\hat{\varphi}^{k+1}, \tilde{u}^{k+1}, \tilde{\zeta}) = \int_{\Omega} \epsilon \beta \nabla \hat{\varphi}^{k+1} : \nabla \hat{\eta} + \frac{9\beta}{\epsilon} \left( \frac{\hat{\varphi}_1^{k+1}(1 - \hat{\varphi}_1^{k+1})(1 - 2\hat{\varphi}_1^{k+1})}{\hat{\varphi}_2^{k+1}(1 - \hat{\varphi}_2^{k+1})(1 - 2\hat{\varphi}_2^{k+1})} \right) \cdot \hat{\eta}$$

$$+ (b \tilde{u}_1^{k+1} + a \tilde{u}_2^{k+1}) \begin{pmatrix} -1 \\ 1 \end{pmatrix} \cdot \hat{\eta} \, d\mathcal{L}^2.$$ 

The code for creating the `PhaseModel` class can be seen in Listing 30.
4.2.3 Fracture

Similarly to the Mullins-Sekerka problem we ensure the bulk field is killed sufficiently well in the crack and by implementing higher order bulk smoothing but we take the terms explicitly along with a small extra small parameter $\eta$ for numerical reasons.

$$q(\hat{\phi}) = \frac{\eta + (\hat{\phi}^k)^2}{\eta + (\hat{\phi}^k)^2}. \tag{4.8}$$

Using Eq. (4.8) in the regularisation of the balance law Algorithm 2 and the subsequent discretisation, the semi-discretised balance law is given by in terms of $J^1_B$ and $J^2_B$ in Section 3.4.2

$$J^1_B(\hat{\phi}^{k+1}, \bar{u}^{k+1}, \tilde{\zeta}) = 0, \tag{4.9}$$

$$J^2_B(\hat{\phi}^{k+1}, \bar{u}^{k+1}, \tilde{\zeta}) = \int_{\Omega} (\hat{\phi}^k)^2 \sigma (\nabla \bar{u}^{k+1}) : \nabla \tilde{\zeta} \, d\mathcal{L}^2. \tag{4.10}$$

The code for editing the ufl class can be seen in Listing 31.

4.2.4 Multi-phase curvature flow

When performing multi-phase field simulations the choice of discretisation can have a big impact on accuracy as well as efficiency. Here we implement a convex concave splitting. The form of the well is taken from [18] where a triple potential well is split into concave and convex parts with the convex taken implicitly and concave explicitly. This ensures the decrease of energy for all time steps. We remark that the tests in [18] were done on the multi-phase Cahn-Hilliard equation rather than multi-phase Allen-Cahn equation as is done here.

Let $g(x) = x^2(1-x)^2$. A three phase consistent potential well is given by

$$W(\hat{\phi}) = \sum_{i=1}^{3} \Sigma_i g(\hat{\phi}_i),$$

where

$$\Sigma_1 = \gamma_{12} + \gamma_{13} - \gamma_{23},$$
$$\Sigma_2 = \gamma_{12} - \gamma_{13} + \gamma_{23},$$
$$\Sigma_3 = -\gamma_{12} + \gamma_{13} + \gamma_{23}.$$

We can split $g(x)$ in to convex part $g^+$ and concave part $g^-$. Letting $\Sigma_i^+ = \Sigma_i$.
max(Σᵢ, 0), Σᵢ⁻ = −min(Σᵢ, 0), i = 1, 2, 3 and

\[ g(x) = \begin{cases} \left( x - \frac{1}{2} \right)^4 + \frac{1}{16}(1 - 2(2x - 1)^2), & x \geq 0 \\ \frac{1}{g^+(x)} & x < 0 \end{cases} \]

and therefore to the following convex concave splitting of \( W \)

\[ W^+(φ) = \sum_{i=1}^{3} \frac{Σᵢ^+}{2} g^+(φᵢ) - \sum_{i=1}^{3} \frac{Σᵢ^-}{2} g^-(φᵢ), \]
\[ W^-(φ) = \sum_{i=1}^{3} \frac{Σᵢ^+}{2} g^-(φᵢ) - \sum_{i=1}^{3} \frac{Σᵢ^-}{2} g^+(φᵢ). \]

Taking the convex parts implicitly and concave explicitly we get the discretised well as

\[ W(φ^{k+1}, φ^k) = W^+(φ^{k+1}) + W^-(φ^k). \]

This function is implemented as the function `ConcaveConvex` in the file `python/auxfun.py` as described in Section 3.4.3. This potential well is then passed into the `PhaseModel` constructor as seen in Listing 32.
class Tumour:
    omega = cartesianDomain([0,0], [12.5, 12.5], [30, 30])
    endTime = 10.0

    mobility = 0

    def dirichlet(t, x):
        return {x[0]*x[1]>1e-12: [None, 1]}  # all except bottom

    def initial(x):
        r = sqrt(dot(x, x))
        theta = atan_2(x[1],x[0]) + conditional(x[1] < 0, 2*pi, 0)
        rTheta = 2 + 0.1 * cos(2*theta)
        return [conditional(r > rTheta, 0, 1), conditional(r < rTheta, 0, 1)], [0, 1]

    def gamma(nu):
        beta = 0.2*pi
        return [[0, beta], [beta, 0]]

    def a(u, gradu, uK, graduK):
        alpha = 5
        b = 1
        return [0, 2*b*u[0] + 2*alpha*uK[1]]

    def distE(u):
        return [[1, 0], [0, 0]]

    def distQ(u):
        D = 1
        return [[-grad(u[0]), zero(2)], [grad(u[1]), grad(u[1])]]

    def distF(u, x, uK)
        lam = 0
        P = 0.1
        C = 2
        A = 0
        return [[ P *( uK[1] + lam) - A, 0], [C *( u[1] + lam ), 0.]]

Listing 29: Semi-discretisation of tumour model.
```
ufModel = PhaseModel(Tumour, epsilon=0.01, dt=0.001)
ufModel.bulkSmoothdict['Q', 0] = lambda phi, phiK: [phiK[0] * phiK[0], phiK[1] * phiK[1]]
```

Listing 30: Entry of PhaseModel class in tumour model.

```
epsilon = 0.625
dt = 1e-2
eta = 1e-5
ufModel = PhaseModel(Fracture, epsilon=epsilon, dt=dt)
ufModel.bulkSmoothdict['Q', 0] = lambda phi, phiK: [eta + phiK[0]*phiK[0]]
ufModel.bulkSmoothdict['Q', 1] = lambda phi, phiK: [eta + phiK[0]*phiK[0]]
ufModel.interpolate = InterpolateOne
```

Listing 31: Bulk smoothing modification in fracture model.

```
from phasefield.auxfun import ConcaveConvex
phaseField = PhaseModel(Mcf3p, epsilon, dt=1e-3, well = ConcaveConvex)
```

Listing 32: PhaseModel class definition for three-phase mean curvature flow with concave convex potential well.
Chapter 5

PHASEFIELD

In this chapter we give a high level overview of PHASEFIELD. This is done with the aim of describing usage without going into implementational details. When describing the classes, we focus on descriptions of signatures and return types, with extra documentation for the interested reader given in Appendix C.

We start by describing the structure of the software in Section 5.1. The most important classes are described first. These are PhaseModel in Section 5.2 and PhaseStepper in Section 5.3. A helper class is briefly described in Section 5.4. Other FEM packages can be used to solve the UFL equations. We describe how to do this in Section 5.5 using an external dispatch class.

