UPPSALA
UNIVERSITET

IT 11 073
Examensarbete 30 hp
November 2011

Preconditioners for the discrete Cahn-Hilliard equation in three space dimensions

Xunxun Wu

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Preconditioners for the discrete Cahn-Hilliard equation in three space dimensions

Xunxun Wu

Multiphase flow problems are often modeled by the diffuse-interface phase-field model, which is based on the so called total free energy of the physical system. The major mathematical tool in the phase-field model is the Cahn-Hilliard equation. Due to finite element discretization in space and the theta-method in time, the arising systems are, in general, of huge size, especially in 3D. In this work, in order to solve large scale nonlinear systems arising from 3D problems, we propose two different inexact Newton methods. The first one is to solve the systems with Jacobian matrix by a preconditioned iterative method. The second one is to replace Jacobian matrix by a high quality approximation. Both inexact Newton methods result in robustness and efficient solution algorithms. The numerical experiments show agreement with the theoretical analysis and the remarkable lower computational cost of the second method.

Handledare: Maya Neytcheva
Ämnesgranskare: Gunilla Kreiss
Examinator: Anders Jansson
IT 11 073
Tryckt av: Reprocentralen ITC

Contents

1 Introduction **1**

2 The mathematical model **5**
2.1 The phase-field model 5
2.1.1 The coupled Cahn-Hilliard-Navier-Stokes system 7
2.2 Boundary conditions 9
2.3 Properties of the interface 9
2.4 Nondimensionalising the PDE model 10
2.5 Governing equations 11

3 Finite element discretizations and the θ-method **13**
3.1 Discretization in space 13
3.2 Discretization in time 15

4 Solution of the nonlinear algebraic system of equations **17**
4.1 Solution of nonlinearity 17
4.2 Inexact Newton methods 19

5 Preconditioned iterative solution method for the linearized Cahn-Hilliard equation **23**
5.1 The idea of preconditioning 23
5.2 Overview of the preconditioned GMRES method 24
5.3 The preconditioning strategy used in this project 25

6 Computer implementation **35**
6.1 The algorithm for solving the C-H equation 35
6.2 The Matlab implementation 36
6.3 The C++ implementation using the library deal.II 41

7 Numerical experiments **43**

8 Conclusions **51**

Bibliography **53**

Acknowledgements

First, I would like to express my heartfelt gratitude towards my supervisor, Dr. Maya Neytcheva at Uppsala University for all the help and support. I also want to thank Petia Boyanova and Martin Kronbichler, who both are Phd students at Uppsala University for their useful advices and encouragements. I wish to thank Prof. Michael Thuné, Stefan Pålsson, Olle Eriksson and Roland Bol for being nice teachers at Uppsala University. Acknowledgement also goes to all my classmates for helping me in various ways and for making the time spent in Sweden a life time experience. And last but certainly not least, I would like to express my deepest love and gratitude to my family, especially my mother. Thank you for all your love, support and encouragements.

Chapter 1

Introduction

In the context of fluid mechanics, mutiphase flows can be viewed as fluid flow system consisting of two or more distinct phases in motion relative to each other in mixture, having some level of phase separation with different chemical compositions and physical properties. Depending on the combinations of phases, a description of the main flow regime in multiphase flows can be characterized according to the state of the different phases: gas-liquid, liquid-liquid, gas-liquid-liquid and solid-liquid-liquid-gas.

Tracking back the history, the existence of multiphase problems has been known to human for thousands of years. Many multiphase phenomena can be often observed in nature, such as gas bubbles in oil, ice melting, wet steam, crystal growth, spinodal decomposition and so on. It is perhaps not unexpected that many practical problems involving mutiphase flows have a broad effect in a range of modern technological industries, as well as within our body system and in the environment we live in. A large number of applications occur in aerospace, atmospheric, biological, chemical, civil, mechanical, and nuclear systems to name a few. However, since the nature of multiphase flows is highly complex, the fundamental understanding of the integrated fluid physics for these phenomena is still not as well developed as those for single-phase flows. Therefore, numerical simulations play a critical role in this area.

Modeling mutiphase flows, in general, encompasses a wide range of engineering disciplines and a multitude of different computational approaches. Over the years, based on the different descriptions (models) used in those engineering disciplines, a wide variety of mathematical theories and computational technologies have been developed for the accurate and efficient numerical simulation of mutiphase flows. These methods can be mainly divided into two large classes, depending on what type of mesh is used in the bulk of the phases, fixed or moving. In moving-mesh methods, the interface between the two fluids is a boundary between two sub-domains of the mesh. The mesh moves with the interface as the system evolves. This simplifies the analysis near the interface. But when there is a large deformation or the topology changes, the domain has to be remeshed or mesh points have to be added and/or removed, which makes the computational procedure

rather complicated. In fixed-mesh methods, the mesh is predefined and does not move with the interface [30]. Due to the simple description and easier extension to 3D, the fixed-mesh methods are more widely used. Some popular fixed-mesh methods are the level set method (see, e.g., [28]) and the volume of fluid method (see, e.g.,[23]), both belonging to the Eulerian specification of the flow field, which depicts fluid motion as a function of fixed position $\mathbf{x}$ and time t. The basic idea of the level set method is to have the zero level set of a signed distance function (level set function) act as a marker that determines the position of the interface. The main drawback of the standard level set method is that mass is not conserved, and significant mass loss may occur. Therefore, in order to improve mass conservation and accuracy, when the interface evolves, some re-initialization schemes of the level-set function are needed [27]. The volume of fluid method is based on the so called fraction function, which is defined as the integral of the fluid's characteristic function in a control volume. The advantage of the volume of fluid method is known to be its ability to conserve the mass of the traced fluid. Even when the fluid interface changes its topology, this change is easily traced, so the interfaces can for example join, or break apart [23]. The volume of fluid method must reconstruct the interface from the discontinuous fraction function at each time step. It turns out that the mean curvature can make it difficult to accurately account for surface tension.

In contrast to the above mentioned lattice-based approaches, the so-called phase-field method can be an attractive alternative for solving multiphase flow problems. The phase-field method offers a systematic physical approach by describing the interface in a physical rather than in a numerical sense. The phase-field method is also known as the diffuse interface model. It replaces sharp interfaces by thin but nonzero thickness regions, where the density, viscosity and other physical quantities undergo rapid but smooth transitions. In order to distinguish one phase from the other, it is necessary to introduce a special parameter, C, called the order parameter, or the phase field, which assumes distinct values in the bulk phases (for two-phase flow, $C = 1$ in one bulk phase and $C = -1$ in the other), and varies continuously over thin interfacial region ($-1 < C < 1$). The relaxation of the order parameter is driven by local minimization of the free energy subject to the phase field conservation and as a result, the interface layers do not deteriorate dynamically. The location of the interface can be defined as the collection of all points where the phase field takes a certain value ($|C| \leq \varepsilon$, $\varepsilon \ll 1$). The most appealing feature of the phase-field method is that, since the solution of all governing equations does not require any priori location of the interfaces in the entire computational domain, interfaces coalescence and separation can be captured naturally without any special procedures [31, 13]. Another feature of the phase field description is that the order parameter has a physical meaning and due to the so called total free energy of the physical system, different phenomena can be easily adapted to this model by a suitable modification of the free energy [13]. However, this feature of the phase field description also causes the main drawback of phase-field method, namely, the interface layer is required to be extremely thin in order to properly model relevant physical phenomena. This means that the conventional numerical methods (i.e.FEM, FVM and others), which are used to discretize and approximate

the solution of the problem, should sustain a high resolution and accuracy, especially in 3D.

The major mathematical tool in the phase-field model is the Cahn-Hilliard equation, which in its original form, is a fourth order parabolic equation. In this project it is used in an equivalent formulation, in the terms of a coupled system of two second order partial differential equations, one of which is time-dependent.

When solving PDE numerically, the most common discretization methods are the finite differences and the finite element methods. In this work, we discretize the problem in space using the standard conforming finite element method. Then we fully discretize the problem in time using the so-called θ-method, which permits to test various time integration schemes of first and second order accuracy. The result is a nonlinear algebraic system to be solved at each time step. The solution of the nonlinear system itself is done by Newton iteration and during each nonlinear iteration, a linear system with the corresponding Jacobian has to be solved. The arising system matrices are, in general, of huge dimension, particularly in 3D, since their size is directly proportional to the number of nodes in the finite element mesh, which is normally up to millions in order to meet the high resolution requirement for the phase-filed method. Direct methods are often used and preferred due to their robustness and the existing highly optimized software packages, providing efficient solvers for sparse matrices. However, especially for large scale linear systems arising from 3D problems, direct solution methods are not a feasible alternative anymore due to their very large demands for computer resources, in particular, computer memory, but also execution time. Therefore, regarding the computational complexity (i.e. number of floating point operations needed), memory requirements and time consumption, iterative methods become nearly mandatory to be used when solving the arising linear systems. However, as is well known, the traditional iterative methods (i.e.CG, GMRES) are all likely to suffer from slow convergence and lack of robustness for problems arising in simulation of fluid dynamics processes. The possibility to overcome this drawback is to use preconditioning, which improves the conditioning of the original linear system so that the preconditioned linear system can converge at a faster rate. In practice, the choice of an efficient preconditioner often depends highly on the properties of the coefficient matrix in the system. For multiphase flow problems, considered here, the system matrix is of 2×2 block form and this is utilized when constructing a preconditioner.

Aim of the thesis

The major goal of this project is to study the suitability of a certain block preconditioner for the linearized discrete Cahn-Hilliard equation in three space dimensions in terms of method parameters, construction, numerical efficiency, robustness with respect to problem parameters, ease of implementation, and development and testing. The preconditioner fully utilizes the two-by-two block structure of the matrix in the arising linear systems. Further, it is subjected to simplifications of the blocks, which substantially facilitate the solution procedure.

The preconditioner is introduced for two dimensional problem in [11] and [1], and is further analysed here in 3D, as well as implemented in C++ using finite element library deal.II in order to test both its numerical and computational efficiency.

Layout of the thesis

The thesis is organized as follows:

- In Chapter 2 we briefly describe the mathematical model used to simulate the physical processes, related to phase separation and interface tracking, as well as some particular formulations and properties.
- The space and time discretization schemes are outlined in Chapter 3.
- In Chapter 4 we introduce the treatment of the nonlinearity using Newton's method and derive the linear system with matrix A, and we also analyze the inexact Newton method.
- Chapter 5 contains a brief overview of preconditioning and a discussion on suitable preconditioners for solving the discrete Cahn-Hilliard equation. We also analyze the convergence properties of the preconditioner of choice.
- In Chapter 6 we describe in detail the computer implementation both in Matlab and deal.II
- The numerical experiments are presented in Chapter 7.
- In Chapter 8 some conclusions are drawn.

Chapter 2

Mathematical model

Due to the complexity of the nature of mutiphase flows, their modeling and simulation has been challenging both mathematically and technically for many years. Besides, the complexities increase with the number of phases encountered in the mutiphase problem. Fortunately, many important applications happen to include only two phases, or can be considered as pseudo-two-phase flows. Hence, this project focuses on preconditioning techniques for a two-phase setting.
In the context of computational fluid dynamics, the equations express the physics. In a fixed and closed domain, each phase of mutiphase flow can be considered to be fractionated by finite units which are defined as volume fractions with a velocity field. These volume fractions always obey three fundamental physical principles both in space and in time:

1 Conservation of mass

2 Newton's second law of motion $F = ma$

3 Conservation of energy

We can capture the evolution of each phase by applying the fundamental physical principles on the volume fractions in integral form. Then it turns out that a partial differential equation has to be solved.

2.1 The phase-field model

Below we briefly describe the derivation of the phase-field model, following [8, 13, 18, 11, 12].
We consider a binary fluid of two immiscible viscous fluids under constant temperature. The mixture of the two incompressible fluids 1 and 2 completely fills a bounded domain $\Omega \subset \mathbb{R}^3$. And let $\partial\Omega$ be its boundary which is assumed to be sufficiently smooth. Let $\mathbf{n}$ denote the unit outward normal vector to $\partial\Omega$.
We consider the total density of the mixture $\rho_m = m/V = constant$, where m is the total mass of the mixture in the representative material volume V (i.e. $m = \int_\Omega \rho_m d\mathbf{x}$). We

also suppose that the two fluids have different apparent densities $\tilde{\rho}_i = m_i/V = constant$, $i = 1, 2$. The apparent densities $\tilde{\rho}_i$ are related to the actual densities $\rho_i = m_i/V_i$ through $\tilde{\rho}_i = k_i\rho_i$. Here, $k_i = V_i/V$ are the volume fractions, and m_i are the masses of the components in Ω, namely

$$m_1 = \int_\Omega \tilde{\rho}_1 d\mathbf{x}, \quad m_2 = \int_\Omega \tilde{\rho}_2 d\mathbf{x},$$

or equivalently

$$dm_1 = \tilde{\rho}_1 dV, \quad dm_2 = \tilde{\rho}_2 dV,$$

where d denotes the differential sign.
Since $m = m_1 + m_2$, we get

$$\rho_m = \tilde{\rho}_1 + \tilde{\rho}_2 \tag{2.1}$$

Following the phase field approach, the model is built on the notion that the interface between the phases is not sharp, but has a thin nonzero thickness and is characterized by a rapid but smooth transition. Then we use a scalar function C, called 'order parameter', to label each phase, for example, for two phases,

$$C = \begin{cases} 1, & \text{in fluid 1,} \\ -1, & \text{in fluid 2,} \\ (-1, 1), & \text{in the interfacial region} \end{cases}$$

Since the fluids 1 and 2 are assumed to be isothermal, incompressible and viscous, they do not undergo phase change. However, due to surface tension, each material can only migrate from one space position to another, close to it. As a consequence, the total amount of each fluid in Ω must be always equal to the given original amount. Hence, this model obeys the laws of conservation of mass and linear momentum of continuum mechanics. The function C can be considered to represent the difference of local mass fraction (concentration differential) of the two fluids, namely

$$C = \frac{dm_1}{dm} - \frac{dm_2}{dm},$$

or equivalently

$$Cdm = dm_1 - dm_2.$$

By dividing by dV on both sides, we get

$$C\rho_m = \tilde{\rho}_1 - \tilde{\rho}_2 \tag{2.2}$$

In this way, according to (2.1) and (2.2), we see that $C \in [-1, 1]$, and when $C = 1$ or $C = -1$ only the fluid 1 or 2 appears. Furthermore, recalling the definition of $\tilde{\rho}_1$ and $\tilde{\rho}_2$, we have

$$\int_\Omega \rho_m C d\mathbf{x} = \int_\Omega (\tilde{\rho}_1 - \tilde{\rho}_2) d\mathbf{x} = m_1 - m_2 = constant.$$

This means that the definition of C ensures the conservation of the concentration difference of the two fluids in Ω. In other words, to make the mass of each component conserved over the whole domain, the evolution of C should be subject to the following constraint

$$\int_{\Omega} \rho_m C(\mathbf{x}, t) d\mathbf{x} = constant. \tag{2.3}$$

In order not to let the interface layers deteriorate dynamically, the relaxation of the function C is driven by local minimization of the free energy subject to the above conservation constraint. The free energy can be defined as a functional of C

$$E(C) = \int_{\Omega} \{\frac{1}{2}\alpha|\nabla C|^2 + \beta\Psi(C)\} d\mathbf{x}, \tag{2.4}$$

The first term $\frac{1}{2}\alpha|\nabla C|^2$ is referred as the gradient energy, or surface energy, with a positive constant α. The second term $\beta\Psi(C)$ is referred to as the 'bulk energy'. The function $\Psi(C)$ is a double-well potential with two minima corresponding to the two stable phases of the fluid. This bulk energy tends to reduce the interfacial region thickness whereas the gradient energy $|\nabla C|^2$ tends to increase it. Since the system evolution is driven by the minimization of the free energy functional E with respect to C, the interface profile C can be found by solving the equation $E' = 0$, thus,

$$\frac{dE}{dC} = \int_{\Omega} \{-\alpha\Delta C + \beta\Psi'(C)\} = 0, \tag{2.5}$$

or equivalently

$$\eta \equiv \frac{\delta E}{\delta C} = -\alpha\Delta C + \beta\Psi'(C) = 0, \tag{2.6}$$

where η is the so called chemical potential and δ denotes the functional derivative $\delta/\delta C = \partial/\partial C - \nabla[\partial/\partial(\nabla C)]$.

2.1.1 The coupled Cahn-Hilliard-Navier-Stokes system

Cahn and Hilliard [15, 16] generalized the problem to a mass diffusion equation, valid in the entire two phase system. The equation was initially developed to describe spinodal decomposition of a binary alloy under isothermal and isochoric conditions. In particular, they assume the equation of conservation of mass to be

$$\frac{\partial C}{\partial t} = \nabla \cdot \mathbf{J}, \tag{2.7}$$

where $\mathbf{J}$ is the local diffusion mass flux $\mathbf{J} = \kappa(C)\nabla\eta$ and subjects to the boundary condition

$$\mathbf{J} \cdot \mathbf{n}|_{\partial\Omega} = 0. \tag{2.8}$$

Here, $\kappa(C) > 0$ is the mobility or so called Onsager coefficient, depending on the concentration.