We expect the reader to broadly understand the following UFL concepts found in [4]

- Coefficients (ufl.coefficient).
- Trial/test functions (ufl.TestFunction/ufl.TrialFunction).

The installation of PHASEFIELD is described in Appendix A.

5.1 Structure of Software

We first list and describe the principle components of PHASEFIELD along with the files that they are stored in

- PhaseModel class - Transforms a $P$ gradient flow into a semi-discrete $P^\epsilon$ gradient flow.
- PhaseStepper class - Steps through the time loop and solves the semi-discrete $P^\epsilon$ gradient flow based on a finite element discretisation.
Figure 5.1: UML diagram with the main components of PhaseField.

- **Well** callable - Potential well $W$ (see Section 3.4.3).
- **Interpolate** callable - Interpolation function $I$ (see Section 3.4.4).
- **InspectSharp** class - Inspects the Sharp class and stores information for use by PhaseModel and PhaseStepper. E.g. Number of phases, number of bulk equation, if energy is isotropic.
- **External** class - Dispatches calls to external FEM software.

The following files are stored within the sub directory PhaseField. We specify the components that they contain.

- auxfun.py - Well, Interpolate.
- stepper.py - PhaseStepper.
- ufl_phasefield.py - PhaseModel.
- InspectSharp.py - InspectSharp.
The dependencies and structure are displayed as a UML diagram in Figure 5.1, which we explain. `PhaseModel` does the manipulation of the UFL forms and produces a semi-discrete $P^n$ gradient flow. It requires the definition of a $P$ gradient flow, provided to it in way of a sharp class, as well as the functions and parameters specified in Table 3.1. These include `well` and `interpolate` which are contained in the `auxfun` module.

The sole public member function in `PhaseModel` is `setupPhase`. This takes as arguments the implicit and explicit coefficients for $\hat{\varphi}$ and $\tilde{u}$. It returns the UFL form of the equation. This is the function called from the `PhaseStepper` class. It is important to remark that the `setupPhase` method does not create the trial/test functions or coefficients that are used to build the model. It only manipulates what it has been passed. It is this feature that makes `PhaseModel` depend upon UFL and not any finite element back-end. We are therefore able to use different FEM packages to solve the resulting system of phase field equations. See Example 5.2.1 for how to build the UFL model without the `PhaseStepper`.

The `PhaseStepper` class constructs the grid, discrete functions, trial/test functions and coefficients; in addition to the time stepping and solving of the semi discrete scheme. It depends heavily on the external dispatch class for providing the function spaces, models and coefficients. These are described in Section 5.5. The `PhaseStepper` additionally depends on the `PhaseModel` class to provide the UFL form of the equations to be solved through the member function `PhaseModel.setupPhase`. In theory, a different class could be passed into the constructor as long as it provides a `setupPhase` method. This would be useful if one wishes to define their own phase field equations but use the `PhaseStepper` class to carry out the computation or do the dimension reduction described in Section 5.3.3. The `PhaseStepper` does the setting up of grids and time stepping. We elaborate on this in Section 5.3.

As stated `PhaseStepper` does not depend on any external FEM packages directly. Instead it depends on the class `External`. Either importing `useDUNE.py` or `useFenics.py` populates the `External` class with the member functions depending on which file is imported. There are a number of methods that need to be defined. We describe this in Section 5.5 Additional packages that can handle UFL forms can be easily added by providing these functions.

5.2 The `PhaseModel` class

The `PhaseModel` class does manipulation of coefficients passed to it in its constructor. In our case these are provided by the `PhaseStepper`. It returns the equations in Section 3.4.2 via the method `setupPhase`.
We give a user-motivated overview of the `PhaseModel` class if one wishes to use it stand-alone. There are two methods that need to be explained for this purpose. The constructor `__init__`, and the `setupPhase` method.

5.2.1 Constructor

We start by describing the constructor. The signature can be seen in Listing 33. We discuss the important parameters. The `sharpCls` must be a class, as described in Section 2.2, that completely specifies a $P$ gradient flow. The well is specified in Section 3.4.3 and is passed an instance of the `PhaseModel` class as an argument. This is because the well often depends on information such as the number of phases within `PhaseModel` through the composite member variable `sharpCls`. If specifying a phase field model, $\epsilon$ must be set, and similarly $\Delta t$ if the problem is time dependent. `well` must be set if choosing a custom potential well. By default this is initialised to a default well which can be seen in the file `auxfun.py`. `constrained` must be set if using an obstacle potential. The last argument `thetaSemi` is a bool that implements the specific discretisation scheme as described in Section 4.1.6.

5.2.2 Additional methods

The signature for the `setupPhase` method that is called from the `PhaseStepper` class can be seen in Listing 45. It takes as arguments UFL coefficients in the following order; $\hat{\varphi}_{k+1}, \hat{\varphi}_{k}, \tilde{u}_{k+1}, \tilde{u}_{k}, \hat{\eta}, \tilde{\zeta}$. These are manipulated and the equations in Section 3.4.2 are returned. The last two arguments are a spatial co-ordinate and constant type. These are not strictly required if the user does not wish to use constants in the ufl form, or does not use a spatial co-ordinate $x$ in the balance laws forcing term $F$.

**Example 5.2.1** (Building UFL model without the `PhaseStepper`). We give an example of how to build the UFL equations without using the `PhaseStepper`. The code in Listing 35 can be found in the “test” direction of `PhaseFIELD`. We build the UFL form of the model described by mean curvature flow example in Listing 7. In the
Listing 34: Signature for \texttt{PhaseModel.\_setupPhase}.

\begin{verbatim}
(self, phi, phiN, u, uN, vPhi, vU, x=None, constant=None)
\end{verbatim}

Listing 35: Code demonstrating the creation of a UFL form without using the \texttt{PhaseStepper} class.

```
phaseField = PhaseModel(Mcf, epsilon=0.03, dt=0.001)
space = VectorElement("Lagrange", triangle, 1, 2)
vPhi = TrialFunction(space)
phi = Coefficient(space)
phiN = Coefficient(space)
form = phaseField.\_setupPhase(phi, phiN, None, None, vPhi, None)

space = FiniteElement("Lagrange", triangle, 1)
vPhi = TrialFunction(space)
phi = Coefficient(space)
phiN = Coefficient(space)
extend = lambda p: as_vector([p,1-p])
form = phaseField.\_setupPhase(extend(phi), extend(phiN), None, None, extend(vPhi), None)
```

The first section of the code in Listing 35 we build a UFL form using both phase field variables one for each phase. First creating the \texttt{PhaseModel} class. Following this the trial functions and coefficients are setup on a space of dim range 2. Finally we call the \texttt{setupPhase} method on the \texttt{PhaseModel} class to return the form.

The second section is similar, but we only setup a space of dim range 1. We therefore here only want $\hat{\phi}_1$ in the resulting ufl form. We then substitute a vector with $\hat{\phi}_1$ and $1 - \hat{\phi}_1$ into the \texttt{setupPhase} method.

### 5.3 The \texttt{PhaseStepper} class

The \texttt{PhaseStepper} class generates the UFL coefficients, trial functions, test functions, grid, space and solves the resulting system. The stepper we present here is a proof of concept and there are many other possible implementations that equally can compute the solution of the forms generated by the \texttt{PhaseModel} class.
After the construction of the `PhaseModel` class, the `PhaseStepper` class must be created. This takes arguments in its constructor that are required for the solving of the system. The constructor can be seen in Listing 36. Here we give a description of the arguments.

**Remark 5.3.1.** When using the TNNMG solver the problem must be linear in this version.

The default solver used is “gmres”. A list of linear algebra solvers used can be seen in Table 5.1 along with the corresponding storage used. The parameters that are passed to the solver are set in `solverParameters`. For a list of possible parameters see [33]. `orderFe` is the order of the finite element space used and by default is set to 1.

### Pre, phase and post bulk

Often the equations (3.4.2) will decouple and can be conceptually split into the following procedure. The balance laws are split into three parts and then the solver consists of the following steps.

- Solve first set of balance laws.
- Solve phase field equations and second set of balance laws.
- Solve final set of balance laws.

If this is the case, one may specify the balance laws to solve first which we call the \texttt{preBulk}, the phase field equations to solve second together with \texttt{phaseBulk} and the balance laws to solve last \texttt{postBulk}. Examples of this where the equations decouple can be found in Sections 6.4 and 6.5.

\textbf{Remark 5.3.2.} \texttt{PhaseStepper} does no validation on whether this splitting is consistent.

\section*{5.3.2 Additional methods and variables}

\subsection*{Adaptivity}

Phase field equations lend themselves to adaptivity schemes as the most variable behaviour happens in a small band around the interface. The simplest marking strategy involves refining when the gradient of the phase field variables are large and coarsening when small. In the bulk a much coarser grid can be used however it still needs to be fine enough to resolve the balance laws defined there.

The \texttt{PhaseStepper} class provides two member variables which are adequate for common strategies. The first variable in Listing 37 is \texttt{indicator}. This is a UFL expression that when evaluated gives a real number. This number is used to either refine or coarsen. By default this is

\begin{equation}
\sum_{i=1}^{n} |\nabla \hat{\phi}_i|^2.
\end{equation}

The second variable is \texttt{defaultRefine}; this is a list with four elements, all floats. The first two are the threshold for refining and coarsening respectively while

\begin{verbatim}
self.indicator = sum(dot(grad(self.solution[i]), grad(self.solution[i])) for i in range(0,self._uflModel.inspectDict['dimRangePhase']))
self.defaultRefine = [1.4, 1.2, 4, 14]
\end{verbatim}

Listing 37: Grid refinement in \texttt{PhaseStepper}. 

1  self.indicator = sum(dot(grad(self.solution[i]), grad(self.solution[i])) for i in range(0,self._uflModel.inspectDict['dimRangePhase']))
2  #[refineTolerance, coarsenTolerance, minLevel, maxLevel]
3  self.defaultRefine = [1.4, 1.2, 4, 14]

expression that when evaluated gives a real number. This number is used to either refine or coarsen. By default this is
the last two are the minimum and maximum number of grid refinements to use. The refinement strategy is described in Algorithm 7 Default parameters are provided,

\begin{algorithm}
\caption{Adaptive grid strategy.}
\begin{algorithmic}
  \IF {$\text{marker} > \text{threshold refine}$ \AND $\text{refinement level} < \text{max level}$}
    \STATE refine grid
  \ELSIF {$\text{marker} < \text{threshold coarsen}$ \AND $\text{refinement level} > \text{min level}$}
    \STATE coarsen grid
  \ENDIF
\end{algorithmic}
\end{algorithm}

as in Listing 37. See Section 6.5 for an example of changing the default indicator. In Listing 39 we change the maximum grid refinement level to 14, from its default of 12, while keeping the default indicator.

After setting up the grid at the start of the simulation, the method \texttt{gridSetup} should be called on the instance of the \texttt{PhaseStepper} object. An example of this can be seen in Listing 39. The first parameter is the number of uniform grid refinements while the second number is adaptive, which in Listing 39 are 8 and 16 respectively. The last parameter is specific to fracture models. It allows one to compute an initial smoothing step of the phase field variable. If set it computes a static iteration step by setting $\tau_0 = 0$ and solving the phase field equation with the $\epsilon$ passed to it, for an example see Listing 42. The signature and doc string for the \texttt{gridSetup} method can be seen in Listing 47.

\textbf{Next time}

Given the structure of the software we describe the basic usage given a simple example. Assuming we have a given \texttt{sharpClass} that defines a $P$ gradient, the code in Listing 39 performs construction, adaptation and the time loop. It is the basic template used to simulate all the examples. We describe each feature in relation to its implementation in Listing 39.

\begin{lstlisting}[language=Python]
    (self, numGlobalRefine, numLocalRefine, smoothingEpsilon=None)
\end{lstlisting}

Listing 38: Signature for \texttt{PhaseStepper.gridSetup}. 112
phaseField = PhaseModel(sharpClass, epsilon = 0.015, dt = 0.0005)
stepper = PhaseStepper(phaseField)
stepper.dt = 0.01
stepper.defaultRefine = [1.4, 1.2, 4, 14]
stepper.gridSetup(8,16)
while stepper.time < sharpClass.endTime:
    stepper.nextTime()
    stepper.adapt()

Listing 39: Basic usage of PhaseField.

Changing $\Delta t$ and $\epsilon$

During the evolution it is often desirable to have a dynamic step size $\Delta t$ or interfacial width $\epsilon$. These are implemented as constants in the UFL model and it is therefore possible to change them by directly accessing them in the PhaseStepper class. In Section 5.3.2 we initially set $\Delta t = 10^{-4}$ before changing it immediately after initialisation to $\Delta t = 0.01$.

Time Loop

Having set up the PhaseStepper class and refined the initial grid, we describe the time loop. PhaseStepper has a public member variable PhaseStepper.time that can store the current time. The member function PhaseStepper.nextTime() must be called to compute the solution at the next time. If one wishes to adapt the grid at any point this is done with the public member function PhaseStepper.adapt(). The time looping can be seen in Listing 39.

Saving output

There are two additional variables that may be prescribed in the SharpClass. These are saveStep and fileBase. They are options that enforce the automatic saving of a vtk file of the solution. fileBase is a string that specifies the start of the name of vtk files saved. saveStep is a float that specifies the intervals that a vtk file is saved at. Internally, there is an output in PhaseStepper that is called after each time loop and when the time is equal to saveStep. This output method can be called
manually with the argument \texttt{force=True} if one wishes to force the output of a vtk file at a specific point in time. The output solution is always of the formal where the first \( m - 1 \) components are the phase field and the result are the balance laws.

### 5.3.3 Dimension reduction

The \texttt{PhaseStepper} class simplifies the UFL form returned by the \texttt{PhaseModel.setupPhase} method. The following is completely internal to \texttt{PhaseStepper} and requires no user input or modification. We describe the procedure for completeness.