Later, this model has appeared in many other contexts ranging from micro to macro scales. It serves as a good model for many systems concerning two phase systems constituted by different substances. Uniqueness and correctness of long-time behavior of the solutions have been proved in practice (see e.g. [9, 11, 12, 13, 18, 26]).
To account for fluid motion, a coupled Cahn-Hilliard-Navier-Stokes system is proposed and analyzed in [18, 22, 21]. In this system, the total rate of change of function C with respect to t refers to the material derivative $\frac{DC}{Dt}$, namely,

$$\frac{DC}{Dt} = \frac{\partial C}{\partial t} + \mathbf{u} \cdot \nabla C, \tag{2.9}$$

where $\mathbf{u}$ denotes the velocity vector. The kinetic equation in (2.7) is substituted by the so-called convective Cahn-Hilliard equation, which takes the form

$$\frac{\partial C}{\partial t} + (\mathbf{u} \cdot \nabla)C = \nabla \cdot [\kappa(C)\nabla\eta], \tag{2.10}$$

where $\eta = -\alpha\Delta C + \beta\Psi'(C)$. The fluid dynamic is described by the coupled time-dependent Navier-Stokes equations

$$\begin{aligned} \rho\tfrac{\partial \mathbf{u}}{\partial t} + (\rho\mathbf{u} \cdot \nabla)\mathbf{u} = &\quad -\nabla p + \nabla \cdot [\mu(\nabla)\mathbf{u} + \nabla\mathbf{u}^T] + \eta\nabla C, \\ \nabla \cdot \mathbf{u} = &\quad 0, \end{aligned} \tag{2.11}$$

where ρ is the density, p is the pressure that enforces incompressibility of flows, and μ is the viscosity. The term $\eta\nabla C$ represents the surface tension force, written in its potential form. At the walls the no-slip boundary condition, $\nabla \cdot \mathbf{u} = 0$, is imposed at the fixed domain boundary.
In this coupled Cahn-Hilliard-Navier-Stokes system, the processes are assumed to be isothermic, thus thermal effects are not included. Hence, the temperature does not appear explicitly in the equation (2.10). Further, elastic, viscoelastic effects, as well as anisotropies are also not explicitly considered. In reality, such an assumption requires very careful condition control and is rarely strictly encountered. However, the equation (2.10) turns out to be the limit and simplified case for other more complex physics phenomena, it plays an important role in understanding the evolution of phase separation. The study of efficient numerical simulations based on this model, can lead to build a structure for solving other more complicated and coupled models.
Due to the high adaptability of this model, if one need to take other physical processes into account, adjustments can be easily done. For example, when consider temperature effect, the equilibrium equation can be coupled with a kinetic equation for the absolute temperature, with the assumption that the temperature of the system rapidly decreases and the system instantaneously equilibrates to that (see e.g. [9, 8]).

2.2 Boundary conditions

Following the phase-field theory, we adopt the following homogeneous Neumann boundary conditions both for the difference concentration C and the chemical potential η, namely,

$$\nabla C \cdot \mathbf{n}|_{\partial\Omega} = 0 \tag{2.12}$$

$$\nabla \eta \cdot \mathbf{n}|_{\partial\Omega} = 0 \tag{2.13}$$

The more general form of the first condition (2.12) is

$$\mathbf{n} \cdot \nabla C = -\gamma \cos\vartheta g'(C),$$

referred to as the wetting boundary condition for the interface, see [18]. Above, γ is a positive constant and $g(C)$ is a local surface energy (which is set to $0.5 + 0.75C - 0.25C^3$). Here, we only consider the case of a contact angle $\vartheta = \pi/2$ (i.e. $\mathbf{n} \cdot \nabla C = 0$), which implies that the interfaces are supposed to be orthogonal to the boundary of the computational domain.
The second condition (2.13) is used to enforce the mass conservation constraint. It ensures that there is no mass flux through the boundary. Applying the material derivative to the mass conservation constraint (2.3), we get the following equalities

$$\int_\Omega \rho_m \dot{C} d\mathbf{x} = \int_\Omega \rho_m \nabla \cdot [\kappa(C)\nabla\eta] d\mathbf{x} = \int_{\partial\Omega} \rho_m \kappa(C) \nabla\eta \cdot \mathbf{n} da = 0.$$

2.3 Properties of the interface

In the current study, we consider the following double-well potential for a binary fluid,

$$\Psi(C) = \frac{1}{4}(C-1)^2(C+1)^2.$$

Recalling equation (2.6), the equilibrium profile is the solution of the equation

$$\eta(C) = -\alpha\Delta C + \beta\Psi'(C) = -\alpha\Delta C + \beta C(C^2-1) = 0 \tag{2.14}$$

This leads to two stable minima at $C = \pm 1$ representing the coexisting bulk phases, and a one-dimensional nonuniform solution (first found by van der Waals) in the form

$$\widehat{C}(\mathbf{x}) = tanh\left(\frac{x}{\sqrt{2}\xi}\right), \tag{2.15}$$

where $\xi = \sqrt{\frac{\alpha}{\beta}}$ is the mean-field thickness. The mixed phase is unstable and phase separation occurs (see Fig. 2.1) As it was stated earlier, the first term in the free energy $\alpha|\nabla C|^2$ is referred to be surface tension. From equation (2.15), the surface tension can be expressed as

$$\sigma = \alpha \int_{-\infty}^{\infty} \left(\frac{d\widehat{C}}{dx}\right)^2 dx = \frac{2\sqrt{2}}{3}\sqrt{\alpha\beta}.$$

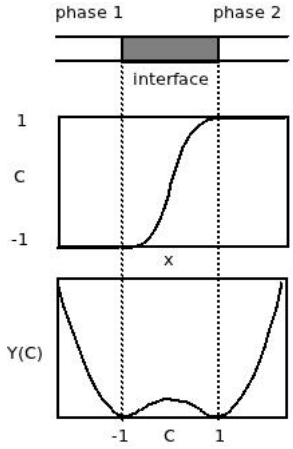


Figure 2.1: Plot of the chemical potential $\Psi = \frac{1}{4}(C^2-1)^2$

In the model, we define the interface thickness as the distance between $\mathbf{x}_1$ and $\mathbf{x}_2$, $C(\mathbf{x}_1) = -0.9$ and $C(\mathbf{x}_2) = 0.9$. With the help of (2.15), the equilibrium interface thickness ϵ is equal to $2\sqrt{2}\xi tanh^{-1}(0.9) = 4.164\xi$.
From $\xi = \sqrt{\frac{\alpha}{\beta}}$ and $\sigma = \frac{2\sqrt{2}}{3}\sqrt{\alpha\beta}$, we obtain an expression for the parameters α and β

$$\alpha = \frac{3}{2\sqrt{2}}\sigma\xi, \quad \beta = \frac{3}{2\sqrt{2}}\frac{\sigma}{\xi}.$$

Thus, via the choice of the parameters α and β, one can control the surface tension σ and equilibrium interface thickness ϵ. On the other hand, the chosen σ and ϵ can bound the value of the parameters α and β.

2.4 Nondimensionalising the PDE model

Following Refs. [13] and [12], we nondimensionalize the governing equations with the following variables

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{L},\ \hat{\mathbf{u}} = \frac{\mathbf{u}}{U},\ \hat{t} = \frac{t}{T} = \frac{U}{L}t,\ \hat{p} = \frac{L}{\mu U}p,$$

where L is the characteristic length, U is the characteristic velocity depending on the problem and T is the characteristic time, $T = L/U$. According to the definition of the rescaled variables and the chain rule, we can easily find that the following equalities hold true

$$\frac{\partial C}{\partial t} = \frac{U}{L}\frac{\partial C}{\partial \hat{t}},\ (\mathbf{u}\cdot\nabla)C = \frac{U}{L}(\hat{\mathbf{u}}\cdot\nabla)C,\ \Delta C(\mathbf{x}) = \frac{1}{L^2}\Delta C(\hat{\mathbf{x}}). \tag{2.16}$$

We use (2.16) to rescale equations (2.10), (2.11) and (2.13). The following dimensionless parameters are introduced

$$Pe = \frac{UL}{\kappa\beta},\ Cn = \frac{\xi}{L},\ Re = \frac{\rho_m UL}{\mu},\ Ca = \frac{\mu U}{\beta\xi}.$$

The dimensionless parameters Pe, Cn, Re, Ca above are the Peclet, Cahn, Reynolds, and Capillary numbers, respectively. Due to the reasonings that the interface thickness ϵ, and, thus ξ, has to be very small, we also see that Cn should be a small positive parameter. Substituting the parameters into the equations (2.10) - (2.13), the system of governing equations for the binary fluid takes the form

$$\begin{aligned} \frac{\partial C}{\partial t} + (\mathbf{u}\cdot\nabla)C &= \tfrac{1}{Pe}\nabla\cdot\lambda\nabla\left(\Psi'(C) - Cn^2\Delta C\right) \\ Re\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) &= -\nabla p + \nabla\cdot\varrho(\nabla)\mathbf{u} + \nabla\mathbf{u}^T + \tfrac{1}{Ca\cdot Cn}\eta\nabla C \\ \nabla\cdot\mathbf{u} &= 0, \end{aligned} \tag{2.17}$$

where λ and ϱ are the normalized mobility and viscosity, respectively and $\eta = \Psi'(C) - Cn^2\Delta C$ is the dimensionless chemical potential.

2.5 Governing equations

In this work, we assume that the computational domain Ω is in three space dimensions and the mobility κ is constant. We also do not consider the coupling to the Navier-Stokes equations. Thus, the constant mobility Cahn-Hilliard model, which is studied here numerically, reads as follow

$$\frac{\partial C}{\partial t} + (\mathbf{u}\cdot\nabla)C = \frac{1}{Pe}\Delta\left(\Psi'(C) - Cn^2\Delta C\right), \tag{2.18}$$

subject to the boundary condition $\nabla C\cdot\mathbf{n}|_{\partial\Omega} = 0$ and $\nabla\eta\cdot\mathbf{n}|_{\partial\Omega} = 0$. The velocity $\mathbf{u}$ satisfies the coupled time-dependent incompressible Navier-Stokes equations in (2.4), and is assumed to be known. When the convection is small or zero, which means $\mathbf{u}\equiv\mathbf{0}$ in (2.18), as for instance, in metallurgy, phase separation is only caused by interface diffusion. The model in this case reads as follows:

$$\frac{\partial C}{\partial t} = \frac{1}{Pe}\Delta\left(\Psi'(C) - Cn^2\Delta C\right) \tag{2.19}$$

where $Pe = 1$ is a usual assumption.
Equation (2.18) is a fourth order nonlinear parabolic equation. If we consider (2.18) directly, we have to discretize higher order derivatives in a suitable way. For example, this problem has been discretized using a discontinuous Galerkin method [25].
In order to use more standard discretization techniques, we reformulate (2.18) as a system of two coupled second order equations. This is done by considering η as an additional unknown. We obtain

$$\begin{aligned} \eta - \Psi'(C) + \epsilon^2\Delta C &= 0, \quad (\mathbf{x},t)\in\Omega_T\equiv\Omega\times(0,T), \Omega\subset\mathbb{R}^3 \\ -\omega\Delta\eta + \frac{\partial C}{\partial t} + (\mathbf{u}\cdot\nabla)C &= 0, \quad (\mathbf{x},t)\in\Omega_T \\ \frac{\partial C}{\partial \mathbf{n}} = 0, \quad \frac{\partial \eta}{\partial \mathbf{n}} &= 0, \quad \mathbf{x}\in\partial\Omega \\ C(\mathbf{x},0) &= C_0(\mathbf{x}), \end{aligned} \tag{2.20}$$

where $0 < \epsilon \ll 1$ denotes Cn and $0 < \omega \leq 1$ denotes $1/Pe$, for notational simplicity. The velocity $\mathbf{u}$ is assumed to be known.
In the sequel, we discretize, and solve numerically the Cahn-Hilliard problem, as stated in (2.20).

Chapter 3

Finite element discretizations and the θ-method

Recalling the previous chapter, the system of PDEs has the following form:

$$\begin{aligned} \eta - \Psi'(C) + \epsilon^2 \Delta C &= 0, \quad (\mathbf{x}, t) \in \Omega_T \equiv \Omega \times (0, T], \Omega \subset \mathbb{R}^3 \\ \omega \Delta \eta - \frac{\partial C}{\partial t} - (\mathbf{u} \cdot \nabla) C &= 0, \quad (\mathbf{x}, t) \in \Omega_T \\ \frac{\partial C}{\partial \mathbf{n}} = 0, \; \frac{\partial \eta}{\partial \mathbf{n}} &= 0, \quad \mathbf{x} \in \partial\Omega \\ C(\mathbf{x}, 0) &= C_0(\mathbf{x}) \end{aligned} \tag{3.1}$$

where $0 < \epsilon \ll 1$ and $0 < \omega \ll 1$.
In the above system (3.1), $\mathbf{n}$ is the unit outward normal to $\partial\Omega$. $\mathbf{u}$ denotes the velocity vector of convection due to fluid flows, which is given. The two unknown functions are $C(\mathbf{x}, t)$ and $\eta(\mathbf{x}, t)$. The function $C(\mathbf{x}, t)$ represents the concentration of the fluid and at the same time serves to model the interface between them. It takes a constant value in each bulk phase and varies smoothly and and continuously in the interface. The second unknown $\eta(\mathbf{x}, t)$ is the so-called chemical potential and plays an auxiliary role.

3.1 Discretization in space

The Cahn-Hilliard equations (3.1) depend on both space and time. We discretize those in space using the standard conforming finite element method and derive the resulting semi-discrete system of ordinary differential equations. For setting up the spatial FE approximation, the first step is to rewrite the system of equations (3.1) in variational form. Let $V = \{v : \|\nabla v\| + \|v\| < \infty\}$. Multiplying (3.1) with a test-function $v \in V$ and integrating over Ω using Green's formula with the homogeneous Neumann conditions, we

have

$$\begin{aligned} 0 &= \int_\Omega \eta\, v\, d\mathbf{x} - \int_\Omega \Psi'(C)\, v\, d\mathbf{x} + \epsilon^2 \int_\Omega \Delta C\, v\, d\mathbf{x} \\ &= \int_\Omega \eta\, v\, d\mathbf{x} - \int_\Omega \Psi'(C)\, v\, d\mathbf{x} + \epsilon^2 \int_{\partial\Omega} \mathbf{n}\cdot\nabla C\, v\, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla C\cdot\nabla v\, d\mathbf{x} \qquad (3.2a) \\ &= \int_\Omega \eta\, v\, d\mathbf{x} - \int_\Omega \Psi'(C)\, v\, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla C\cdot\nabla v\, d\mathbf{x} \\ 0 &= \omega \int_\Omega \Delta\eta\, v\, d\mathbf{x} - \int_\Omega \frac{\partial C}{\partial t} v\, d\mathbf{x} - \int_\Omega (\mathbf{u}\cdot\nabla) C v\, d\mathbf{x} \\ &= \omega \int_{\partial\Omega} \mathbf{n}\cdot\nabla\eta\, v\, d\mathbf{x} - \omega \int_\Omega \nabla\eta\cdot\nabla v\, d\mathbf{x} - \int_\Omega \frac{\partial C}{\partial t} v\, d\mathbf{x} - \int_\Omega (\mathbf{u}\cdot\nabla) C v\, d\mathbf{x} \qquad (3.2b) \\ &= -\omega \int_\Omega \nabla\eta\cdot\nabla v\, d\mathbf{x} - \int_\Omega \frac{\partial C}{\partial t} v\, d\mathbf{x} - \int_\Omega (\mathbf{u}\cdot\nabla) C v\, d\mathbf{x}, \quad t\in(0,T] \end{aligned}$$

The variational form of (3.1) is thus defined to be the following problem:
Find a pair $C(t), \eta(t) \in V$ such that the equations

$$\begin{aligned} \int_\Omega \eta\, v\, d\mathbf{x} - \int_\Omega \Psi'(C)\, v\, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla C\cdot\nabla v\, d\mathbf{x} &= 0 \\ \omega \int_\Omega \nabla\eta\cdot\nabla v\, d\mathbf{x} + \int_\Omega \frac{\partial C}{\partial t} v\, d\mathbf{x} + \int_\Omega (\mathbf{u}\cdot\nabla) C v\, d\mathbf{x} &= 0 \end{aligned} \qquad (3.3)$$

hold $\forall v \in V$, $t \in (0,T]$.
In order to formulate the discrete method, we decompose the infinite-dimensional computational domain Ω into finite-dimensional subsets (elements) with a characteristic size h. (In this thesis, we use hexahedral (cube) elements in 3D.) Let $V_h \subset V$ be the subspace of continuous piecewise linears on a mesh $\mathcal{K}$ of Ω. The discrete space counterpart of (3.5) reads:
Find $C^h, \eta^h \in V_h$ such that the equations

$$\begin{aligned} \int_\Omega \eta^h\, v\, d\mathbf{x} - \int_\Omega \Psi'(C^h)\, v\, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla C^h\cdot\nabla v\, d\mathbf{x} &= 0 \\ \omega \int_\Omega \nabla\eta^h\cdot\nabla v\, d\mathbf{x} + \int_\Omega \frac{\partial C^h}{\partial t} v\, d\mathbf{x} + \int_\Omega (\mathbf{u}\cdot\nabla) C^h v\, d\mathbf{x} &= 0 \end{aligned} \qquad (3.4)$$

hold $\forall v \in V_h$, $t \in (0,T]$.
Next, to compute the finite element approximation C^h and η^h we let $\{\varphi_i\}_{i=1}^N$ be the trilinear basis for the subspace V_h. Since C^h and η^h belong to V_h they can be written as:

$$C^h(t) = \sum_{j=1}^N c_j(t)\varphi_j, \quad \eta^h(t) = \sum_{j=1}^N d_j(t)\varphi_j \qquad (3.5)$$

with N time-dependent unknowns $c_j(t)$, $d_j(t), j = 1, 2, \cdots, N$, to be found.
Inserting (3.5) and test function $v = \varphi_i$ into (3.3) we obtain

$$\begin{aligned}
0 = & \int_\Omega \eta^h \, v \, d\mathbf{x} - \int_\Omega \Psi'(C^h) \, v \, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla C^h \cdot \nabla v \, d\mathbf{x} \\
= & \int_\Omega \left(\sum_{j=1}^{N} d_j(t) \varphi_j \right) \varphi_i \, d\mathbf{x} - \int_\Omega \Psi'(C^h(t)) \, \varphi_i \, d\mathbf{x} - \epsilon^2 \int_\Omega \nabla \left(\sum_{j=1}^{N} c_j(t) \varphi_j \right) \cdot \nabla \varphi_i \, d\mathbf{x} \\
= & \sum_{j=1}^{N} d_j(t) \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x} - \int_\Omega \Psi'(C^h(t)) \, \varphi_i \, d\mathbf{x} - \epsilon^2 \sum_{j=1}^{N} c_j(t) \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\mathbf{x}
\end{aligned} \tag{3.6a}$$

$$\begin{aligned}
0 = & \omega \int_\Omega \nabla \eta^h \cdot \nabla v \, d\mathbf{x} + \int_\Omega \frac{\partial C^h}{\partial t} v \, d\mathbf{x} + \int_\Omega (\mathbf{u} \cdot \nabla) \, C^h v \, d\mathbf{x} \\
= & \omega \int_\Omega \nabla \left(\sum_{j=1}^{N} d_j(t) \varphi_j \right) \cdot \nabla \varphi_i \, d\mathbf{x} + \int_\Omega \left(\sum_{j=1}^{N} \frac{\partial c_j(t)}{\partial t} \varphi_j \right) \varphi_i \, d\mathbf{x} + \int_\Omega \left(\mathbf{u} \cdot \left(\sum_{j=1}^{N} c_j(t) \nabla \varphi_j \right) \right) \varphi_i \, d\mathbf{x} \\
= & \omega \sum_{j=1}^{N} d_j(t) \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\mathbf{x} + \sum_{j=1}^{N} \frac{\partial c_j(t)}{\partial t} \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x} + \sum_{j=1}^{N} c_j(t) \int_\Omega (\mathbf{u} \cdot \nabla \varphi_j) \, \varphi_i \, d\mathbf{x}
\end{aligned} \tag{3.6b}$$

where $i, j = 1, 2, \cdots, N$
Using the notation

$$\begin{aligned}
M_{ij} &= \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x}, \quad i, j = 1, 2, \cdots, N \\
K_{ij} &= \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\mathbf{x}, \quad i, j = 1, 2, \cdots, N \\
W_{ij} &= \int_\Omega (\mathbf{u} \cdot \nabla \varphi_j) \, \varphi_i \, d\mathbf{x}, \quad i, j = 1, 2, \cdots, N \\
f(\mathbf{c}(t))_{ij} &= \int_\Omega \Psi'(C^h(t)) \, \varphi_i \, d\mathbf{x}, \quad i, j = 1, 2, \cdots, N
\end{aligned}$$

the spatial semi-discretization of (3.3) in matrix form reads as follows:
Find $\mathbf{c}(t) = \{c_i(t)\}_{i=1}^N$, $\mathbf{d}(t) = \{d_i(t)\}_{i=1}^N$ such that

$$\begin{aligned}
M\mathbf{d}(t) - f(\mathbf{c}(t)) - \epsilon^2 K \mathbf{c} &= 0 \\
\omega K \mathbf{d}(t) + M \frac{\partial \mathbf{c}(t)}{\partial t} + W \mathbf{c}(t) &= 0
\end{aligned} \tag{3.7}$$

where $t \in (0, T]$. M, K and W are the Gramian mass matrix, the discrete Laplacian stiffness matrix and the matrix, corresponding to the discrete convective term in (3.1), respectively.