As the phase field variables must sum to 1 (\( \mathbf{\varphi} \in \Sigma \)) we can reduce the dimension of the system by performing a transformation of variables. This enables us to solve \( m - 1 \) equations for \( \mathbf{\varphi} \) instead of the full system. To reduce the dimensions of the problem we replace \( \mathbf{\varphi}_m \) by \( 1 - \sum_{i=1}^{m-1} \mathbf{\varphi}_i \). In the two phase case this gives

\[
\mathbf{\varphi} = \begin{pmatrix} \mathbf{\varphi}_1 \\ \mathbf{\varphi}_2 \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{\varphi}_1 \\ 1 - \mathbf{\varphi}_1 \end{pmatrix}.
\]

For the general multi-phase case we make the following substitution

\[
\mathbf{\varphi} = A \mathbf{\varphi} + \mathbf{d}, \quad (5.1)
\]

with \( A \) and \( \mathbf{d} \) defined by

\[
A_{ij} = \begin{cases} 
\delta_{ij} & \text{if } 1 \leq i, j \leq m - 1, \\
-1 & \text{if } i = m \text{ and } 1 \leq j \leq m - 1, \\
0 & \text{if } i = j = m.
\end{cases}
\]

\[
\hat{d}_i = \begin{cases} 
1 & \text{if } i = m, \\
0 & \text{otherwise}.
\end{cases}
\]

Let the matrix \( B \) be given by

\[
B_{ij} = \begin{cases} 
\delta_{ij} & \text{if } 1 \leq i, j \leq m - 1, \\
0 & \text{if } i = j = m.
\end{cases}
\]

After the substitution we are left with the following set of equations returned from
the PhaseModel class simplified from Section 3.4.2.

\[
\begin{pmatrix}
J^1_{PF}(A\hat{\varphi}^{k+1} + \hat{d}, \tilde{u}^{k+1}, B\tilde{n}) & J^2_{PF}(A\hat{\varphi}^{k+1} + \hat{d}, \tilde{u}^{k+1}, B\tilde{n}) \\
J^1_{B}(A\hat{\varphi}^{k+1} + \hat{d}, \tilde{u}^{k+1}, B\hat{\zeta}) & J^2_{B}(A\hat{\varphi}^{k+1} + \hat{d}, \tilde{u}^{k+1}, B\hat{\zeta})
\end{pmatrix}
\]

(5.2)

**Remark 5.3.3.** Generic transformations of the form Eq. (5.1) have been considered in [74].

### 5.4 The InspectSharp class

The class `InspectSharp` is in the file `python/InspectSharp.py`. This is a helper class; it is used to inspect the Sharp class as a number of attributes are needed by the PhaseStepper and PhaseModel. When debugging models, checking that the dictionary variables contained within the InspectSharp is often the first step.

One of the important detections the InspectSharp class does is whether the user-defined $Q$, and $F$ are independent of the phase. For example, take the result of the regularised distribution Eq. (3.62) which is

\[ Q^\epsilon = -\alpha(\hat{\varphi}_1 + \hat{\varphi}_2)\nabla \tilde{u}. \]

After the transformation described in Section 5.3.3 this is

\[ Q^\epsilon = -\alpha(\hat{\varphi}_1 + (1 - \hat{\varphi}_1))\nabla \tilde{u}. \]

UFL does some simplification but often does not pick up cancellations such as this. As situation is common we build it into InspectPhase to detect this scenario and produce the regularisation as

\[ Q^\epsilon = -\alpha \nabla \tilde{u}. \]

### 5.5 The External class

In order to use PHASEFIELD with other FEM software External needs to be dispatched too. This is done currently in the files `useDune.py` and `useFenics.py`, which must be run to populate External with the appropriate members. We list the necessary members that need to be defined for the PhaseStepper class to work:

- **constant** - Constant expressions for use in UFL form.
- **dirichletBC** - Dirichlet boundary conditions for $\tilde{u}$.  

115
• **mesh** - Gridview for computations.

• **adaptMesh** - Adaptive mesh with marker.

• **globalRefine** - Global refinement of the mesh.

• **discreteFunctionSpace** - Discrete function space.

• **interpolate** - Interpolates an expression over the discrete function space.

• **assign** - Assigns the degrees of freedom from one discrete function to another.

• **scheme** - Scheme for solving UFL expression with parameters.

• **solve** - Solves the scheme and assigns solution into target.
Chapter 6

Software Validation

The aim of this chapter is to validate PHASEFIELD. We run simulations of the examples described previously and compare them to exact solutions, or the result of the same simulations presented in the literature. This is done to confirm that the software is working as desired. We remark that the aim is not to present new numerical results but to test that PHASEFIELD works as expected.

In Section 6.1 we compute the solution to mean curvature flow with the three different potential wells available in PHASEFIELD. Our aim is not to compare the relative merits and disadvantages of using each well as this has already been well documented, but to show that PHASEFIELD works as required in approximating mean curvature flow. The Mullins-Sekerka problem is known to conserve the area inside the interface $\Gamma^{12}$ we perform a long time simulation in Section 6.2 to ensure that this behaviour is replicated in the phase field setting. In Section 6.3 we compute the solution to the dendritic growth problem. This model has already been implemented in both UFL and DUNE. Here the aim was to show how it fit into the framework and ensure that the automatic differentiation of the energy has worked. This is done by comparing to a solution computed in DUNE-FEMPY. In Section 6.4 we compute the tumour growth model matching the parameters from [43]. We qualitatively compare the solution to ensure the correct behaviour is obtained. In Section 6.5 we validate the fracture model by comparing to the analytical solution for the starting time. Finally, in Section 6.6, we perform a number of tests for multi-phase mean curvature flow. In the first we set one of the phase field variables to zero and ensure that we recover mean curvature flow as computed in Section 6.1. We then compute a three-phase simulation with a triple junction.
6.1 Mean curvature flow

We the solution of our phase field simulations to the well known exact solution of a shrinking circle. Given an initial circle of radius $r$, the radius of the exact solution at time $t$ is

$$r(t) = \sqrt{r_0^2 - 2t}.$$ 

We compute the radius for 3 different potential wells, Implicit, ConcaveConvex and Obstacle. We use a direct solver for the linear system in the case of the smooth wells. For the Obstacle well we use the TNNMG solver. An approximate radius $r_{pf}$ can be obtained by integrating $\hat{\phi}_1$ over the domain and identifying $\hat{\chi}_1$ with the points where $\hat{\phi}_1 < 0.5$.

In addition to the sharp model for mean curvature flow summarised in Table 3.2 we took the following parameters. For the Implicit, Concave and Obstacle simulations respectively $\Delta t = 10^{-4}, 10^{-5}, 10^{-4}$, as the arguments to the gridSetup method $(14,0), (14,0), (13,15)$ and the maximum refinement for the adaptivity was taken as $16, 13, 15$

In Figures 6.1 to 6.3 are plots of time vs the approximate radius. In each plot is also the exact solution. In all three cases as we decrease $\epsilon$ we get convergence to the exact solution.

6.2 Mullins-Sekerka

It is well known that the Mullins-Sekerka problem decreases the length of the interface $\Gamma^{12}$ and preserves the area inside it. We choose this attribute to test in the phase field setting. The sharp model defined in Listing 8 is used along with the creation of the PhaseModel class from Table 3.3, where $\Delta t = 10^{-3}$ and $\epsilon = 0.05$. We use a maximum level of 15 with 8 initial refinements. We integrate the component $\hat{\phi}_1$ to approximate the area inside $\Gamma^{12}$. The results are shown in Figure 6.5. The area becomes less volatile as $\epsilon$ is taken sufficiently small. We highlight the scales on the $y$-axis which are extremely small.

6.3 Dendritic growth

We produce a simulation of the example presented in Section 3.5.3. This was first computed in [53] and later replicated in the DUNEmodule DUNE-FEMPY. With the
Figure 6.1: Mean curvature flow with implicit well.

Figure 6.2: Mean curvature flow with concave convex well.

Figure 6.3: Mean curvature flow with obstacle well.

Figure 6.4: Mean curvature flow with different potential wells, the legends show the radius for different values of $\epsilon$. 

119
Figure 6.5: Tracking the volume in Mullins-Sekerka example, $\epsilon = 0.2, 0.1, 0.05$.

Figure 6.6: Mullins-Sekerka simulation, $\hat{\varphi}_1$ displayed at times $t = 0, 5, 10, \epsilon = 0.025$.

sharp class in Listing 9 and the construction of the `PhaseModel` class in Listing 20 this is supplemented with the code to run and plot the simulation in Listing 40.

The result of the simulations can be seen in Figure 6.7. The difference in the UFL forms between the version presented here and in DUNE-FEMPY is that here we perform the differentiation symbolically from a given energy. The get to the resulting equations we have had to perform these extra steps

- Differentiate the energy symbolically.
- Project onto tangent space $T_G^\epsilon$.
- Reduce dimension of phase field equations from 2 to 1.

Even with this extra manipulation of UFL forms we have managed to replicate the previous results exactly.
Figure 6.7: Dendritic growth, $\hat{\varphi}_1$ displayed at $t = 0.8$, DUNE-FEMPY (top), PHASEFIELD (bottom).
stepper = PhaseStepper(phaseField, solverParameters)
stepper.gridSetup(12,12)

import os
currentPath = os.path.dirname(os.path.abspath(__file__))

while stepper.time < Crystal.endTime:
    stepper.nextTime()
    stepper.adapt()

plotComponents(stepper.solution, saveString=currentPath + '/images/crystalFinal.pdf')

Listing 40: The stepper code for dendritic growth.

6.4 Tumour growth

The aim of these simulations is to qualitatively replicate the results from [43]. The following code snippets specify the problem. The tumour sharp class is in Section 4.2.2 and the creating of the PhaseModel class is in Listing 30. These are taken together with the stepping code in Listing 41 which produces the simulations seen in Figure 6.8. There are a number of parameters passed to the solver, these include "ilu" preconditioning to speed up the computations. As expected we have a four fold symmetry and witness the tumour extending side branches into the healthy tissue in order to maximise surface area and absorb nutrient.

We remark that there are so slight differences between the simulation presented here and in [43]. The exact initial conditions are never stated so we took similar initial conditions from the authors publications, in addition to this we have used a smooth rather than an obstacle potential.

Figure 6.8: Tumour growth, $\hat{\phi}_1$ displayed at times $t = 5, 10, 20$. 

122
Listing 41: Stepper code for tumour growth.

6.5 Fracture

We compare our fracture model to a known analytical solution of the first time the crack starts to propagate. An analytical solution in terms of the stress intensity factors is available in [51]. We aim replicate the results from [54]. Let the stress intensity factor $K_I$ be given by

$$K_I = \sigma \sqrt{\pi a} \sqrt{\frac{2b}{\pi a}} \tan \frac{\pi a}{2b} C \left( \frac{a}{b} \right)$$

with

$$C(x) = \frac{0.752 + 2.02x + 0.37(1 - \sin(x))^3}{\cos(x)}.$$

The length of the initial crack is $a$ and $b$ is the width of the domain. From Table 2.9, $a = 25$, $b = 100$. $\sigma$ is the Cauchy stress tensor in Eqs. (2.92) and (2.93). Griffith’s growth criteria is that a crack will grow if the released strain energy is large enough to form new crack surfaces. We introduce the following parameters from linear elasticity

- $\nu$ - Poisson ratio.
- $E$ - Young’s modulus.
- $K$ - Bulk modulus.
These are related by the following formula

\[
\begin{align*}
K &= \lambda + \frac{2}{3}\mu, \\
E &= \frac{9K(K - \lambda)}{3K - \lambda}, \\
\nu &= \frac{1}{2} - \frac{E}{6K}.
\end{align*}
\]

In a plane setting Griffith’s growth criterion reads [50]

\[
\frac{1 - \nu^2}{E} K_T^2 \geq 2\gamma_{11}.
\]

The critical value for the mode I stress load is given by

\[
\sigma_{\text{cric}} = \sqrt{\frac{E}{1 - \nu^2}} \frac{2\gamma_{11}}{2b \tan \left( \frac{\pi a}{2b} \right)} C \left( \frac{a}{b} \right)^{-1}
\]

The parameters taken for the simulations can be seen in Listing 42. The computation uses Listing 12 for the sharp class along with the code for creating the `PhaseModel` class in Listing 31. The stepper for the time looping can be seen in Listing 42. As the equations decouple this implements a staggered scheme commonly seen in fracture simulations, see e.g [15]. Because of the symmetry of the problem we compute our solution on the top half of the domain and take appropriate Dirichlet boundary conditions along the bottom of the reduced domain. We define the start of the propagation to be the time when the node at the point \((25.025, 0)\) becomes less than 0.05.

A plot of the start time vs crack resistance can be seen in Figure 6.9. The crack resistance is defined to be two times the surface tension. It can be seen that there is good agreement between the theoretical start times and our simulations. We note that the use of the TNNMG solver in our opinion is the natural way to handle the constraints on \(\hat{\phi}\). It is beyond the scope of this thesis to compare the computations done with traditional methods of enforcing the irreversibility, however `PhaseField` provides the tools to carry these investigations out.

Remark 6.5.1. There are a number of other UFL packages for fracture simulations, see [1, 60, 2]. These are all dependent on the back end FEniCS.

6.6 Multi-phase curvature flow

We perform a number of separate test to ensure that the multi-phase models are working as required.
Figure 6.9: Starting time of crack propagation for different resistances.

Listing 42: Stepper code for fracture example.

```python
stepper = PhaseStepper(uflModel, preBulk = [0,1], solver="cg",
                      storage="istl", staticIt = 6)

solution = stepper.solution
stepper.indicator = 1-solution[0]
stepper.defaultRefine = [2e-1, 1e-1, 4, 11]
stepper.gridSetup(8,11, smoothingEpsilon = 6)

while stepper.time <= Fracture.endTime:
    stepper.nextTime()
    stepper.adapt()
```

Consistency with two phase simulations

We set the initial condition \( \dot{\varphi}_3 = 0 \) and again compute the shrinking circle from Section 6.1. We see that as expected Figure 6.10 is the same as Figure 6.2.
```python
solverParameters = {
    "tolerance": 1e-5,
    "verbose": True,
    "linear.tolerance": 1e-8,
    "linear.preconditioning.method": "ilu",
    "linear.preconditioning.iterations": 1,
    "linear.preconditioning.relaxation": 1.2,
}

stepper = PhaseStepper(phaseField, solverParameters)
maxLevel = 14
stepper.defaultRefine = [1.4, 1.2, 4, maxLevel]
stepper.gridSetup(12, maxLevel)

while stepper.time < Mcf3p.endTime:
    stepper.nextTime()
    stepper.adapt()
```

Listing 43: Stepper code for three-phase example.