3.2 Discretization in time

To discretize also in time, let $0 = t_0 < t_1 < t_2 < ... < t_L = T$ be a time grid on the time interval $(0, T]$ with steps $\Delta t_k = t_k - t_{k-1}$, $k = 1, 2, \cdots, L$. Also, let $\mathbf{c}^k$ and $\mathbf{d}^k$ denote approximations to $\mathbf{c}(t_k)$ and $\mathbf{d}(t_k)$. For the discretisation of the time derivative we use the

θ-method. This method is also known as the weighted method, which has the following form

$$y_k = y_{k-1} + \Delta t_k[\theta f(t_k, y_k) + (1-\theta) f(t_{k-1}, y_{k-1})], \quad k = 1, 2 \cdots \tag{3.8}$$

where $\theta \in [0, 1]$. We see that three classical methods for time-dependent problems fit in the framework (3.8), namely, the forward Euler, the backward Euler and the Trapezoidal (i.e. Crank-Nicolson) methods. For $\theta = 1$ we obtain the backward Euler method and for $\theta = 1/2$ we obtain the Crank-Nicolson method.
Applying the θ-method to the semi-discrete problem (3.7) gives raise to the following fully discretized Cahn-Hilliard equations (3.9):

$$\begin{aligned} M\mathbf{d}^k - f(\mathbf{c}^k) - \epsilon^2 K\mathbf{c}^k &= 0 \\ M(\mathbf{c}^k - \mathbf{c}^{k-1}) + \Delta t_k[\theta(\omega K\mathbf{d}^k + W\mathbf{c}^k) + (1-\theta)(\omega K\mathbf{d}^{k-1} + W\mathbf{c}^{k-1})] &= 0 \end{aligned} \tag{3.9}$$

In this work, we choose $\theta = 1$, thus, the fully implicit backward Euler scheme is as follows:

$$\begin{aligned} M\mathbf{d}^k - f(\mathbf{c}^k) - \epsilon^2 K\mathbf{c}^k &= 0, \\ \omega\Delta t_k K\mathbf{d}^k + M\mathbf{c}^k + \Delta t_k W\mathbf{c}^k - M\mathbf{c}^{k-1} &= 0. \end{aligned} \tag{3.10}$$

Here, the starting vectors $\mathbf{c}_0$ and $\mathbf{d}_0$ are the nodal interpolations of C_0 and η_0 on the mesh. According to the initial condition in (3.1), the initial vectors $\mathbf{c}_0 = \{C_0(x_i)\}_{i=1}^N$ and $\mathbf{d}_0 = \{(\Psi'(C_0(x_i)) - \epsilon^2\Delta C_0(x_i))\}_{i=1}^N$. At each time step, from the known approximate solutions $\mathbf{c}^{k-1}$ and $\mathbf{d}^{k-1}$ at time t_{k-1}, the solutions $\mathbf{c}^k$ and $\mathbf{d}^k$ at time $t_k = t_{k-1} + \Delta t_k$ are obtained by solving the system (3.10). Since this problem is nonlinear, its solution requires in general an iterative method, such as functional iteration or Newton's method. In this project, we use Newton's method to handle the nonlinearity. The details are introduced in Section (4.1).

A priori error estimates

For completeness, we state here the error bounds for the well-known elliptic problems of second order in space and first order in time. As is shown for example in [24] and [10], the discretized solution $U(t)$ satisfy a priori error estimate

$$\|u(t) - U(t)\| \leq C^S h^2(\|\Delta u(\cdot, 0)\| + \int_0^t \|\Delta u_t(\cdot, s)\| ds)$$

where $u(t)$ is the exact solution. For the fully discrete solution U_n, the following priori estimate holds,

$$\|u(t) - U_n\| \leq C^S h^2(\|\Delta u(\cdot, 0)\| + \int_0^t \|\Delta u_t(\cdot, s)\| ds) + C^T \Delta t \int_0^t \|\Delta u_{tt}(\cdot, s)\| ds$$

where Δt is a uniform time step, C^S and C^T are constants, independent of h and t. This shows that the error for $\theta = 1$ is bounded by $O(h^2) + O(\Delta t)$. We also note that the choice of Δt, is related to h, which controls the total discretisation error.
If one uses $\theta = \frac{1}{2}$ (rather $\theta = \frac{1}{2} + \zeta$ for small enough ζ), the total discretization error is then $O(h^2 + \Delta t^2)$. More details on the error estimates can be found in [5].

Chapter 4

Solution of the nonlinear algebraic system of equations

4.1 Solution of nonlinearity

Due to the presence of the term $\Psi'(C)$ in (3.1), the Cahn-Hilliard mathematical model and, thus, the arising discrete system of algebraic equations is nonlinear. A common approach to nonlinear problems is linearization. Here, as mentioned in the last chapter, after space and time discretization, we obtain the nonlinear algebraic system of equations (3.10). At each time step t_k, we use classical Newton's method to solve it. Let $\mathbf{X}^k = \left[\mathbf{d}^k, \mathbf{c}^k\right]^T$ denote the combined vector of unknowns. We then rewrite the time stepping scheme (3.10) into the equivalent form

$$F_k(\mathbf{X}^k) = 0, \tag{4.1}$$

where F_k is defined by

$$F_k(\mathbf{X}^k) = \begin{bmatrix} M\mathbf{d}^k - f(\mathbf{c}^k) - \epsilon^2 K\mathbf{c}^k \\ \omega\Delta t_k K\mathbf{d}^k + M\mathbf{c}^k + \Delta t_k W\mathbf{c}^k - M\mathbf{c}^{k-1} \end{bmatrix}. \tag{4.2}$$

As is well known, Newton's method is an iterative method. It assumes that we possess an initial guess $\mathbf{X}^{k,0}$ of the solution $\mathbf{X}^k = \left[\mathbf{d}^k, \mathbf{c}^k\right]^T$. Then we repeatedly solve the linearized equation and update the solution as follows

$$\mathcal{J}(\mathbf{X}^{k,s})\Delta\mathbf{X}^{k,s} = -F_k(\mathbf{X}^{k,s}),\ \mathbf{X}^{k,s+1} = \mathbf{X}^{k,s} + \Delta\mathbf{X}^{k,s},\ s = 0, 1, \cdots \tag{4.3}$$

where the entries of the Jacobian matrix $\mathcal{J}$ are $\mathcal{J}_{i,j}(\mathbf{X}^k) = \frac{\partial(F_k)_i}{\partial(\mathbf{X}^k)_j}$, $i, j = 1, 2, \cdots, 2N$. When a desired stopping criterion is met (for instance, $\|\Delta\mathbf{X}^{k,s}\| < \varepsilon$), the iteration process is stopped. Regarding $\mathcal{J}$, we note that the linear term of (4.2) is easy to differentiate with

respect to $\mathbf{d}^k$ and $\mathbf{c}^k$. We have

$$\begin{array}{rcl} \frac{\partial(M\mathbf{d}^k-\epsilon^2K\mathbf{c}^k)_i}{\partial(\mathbf{d}^k)_j} & = & M_{ij} \\ \frac{\partial(M\mathbf{d}^k-\epsilon^2K\mathbf{c}^k)_i}{\partial(\mathbf{c}^k)_j} & = & -\epsilon^2K_{ij} \\ \frac{\partial(\omega\Delta t_kK\mathbf{d}^k+M\mathbf{c}^k+\Delta t_kW\mathbf{c}^k-M\mathbf{c}^{k-1})_i}{\partial(\mathbf{d}^k)_j} & = & \omega\Delta t_kK_{ij} \\ \frac{\partial(\omega\Delta t_kK\mathbf{d}^k+M\mathbf{c}^k+\Delta t_kW\mathbf{c}^k-M\mathbf{c}^{k-1})_i}{\partial(\mathbf{c}^k)_j} & = & M_{ij}+\Delta t_kW_{ij} \end{array} \tag{4.4}$$

Since the function $f(\mathbf{c})$ is nonlinear and only depends on $\mathbf{c}$, it is differentiated by using the chain rule as follows

$$\frac{\partial f((\mathbf{c}))_i}{\partial c_j} = \frac{\partial}{\partial c_j}\int_\Omega \Psi'(C^h)\varphi_i d\mathbf{x} = \int_\Omega \frac{\partial\Psi'(C^h)}{\partial C^h}\frac{\partial C^h}{\partial c_j}\varphi_i d\mathbf{x} = \int_\Omega \frac{\partial\Psi'(C^h)}{\partial C^h}\varphi_j\varphi_i d\mathbf{x}.$$

Since $\Psi'' = 3C^2-1$ and $C^h = \sum_{l=1}^{N} c_l\varphi_l$, we obtain the explicit matrix form J

$$J = \frac{\partial f((\mathbf{c}))_i}{\partial c_j} = \begin{bmatrix} ((3(\sum_{l=1}^{N} c_l\varphi_l)^2-1)\varphi_1,\varphi_1) & \cdots & ((3(\sum_{l=1}^{N} c_l\varphi_l)^2-1)\varphi_N,\varphi_1) \\ \vdots & \ddots & \vdots \\ ((3(\sum_{l=1}^{N} c_l\varphi_l)^2-1)\varphi_1,\varphi_N) & \cdots & ((3(\sum_{l=1}^{N} c_l\varphi_l)^2-1)\varphi_N,\varphi_N) \end{bmatrix}. \tag{4.5}$$

The matrix J is similar to the mass matrix M with coefficient $j_e = 3(\sum_{l=1}^{N} c_l\varphi_l)^2-1$, which depends on the solution of the concentration vector $\mathbf{c}$ at the previous Newton step. Since there holds $-1 \le c_i \le 1$, one can see that $-1 \le j_e \le 2$. From the above representations (4.4) and (4.5), we obtain that the Jacobian matrix of $F_k(\mathbf{X}^k)$ is given by

$$\mathcal{J}(\mathbf{X}) = \begin{bmatrix} M & -J(\mathbf{c})-\epsilon^2K \\ \omega\Delta t_kK & M+\Delta t_kW \end{bmatrix}, \tag{4.6}$$

where $J(\mathbf{c})$ is the Jacobian of the nonlinear term $f(\mathbf{c})$ only.
The reason we choose Newton's method is that, provided that the Jacobian is invertible and $\mathbf{x}_s$ is sufficiently close to the exact solution $\mathbf{x}^*$, it usually converges rapidly, namely, it holds

$$\|\mathbf{x}_{s+1}-\mathbf{x}^*\| \le L\|\mathbf{x}_s-\mathbf{x}^*\|^2.$$

As already stated, the general idea of Newton's method is to linearize F around a current guess, $\mathbf{x}_s$, and compute a correction δ_s expecting that $\mathbf{x}_{s+1} = \mathbf{x}_s+\delta_s$ is a better approximation of the root of the nonlinear problem than $\mathbf{x}_s$ (see Algorithm 1). The method is attractive due to its rapid convergence from any sufficiently good initial guess $\mathbf{x}_0$. However, there are two drawbacks with this method.

Algorithm 1 Newton's method

1. Choose a starting guess $\mathbf{x}_0$, and a desired accuracy ε
4. **for** $s = 0, 1, 2 \cdots$ **do**
5. Compute the correction $F'(\mathbf{x}_s)\delta_s = -F(\mathbf{x}_s)$
6. Update the solution guess $\mathbf{x}_{s+1} = \mathbf{x}_s + \delta_s$
7. **if** $|\delta_s| < \varepsilon$ **then**
8. Stop
9. **end if**
10. **end for**

- First, at every Newton iteration, we have to solve exactly the Jacobian equation $F_k'(\mathbf{X}^{k,s})\Delta\mathbf{X}^{k,s} = -F_k(\mathbf{X}^{k,s})$. Computing the exact Jacobian of large-scale problems can be very expensive.

- Second, when $\mathbf{x}_k$ is far from $\mathbf{x}^*$, it may not be a justified approximation of the root of the nonlinear problem (i.e. no guarantee of global convergence).

Therefore, we consider to replace the Newton step with an "inexact" Newton step.

4.2 Inexact Newton methods

In step 5 of Algorithm 1, if $F'(\mathbf{x}_s)$ is the exact Jacobian matrix of $F(\mathbf{x}^{k,s})$ and we solve the system with $F(\mathbf{x}^{k,s})$ exactly, the method is referred to as the exact Newton method. If the system in step 5 is not solved exactly or $F'(\mathbf{x}_s)$ is an approximation to the exact Jacobian matrix, the method is referred to as the "inexact Newton method".
In this work, the nonlinear operator F, which we deal with, is differentiable. F' exists and is nonsingular. We assume that we possess a good enough initial guess $\mathbf{x}_0$. We consider two following variants of an inexact Newton method:

(i1) Solve the systems with F' by a preconditioned iterative method. The algorithm takes the form:

Given $\mathbf{x}_0$ and ε,
For $s = 0, 1, 2 \cdots$ **do** until convergence
Compute δ_s as a solution of the system
(i1-1) $F'(\mathbf{x}_s)\delta_s = -F(\mathbf{x}_s) + \mathbf{r}_s$,
where $\frac{\|\mathbf{r}_s\|}{\|F(\mathbf{x}_s)\|} \leq \mu_s$, $\mu_s \in (0, 1)$
(i1-2) Update $\mathbf{x}_{s+1} = \mathbf{x}_s + \varsigma_s\delta_s$, $\varsigma_s \in (0, 1]$

(i2) The second one is to replace F' by a approximation $\widehat{F}_0$, thus, we solve

Given $\mathbf{x}_0$ and ε,
For $s = 0, 1, 2 \cdots$ **do** until convergence
Compute δ_s as a solution of the system
(i2-1) $\widehat{F}_0 \delta_s = -F(\mathbf{x}_s)$
(i2-2) Update $\mathbf{x}_{s+1} = \mathbf{x}_s + \varsigma_s \delta_s$, $\varsigma_s \in (0, 1]$

In Chapter 5 we make a proper choice of $\widehat{F}_0$ and use it as a preconditioner for the iterative method in (i1-1) as well as in (i2-1). As it turns out, for the Cahn-Hilliard problem, $\widehat{F}_0$ does not depend on $\mathbf{x}_s$ and we can efficiently solve systems with it to arbitrary accuracy.

Local convergence

The method (i1) is studied in [19] and it is shown that for good enough $\mathbf{x}_0$, the method is locally convergent. The sequence $\{\mu_s\}, s = 1, 2, \cdots$ is referred to as the forcing sequence. The choice of μ_s affects local convergence properties. Clearly, for $\mu_s = 0$, we obtain the exact Newton method.
A well known effect of using inexact Newton method is that when an iterative method is used in (i1-1), the stopping criterion for the iterations may be chosen much smaller than that needed to meet the condition on the norm of scaled residual for a given μ_s. Further discussion on this issue falls out of the scope of this work.
Consider the inexact Newton method (i1). Assume, that to solve systems with F' we use a preconditioned iterative method with a preconditioner $\widehat{F}_0$, which is nonsingular and for which the condition number of $\widehat{F}_0^{-1} F'$ is much smaller than that of F'. The operator $\widehat{F}_0$ may possibly also depend on $\mathbf{x}_s$. When F' is preconditioned by $\widehat{F}_0$, the system is modified as below:

$$\widehat{F}_0^{-1} F'(\mathbf{x}_s)\delta_s = -\widehat{F}_0^{-1} F(\mathbf{x}_s) + \widehat{\mathbf{r}}_s \,,$$

where the preconditioned residual $\widehat{\mathbf{r}}_s \equiv \widehat{F}_0^{-1}(F'(\mathbf{x}_s)\delta_s + F(\mathbf{x}_s))$ and the norm of scaled residual $\frac{\widehat{\mathbf{r}}_s}{\|\widehat{F}_0^{-1} F(\mathbf{x}_s)\|} \leq \mu_s$, $\mu_s \in (0, 1)$
This case is studied earlier in [6]. There, local convergence of the inexact Newton method is shown, provided that the preconditioned operator $\widehat{F}_0^{-1} F'$ is continuous in the following sense (Hölder-continuous):

$$\|\widehat{F}_0^{-1} F'(\mathbf{x}) - \widehat{F}_0^{-1} F'(\mathbf{y})\| \leq \tau \|\mathbf{x} - \mathbf{y}\|^{\varrho} \,,$$

where τ is constant, $\varrho \in (0, 1]$ and the above equation holds for any $\mathbf{x}$, $\mathbf{y}$ in a certain subspace.
If $\widehat{F}_0$ and F' are linear operators, as for the considered problem, the above relation takes the form

$$\|\widehat{F}_0^{-1} F'(\mathbf{x}) - \widehat{F}_0^{-1} F'(\mathbf{y})\| = \|\widehat{F}_0^{-1} F'(\mathbf{x} - \mathbf{y})\| \leq \|\widehat{F}_0^{-1} F'\| \|\mathbf{x} - \mathbf{y}\| \,. \tag{4.7}$$

Thus, $\varrho = 1$ and $\tau = \|\widehat{F}_0^{-1} F'\|$, which is clearly a bounded quantity. Therefore, in our case, the inexact Newton method is locally convergent.

Global convergence

Globally convergent inexact Newton methods are studied in [6], [29], [14] and others. The global convergence is derived for the so-called damped inexact Newton method (DINM), where, as indicated in (i1-2) and (i2-2)

$$\mathbf{x}_{s+1} = \mathbf{x}_s + \varsigma_s \delta_s \,,$$

where $\varsigma_s \in (0, 1]$ is the so-called damping parameter. The meaning of $\varsigma_s < 1$ is that when we are far away from the exact solution, we could better take a shorter step along the so-computed search direction δ_s, since it is obtained via an inexact Newton method. In our experiments, due to the observed fast convergence, we use $\varsigma_s = 1$.
Global convergence of inexact Newton methods of type (i1) is analysed in [6] and [29]. The conditions, under which global convergence is shown, constitute that F is Fréchet-differentiable, F' is Lipschitz-continuous and the preconditioned operator $\widehat{F}_0^{-1} F'$ is Hölder-continuous.
It is then shown that there exists a sequence of $\{\varsigma_s\} \in (0, 1]$, such that

$$\|F(\mathbf{x}_{s+1})\| = \|F(\mathbf{x}_s + \varsigma_s \delta_s)\| < q \|F(\mathbf{x}_s)\| \,,$$

where $q < 1$.
Inexact Newton methods of type (i2) are analysed in [14]. There, global convergence is shown under the conditions that $\widehat{F}_0$ is uniformly bounded, i.e. $\|\widehat{F}_0\| \leq \alpha$, and F' is Lipschitz-continuous and nonsingular and

$$\frac{\|(\widehat{F}_0 - F'(\mathbf{x}^*))\mathbf{x}_s\|}{\mathbf{x}_s} \underset{s \to \infty}{\longrightarrow} 0$$

where $\mathbf{x}^*$ is the exact solution of $F(\mathbf{x}) = 0$
Under the above conditions and suitable choice of ς_s, both methods (i1) and (i2) are globally convergent and retain a superlinear rate of convergence.

Chapter 5

Preconditioned iterative solution method for the linearized Cahn-Hilliard equation

In this chapter we discuss the solution of the linear system (4.3). First of all, we note that the system tends to be very large, in particular, in 3D. More to that, it needs to be solved many times during each nonlinear iteration. And the nonlinear system is solved during each time step. Last, but not least, the matrix in (4.3) has to be recomputed, at least partly, every time when we solve with it. For such problems, iterative methods become nearly mandatory to use, due to their low computational complexity per iteration and memory requirements. Taking the above into account, we choose to solve (4.3) by a preconditioned iterative solution method.

5.1 The idea of preconditioning

For simplicity, in what follows, we use the generic name A to denote the system matrix, namely, we let

$$F_k'(\mathbf{X}) \equiv A = \begin{bmatrix} M & -J + \epsilon^2 K \\ \omega \Delta t_k K & M + \Delta t_k W \end{bmatrix}, \tag{5.1}$$

The FE discretization and Newton formulation of the Cahn-Hilliard problem leads to a large sparse linear system of the form:

$$A\mathbf{x} = \mathbf{b} \tag{5.2}$$

where $A \in \mathbb{R}^{n \times n}$ is the coefficient matrix, $\mathbf{b} \in \mathbb{R}^n$ is the known right hand side (RHS) and $\mathbf{x} \in \mathbb{R}^n$ is the unknown vector. Here n is the total number of degrees of freedom.
The convergence rate of iterative methods depends on the properties (i.e. the condition number) of the coefficient matrix A, which is denoted by $\kappa(A) = \|A^{-1}\| \, \|A\|$. The general theory of iterative methods shows that $\kappa(A)$ influences the number of iterations, needed to reduce the residual $\mathbf{b} - A\mathbf{x}$ below a given tolerance. Thus, the larger the condition number,

the more iterations we need. Furthermore, $\kappa(A)$ in our case depends on h and increases when the mesh is refined. To accelerate the convergence one usually tries to transform the original linear system $A\mathbf{x} = \mathbf{b}$ into one which has the same solution, but which has smaller condition number and is easier to solve. Generally speaking, preconditioning is such a transformation procedure. A preconditioner A_0 is any form of implicit or explicit approximation of A, which effects such a transformation. Instead of solving a system $A\mathbf{x} = \mathbf{b}$, we solve a left-preconditioned system

$$A_0^{-1}A\mathbf{x} = A_0^{-1}\mathbf{b}\,. \tag{5.3}$$

The choice of a preconditioner could be based on algebraic properties or could be problem-dependent, but generally must possess a number of properties. Clearly, one requirement for A_0 is that it should be easily "invertible" which means that it should be easy to solve system of the form $A_0\mathbf{y} = \mathbf{z}$. To reduce the number of iterations, it is also desirable that $A_0^{-1}A$ has a small condition number. For large systems, the solution with A_0 should also be easily parallelizable.

5.2 Overview of the preconditioned GMRES method

The matrix A in (5.1) is typically very large, sparse and non-symmetric. The lack of regular structure limits efficient utilization of certain iterative methods, such as traditional iteration method and multigrid method. As a result, the Krylov subspace methods become the natural choice for the problem. The Generalized Minimum Residual Method (GMRES) (see [32] and the references there in) is a standard Krylov subspace method used for non-symmetric problems. At step m we find an approximate solution $\mathbf{x}_m$ in the Krylov subspace

$$\mathcal{K}_m(A, \mathbf{r}_0) = \text{span}\,\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \cdots, A^{m-1}\mathbf{r}_0\}$$

where $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ is the initial residual and $\mathbf{x}_0$ is an initial guess, such that the residual

$$\|\mathbf{r}_m\|_2 = \|\mathbf{b} - A\mathbf{x}_m\|_2$$

is minimized. In practice, unpreconditioned GMRES is rarely used, since many realistic matrices are ill-conditioned and the GMRES method applied to such matrices converges very slowly or sometimes even diverges. Thus, in general, GMRES is always applied with a suitable preconditioner. In the case of a left preconditioner, we apply GMRES to the equivalent system of equations (5.3). The Arnoldi loop constructs an orthogonal basis in the Krylov subspace

$$\mathcal{K}_m(A_0^{-1}A, \mathbf{z}_0) = \text{span}\,\{\mathbf{z}_0, A_0^{-1}A\mathbf{z}_0, (A_0^{-1}A)^2\mathbf{z}_0, \cdots, (A_0^{-1}A)^{m-1}\mathbf{z}_0\}$$

where $\mathbf{z}_0 = A_0^{-1}(\mathbf{b} - A\mathbf{x}_0)$. The residual obtained is $A_0^{-1}(\mathbf{b} - A\mathbf{x}_m)$, instead of $\mathbf{b} - A\mathbf{x}_m$. The left preconditioned GMRES (PGMRES) algorithm (Algorithm 2) described above is given in [32].

Algorithm 2 left preconditioned GMRES

1. Choose a starting guess $\mathbf{x_0}$ for preconditioned GMRES method
2. Compute $\mathbf{r}_0 = A_0^{-1}(\mathbf{b} - A\mathbf{y}_0)$ and set $\mathbf{v}_1 = \mathbf{r}_0/\|r_0\|$
3. **for** $j = 1, 2\cdots$ until convergence **do**
4. Compute $\mathbf{z}_j = A\mathbf{v}_j$
5. Compute $\omega = A_0^{-1}\mathbf{z}_j$
6. **for** $i = 1, 2\cdots, j$ **do**
7. $\mathbf{h}_{i,j} = (\omega - \mathbf{v}_i)$
8. $\omega = \omega - \mathbf{h}_{i,j}\mathbf{v}_i$
9. **end for**
10. $\mathbf{h}_{i,j+1} = \|\omega\|_2,\ \mathbf{v}_{j+1} = \omega/\mathbf{h}_{i,j+1}$
11. **end for**
12. Define $Q_m = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m]$ and $\overline{H_m} = \{\mathbf{h}_{i,j}\}$
13. Compute $y_m = \text{argmin}_y\|\beta\mathbf{e}_1 - \overline{H_m}y\|$
14. Set $\mathbf{x}_m = \mathbf{x}_0 + Q_m y_m$

5.3 The preconditioning strategy used in this project

Although there are some robust preconditioners developed for general purpose, in practice, the choice of efficient preconditioner often depends on the properties of the coefficient matrix A. The proposed preconditioner in [11] described below is targeted on this specific Cahn-Hilliard model.

First, we take a closer look at the system matrix A in (5.1). The block M (the mass matrix) is symmetric, positive definite. The block K (the stiffness matrix) is also symmetric, positive semi-definite. Since the entries in J ($-1 \leq j_e \leq 2$) have different signs, $-(J+\epsilon^2 K)$ is symmetric and indefinite. Due to the presence of the convection part W, the block $M + \Delta t_k W$ is in general non-symmetric. In the case of diffusion with zero convection, W is zero. We note that the two-by-two block system matrix A is clearly non-symmetric.

Splitting the matrix A in (5.1),

$$A = \begin{bmatrix} M & -\epsilon^2 K \\ \omega\Delta t_k K & M \end{bmatrix} + \begin{bmatrix} 0 & -J \\ 0 & \Delta t_k W \end{bmatrix} = A_0 + \begin{bmatrix} 0 & -J \\ 0 & \Delta t_k W \end{bmatrix} \tag{5.4}$$

we can easily notice that if the influence of J and W becomes negligible, the matrix without J and $\Delta t_k W$ is close to A but much simpler to work with. Hence, the matrix A_0 is a good candidate for a preconditioner for A.

$$A_0 = \begin{bmatrix} M & -\epsilon^2 K \\ \omega\Delta t_k K & M \end{bmatrix} \tag{5.5}$$

An earlier work in [12] and [1] shows that in 2D for small enough h and Δt the idea to neglect the blocks J and W is justified. As is well-known, there are two important issues to consider when solving large scaled problem, the preconditioner and the choice of

the iterative method itself. As already stated, we use a preconditioned Krylov subspace method, namely, preconditioned GMRES.
However, in order to show the quality and the robustness of A_0 as a preconditioner to A, we analyse the convergence of a basic stationary iteration method with the preconditioner A_0.
To define a stationary iteration method for $A\mathbf{x} = \mathbf{b}$, we split the matrix A as $A = A_0 - R$, where R is

$$R = A_0 - A = \begin{bmatrix} 0 & J \\ 0 & -\Delta W \end{bmatrix}$$

Then we rewrite the equation $A\mathbf{x} = \mathbf{b}$ as

$$A_0\mathbf{x} = (A_0 - A)\mathbf{x} + \mathbf{b} \tag{5.6}$$

and define a simple iteration method as:

$$A_0\mathbf{x}_{k+1} = (A_0 - A)\mathbf{x}_k + \mathbf{b}. \tag{5.7}$$

Subtracting equation (5.6) from (5.7), we obtain the error equation, which connects $\mathbf{e}_k$ to $\mathbf{e}_{k+1}$:

$$A_0\mathbf{e}_{k+1} = (A_0 - A)\mathbf{e}_k \;\Leftrightarrow\; \mathbf{e}_{k+1} = (I - A_0^{-1}A)\mathbf{e}_k = E\mathbf{e}_k \tag{5.8}$$

At each iterative step, the error is multiplied by the matrix E. In order to converge, every eigenvalue of $E = I - A_0^{-1}A$ must have $|\lambda(E)| < 1$. The speed of the convergence entirely depends on the spectral radius of the matrix $E = I - A_0^{-1}A$. Thus, to determine whether the preconditioner A_0 is efficient or not, we analyse the following standard eigenvalue problem

$$(I - A_0^{-1}A)\mathbf{q} = \lambda\mathbf{q}. \tag{5.9}$$

Using (5.5), we obtain

$$A_0^{-1} = \begin{bmatrix} S_0^{-1} & \epsilon^2 M^{-1}KS_0^{-1} \\ -\delta S_0^{-1}KM^{-1} & S_0^{-1} \end{bmatrix}, \tag{5.10}$$

where $\delta = \omega\Delta t_k$ and $S_0 = M + \epsilon^2\delta KM^{-1}K$. Then, we notice that the preconditioned matrix $A_0^{-1}A$ is of the form

$$A_0^{-1}A = I + \begin{bmatrix} 0 & -S_0^{-1}J + \epsilon^2\Delta tM^{-1}KS_0^{-1}W \\ 0 & \delta S_0^{-1}KM^{-1}J + \Delta tS_0^{-1}W \end{bmatrix} \tag{5.11}$$

Hence,

$$E_q = (I - A_0^{-1}A)_q = \begin{bmatrix} 0 & S_0^{-1}J - \epsilon^2\Delta tM^{-1}KS_0^{-1}W \\ 0 & -\delta S_0^{-1}KM^{-1}J - \Delta tS_0^{-1}W \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \end{bmatrix} \tag{5.12}$$

From the characteristic equation $det(E - \lambda I) = 0$, namely,

$$\lambda I(\delta S_0^{-1}KM^{-1}J + \Delta tS_0^{-1}W + \lambda I) = 0$$

we see that N of the eigenvalues λ (i.e. λ_1) are equal to 0 with eigenvectors $\begin{bmatrix}\mathbf{q}_1\\ \mathbf{0}\end{bmatrix}$, where $\mathbf{q}_1$ is a vector of size N, where N is the size of the matrix K.
The rest of the eigenvalues are not equal to zero and from (5.12) we see that they satisfy the system

$$\begin{cases} J\mathbf{q}_2 &= \lambda(M\mathbf{q}_1 - \epsilon^2 K\mathbf{q}_2) \\ -\Delta t W\mathbf{q}_2 &= \lambda(\delta K\mathbf{q}_1 + M\mathbf{q}_2) \end{cases} \tag{5.13}$$

After some algebraic transformations in (5.13), we obtain the following equation of eigenvector $\mathbf{q}_2$

$$-\Delta t(M^{-1}W + \omega M^{-1}KM^{-1}J)\mathbf{q}_2 = \lambda(I + \Delta t\omega\epsilon^2(M^{-1}K)^2)\mathbf{q}_2. \tag{5.14}$$

or equivalently

$$-\Delta t(I + \Delta t\omega\epsilon^2(M^{-1}K)^2)^{-1}(M^{-1}W + \omega M^{-1}KM^{-1}J)\mathbf{q}_2 = \lambda\mathbf{q}_2. \tag{5.15}$$

Hence, the eigenvalues λ ($\lambda \neq 0$) of problem (5.9) consider with those of the matrix Q, where

$$Q \equiv -\Delta t(I + \Delta t\omega\epsilon^2(M^{-1}K)^2)^{-1}(M^{-1}W + \omega M^{-1}KM^{-1}J).$$

By definition, for any given real square matrix Q, the spectral radius $\rho(Q)$ satisfies

$$\rho(Q) = |\lambda|_{max} \leqslant \|Q\|.$$

From here on, we follow the derivations in 2D, presented in [12] and [1]. In order to estimate the nonzero eigenvalues λ, we first estimate the norm of the matrix Q. Based on the relation $|a|/(1 + a^2b^2) \leq 1/(2b)$, which holds for any real a and b, and $\|M^{-1}W\| = \|M^{-1}KK^{-1}W\| \leq \|M^{-1}K\|\|K^{-1}W\|$, it follows that

$$\begin{aligned} \|Q\| &\leq \Delta t\frac{\|M^{-1}W\|}{\|I + \Delta t\omega\epsilon^2(M^{-1}K)^2)\|} + \Delta t\omega\frac{\|M^{-1}K\|\,\|M^{-1}J\|}{\|I + \Delta t\omega\epsilon^2(M^{-1}K)^2)\|} \\ &\leq \frac{\Delta t}{2\sqrt{\Delta t\omega}\epsilon}\frac{\|M^{-1}W\|}{\|(M^{-1}K))\|} + \frac{\Delta t\omega}{2\sqrt{\Delta t\omega}\epsilon}\|M^{-1}J\| \end{aligned} \tag{5.16}$$