**Equilibrium and triple junctions**

We next test an equilibrium configuration by setting the angle between corresponding phases as 120° on a circular grid of radius \( r \). For computational parameters we set \( \epsilon = 0.02, \Delta t = 10^{-2} \). The initial grid refinement is (12, 14). As in this case there is no curvature present and the angle is 120° the velocity should be zero on the interfaces according to Section 2.2.6 and Eq. (2.58). The results can be seen in Figure 6.11 where it is observed that the angle does not change when set in initial equilibrium.

We next run the same simulation but with an acute angle at the start and expect this to be unstable and the evolution to move to minimise both curvature and interface length. The expected result can be seen in Figure 6.12.

We next compute the solution of a specific configuration constructed in [44]. We choose the surface tensions so that from Eq. (2.61) we get \( \theta_2 = \theta_3, \theta_1 = \frac{\pi}{2} \). This is done by choosing \( \sigma_{12} = \sigma_{13} = 1, \sigma_{23} = \frac{1}{\sqrt{3}} \).

A grid of \([0, 0.2] \times [0, 0.2]\) is chosen with \( dt = 5 \times 10^{-5}, \epsilon = 3 \times 10^{-3} \). The initial configuration chosen can be seen in the first image in Figure 6.13. We use a direct solver by passing `solver="direct"` as a named argument to the `PhaseStepper` class. A maximum level of 13 adaptive refinements was used with a initial uniform
Figure 6.10: Consistency of multi-phase simulation with two phases, using a concave convex well, $\epsilon = 0.9, 0.8, 0.7, 0.6$.

Figure 6.11: Equilibrium shape of multi-phase simulation at $t = 0, 1$.

Figure 6.12: Unstable initial state for multi-phase simulation at $t = 0, 1$. 
refinement of 3. The solution can be seen in Figure 6.13. \( \Omega^1 \) is coloured blue, \( \Omega^2 \) pink and \( \Omega^3 \) in yellow. As calculated in [44] after the initial rapid change we expect a constantly transported profile to the right with a angle at the triple junction of 90°. When the triple junction hits the right edge we edge the remaining interfaces to shrink to the corners.
Chapter 7

Conclusion and Extensions

7.1 Extensions

As demonstrated, our software can solve a large class of interfacial problems. There are a number of possible extensions that we have not been able to cover. We describe some possible extensions and the steps necessary to implement them.

7.1.1 Linearisation for anisotropic energy

In Section 4.1.6 we gave an implementation of a natural time discretisation when the energy density is written in terms of an orientation angle $\theta$. In a general anisotropic energy written in terms of $\nu$, the fully implicit scheme is often numerically difficult to solve because of the non-linearities. In [48] the following time discretisation for the energy densities was suggested

$$\langle \nabla \gamma^2(\nabla \hat{\phi}_1^{k+1}), \hat{\zeta} \rangle \approx \langle \gamma^2(\hat{\phi}_1^k), \hat{\zeta} \rangle + \epsilon \lambda \langle \nabla (\hat{\phi}_1^{k+1} - \hat{\phi}_1^k), \nabla \hat{\zeta} \rangle,$$

with a suitable $\lambda > 0$. This could be implemented with a flag similarly to the $\text{thetaSemi}$ flag in Section 4.1.6.

7.1.2 Complete bindings for TNNMG solver

Python bindings were created for a TNNMG solver provided in the module DUNE-TNNMG. This binding was done directly to a function that takes as arguments the appropriate matrix and vectors to define the linear system. Within DUNE-TNNMG there are many additional options for the solving. It was outside the scope of this project to provide bindings for these. Providing additional control over these options from within PHASEFIELD would give the user more control over the solving
process. A mechanism is already in place for passing parameters to the solvers via the `solverParameters` dictionary that is passed into `Stepper`.

As we have only bound to the solver for a linear system `PhaseField` can only use quadratic wells which become linear in the phase field equation, when using the TNNMG solver. Improved bindings would also provide the functionality of using other wells and solve the phase field equation together with non-linear bulk equations.

### 7.1.3 N-phase energies

Currently `PhaseField` can only handle up to three phases. The extension to n-phases could be done along the lines of [21], where an n-phase energy is proposed and shown to Γ converges to the multi-phase surface energy. This would also require an extension to the mobility.

In the inner product Eq. (2.51), which is used in the definition of the $\mathcal{P}$ gradient flow, the mobility $\tau_G$ only depends on a scalar $\tau_0$. It is possible to extend this to a form where the mobility has different values depending on the $(n - 1)$ hypersurface. In the sharp setting the problem of well-posedness of multi-phase mean curvature flow with general mobilities and surface tensions is still an open problem. The convergence can still be shown at least formally see [21].

### 7.1.4 Time discretisation of $E$

The interpolation functions for $Q_{\hat{\phi}}$ and $F_{\hat{\phi}}$ can easily be modified as described in Section 3.4.5. For the transport term $E_{\hat{\phi}}$ only the default implementation is possible. This would be a simple extension.

### 7.1.5 Geometric constraints

In interface equations, often one can have additional geometric constraints such as conservation of mass. `PhaseField` can currently not handle these types of constraints, although we give some preliminary computations. The following model is taken from [20]. We state the required extension to our framework which would be necessary. The model is for vapour, solid, liquid, nano wire growth. There is a liquid phase $\Omega^1$, solid phase $\Omega^2$ and vapour phase $\Omega^3$. The configuration must satisfy the following constraints

---

130
• The liquid phase is conserved
\[ \frac{d}{dt} \int_{\Omega_1} d\mathcal{L}^3 = 0. \] (7.1)

• The velocity of the nanowire growth is proportional to the area of the solid-liquid interface, let \( c_s \in \mathbb{R} \) then
\[ \frac{d}{dt} \int_{\Omega_2} d\mathcal{L}^3 = \frac{d}{dt} \int_{\Omega_3} d\mathcal{L}^2 = c_s \int_{\Gamma_{12}} d\mathcal{H}^2. \] (7.2)

In addition to this the interfaces must evolve according to multi-phase curvature flow with forcing terms \( \mu_1, \mu_2, \mu_3 \). These forcing terms enforce the constraints in Eqs. (7.1) and (7.2).
\[
\begin{align*}
\tau_{12} \vec{V}_t \cdot \vec{v}_{12} &= -\gamma_{12} \kappa_{12} + \mu_1 - \mu_2 & \text{on } \Gamma_{12}, \\
\tau_{13} \vec{V}_t \cdot \vec{v}_{13} &= -\gamma_{13} \kappa_{13} + \mu_1 - \mu_3 & \text{on } \Gamma_{12}, \\
\tau_{23} \vec{V}_t \cdot \vec{v}_{23} &= -\gamma_{23} \kappa_{23} + \mu_2 - \mu_3 & \text{on } \Gamma_{12}.
\end{align*}
\]

Clearly this form is similar to a \( \mathcal{P} \) gradient flow, with \( \mu \) taking the place of the bulk terms \( \hat{a} \) must now depend on the state of the current configuration \( G \). We however have not found a generic way of encoding these constraints into the phase field equations and the following computations are done by way of an explicit computation depending on the phase field variables.

The preliminary computations are shown in Figure 7.1. Here a small ball of liquid (yellow) is rested on the solid (red) with the rest of the domain being vapour.

![Figure 7.1: Nanowire growth at times \( t = 0, 10, 20 \).](image)

### 7.1.6 Surface quantities

In the balance law defined by Eq. (2.24) we have only allowed contributions from the \( (n) \) hypersurfaces. In many problems the field \( \tilde{u} \) may also have contributions on the interface. This then involves approximating the lower dimensional measure \( \mathcal{H}^1 \)
in the balance law. The extension to the equations in the sharp setting is straightforward although one has to be careful to include appropriate extra terms in the multi-phase case. See [62] for an example of mean curvature flow coupled to lateral diffusion.

**Willmore and other flows**

The only geometric quantities that our interface equations have involved are the unit normal and mean curvature, it is possible to approximate higher order interface equations with phase field equations. These include Willmore flow and motion by surface diffusion. See [22] for an overview for current phase field approaches to approximate of Willmore flow.

**Analytical extensions to framework**

In addition to the extension for surface quantities as described above there are a number of important analytical extensions.

In [29] asymptotics were derived in the case where the crack was travelling in straight lines. Our framework would in theory allows for curved cracks, and asymptotics should be performed to recover this at the tip. Furthermore, crack propagation also extends to the setting when $n = 3$, our variation calculations do not cover this.

**7.2 Conclusion**

Our aim was to create a framework for the translation of a sharp interfacial problem into a phase field formulation. We believe we have succeeded in this regard. In Chapter 2 we defined what we called a $\mathcal{P}$ gradient flow which was an abstract formulation able to describe a large class of interfacial problems.

The formulation of this abstract framework was always done in such a way that the transformation to the phase field equations was straightforward; we described this transformation via a number of algorithms. This culminated in the specification of the UFL equations that the $\text{PhaseModel}$ class produced. Of particular interest was the link and generalisation of the framework that meant it was able to handle both regular and obstacle potentials in a unified manner.

In our opinion, one of the bigger shortcomings of the software is the need to split the energies into four distinct models. While Section 3.2.1 is specific to time discretisation and unavoidable, the splitting of the other models is necessary as to our knowledge there appears to be no tractable $n − phase$ anisotropic potentials.
In Chapter 6, we carried out a number of simulations, in order to demonstrate the validity of the software as well as its flexibility. The fracture simulations in Section 6.5 were novel in the sense that they properly enforced the irreversibility of the crack as opposed to ad-hoc methods. We believe the solver and setting presented are the natural ones to enforce these conditions, while the other simulations were shown in order to prove the software works as desired.

In conclusion, it is our hope that this thesis and accompanying software not only provides practitioners for the means to easily compute solutions to interfacial problems, but also gives a unified framework from within to see phase field and interfacial problems.
Appendix A

Installation

PhaseField comes as a Python module and can therefore be pip installed as usual. The package provided the functionality for constructing the UFL form from a sharp interface description class. There is also a time evolution class included which works with either a DUNE or a FEniCS backend. Using the DUNE backend, the available functionality will depend on the DUNE modules available in the system. A requirement is DUNE-FEM and for more advanced solvers PETSc and SuiteSparse can be used if available. For the constraint solvers the DUNE-TNNMG module and its dependencies are also required. Here we talk through the steps if one would like to carry out computations using DUNE. The easiest way to get up and running is by using the provided docker container.

1. Clone git repository
   https://gitlab.dune-project.org/dune-fem/phasefield.git

2. Run the setupdune.sh script located in the repository, this will start up the docker container. When executed the first time the docker image needs to be downloaded which can take some time.

3. The first time the container is used, the setupPhase.sh script needs to be run from within the docker container. This sets up and installs all the required additional modules in the container (especially the modules required for the TNNMG solver).

Many of the examples from within this thesis can be found in the tutorials folder.
Appendix B

Calibration calculations

B.1 Smooth potential

Garcke et al [45] derive the following formula for anisotropic surface tensions and mobility where \( \hat{q} \) is the fundamental two-phase profile.

\[
\gamma_{ij} = 2 \int_{-\infty}^{\infty} \sqrt{W(\hat{q})\Lambda(\hat{q}')(\vec{z})} d\vec{z},
\]

\[\tag{B.1}\]

\[
\tau_0 = \int_{-\infty}^{\infty} |\hat{q}'|^2 d\vec{z}.
\]

\[\tag{B.2}\]

Where the energy is

\[
\mathcal{F}(\varphi) = \int_{\Omega} \epsilon \Lambda(\nabla \varphi) + \frac{1}{\epsilon} W(\varphi).
\]

\[\tag{B.3}\]

We show the calibration for a two phase system so

\[
\Lambda(\nabla \varphi) = \frac{\beta}{4} |\nabla \varphi|^2,
\]

\[\tag{B.4}\]

\[
W(\varphi) = \alpha \varphi_1^2 \varphi_2^2.
\]

\[\tag{B.5}\]

Taking the \( L^2 \) gradient flow with respect to \( c_1 \) and \( c_2 \)

\[
\partial \varphi_1 = \frac{\beta}{2} \Delta \varphi_1 - \alpha \varphi_1 (1 - \varphi_1)(1 - 2\varphi_1).
\]

\[\tag{B.6}\]
The standing wave solution for this system is

\[
\hat{q}(\vec{z}) = \frac{1}{2} \left[ 1 + \tanh \left( \frac{\alpha}{2\beta} \vec{z} \right) \right] \left[ 1 - \tanh \left( \frac{\alpha}{2\beta} \vec{z} \right) \right].
\]  

(B.7)

which solve

\[
\beta \hat{q}_1'' - 2\alpha \hat{q}_1 (1 - \hat{q}_1)(1 - 2\hat{q}_1) = 0,
\]

(B.8)
a short calculation shows

\[
W(\hat{q}) = \frac{\alpha}{16} \left( 1 + \tanh \left( \frac{\alpha}{2\beta} \vec{z} \right) \right)^2 \left( 1 - \tanh \left( \frac{\alpha}{2\beta} \vec{z} \right) \right)^2.
\]  

(B.9)

Calculating the gradient term

\[
\Lambda(\hat{q}') = \frac{\beta}{4} (|\hat{q}_1'|^2 + |\hat{q}_2'|^2) = \frac{\beta}{2} |\hat{q}_1'|^2 |\vec{v}| = \frac{\alpha}{16} \text{sech}^4 \left( \frac{\alpha}{2\beta} \vec{z} \right).
\]  

(B.10)

The definition of the surface tension gives

\[
\gamma_{12} = 2 \int_{-\infty}^{\infty} \frac{\alpha}{16} \text{sech}^4 \left( \frac{\alpha}{2\beta} \vec{z} \right) = \frac{\sqrt{\alpha \beta}}{6} \sqrt{2}.
\]  

(B.11)

The Boyer-Minjaud potential is equivalent to setting \(\alpha = 12\) and \(\beta = \frac{3}{2}\) which we see gives the correct surface tension of 1. Now for the mobility calculation, using the definition of the standing wave \(\hat{q}\)

\[
|\hat{q}'|^2 = \frac{\alpha}{4\beta} \text{sech}^4 \left( \frac{\alpha}{2\beta} \vec{z} \right),
\]  

(B.12)

\[
\tau_0 = \int_{-\infty}^{\infty} \frac{\alpha}{4\beta} \text{sech}^4 \left( \frac{\alpha}{2\beta} \vec{z} \right) = \frac{2}{3\sqrt{2}} \sqrt{\frac{\alpha}{\beta}}.
\]  

(B.13)