Recalling that the matrix J has the form $J_e = j_e M_e$, where M_e is the corresponding element mass matrix and $-1 \leq j_e \leq 2$, it is then straightforward to see that $\|M^{-1}J\| = O(1)$.
Following Refs. [33], for the mass, stiffness and convective matrices of trilinear quadrilateral elements on mesh in 3D, it holds

$$c_M h^3 \leq \frac{v^T M v}{v^T v} \leq C_M h^3$$

$$c_K h^3 \leq \frac{v^T K v}{v^T v} \leq C_K h$$

$$c_W h^2 \leq \frac{v^T W v}{v^T v} \leq C_W h^2;$$

where c_M, c_K, c_W, C_M, C_K, C_W are some constants, independent of h, and thus,

$$\frac{c_W}{C_M}h^{-1} \leq \frac{v^T W v}{v^T M v} \leq \frac{C_W}{c_M}h^{-1}$$

$$\frac{c_K}{C_M} \leq \frac{v^T K v}{v^T M v} \leq \frac{C_K}{c_M}h^{-2}$$

Hence, $\|M^{-1}W\| = O(h^{-1})$ and $\|M^{-1}K\| = O(h^{-2})$. In order to resolve the interface, we choose the mesh size $h = \epsilon/r$, where typically we have $5 \leq r \leq 10$. Recall also that for the Peclet number we have $Pe = 1/\omega$, where $0 < \omega \leq 1$. Since $\|M^{-1}J\| = O(1)$, it then follows that

$$\begin{aligned} \|Q\| &= O(\frac{\Delta t}{2\sqrt{\Delta t\omega\epsilon}}h) + O(\frac{\Delta t\omega}{2\sqrt{\Delta t\omega\epsilon}}) \\ &= O(\sqrt{Pe}\sqrt{\Delta t}) + O(\frac{1}{\sqrt{Pe}}\sqrt{\Delta t}\,h^{-1}) \\ &= O(\sqrt{\zeta}\,\sqrt{\Delta t\,h^{-1}}) + O(\frac{1}{\sqrt{\zeta}}\sqrt{\Delta t\,h^{-1}}) \end{aligned} \tag{5.17}$$

where $\zeta = Peh$ is called the mesh Peclet number.
As we know, one of the conditions for a successful preconditioner is that the iterative errors $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$ should approach zero as fast as possible (i.e. fast rate of convergence), which means that the eigenvalues λ of the error matrix E in (5.8) should be close to zero. As a consequence, $\|Q\|$ should be very small. From equation (5.17), we see that the order of $\|Q\|$ is highly depended on the mesh Peclet number ζ and the relationship between the time step size Δt and the mesh size h.
We see now that, in order to perform fast, reliable and accurate numerical simulations, we have to balance two factors. If we want to test the stationary state solution, we should take Δt as big as possible to reduce the computational costs. On the other hand, the choice of Δt, related to h, which controls the total discretisation error, also determines the quality of the preconditioner A_0. Therefore, in order to ensure high quality of the preconditioner A_0, but also within the desired discretisation error bounds, the choice of Δt plays an important role in this model. Since $\zeta = Peh$ and $h = \epsilon/r$, $\zeta = Peh = Pe\,\epsilon/r$, where $Pe \geq 1$ and $5 \leq r \leq 10$. We recall that ϵ is referred to the equilibrium interface thickness, which should be small enough to allow the surface deformations to be resolved. However if the thickness is too thin, the computational costs increases. We must, therefore, choose Δt as follows

$$\Delta t \ll \min\{\sqrt{\zeta}, 1/\sqrt{\zeta}\} = \min\{\sqrt{Pe\,\epsilon/r}, 1/\sqrt{Pe\,\epsilon/r}\}\,.$$

Thus, when $Pe \cong r/\epsilon$ (i.e. $\zeta \cong 1$), we can choose $\Delta t < h$ independently of these parameters. In this case the Peclet number is relatively large, which says that the flows in the bounded domain are convective. However, in absence of convection, $Pe \approx 1$, and then we must choose Δt

$$\Delta t \ll \min\{\sqrt{\epsilon/r}, 1/\sqrt{\epsilon/r}\} = 1/\sqrt{\epsilon/r}\,.$$

In this case, from the above theoretical analysis we see, that Δt must be chosen significantly smaller than h (i.e. $\Delta t < h^2$) to make λ close to zero, and ensure a rapid convergence. Thus, we can see that the eigenvalues of problem (5.9) satisfy the estimate

$$|\lambda| \leq \varsigma, \tag{5.18}$$

where,

(1) for diffusion dominated problems ($Pe = 1$), $\varsigma \to 0$ when $\Delta t < h^2$,

(2) for convection dominated problems ($Pe \gg 1$), $\varsigma \to 0$, when $\Delta t < h$.

Efficient solution of systems with A_0

Based on the derivations above, we see that A_0 is a suitable preconditioner of A. Next, we address the issue how to efficiently solve systems with it. One possible approach is to solve A_0 by another, inner, preconditioned iterative method. Earlier work in [2] and [1], shows that the matrix

$$\widehat{A}_0 = \begin{bmatrix} M & -\epsilon^2 K \\ \delta K & M + 2\epsilon\sqrt{\delta} K \end{bmatrix}$$

is an optimal preconditioner for A_0. We note that

$$A_0 = \widehat{A}_0 - \begin{bmatrix} 0 & 0 \\ 0 & 2\epsilon\sqrt{\delta} K \end{bmatrix}.$$

For completeness, we include the derivations here. To study the convergence properties of an iterative method with $\widehat{A}_0$ as a preconditioner to A_0, we analyze the following generalized eigenvalue problem

$$A_0 \mathbf{v} = \lambda \widehat{A}_0 \mathbf{v} \quad \Leftrightarrow \quad (\widehat{A}_0 - A_0)\mathbf{v} = (1-\lambda)\widehat{A}_0 \mathbf{v}\,. \tag{5.19}$$

Inserting $\widehat{A}_0$ and A_0 into (5.19), it follows

$$\begin{bmatrix} 0 & 0 \\ 0 & 2\epsilon\sqrt{\delta} K \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = (1-\lambda) \begin{bmatrix} M & -\epsilon^2 K \\ \delta K & M + 2\epsilon\sqrt{\delta} K \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}.$$

It holds

$$\begin{array}{rcl} (1-\lambda)(M\mathbf{v}_1 - \epsilon^2 K \mathbf{v}_2) & = & 0 \\ (1-\lambda)(\delta K \mathbf{v}_1 + (M + 2\epsilon\sqrt{\delta} K)\mathbf{v}_2) & = & 2\epsilon\sqrt{\delta} K \mathbf{v}_2 \end{array}. \tag{5.20}$$

Solving equations (5.20), we get that $\lambda = 1$ and the corresponding eigenvector $\mathbf{v}_2$ in the null space of K (i.e. $\mathbf{v}_2 \in \mathcal{N}(K)$).
If $\lambda \neq 1$, then

$$M\mathbf{v}_1 - \epsilon^2 K \mathbf{v}_2 = 0$$

Hence, $\mathbf{v}_1 = \epsilon^2 M^{-1} K \mathbf{v}_2$ and

$$(1-\lambda)(\epsilon^2 \delta K M^{-1} K \mathbf{v}_2 + (M + 2\epsilon\sqrt{\delta} K)\mathbf{v}_2) = 2\epsilon\sqrt{\delta} K \mathbf{v}_2$$

that is,

$$(1-\lambda)(\epsilon^2 \delta K M^{-1} K + M)\mathbf{v}_2 = 2\lambda\epsilon\sqrt{\delta} K \mathbf{v}_2$$

By multiplying by M^{-1} on both sides, we obtain

$$(1-\lambda)(BB + I)\mathbf{v}_2 = 2\lambda B \mathbf{v}_2\,,$$

where $B = \epsilon\sqrt{\delta} K$. Hence,

$$\frac{1}{\lambda} - 1 = \frac{2\mathbf{v}_2^T B \mathbf{v}_2}{\mathbf{v}_2^T \mathbf{v}_2 + \mathbf{v}_2^T B B \mathbf{v}_2}.$$

Using Cauchy-Schwartz inequality and the fact that the stiffness matrix K is symmetric and positive semi-definite, we obtain

$$0 \le 2\mathbf{v}_2^T B \mathbf{v}_2 \le \mathbf{v}_2^T \mathbf{v}_2 + (B\mathbf{v}_2)^T (B\mathbf{v}_2) = \mathbf{v}_2^T \mathbf{v}_2 + \mathbf{v}_2^T B^T B \mathbf{v}_2 = \mathbf{v}_2^T \mathbf{v}_2 + \mathbf{v}_2^T B B \mathbf{v}_2.$$

so

$$0 \le \frac{1}{\lambda} - 1 \le 1,$$

that is

$$\frac{1}{2} \le \lambda \le 1.$$

In conclusion, for the eigenvalue problem (5.19), there holds that

$$\begin{cases} \text{all eigenvalues } \lambda \in [0.5, 1], \\ \lambda = 1 \text{ for eigenvectors } \mathbf{v}_2 \text{ in the null space of } K. \end{cases} \tag{5.21}$$

All of the eigenvalues λ of the problem are contained in the interval $[0.5, 1]$. Thus, the condition number $\kappa(\widehat{A}_0^{-1} A_0) = |\frac{\lambda_{max}}{\lambda_{min}}| = 2$. This means that the preconditioned inner iterations will converge rapidly with the conjugate gradient method. Systems of A_0 can be solved very efficiently when preconditioned by $\widehat{A}_0$.
Now we consider the solution of the systems with the matrix $\widehat{A}_0$.

$$\widehat{A}_0 \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} M & -\epsilon^2 K \\ \delta K & M + 2\epsilon\sqrt{\delta} K \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}$$

First, we consider the exact block LU factorization of $\widehat{A}_0$(cf. [2], [1])

$$\widehat{A}_0 = \begin{bmatrix} M & 0 \\ \delta K & M + \epsilon\sqrt{\delta} K \end{bmatrix} \begin{bmatrix} I & -\epsilon^2 M^{-1} K \\ 0 & M^{-1}(M + \epsilon\sqrt{\delta} K) \end{bmatrix}. \tag{5.22}$$

The straightforward implementation of the block LU solver for (5.22) is given in Algorithm 3.

Algorithm 3 Straightforward LU solve

1. Compute $\mathbf{b}_1$ by solving $M\mathbf{b}_1 = \mathbf{f}$
2. Compute $\mathbf{b}_2$ by solving $(M + \epsilon\sqrt{\delta}K)\mathbf{b}_2 = \mathbf{g} - \delta K\mathbf{b}_1$
3. Compute $\mathbf{y}$ by solving $(M + \epsilon\sqrt{\delta}K)\mathbf{y} = M\mathbf{b}_2$
4. Compute $\mathbf{x} = \mathbf{b}_1 - \frac{\epsilon}{\sqrt{\delta}}(\mathbf{b}_2 - \mathbf{y})$

From Algorithm 3, we see that there are three solutions of systems required, one solution with M and two with $M + \epsilon\sqrt{\delta}K$.
Next, we consider a block LDU factorization in a block lower-triangular, block-diagonal and block upper-triangular form (cf. [2], [1])

$$\widehat{A}_0 = \begin{bmatrix} I & 0 \\ \delta K M^{-1} & I + \epsilon\sqrt{\delta}KM^{-1} \end{bmatrix} \begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} I & -\epsilon^2 M^{-1}K \\ 0 & I + \epsilon\sqrt{\delta}M^{-1}K \end{bmatrix}. \tag{5.23}$$

From the factorization (5.23) we obtain the exact form of $\widehat{A}_0^{-1}$ as follows

$$\begin{aligned}
&\widehat{A}_0^{-1} \\
&= \begin{bmatrix} I & \epsilon^2 M^{-1}K(I + \epsilon\sqrt{\delta}M^{-1}K)^{-1} \\ 0 & (I + \epsilon\sqrt{\delta}M^{-1}K)^{-1} \end{bmatrix} \begin{bmatrix} M^{-1} & 0 \\ 0 & M^{-1} \end{bmatrix} \times \\
&\quad \begin{bmatrix} I & 0 \\ -\delta K M^{-1}(I + \epsilon\sqrt{\delta}KM^{-1})^{-1} & (I + \epsilon\sqrt{\delta}KM^{-1})^{-1} \end{bmatrix} \\
&= \begin{bmatrix} M^{-1} & \frac{\epsilon}{\sqrt{\delta}}(I - (I + \epsilon\sqrt{\delta}M^{-1}K)^{-1})M^{-1} \\ 0 & (M + \epsilon\sqrt{\delta}K)^{-1} \end{bmatrix} \times \\
&\quad \begin{bmatrix} I & 0 \\ -\frac{\sqrt{\delta}}{\epsilon}(I - (I + \epsilon\sqrt{\delta}KM^{-1})^{-1}) & M(M + \epsilon\sqrt{\delta}K)^{-1} \end{bmatrix} \\
&= \begin{bmatrix} (M + \epsilon\sqrt{\delta}K)^{-1}(I - M(M + \epsilon\sqrt{\delta}K)^{-1})) & \frac{\epsilon}{\sqrt{\delta}}(I - M(M + \epsilon\sqrt{\delta}K)^{-1}))(M + \epsilon\sqrt{\delta}K)^{-1} \\ +(M + \epsilon\sqrt{\delta}K)^{-1} & \\ -\frac{\sqrt{\delta}}{\epsilon}(M + \epsilon\sqrt{\delta}K)^{-1}(I - M(M + \epsilon\sqrt{\delta}K)^{-1})) & (M + \epsilon\sqrt{\delta}K)^{-1}M(M + \epsilon\sqrt{\delta}K)^{-1} \end{bmatrix}
\end{aligned}$$

Multiplying a vector $\begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}$ with $\widehat{A}_0^{-1}$ we find

$$\widehat{A}_0^{-1} \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} (M + \epsilon\sqrt{\delta}K)^{-1}(2\mathbf{f} - \mathbf{f}^1) + \frac{\epsilon}{\sqrt{\delta}}(\mathbf{g} - \mathbf{g}^1)(M + \epsilon\sqrt{\delta}K)^{-1} \\ -\frac{\sqrt{\delta}}{\epsilon}(M + \epsilon\sqrt{\delta}K)^{-1}(\mathbf{f} - \mathbf{f}^1) + (M + \epsilon\sqrt{\delta}K)^{-1}\mathbf{g}^1 \end{bmatrix}, \tag{5.24}$$

Algorithm 4 LPU factorization

1. Compute $\mathbf{b}_1 = \frac{\sqrt{\delta}}{\epsilon}\mathbf{f} + \mathbf{g}$
2. Solve $(M + \epsilon\sqrt{\delta}K)\,\mathbf{g}^2 = \mathbf{b}_1$ by PCG with AMG preconditioner
3. Compute $\mathbf{b}_2 = \frac{\sqrt{\delta}}{\epsilon}\mathbf{f} - M\mathbf{g}^2$
4. Solve $(M + \epsilon\sqrt{\delta}K)\,\mathbf{g}^3 = \mathbf{b}_2$ by PCG with AMG preconditioner
5. Compute $\mathbf{x} = \frac{\epsilon}{\sqrt{\delta}}(\mathbf{g}^2 + \mathbf{g}^3)$
6. Set $\mathbf{y} = -\mathbf{g}^3$

where $\mathbf{f}^1 = M(M + \epsilon\sqrt{\delta}K)^{-1}d$ and $\mathbf{g}^1 = M(M + \epsilon\sqrt{\delta}K)^{-1}c$.
Hence,

$$\begin{aligned}
\widehat{A}_0^{-1}\begin{bmatrix}\mathbf{f}\\ \mathbf{g}\end{bmatrix} &= \begin{bmatrix} \frac{\epsilon}{\sqrt{\delta}}(M + \epsilon\sqrt{\delta}K)^{-1}[\frac{\delta}{\sqrt{\epsilon}}(2\mathbf{f} - \mathbf{f}^1) + (\mathbf{g} - \mathbf{g}^1)] \\ -(M + \epsilon\sqrt{\delta}K)^{-1}[\frac{\sqrt{\delta}}{\epsilon}(\mathbf{f} - \mathbf{f}^1) - \mathbf{g}^1] \end{bmatrix} \\
&= \begin{bmatrix} \frac{\epsilon}{\sqrt{\delta}}(M + \epsilon\sqrt{\delta}K)^{-1}[\frac{\delta}{\sqrt{\epsilon}}\mathbf{f} + \mathbf{g}] + \frac{\epsilon}{\sqrt{\delta}}(M + \epsilon\sqrt{\delta}K)^{-1}[\frac{\delta}{\sqrt{\epsilon}}\mathbf{f} - M\mathbf{g}^2] \\ -(M + \epsilon\sqrt{\delta}K)^{-1}[\frac{\sqrt{\delta}}{\epsilon}\mathbf{f} - M\mathbf{g}^2] \end{bmatrix} \\
&= \begin{bmatrix} \frac{\epsilon}{\sqrt{\delta}}(\mathbf{g}^3 + \mathbf{g}^2) \\ -\mathbf{g}^3 \end{bmatrix},
\end{aligned} \tag{5.25}$$

where $\mathbf{g}^2 = (M + \epsilon\sqrt{\delta}K)^{-1}(\frac{\sqrt{\delta}}{\epsilon}\mathbf{f} + \mathbf{g})$ and $\mathbf{g}^3 = (M + \epsilon\sqrt{\delta}K)^{-1}(\frac{\sqrt{\delta}}{\epsilon}\mathbf{f} - M\mathbf{g}^2)$.
From (5.25), we see that there are only two solutions with the matrix $M + \epsilon\sqrt{\delta}K$ required in order to solve a system with $\widehat{A}_0$. Compared with Algorithm 3, which involves three solutions per each inner iteration, this implementation is cheaper. Moreover, due to the properties of the mass matrix M and the stiffness matrix K, systems with $M + \epsilon\sqrt{\delta}K$ can be solved very efficiently by the Preconditioned Conjugate Gradient (PCG) method, using, for example, an Algebraic Multigrid (AMG) preconditioner. The implementation of the improved solution with $\widehat{A}_0$ is described in Algorithm 4.

We have shown that A_0 is a high quality preconditioner to A, and $\widehat{A}_0$ is high quality preconditioner to A_0. Moreover, systems with $\widehat{A}_0$ can be solved very efficiently, see Algorithm 4. Therefore, we choose to precondition A directly by $\widehat{A}_0$. This leads to a simpler and less expensive procedure, see Fig 7.1.
The choice of $\widehat{A}_0$ as a preconditioner to A is justified by the following analysis. Consider the generalized eigenvalue problem

$$A\mathbf{v} = \lambda\widehat{A}_0\mathbf{v} \quad \Leftrightarrow \quad (\widehat{A}_0 - A)\mathbf{v} = (1 - \lambda)\widehat{A}_0\mathbf{v} \tag{5.26}$$

Inserting $\widehat{A}_0$ and A into the equation (5.26), we get

$$\begin{bmatrix} 0 & J \\ 0 & 2\epsilon\sqrt{\delta}K - \Delta t W \end{bmatrix}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\end{bmatrix} = (1 - \lambda)\begin{bmatrix} M & -\epsilon^2 K \\ \delta K & M + 2\epsilon\sqrt{\delta}K \end{bmatrix}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\end{bmatrix},$$

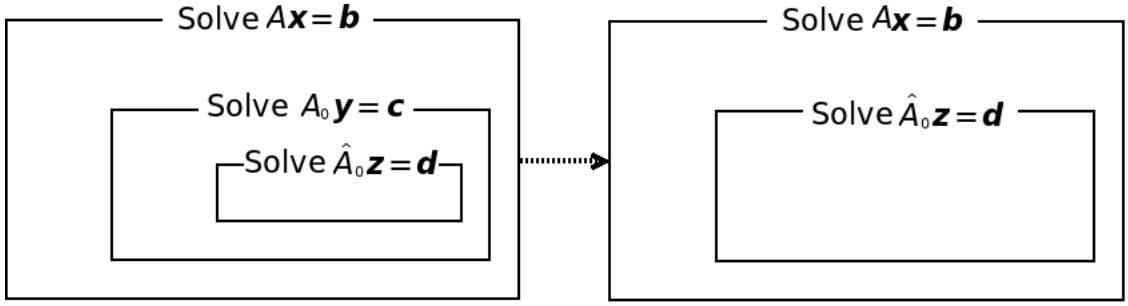


Figure 5.1: Simplification of the solution procedure

It holds

$$\begin{aligned}(1-\lambda)(M\mathbf{v}_1-\epsilon^2K\mathbf{v}_2) &= J\mathbf{v}_2\\ (1-\lambda)(\delta K\mathbf{v}_1+(M+2\epsilon\sqrt{\delta}K)\mathbf{v}_2) &= (2\epsilon\sqrt{\delta}K-\Delta tW)\mathbf{v}_2.\end{aligned} \tag{5.27}$$

Solving the equations, we see that

$$\lambda(\epsilon^2\delta KM^{-1}K+M+2\epsilon\sqrt{\delta}K)\mathbf{v}_2=(\Delta tW+\delta KM^{-1}J+M+\epsilon^2\delta KM^{-1}K)\mathbf{v}_2.$$

Multiplying M^{-1} on both sides, we obtain

$$\lambda(\epsilon^2\delta(M^{-1}K)^2+I+2\epsilon\sqrt{\delta}M^{-1}K)\mathbf{v}_2=(\Delta tM^{-1}W+\delta M^{-1}KM^{-1}J+I+\epsilon^2\delta(M^{-1}K)^2)\mathbf{v}_2.$$

Consider the matrix Q, defined as

$$\begin{aligned}Q &\equiv (I+\epsilon^2\delta(M^{-1}K)^2+2\epsilon\sqrt{\delta}M^{-1}K)^{-1}(\Delta tM^{-1}W+\delta M^{-1}KM^{-1}J+I+\epsilon^2\delta(M^{-1}K)^2)\\ &= (I+\epsilon\sqrt{\delta}(M^{-1}K))^{-2}(\Delta tM^{-1}W+\delta M^{-1}KM^{-1}J+I+\epsilon^2\delta(M^{-1}K)^2).\end{aligned}$$

Since the stiffness matrix K is symmetric positive semi-definite and the mass matrix M is symmetric positive definite, for any vector $\mathbf{y}$, it holds

$$\frac{\mathbf{y}^T(I+\epsilon^2\delta(M^{-1}K)^2+2\epsilon\sqrt{\delta}M^{-1}K)\mathbf{y}}{\mathbf{y}^T(2\epsilon\sqrt{\delta}M^{-1}K)\mathbf{y}}=\frac{\mathbf{y}_0^T(I+BB+2B)\mathbf{y}_0}{\mathbf{y}_0^T(2B)\mathbf{y}_0},$$

where $B=\epsilon\sqrt{\delta}M^{-\frac{1}{2}}KM^{-\frac{1}{2}}$ and $\mathbf{y}_0=M^{\frac{1}{2}}\mathbf{y}$.
Using Cauchy-Schwartz inequality, we obtain

$$0\le 2\mathbf{y}_0^TB\mathbf{y}_0\le\mathbf{y}^T\mathbf{y}_0+(B\mathbf{y}_0)^T(B\mathbf{y}_0)=\mathbf{y}_0^T\mathbf{y}_0+\mathbf{y}_0^TB^TB\mathbf{y}_0=\mathbf{y}_0^T\mathbf{y}_0+\mathbf{y}_0^TBB\mathbf{y}_0.$$

It follows

$$\frac{\mathbf{y}_0^T(I+BB+2B)\mathbf{y}_0}{\mathbf{y}_0^T(2B)\mathbf{y}_0}\ge\frac{\mathbf{y}_0^T(2B+2B)\mathbf{y}_0}{\mathbf{y}_0^T(2B)\mathbf{y}_0}=2.$$

As a consequence, $\|(I+\epsilon\sqrt{\delta}(M^{-1}K))^2\|\ge 4\epsilon\sqrt{\delta}\|(M^{-1}K))\|$ and $\|I+\epsilon^2\delta(M^{-1}K)^2\|\ge 2\epsilon\sqrt{\delta}\|(M^{-1}K))\|$.

Recall that $\delta = \Delta t\omega$, then,

$$\begin{aligned}\|Q\| &\leq \frac{\|I+\epsilon^2\Delta t\omega(M^{-1}K)^2\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} + \Delta t\frac{\|M^{-1}W\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} + \Delta t\omega\frac{\|M^{-1}K\|\,\|M^{-1}J\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} \\ &\leq \frac{\|2\epsilon\sqrt{\Delta t\omega}M^{-1}K\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} + \Delta t\frac{\|M^{-1}W\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} + \Delta t\omega\frac{\|M^{-1}K\|\,\|M^{-1}J\|}{\|(I+\Delta t\omega\epsilon^2(M^{-1}K))^2\|} \\ &\leq \frac{1}{2} + \frac{\Delta t}{4\sqrt{\Delta t\omega}\epsilon}\frac{\|M^{-1}W\|}{\|(M^{-1}K))\|} + \frac{\Delta t\omega}{4\sqrt{\Delta t\omega}\epsilon}\|M^{-1}J\|\end{aligned} \tag{5.28}$$

Similarly to the analysis for (5.9). For the eigenvalue problem, there holds

$$\text{all eigenvalues } |\lambda| \in [\tfrac{1}{2}, \tfrac{1}{2}+\varsigma], \tag{5.29}$$

where,

(1) for diffusion dominated problems ($Pe = 1$), $\varsigma \to 0$ when $\Delta t < h^2$,

(2) for convection dominated problems ($Pe \gg 1$), $\varsigma \to 0$, when $\Delta t < h$.

Later, via the numerical experiments, we find out that Δt does not need to be chosen smaller than h^2. It turns out that $\Delta t < h/4$ is already small enough to ensure fast convergence. The robustness of the preconditoner $\widehat{A}_0$ is enhanced by this advantage. Within the desired discretisation error bounds, we can chose the time step Δt much more flexible.

Chapter 6

Computer implementation

In this chapter, we discuss the implementation of the solution algorithm we introduced in Chapters 3 - 5. The program is first implemented in Matlab. As a starting point, a 2D Matlab code was used, developed for the numerical experiments in [12]. The existing codes have been rewritten for the 3D case and some relatively coarse resolution test problems have been run in Matlab. However, for higher resolution meshes, the number of mesh points easily reaches a million and more. Therefore, a C++ implementation using the open source finite element library deal.II has been developed and tested.

6.1 The algorithm for solving the C-H equation

To solve the time dependent differential equations (3.1), we first discretize them in space using finite elements. In finite element modeling, the underlying domain is divided into a large number of smaller parts, also called "elements". Elements may be of different shapes and sizes, such as triangles, or quadrilaterals in 2D, and bricks or tetrahedra in 3D. By breaking down the domain into elements, the complicated PDEs are transformed into a finite set of ODEs. We then discretize in time and solve this ODE system numerically using the θ-method. As a result we obtain a nonlinear system to be solved at each time step. To handle the nonlinearity, we use nonlinear solution technique (i.e. Inexact Newton method) to linearize the system. Finally, we use preconditioned iterative method to solve the large sparse linear system. From the above process, we notice that for a fixed mesh, the mass matrices M, the stiffness matrices K, and the convective matrices W are time-independent. They only need to be assembled once in the beginning of the time stepping procedure.
In finite element method, one simple way to create a mesh is to define a coarse mesh and then refine it by the so-called regular refinement. We begin with an initial mesh in the computational domain Ω, and then perform a number of the regular refinement steps. In 3D, one refinement replaces each cube in the initial cube with eight smaller cubes. The mesh size of the refined mesh h is half the size of its parent mesh. In another word,

$h_{refined} = h_{initial}2^{-i}$, where i is the number of refinements.
In general, when solving the Cahn-Hilliard equations in (3.1) there are several important steps:

- Creating a mesh in the computational domain Ω.
- Computing the mass matrices M, the stiffness matrices K, and the convective matrices W.
- Creating a time grid on the interval $(0, T]$.
- At each time step, solving the nonlinear system in (4.1) by the inexact Newton method (i1) or (i2).
- At each inexact Newton iteration, computing the matrix J related to the Jacobian of nonlinear part $f(\mathbf{c})$ in the Cahn-Hilliard equation.

The full algorithms with the inexact Newton method (i1) and (i2) are shown in Algorithm 5 and Algorithm 6 respectively. Below, we describe the program implementation both in Matlab and deal.II.

6.2 The Matlab implementation

In Matlab, we first create a mesh by regular refinements. The most important information related to the mesh, needed for further calculations is the list of nodes with their coordinates, the list of elements, and the consistently ordered list of nodes in every element.

Assemble of the matrices M, K, W, J

Let $\Omega_h = \{\Omega_i, i = 1, 2, \cdots, N_e\}\} \subseteq \Omega$ is the set of all elements in the mesh. For each element, there are eight local basis functions. Therefore, we obtain 8×8 small element matrices in each Ω_i, $M^{(e)}$, $K^{(e)}$, $W^{(e)}$, $J^{(e)}$, with entries

$$M_{ij}^{(e)} = \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x}, \quad K_{ij}^{(e)} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\mathbf{x}, \quad W_{ij}^{(e)} = \int_\Omega (\mathbf{u} \cdot \nabla\varphi_j) \, \varphi_i \, d\mathbf{x}$$

$$J_{ij}^{(e)} = \int_\Omega (3(\sum_{l=1}^{N} c_l \varphi_l)^2 - 1)\varphi_j \varphi_i d\mathbf{x}$$

The integrals are usually evaluated by some quadrature. In this project, we use an 8-point Gaussian quadrature rule which takes the form

$$\int_{\Omega_i} f d\mathbf{x} \approx \sum_{j=1}^{8} \omega_j f(\mathbf{q}_j),$$

Algorithm 5 Full algorithm with the inexact Newton method (i1)

1. Generate a mesh for Ω and define the corresponding space of continuous piecewise linear functions V_h. Let $\{\varphi_i\{\}_{i=1}^N$ be the basis function for the subspace V_h.
2. Assemble the $N \times N$ mass matrix M, the stiffness matrix K, and the convective matrix W, with entries

$$M_{ij} = \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x}, \; K_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\mathbf{x}, \; W_{ij} = \int_\Omega (\mathbf{u} \cdot \nabla\varphi_j) \, \varphi_i \, d\mathbf{x}$$

3. Create a time grid $0 = t_0 < t_1 < t_2 < ... < t_L = T$ on $0 < t < T$ with time steps $\Delta t_k = t_k - t_{k-1}$, $k = 1, 2 \cdots, L$
4. Choose initial condition $\mathbf{d_0}$ and $\mathbf{c_0}$
5. **for** $k = 1, 2 \cdots, L$ **do**
6. Choose the starting guess $\mathbf{x}^{k,0} = \left[\mathbf{d}^{k-1}, \mathbf{c}^{k-1}\right]^T$ and a desired Newton tolerance ε
7. **for** $q = 1, 2 \cdots$ **do**
8. Assemble the $N \times N$ Jacobian matrix J of the nonlinear term $f(\mathbf{c})$, and the $2N \times 1$ right hand side of Newton equation $\mathbf{b}$, with entries

$$J_{ij}^{k,q} = \int_\Omega (3(\sum_{l=1}^{N} c_l \varphi_l)^2 - 1)\varphi_j \varphi_i d\mathbf{x},$$

$$Jrhs_{ij}^{k,q} = \int_\Omega ((\sum_{l=1}^{N} c_l \varphi_l)^3 - (\sum_{l=1}^{N} c_l \varphi_l))\varphi_j \varphi_i d\mathbf{x},$$

$$\{b_i^{k,q}\}_{i=1}^N = M_{ij}\mathbf{d_i}^{k,q} - Jrhs_{ij}^{k,q} - \epsilon^2 K_{ij}\mathbf{c_i}^{k,q},$$

$$\{b_i^{k,q}\}_{i=N+1}^{2N} = \omega\Delta t_k K_{ij}\mathbf{d}^{k,q} + M_{ij}\mathbf{c}^{k,q} + \Delta t_k W_{ij}\mathbf{c}^{k,q} - M_{ij}\mathbf{c}^{k-1}.$$

9. Set $A = \begin{bmatrix} M & -J(\mathbf{c}) - \epsilon^2 K \\ \omega\Delta t_k K & M + \Delta t_k W \end{bmatrix}$
10. choose a starting guess $\mathbf{y_0}$ for preconditioned GMRES method
11. Compute $\mathbf{r} = \mathbf{b} - A\mathbf{y}_0$
12. Solve $\mathbf{r}_0 = \widehat{A}_0^{-1} r$ via the computational steps in Algorithm 4, and set $v_1 = r_0/\|r_0\|$
13. **for** $m = 1, 2 \cdots$ until convergence **do**
14. Compute $Z_m = Av_m$
15. Solve $\omega = \widehat{A}_0^{-1} Z_m$ via the computational steps in Algorithm 4
16. Compute Q_m as in Algorithm 2
17. Compute $Y_m = \text{argmin}_Y \|\beta e_1 - \overline{H_m} Y\|$
18. Set $y_m = y_0 + Q_m Y_m$
19. **end for**
20. Set the correction $\delta^{k,q} = -y_m$
21. Update the solution guess $\mathbf{x}^{k,q+1} = \mathbf{x}^{k,q} + \delta^{k,q}$
22. **if** $|\delta^{k,q}| < \varepsilon$ **then**
23. Stop
24. **end if**
25. **end for**
26. Set the solution vector $\mathbf{c^k}$ and $\mathbf{d^k}$, with entries $\mathbf{c}_i^k = \{\mathbf{x}_i^{k,q}\}_{i=N+1}^{2N}$ and $\mathbf{d}_i^k = \{\mathbf{x}_i^{k,q}\}_{i=1}^N$
27. **end for**

Algorithm 6 Full algorithm with the inexact Newton method (i2)

1. Generate a mesh for Ω and define the corresponding space of continuous piecewise linear functions V_h. Let $\{\varphi_i\{\}_{i=1}^N$ be the basis function for the subspace V_h.
2. Assemble the $N \times N$ mass matrix M, the stiffness matrix K, and the convective matrix W, with entries

$$M_{ij} = \int_\Omega \varphi_i \, \varphi_j \, d\mathbf{x},\ K_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\mathbf{x},\ W_{ij} = \int_\Omega (\mathbf{u} \cdot \nabla\varphi_j) \, \varphi_i \, d\mathbf{x}$$

3. Create a time grid $0 = t_0 < t_1 < t_2 < ... < t_L = T$ on $0 < t < T$ with time steps $\Delta t_k = t_k - t_{k-1}$, $k = 1, 2 \cdots, L$
4. Choose initial condition $\mathbf{d_0}$ and $\mathbf{c_0}$
5. **for** $k = 1, 2 \cdots, L$ **do**
6. Choose the starting guess $\mathbf{x}^{k,0} = \left[\mathbf{d}^{k-1}, \mathbf{c}^{k-1}\right]^T$ and a desired Newton tolerance ε
7. **for** $q = 1, 2 \cdots$ **do**
8. Assemble the $N \times N$ Jacobian matrix J of the nonlinear term $f(\mathbf{c})$, and the $2N \times 1$ right hand side of Newton equation $\mathbf{b}$, with entries

$$J_{ij}^{k,q} = \int_\Omega (3(\sum_{l=1}^{N} c_l\varphi_l)^2 - 1)\varphi_j\varphi_i d\mathbf{x},$$

$$Jrhs_{ij}^{k,q} = \int_\Omega ((\sum_{l=1}^{N} c_l\varphi_l)^3 - (\sum_{l=1}^{N} c_l\varphi_l))\varphi_j\varphi_i d\mathbf{x},$$

$$\{b_i^{k,q}\}_{i=1}^N = M_{ij}\mathbf{d_i}^{k,q} - Jrhs_{ij}^{k,q} - \epsilon^2 K_{ij}\mathbf{c_i}^{k,q},$$

$$\{b_i^{k,q}\}_{i=N+1}^{2N} = \omega\Delta t_k K_{ij}\mathbf{d}^{k,q} + M_{ij}\mathbf{c}^{k,q} + \Delta t_k W_{ij}\mathbf{c}^{k,q} - M_{ij}\mathbf{c}^{k-1}.$$

9. Solve $\widehat{A}_0\delta^{k,q} = \mathbf{b}$ via the computational steps in Algorithm 4
10. Update the solution guess $\mathbf{x}^{k,q+1} = \mathbf{x}^{k,q} + \delta^{k,q}$
11. **if** $|\delta^{k,q}| < \varepsilon$ **then**
12. Stop
13. **end if**
14. **end for**
15. Set the solution vector $\mathbf{c^k}$ and $\mathbf{d^k}$, with entries $\mathbf{c}_i^k = \{\mathbf{x}_i^{k,q}\}_{i=N+1}^{2N}$ and $\mathbf{d}_i^k = \{\mathbf{x}_i^{k,q}\}_{i=1}^N$
16. **end for**

where $\mathbf{q}_j$ are the quadrature nodes in Ω_i, and ω_j-the corresponding quadrature weights. In order to apply the quadrature rules, it is necessary to evaluate the basis functions $\varphi_1, \varphi_2, \cdots, \varphi_8$ and their gradients $\nabla\varphi_i$ over each finite element Ω_i. In practice, the basis functions are not computed on Ω_i. The usual way to define shape functions on a mesh is to view each element Ω_i as the image of a reference element $R = [-1, 1]^3$. In this way the relatively simple basis functions $\gamma_1, \gamma_2, \cdots, \gamma_8$ and their gradients $\nabla\gamma_i$ on the reference element can be computed once and then mapped on each original element by the corresponding coordinate transformation. Denote the transformation from R to Ω_i by F.

$$\mathbf{u} = (s, t, w) \in R = [-1, 1]^3 \quad \longleftrightarrow \quad \mathbf{z} = (x, y, z) \in \Omega_i, \quad \mathbf{z} = F(\mathbf{u}).$$

The Jacobian matrix of this transformation is

$$\Gamma(\mathbf{u}) = F'(\mathbf{u}) = \begin{bmatrix} \frac{\partial F_1}{\partial s}(s,t,w) & \frac{\partial F_1}{\partial t}(s,t,w) & \frac{\partial F_1}{\partial w}(s,t,w) \\ \frac{\partial F_2}{\partial s}(s,t,w) & \frac{\partial F_2}{\partial t}(s,t,w) & \frac{\partial F_2}{\partial w}(s,t,w) \\ \frac{\partial F_3}{\partial s}(s,t,w) & \frac{\partial F_3}{\partial t}(s,t,w) & \frac{\partial F_3}{\partial w}(s,t,w) \end{bmatrix} = (\nabla\gamma)Coord,$$

where the 3×8 matrix $\nabla\gamma$ denotes the gradients of the reference basis functions, and the 8×3 matrix $Coord$ contains the coordinates of eight vertices in Ω_i.
The rule for changing variables in an integral is as follows

$$\int_{\Omega_i} f(\mathbf{z})d\mathbf{z} = \int_{R_i} f(F(\mathbf{u}))|det(\Gamma(\mathbf{u}))|d\mathbf{u}\,.$$

Since $\varphi_i(F(u)) = \gamma_i(u)$, applying the above result and Gaussian quadrature rule to the problem of computing mass matrix M, we get

$$\int_{\Omega_i} \varphi_i\,\varphi_j = \int_{R_i} \kappa\gamma_i\,\gamma_j \approx \kappa \sum_{k=1}^{n} \omega_k \gamma_i(\mathbf{q}_k)\,\gamma_j(\mathbf{q}_k)$$

where $\kappa = |det(\Gamma(\mathbf{u}))|$. As we mentioned before, the matrix J of the nonlinear term $f(\mathbf{c})$ is very similar to the mass matrix M. Applying the same rules, it follows

$$\begin{aligned} \textstyle\int_{K_i}(3(\sum_{l=1}^{8} c_l\varphi_l)^2 - 1)\varphi_j\varphi_i &= \kappa \textstyle\int_{R_i}(3(\sum_{l=1}^{8} c_l\varphi_l)^2 - 1)\gamma_i\,\gamma_j \\ &\approx \kappa \textstyle\sum_{k=1}^{n} \omega_k(3(\sum_{l=1}^{8} c_l\varphi_l(\mathbf{q}_k))^2 - 1)\gamma_i(\mathbf{q}_k)\,\gamma_j(\mathbf{q}_k)\,. \end{aligned}$$

Since both the stiffness matrix K and convective matrix W contain the gradients $\nabla\varphi_i$, it is necessary to know the transformation from $\nabla\varphi_i$ on Ω_i to $\nabla\gamma_i$ on R. According to the chain rule, it follows

$$\begin{aligned} \varphi_i(F(\mathbf{u})) = \gamma_i(\mathbf{u}) \quad &\Rightarrow \nabla\gamma_i(\mathbf{u}) = F'(\mathbf{u})^T\nabla\varphi_i(F(\mathbf{u})) \\ &\Rightarrow \nabla\gamma_i(\mathbf{u}) = \Gamma(\mathbf{u})^T\nabla\varphi_i(F(\mathbf{u})) \\ &\Rightarrow \nabla\varphi_i(F(\mathbf{u})) = \Gamma(\mathbf{u})^{-T}\nabla\gamma_i(\mathbf{u}) \end{aligned}$$

Algorithm 7 Assembly of the matrices M, K, W, J

1. Let N be the number of nodes and $Nbody$ the number of elements in a mesh.
2. Allocate memory for the $N \times N$ sparse matrices and initialize $10 \times 10 \times N$ matrices entries to zero.
3. **for** $K = 1, 2, \cdots, Nbody$ **do**
4. Compute the 8×8 local element matrices M^K, K^K, W^K, J^K by Gaussian quadrature, with entries
   $M_{ij}^K = \kappa \sum_{k=1}^n \omega_k \gamma_i(\mathbf{q}_k)\, \gamma_j(\mathbf{q}_k)$,
   $J_{ij}^K = \kappa \sum_{k=1}^n \omega_k (3(\sum_{l=1}^{8} c_l \varphi_l(\mathbf{q}_k))^2 - 1)\gamma_i(\mathbf{q}_k)\, \gamma_j(\mathbf{q}_k)$,
   $K_{ij}^K = \kappa \sum_{k=1}^n \omega_k (T^{-T}\nabla\gamma_j(\mathbf{q}_k)) \cdot (T^{-T}\nabla\gamma_i(\mathbf{q}_k))$,
   $W_{ij}^K = \kappa \sum_{k=1}^n \omega_k (\mathbf{v} \cdot (T^{-T}\nabla\gamma_j(\mathbf{q}_k)))\gamma_i(\mathbf{q}_k)$
5. Set up the local to global mapping
6. **for** $i = 1, 2, \cdots, 8$ **do**
7. **for** $j = 1, 2, \cdots, 8$ **do**
8. $M = M + M_{ij}^K$, $J = J + J_{ij}^K$, $K = K + K_{ij}^K$, $W = W + W_{ij}^K$
9. **end for**
10. **end for**
11. **end for**

Therefore, the integrals of the stiffness matrix are approximated as

$$\int_{\Omega_i} \nabla\varphi_j \cdot \nabla\varphi_i = \int_{R_i} \kappa(\Gamma^{-T}\nabla\gamma_j) \cdot (\Gamma^{-T}\nabla\gamma_i) \approx \kappa \sum_{k=1}^{n} \omega_k (\Gamma^{-T}\nabla\gamma_j(\mathbf{q}_k)) \cdot (\Gamma^{-T}\nabla\gamma_i(\mathbf{q}_k))\,.$$

In the same way, we get the convective matrix W

$$\int_{\Omega_i} (\mathbf{v} \cdot \nabla\varphi_j)\, \varphi_i = \kappa \int_{R_i} (\mathbf{v} \cdot (\Gamma^{-T}\nabla\gamma_j))\gamma_i \approx \kappa \sum_{k=1}^{n} \omega_k (\mathbf{v} \cdot (\Gamma^{-T}\nabla\gamma_j(\mathbf{q}_k)))\gamma_i(\mathbf{q}_k)\,.$$

After assembly of the local element matrices, we need to map the local node numbers to he global node numbers. This is done by cycling the index i and j over each element while adding the entries of the local element matrices on K_i into their appropriate positions in the global matrices. We summarize this assembly technique in Algorithm 7.

Assemble the vector b

The right hand side vector of Newton method $\mathbf{b}$ is assembled using the same technique as the assembly of the matrices, which is breaking into a sum of integrals over the finite elements K_i. On each element K_i we get a 16×1 local element vector b^K, with entries

$$\{b^K\}_{i=1}^{8} = M_{ij}^K \mathbf{d_i}^t - Jrhs_{ij}^K - \epsilon^2 K_{ij}^K \mathbf{c_i}^t, \{b^K\}_{i=9}^{16} = \omega\Delta t_k K_{ij}^K \mathbf{d_i}^t + M_{ij}^K \mathbf{c_i}^t + \Delta t_k W_{ij}^K \mathbf{c_i}^t - M_{ij}^K \mathbf{c_i}^{t-1}$$

where $Jrhs_{ij}^K = \kappa \sum_{k=1}^n ((\sum_{l=1}^8 c_l^t \varphi_l)^3 - (\sum_{l=1}^8 c_l^t \varphi_l))\varphi_j(\mathbf{q}_k))\varphi_i(\mathbf{q}_k))$, M_{ij}^K, K_{ij}^K and W_{ij}^K are the local element matrices on element K_i.

6.3 The C++ implementation using the library deal.II

Deal.II is a finite element differential equations analysis library. The library uses advanced object-oriented and data encapsulation techniques. It provides all tools needed for simulations with the finite element method. In particular, the strict separation of meshes, finite element spaces and linear algebra classes allows for a very modular approach in programming applications built on deal.II, and to combine the provided functionality in many different ways, suiting the particular needs of the applications. The main structure of this program is as follows:

- Set up a mesh and the finite element space: *make_grid_and_dofs ()*
- Assemble a discrete linearized system of equations: *assemble_constant_matrix ()* and *assemble_system ()*
- Solve this linearized system: *solve()*
- Output the solution: *output_results()*

The member functions

make_grid_and_dofs (): This is what one could call a pre-processing function. As its name suggests, the first thing we need to do is to generate the mesh in the computation domain Ω and associate degree of freedom numbers to each vertex. Next we enumerate all the degrees of freedom and set up matrix and vector objects to hold the sparse system data structure. Enumerating is done by using DoFHandler::distribute_ dofs(). We use the FE_ Q class and set the polynomial Q1 in the constructor to associates one degree of freedom with each vertex. We also set up a block sparsity pattern by creating a temporary block structure in order to allow several matrices to use the same sparsity pattern.

assemble_constant_matrix (): In this function, we assemble the time-independent matrices, the mass matrix M, the stiffness matrix K, and the convective matrix W. The same as in Matlab, the general approach to assemble matrices is to loop over all elements, and each element compute by quadrature formula, then mapping to the global matrices. To assemble the local matrices we need the values of the shape functions at the locations of the quadrature points on the real element. Both the finite element shape functions as well as the quadrature points are defined on the reference element. In deal.II there is a class called FEValues which is used to map data from the reference element to the real element. If given one instance of three objects (i.e. finite element, quadrature, and mapping objects),

this class provides the values and gradients of shape functions at quadrature points on a real element. Using this class, we can easily compute matrix M, K, and W.

assemble_system (): In this function, we assemble the time-dependent Jacobian matrix J of the nonlinear term $f(\mathbf{c})$ and the right hand side $\mathbf{b}$ of the Newton equation the same way as in assemble_ constant_ matrix ().

solve(): At each time step, this nonlinear solver builds Newton loops to linearize the nonlinear equations. At each Newton iteration, it solves the linearized equations by preconditioned GMRES method, or directly by the linear solver which is built in the class LinearSolvers. The algorithm of the linear solver is introduced in the last chapter in the Algorithm 4. The preconditioner $\widehat{A}_0$ is implemented as a separate routine, which is called within each GMRES iteration.

output_results(): This is the last part of the finite element program, which is to write the solution to a file.

Chapter 7

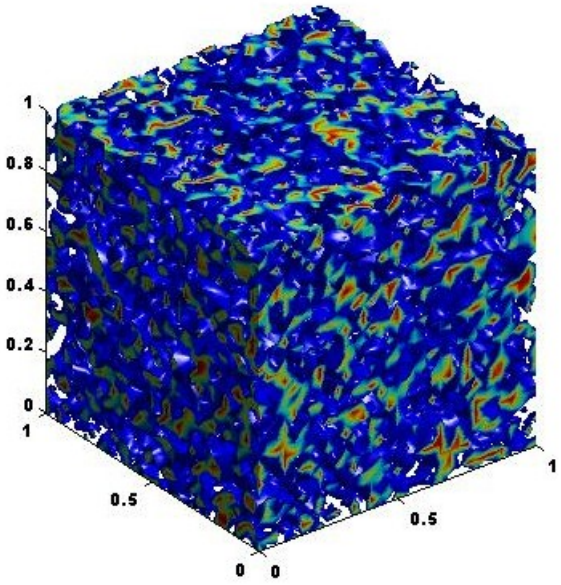
Numerical experiments

We use the following two problems to evaluate the proposed preconditioning techniques for the three-dimensional Cahn-Hilliard equation:

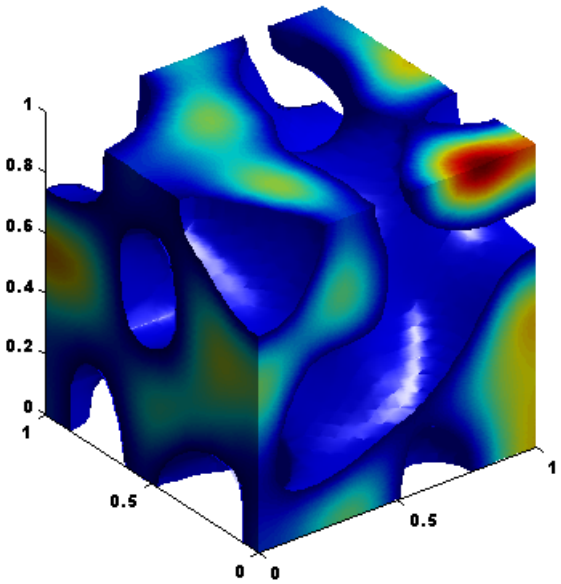
Problem 1: Only diffusion
We consider the system in $\Omega = [0,1] \times [0,1] \times [0,1]$ with parameters $\omega = 1, \epsilon = 0.0625$ and $u = (0,0,0)$. The process of phase separation and coarsening takes place only due to diffusion. Figure 7.1 illustrates the evolution of a binary mixture in time.

Problem 2: Diffusion combined with convection
We consider the system in $\Omega = [-0.5, 0.5] \times [0,1] \times [0,1]$ with parameters $\omega = 1/300, \epsilon = 0.1$ and $u = (1,0,0)$, which is referred to as horizontal wind. The process of phase separation and coarsening takes place due to both diffusion and convection. The initial condition is assumed to be $C_0 = -tanh(10x_1)$. Figure 7.2 illustrates the evolution of the binary mixture in time.

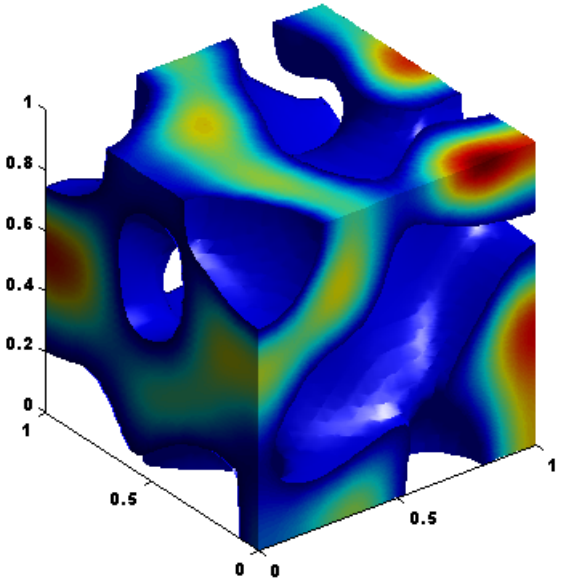
The numerical experiments are done on a cluster with HP SL170h G6 compute servers. Each compute server has two Intel Xeon 5520 Quad core (Nehalem 2.26 Ghz, 8MB cache) processors and 24 GB memory and 250 GB local disk (per server).
In deal.II, there are two libraries: one for the debug mode and one for optimized mode. We use the optimized version of the library, which does not contain the safety checks and is compiled with aggressive compiler optimizations (C++ compiler flag with $-O2$; C compiler flag with $-O3$). The resulting executables are smaller and run between 2 and 10 times faster than the debug executables.
As is mentioned before, in this case, since we need thin interfaces, higher resolution is required in the simulation process. We need to refine the initial mesh 4 or 5 times. The stopping criterion for the Newton iterations at each time step is always taken to be $||\Delta \mathbf{x}^{k,s}|| < 10^{-6}$. In Matlab, the iterative method used to solve the arising system is the Generalised Conjugate Gradient - Minimal Residual (GCG-MR) method, and the solution process is stopped when the norm of the residual is reduced by a factor 10^{-6} or the norm itself is smaller than 10^{-12}. In deal.II, the iterative method used to solve the



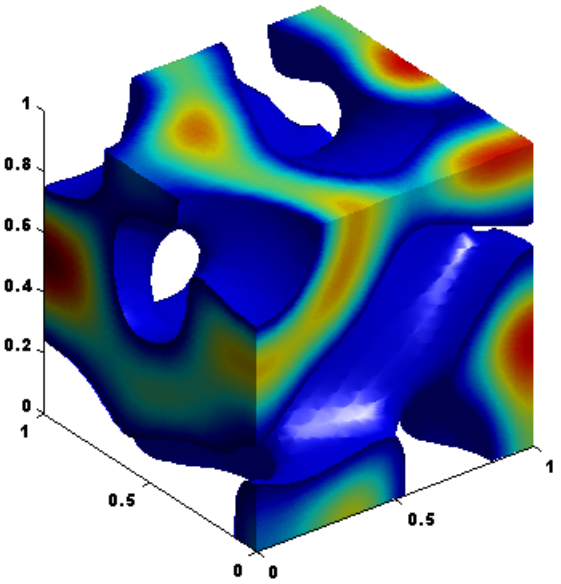
(a) random



(b) $t_{next} = 0.01875$

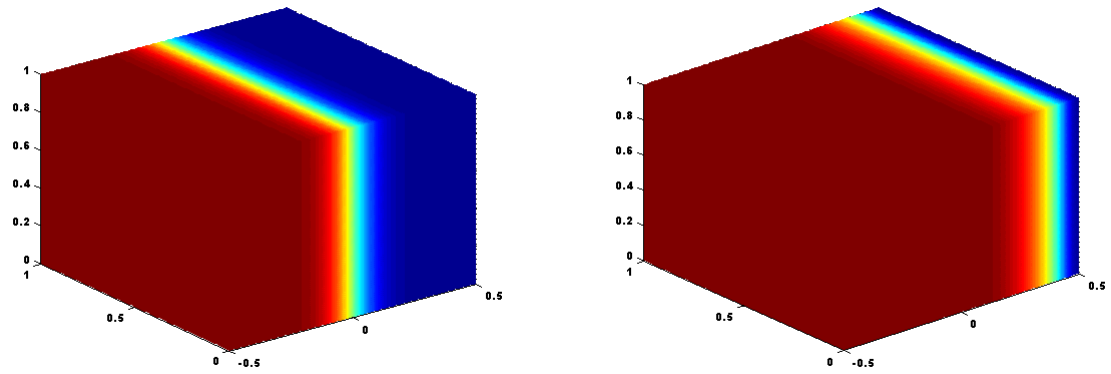


(c) $t_{next} = 0.0375$



(d) $t_{next} = 0.065625$

Figure 7.1: Problem 1: only diffusion



(a) initial convection (b) $conv_t = 0.4687$

Figure 7.2: Problem 2: convection and diffusion

arising systems is GMRES with left preconditioning. Here, the residuals are absolute. We choose the stopping criterion $||\widehat{A}_0^{-1}(\mathbf{b} - A\mathbf{x})|| \leq max\{10^{-12}, 10^{-6}||\widehat{A}_0^{-1}(\mathbf{b} - A\mathbf{x})||\}$.

The numerical tests in Tables 7.1-7.7 are performed both in Matlab and deal.II. Here, we monitor the convergence of the nonlinear method, as well as that of various combinations of inner solvers and their influence on the number of the nonlinear iterations. We report results, averaged over five time steps. Each table cell contains two or three integer digits of the form N_1/N_2 or $N_1/N_2/N_3$, where N_1 denotes the average number of Newton iterations per time step, N_2 is the average number of GCG-MR iterations or GMRES iterations per Newton iteration and N_3, whenever present, shows the average number of AMG-preconditioned conjugate gradient (PCG) iterations or the standard unpreconditioned conjugate gradient (CG) iterations to solve iteratively systems with $M + \epsilon\sqrt{\delta}K$.
Tables 7.1 to 7.3 (Problem 1) and Tables 7.5 to 7.7 (Problem 2) illustrate the numerical performance of the inexact Newton method (i1) in Algorithm 5. We use $\widehat{A}_0$ as a preconditioner for A. The algorithm for solving $\widehat{A}_0$ is the one, presented in Algorithm 4. First we test the convergence when systems with $M + \epsilon\sqrt{\delta}K$ are solved via direct method (Tables 7.1 and 7.5) and next when $M + \epsilon\sqrt{\delta}K$ is solved by CG in Matlab without preconditioner, and next when $M + \epsilon\sqrt{\delta}K$ is solved by PCG with AMG precondioner HSL-MI20. The stopping criterion for the two inner solvers (both CG and PCG) is taken as 10^{-3} in Matlab. In deal.II, $M + \epsilon\sqrt{\delta}K$ is solved by CG method without preconditioner, and the stopping criterion is taken as the norm of the residual reduced by a factor 10^{-3}.
We also test the inexact Newton method (i2) where A is directly replaced by $\widehat{A}_0$ in Algorithm 6. The results are presented in Table 7.4 and 7.8, where we show the number of nonlinear iterations only.
The numerical tests in Table 7.9 are performed in Deal.II for Problem 1, for time step $\Delta t = h^2$. The elapsed CPU time, indicated in the table is to account for the time for one time step. The result shown in Figure 7.3 illustrates that the elapsed CPU time of both inexact Newton method (i1) and (i2) increases with the problem size almost linearly. The

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	> 50	> 50	8 / 41	3 / 12	2 / 10
71874	> 50	> 50	3 / 11	2 / 10	2 / 9
549250	> 50	> 50	3 / 8	3 / 7	3 / 7

Table 7.1: Problem 1, no convection: A is preconditioned with $\widehat{A}_0$

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	> 50	> 50	8 / 44 / 14	3 / 12 / 14	2 / 11 / 13
71874	> 50	> 50	3 / 11 / 21	3 / 9 / 17	3 / 8 / 15
			Deal.II		
9826	> 50	> 50	8/ 45/ 12	4/ 17 / 12	4 / 14 / 11
71874	> 50	> 50	4 / 21 / 19	4 / 14 / 16	4 / 11 / 13
549250	> 50	> 50	4 / 15 / 32	4 / 13 / 24	4 / 11 / 16

Table 7.2: Problem 1, no convection: A is preconditioned with $\widehat{A}_0$, CG applied to the matrix $M + \epsilon\sqrt{\beta}K$ with no preconditioner

problem size increases with a factor 8, and the elapsed CPU time per time step increases with a factor ≈ 11 for the method (i1) and a factor ≈ 8 for the method (i2). We also see that the method (i2) entails more nonlinear iterations (see Table Table 7.4 and 7.8), however it has a lower computational cost than (i1). The larger the size of the matrix (i.e. the more the refinement) is, the more remarkable the advantage of the method (i2) is.
The numerical results are in agreement with the theoretical analysis introduced in previous chapters. For the diffusion dominated Problem 1, where the Peclet number is equal to one, the numerical results are even better than we anyalyize and estimate in theory. As is stated before, in order to obtain high quality of the preconditioner A_0, the time step size Δt should be chosen significantly smaller than h, in that case $\Delta t < h^2$. The numerical results show that $\Delta t \leq h/4$ is already small enough to converge. For the convection dominated Problem 2, the numerical results confirm the theoretical analysis (i.e. the time step size Δt can be of order $O(h)$). We also notice that both inner solvers for $M + \epsilon\sqrt{\delta}K$ do not influence the outer convergence of both the nonlinear and the linear solution method.

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
9826	> 50	> 50	12 / 58 / 3	2 / 13 / 3	2 / 11 / 3
71874	> 50	> 50	3 / 11 / 3	2 / 10 / 3	2 / 9 / 3
549250	> 50	> 50	3 / 8 / 3	3 / 7 / 3	3 / 7 / 3

Table 7.3: Problem 1, no convection: A is preconditioned with $\widehat{A}_0$, AMG used for the matrix $M + \epsilon\sqrt{\beta}K$

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	> 50	> 50	> 50	35	23
71874	> 50	> 50	47	21	20
			Deal.II		
9826	> 50	> 50	> 50	37	27
71874	> 50	> 50	50	25	20
549250	> 50	> 50	28	22	18

Table 7.4: Problem 1, no convection: A is replaced by $\widehat{A}_0$ (method (i2))

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	4 / 10	4 / 9	4 / 7	3 / 8	3 / 7
71874	4 / 7	4 / 6	4 / 6	4 / 6	3 / 6
549250	4 / 6	4 / 6	4 / 5	3 / 6	3 / 6

Table 7.5: Problem 2, convection-diffusion: A is preconditioned with $\widehat{A}_0$

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	4 / 10 / 10	4 / 8 / 9	4 / 7 / 9	3 / 8 / 10	3 / 7 / 11
71874	4 / 9 / 11	4 / 7 / 10	4 / 6 / 10	3 / 7 / 8	3 / 6 / 8
			Deal.II		
9826	6 / 20 / 9	6 / 16 / 8	6 / 13 / 9	5 / 12 / 9	5 / 11 / 12
71874	6 / 17 / 10	5/ 15 / 9	5 / 13 / 10	4 / 13 / 9	4 / 12 / 8
549250	5 / 16 / 14	5 / 14 / 13	5/ 13 / 12	4 / 13 / 9	4 / 11 / 7

Table 7.6: Problem 2, convection-diffusion: A is preconditioned with $\widehat{A}_0$, CG applies to the matrix $M + \epsilon\sqrt{\beta}K$ with no preconditioner

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
9826	4 / 10 / 2	4 / 8 / 2	4 / 7 / 3	3 / 7 / 3	3 / 7 / 3
71874	4 / 9 / 3	4 / 7 / 3	4 / 6 / 3	3 / 7 / 2	3 / 6 / 3
549250	4 / 7 / 3	4 / 6 / 3	4 / 6 / 3	4 / 6 / 3	3 / 6 / 3

Table 7.7: Problem 2, convection-diffusion: A is preconditioned with $\widehat{A}_0$, AMG used for the matrix $M + \epsilon\sqrt{\beta}K$

	Δt				
Size	h	$h/2$	$h/4$	$h/10$	h^2
			Matlab		
9826	34	25	23	21	20
71874	28	25	23	21	18
			Deal.II		
9826	31	23	19	17	16
71874	28	21	19	16	14
549250	22	20	18	15	10

Table 7.8: Problem 2, convection-diffusion: A is replaced by $\widehat{A}_0$ (method (i2))

Elapsed CPU time	Size			
(seconds)	9826	71874	549250	4293378
inexact Newton (i1)	1.77873	12.0412	136.344	1474.05
inexact Newton (i2)	1.58076	9.89750	79.3449	660.035

Table 7.9: Problem 1, no convection: A is solved by methods (i1) and (i2)

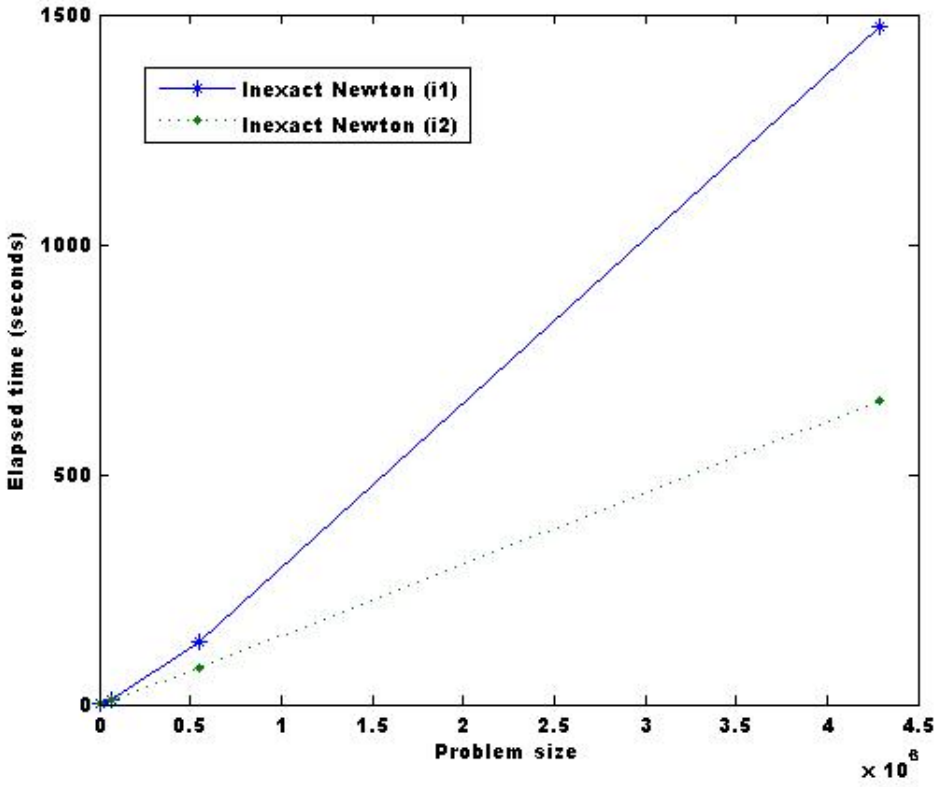


Figure 7.3: Elapsed CPU time per time step vs. problem size

Chapter 8

Conclusions

In this work we study preconditioning techniques for the iterative solution of two-phase flow problems, based on Cahn-Hilliard equation. The original form of Cahn-Hilliard equation is a fourth order parabolic partial differential equation, however, in this study it is reformulated as a coupled nonlinear system of two second order equations, one of which is time-dependent.
We consider three-dimensional domains, discretized using cube meshes and conforming trilinear basis functions. At each time step we use inexact Newton method to solve the arising nonlinear systems. Then, the main computational effort is related to solution of the resulting linear system with a Jacobian matrix A, that is non-symmetric and of very large size, especially in 3D. In order to meet high resolution requirements for the phase-field method, the number of mesh nodes are normally up to millions.
Here, in order to solve the linear system with A, we consider two different inexact Newton methods, that both result in efficient solution algorithms. In the first one (method (i1)), instead of solving the system with A exactly, we apply a preconditioned iterative method. To construct the preconditioner, we utilize the natural block form of the matrix. In this work, we propose to use a high quality approximation, $\widehat{A}_0$, of the original system matrix A, that turns out to be an optimal preconditioner. Its use leads to both optimal order convergence rate and optimal order computational complexity. The latter is due to the special structure of $\widehat{A}_0$.
In the second inexact Newton method (method (i2)), we consider that the solution of the system with the Jacobian matrix is directly replaced by the solution of the system with the matrix $\widehat{A}_0$. In this case, at each Newton step, the process of solving the linear system by preconditioned iterative method, reduces to directly solving a linear system. Since $\widehat{A}_0$ is a very close approximation of the Jacobian matrix, for small enough Δt, we can expect that the second inexact Newton method (i2) is a feasible alternative to the first inexact Newton method (i1). Numerical experiments show that the number of nonlinear iterations do raise a lot, however we save computational cost by avoiding the iterative solution method.
The main contributions of this work are the following:

- We have derived estimates for the preconditioned system for the presented precondi-

tioners for three-dimensional problems. The results are similar to the two-dimensional case. The numerical experiments confirm the efficiency of the preconditioner in 3D. Moreover, for problems with both cases of Peclet numbers, the preconditioned methods show optimal behavior for time steps Δt of order $O(h)$. The numerical experiments show that $\Delta t \leq h/4$ is already small enough to ensure fast convergence. Within the desired discretisation error bounds, we can chose the time steps Δt much flexible. The robustness of the preconditoner $\widehat{A}_0$ is enhanced by this advantage.

- We have observed that the quality of the preconditioner $\widehat{A}_0$ is not affected much if the subsystems with its blocks are not solved accurately. We can use CG or PCG, and both lead to efficient methods.

- In the case of 3D, the simulations have very big computer system demands. The matrices are of huge size and the program implementation is crusial. We have implemented the methods in C++ using deal.II. Thus, we were able to perform experiments for problems with size more than 4 million.

Bibliography

[1] O. Axelsson, M. Neytcheva, Operator splittings for solving nonlinear, coupled multiphysics problems with an application to the numerical solution of an interface problem. TR 2011-009, Institute for Information Technology, Uppsala University, April 2011. Submitted.

[2] O. Axelsson, A. Kucherov, Real valued iterative methods for solving complex symmetric linear systems. *Numerical Linear Algebra with Applications* 7 (2000), 197–218.

[3] O. Axelsson and G. Lindskog, On the Eigenvalue Distribution of a Class of Preconditioning Methods, *Numer. Math.*, 48 (1986), 479-498.

[4] O. Axelsson and H. Lu, On Eigenvalue Estimates for Block Incomplete Factorization Methods, *SIAM J. Matrix. Anal. Appl.*, 16 (1995). To appear.

[5] O. Axelsson, High-order methods for parabolic problems, *Journal of Computational and Applied Mathematics*, 1(1) 1975, 5–16.

[6] O. Axelsson, On global convergence of iterative methods, *Iterative solution of nonlinear systems of equations* (1982), 1–19, Springer.

[7] B. L. Beegle, M. Modell, R. C. Reid, Legendre transforms and their application in thermodynamics, *AIChE Journal*, 20 (1974): 1194-1200.

[8] A. Berti, I. Bochicchio, A mathematical model for phase separation: a generalized Cahn-Hilliard equation, *Mathematical Methods in the Applied Sciences*, 34 (2011): 1193-1201.

[9] M. Brokate, J. Sprekels, *Hysteresis and Phase Transitions*, Springer, New York, 1996.

[10] D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, Cambridge University Press, 1997.

[11] Petia Boyanova, Minh Do-Quang, Maya Neytcheva, Efficient preconditioners for large scale binary Cahn-Hilliard models. *Technical report / Department of Information Technology*, Uppsala University nr 2011-011, 2011.

[12] Petia Boyanova, Minh Do-Quang, Maya Neytcheva, Solution methods for the Cahn-Hilliard equation discretized by conforming and non-conforming finite elements. *Technical report / Department of Information Technology*, Uppsala University nr 2011-004, 2011.

[13] V. E. Badalassi, H. D. Ceniceros, S. Banerjee, Computation of multiphase systems with phase field models, *Journal of computational physics*, 32(2003): 371-397, Elsevier.

[14] R. E. Bank, D. J. Rose, Analysis of a multilevel iterative method for nonlinear finite element equations, *Mathematics of Computation*, 39(160) 1982, 453–465.

[15] J. W. Cahn, J. E. Hilliard, Free energy of a nonuniform system I, J. *Chem. Phys.* 28 (1958): 258.

[16] J.C. Cahn, On spinodal decomposition, *Acta Metall*, 9 (1961) 795-801.

[17] J. Du, B. Fix, J. Glimm, X. C. Jia, X. L. Li, Y. H. Li, and L. L. Wu, A simple package for front tracking, *J. Comput. Phys.*, 213(2) (2006), 613-628.

[18] M. Do-Quang, G. Amberg, The splash of a ball hitting a liquid surface: Numerical sim- ulation of the influence of wetting, *Physics of Fluids*, 2008.

[19] R. S. Dembo, S. C. Eisenstat, T. Steihaug, Inexact newton methods, *SIAM Journal on Numerical analysis* (1982), 400–408, JSTOR.

[20] J. W. Demmel, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics Philadelphia, PA,, USA, 1997.

[21] C.G. Gal, M. Grasselli, Asymptotic behavior of a Cahn-Hilliard-Navier-Stokes system in 2D, *Ann. I. H. Poincare AN* 27 (2010),p. 401436.

[22] M.E. Gurtin, D. Polignone, J. Vials Two-phase binary fluids and immiscible fluids described by an order parameter, *Math. Models Methods Appl. Sci.* 6 (1996): 815-831.

[23] C.W. Hirt, B.D. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, *Journal of Computational Physics* 39 (1981): 201-225.

[24] M. G. Larson, F. Bengzon, *The Finite Element Method: Theory, Implementation, and Practice*, Springer, November 9, 2010.

[25] G.N. Wells, E. Kuhl, K. Garikipati, A discontinuous Galerkin method for the Cahn-Hilliard equation, *Journal of Computational Physics*, volume 218 (2006): 860–877, Elsevier.

[26] A. Novick-Cohen, *The Cahn-Hilliard Equation: From Backwards Diffusion to Surface Diffusion*. Draft. n preparation under contract with Cambridge University Press.
http://www.math.technion.ac.il/ amync/

[27] S. Osher, R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, 2002.

[28] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations", *J. Comput. Physics*, 79(1988): 12-49.

[29] D. Ralph, Global convergence of damped Newton's method for nonsmooth equations via the path search, *Mathematics of Operations Research* (1994), 352–389, JSTOR.

[30] R. Scardovelli, S. Zaleski. Direct numerical simulation of free-surface and interfacial flow. *Annual Review of Fluid Mechanics*, Vol. 31: 567-603.

[31] Y. Sun, C. Beckermann, Sharp interface tracking using the phase-field equation, *Journal of Computational Physics archive*, Volume 220 Issue 2, January, 2007.

[32] Y. Saad, *Iterative Methods for Sparse Linear System*, PWS: New York, 1996.

[33] A.J. Wathen, Realistic eigenvalue bounds for the Galerkin mass matrix, *IMA Journal of Numerical Analysis*, 7 (1987), 449–457.