Equating both the mobility and surface tension equal to 1 gives \(\beta = 2\) and \(\alpha = 9\).
B.2 Obstacle potential

We assume that $N$ phases are present and have a multiphase order parameter $\vec{\phi}$ which lies in $\mathbb{R}^N$. The energy for the system is given by

$$F(\vec{\phi}) = \int_{\Omega} \epsilon \Lambda(\nabla \vec{\phi}) + \frac{1}{\epsilon} W(\vec{\phi}), \quad (B.14)$$

Specialising to the 2 phase case so we will assume $N = 2$

$$\Lambda(\nabla \vec{\phi}) = \frac{\beta}{4} |\nabla \vec{\phi}|^2, \quad (B.15)$$

$$W(\vec{\phi}) = \alpha \phi_1 \phi_2 = \alpha \phi_1 (1 - \phi_1) + 1_{G^c}(\vec{\phi}). \quad (B.16)$$

Where

$$1_{G^c}(\vec{\phi}) = \begin{cases} 0 & \text{if } \vec{\phi} \in G^c \\ \infty & \text{otherwise} \end{cases}$$

Taking the $L^2$ gradient flow of the free energy $\xi$ a solution to the multi phase obstacle problem must satisfy

$$(\epsilon \vec{\phi}_t + \frac{1}{\epsilon} \mathbb{P}_{T\Sigma}[W'(\vec{\phi})], \vec{v} - \vec{\phi})_{L^2} + (\epsilon \mathbb{P}_{T\Sigma}[\Lambda'(\nabla \vec{\phi})], \nabla (\vec{v} - \vec{\phi}))_{L^2} \geq 0 \quad (B.17)$$

for all $\vec{v} \in G^c$. The quantities we would like to calibrate are the surface energy and mobility defined as

$$\gamma_{12} = 2 \int_{-\infty}^{\infty} \sqrt{W(q)\Lambda'(q')}(z) \, dz, \quad (B.18)$$

$$\tau_0 = \int_{-\infty}^{\infty} |q'|^2 \, dz. \quad (B.19)$$

Substituting (B.15) and (B.15) into (B.17) we recover

$$\partial_t \phi_1 = \mathbb{P}_{T\Sigma}[\frac{\beta}{2} \epsilon \Delta \phi_1 - \frac{1}{\epsilon} \alpha \phi_2], \quad (B.20)$$

$$\partial_t \phi_1 = \mathbb{P}_{T\Sigma}[\frac{\beta}{2} \epsilon \Delta \phi_2 - \frac{1}{\epsilon} \alpha \phi_1]. \quad (B.21)$$

Expanding and simplifying the (B.20)

$$\partial_t \phi_1 = \frac{\beta}{2} \epsilon \Delta \phi_1 - \frac{\alpha}{2\epsilon} (1 - 2\phi_1). \quad (B.22)$$
To first order inside an appropriate region we must have

$$\beta \dot{q}_1'' - \alpha (1 - 2\dot{q}_1) = 0. \quad (B.23)$$

$q_1$ is the first component of the fundamental transition profile $\dot{q}$. Taking into consideration the appropriate boundary conditions after some elementary calculations we recover

$$\dot{q}_1 = \begin{cases} 
0 & \text{if } \bar{z} \leq -\frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} \\
\frac{1}{2} \left[ 1 + \sin \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right) \right] & \text{if } |\bar{z}| < \frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} \\
1 & \text{if } \bar{z} \geq \frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} 
\end{cases} \quad (B.24)$$

$$\dot{q}_2 = \begin{cases} 
1 & \text{if } \bar{z} \leq -\frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} \\
\frac{1}{2} \left[ 1 - \sin \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right) \right] & \text{if } |\bar{z}| < \frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} \\
0 & \text{if } \bar{z} \geq \frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}} 
\end{cases} \quad (B.25)$$

$$W(\dot{q}) = \frac{\alpha}{4} \left( 1 - \sin^2 \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right) \right) = \frac{\alpha}{4} \cos^2 \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right), \quad (B.26)$$

$$\dot{q}' = \sqrt{\frac{\alpha}{2\beta}} \cos \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right) \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad (B.27)$$

$$\Lambda(\dot{q}') = \frac{\beta}{4} |\dot{q}'|^2 = \frac{\alpha}{4} \cos^2 \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right). \quad (B.28)$$

Equation (B.26) and (B.28) are the same so we have the equi-partition of energy property. Substituting these into expression (B.18) the surface energy is

$$\gamma_{12} = \frac{\alpha}{2} \int_{-\frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}}}^{\frac{\pi}{2\sqrt{2} \sqrt{\beta \alpha}}} \cos^2 \left( \sqrt{\frac{2\alpha}{\beta}} \bar{z} \right) = \frac{\sqrt{\alpha \beta}}{2 \sqrt{2}} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2(u) = \frac{\pi \sqrt{\alpha \beta}}{4 \sqrt{2}} \quad (B.29)$$

likewise for the mobility

$$\tau_0 = \int_{-\infty}^{\infty} |\dot{q}'|^2 = \frac{\pi}{2\sqrt{2}} \sqrt{\frac{\alpha}{\beta}}. \quad (B.30)$$
Equating expressions (B.29) and (B.30) equal to one we have that $\beta = 2$ and $\alpha = \frac{16}{\pi^2}$. 
Appendix C

Software documentation

__init__(self, sharpCls, epsilon=None, dt=None, well=<function Default at 0x7f66a5adaa60>, constrained=False, thetaSemi=False)

Parameters

----------

sharpCls : object specifying a P gradient flow. Contains the required methods
and parameters in order to define P gradient flow.

epsilon : float
Small parameter proportional to the width of the interface. Default is
(None)

dt : float
Time step (default is None)

well : function, optional function that has one argument a PhaseModel class
Potential well W that also specifies the time
discretisation (default is Implicit).

thetaSemi : bool, optional
Flag used to specify whether a specific explicit implicit time
discretisation is used of the surface energy if it is written in terms of
an orientation angle theta (default is False).

Listing 44: Initialiser in PhaseModel.
setupPhase(self, phi, phiN, u, uN, vPhi, vU, x=None, constant=None)

UFL form for balance + phase field.

Parameters
----------
phi : Coefficient
    Implicit phase field function.

phiN : Coefficient
    Explicit phase field function.

u : Coefficient
    Implicit bulk function.

uN : Coefficient
    Explicit bulk function.

vPhi : Coefficient
    Testfunction for phase field equations.

vU : Coefficient
    Testfunction for bulk equations.

x : SpatialCoordinate
    Point of the domain used for Dirichlet boundary conditions.

constant : External.constant class
    Needed for storage of epsilon, dt and t so are allowed to change from
    within time loop. Defaults to 'None' leading to the provided
    values being used directly within the ufl forms.

Returns
-------
UflForm object :
    Form used for computation, in format
    [[lhsBalance, RhsBalance],[lhsPf,rhsPf]]

Listing 45: **setupPhase** method in the **PhaseModel** class.
Listing 46: Initialiser in Stepper.
gridSetup(self, numGlobalRefine, numLocalRefine, smoothingEpsilon=None)

Performs the initial refinement of the grid.

Parameters
----------
numGlobalRefine : int
    Number of global refinements to do.

numLocalRefine : int
    Number of adaptive refinements, depends on member variables indicator
    and defaultRefine.

Bibliography


