Project Thesis

Jonathan Ulmer

December 11, 2024

Contents

Abstract

This work shows sensitivity of boundary conditions for two different finite difference approaches to solving the Cahn-Hilliard equation

1 Introduction

This project thesis builds upon the work in our bachelor thesis, by introducing a simple boundary condition approach to a variation of the solver used therein. In Chapter 2 we introduce the Cahn-Hilliard equation in the formulation that we use for this project. This project used a two-dimensional second order version of this CH equation rather than the usual 1D 4th order one, to simplify the numerical implementation. The solver itself builds upon a finite difference discretization of this equation. In Chapter 4, together with the discrete domains, on which we run our numerical solver, we introduce a Jacoby iteration to solve the linear system derived from the aforementioned discretization. The numerical solver in this thesis is GPU accelerated, and the discretization we chose to base our solver on is capable to calculate on all domains as long as a characteristic function is given. Therefore, we introduce two domains, on which we present our findings. The primary goal of this work is then the boundary condition approach in Chapter 3. Conceptionally the boundary condition we introduce simply consists of a constant value added in the linear system to all equations corresponding to grid-cells on the boundary. The actual implementation is capable of doing this for arbitrary domains. We present the results of this method on two Domains, a square one in 5 and a circular domain in 6 where we show the phase field for different boundary conditions which manifest in a variable contact angle of the interface on the boundary mimicking the behavior of hydrophobic/hydrophilic material. While we are unable to provide explicit formulae in relation to the constant, in Chapter 7 we provide numerical insight in this relationship, and a table with precomputed values.

2 Fundamentals

This work concerns itself with boundary conditions on the CH equation. The CH equation is a fourth order differential equation that provides a phasefield ϕ which is used for implicit interface formulation. The Cahn Hilliard equation, in the formulaton we use here, is derived from the **Ginzburg-Landau** energy, an example on how this is done is given by [3].

$$E^{\text{bulk}}[\phi] = \int_{\Omega} \frac{\varepsilon^2}{2} |\nabla \phi|^2 + W(\phi)\, dx, \tag{2.1}$$

There they introduce a chemical potential μ derived as derivative of the **Ginzburg-Landay** energy.

$$\mu = \frac{\delta E_{bulk}(\phi)}{\delta \phi} = -\varepsilon^2 \Delta \phi + W'(\phi) \tag{2.2}$$

where $W(\phi)$ in the energy,is a double well potential. In our case we orient us at the work of[1], where they use

$$W(\phi) = \frac{(1-\phi^2)^2}{4}. \tag{2.3}$$

they Cahn Hilliard equation in this thesis is then given as

$$\begin{aligned} \partial_t \phi(x,t) &= \Delta \mu \\ \mu &= -\varepsilon^2 \Delta \phi + W'(\phi). \end{aligned} \tag{2.4}$$

one thing to note is, that this way of writing the CH equation is secend order two dimensional rather than the one dimensional fourth order often given.

$$\partial_t \phi(\vec{x}, t) = -\Delta(\varepsilon^2 \Delta \phi + W'(\phi)) \tag{2.5}$$

this choice is deliberate, and aligns with the numerical implementation.

2.1 Notation

This project solves the CH equation on a regular rectangular grid with grid-size h. The computational domain, is therefore discretized as

$$\vec{x}_{ij} := \frac{i}{h} * e_1 + \frac{j}{h} e_2 \tag{2.6}$$

where $i, j \in [0, ... N]$ and N is chosen arbitrarily ,such that the resulting rectangle $[0, Nh] \times [0, Nh]$ acts as bounding box of the domain Ω. For our imlpementation we use $N = 256$ as it gives a good compromize bewteen resolution and compute time. We denote a discrete version of the domain Ω_d where

$$\Omega_d := \{x_{ij} | x_{ij} \in \Omega\} \tag{2.7}$$

On this discrete domain our solver calulates solutions for discrete fields

$$\phi_{ij}^n : \Omega_d \times \{0, ... \} \to \mathbb{R}\,, \tag{2.8}$$

$$\phi_{ij} := \phi(\vec{x}_{ij}) \qquad \vec{x}_{ij} \in \Omega_d \tag{2.9}$$

$$\mu_{ij}^n : \Omega_d \times \{0, ... \} \to \mathbb{R}\,, \tag{2.10}$$

$$\mu_{ij} := \mu(\vec{x}_{ij}) \tag{2.11}$$

We use the following differential quotients:

$$D_x f_{i+\frac{1}{2}j} = \frac{f_{i+1j} - f_{ij}}{h} \qquad D_y f_{ij+\frac{1}{2}} = \frac{f_{ij+1} - f_{ij}}{h} \tag{2.12}$$

And define a discrete gradient as.

$$\nabla_d f_{ij} = (D_x f_{i+1j},\ D_y f_{ij+1}) \tag{2.13}$$

See [2] Furthermore, our solver implements the ansatz proposed by the authors [1].

$$\begin{aligned} \frac{\phi_{ij}^{n+1} - \phi_{ij}^n}{\Delta t} &= \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2}}) \\ \mu_{ij}^{n+\frac{1}{2}} &= 2\phi_{ij}^{n+1} - \varepsilon^2 \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1}) + W'(\phi_{ij}^n) - 2\phi_{ij}^n \end{aligned} \tag{2.14}$$

This approach provides a semi implicit time discretization were linear terms are evaluated implicitly and the nonlinear double well potential is evaluated explicitly.

3 Boundary adaptation

The solver from [1], that we use as reference guaranties no flux boundary conditions at a discrete level by setting $\nabla\phi_{ij} = 0$ for $\phi_{ij} \in \partial\Omega_d$ this is done by multiplying with the Characteristic function of Ω_d

$$G_{ij} = \begin{cases} 1\,, x_{ij} \in \Omega \\ 0\,, x_{ij} \notin \Omega \end{cases} \tag{3.1}$$

To accommodate different boundary conditions, we modify $\nabla_d \cdot (G_{ij}\nabla_d\phi_{ij})$ with a constant term C on grid points next to the boundary. To do this, we introduce an boundary field B_{ij} that we add to μ_{ij}. We determine the value of B_{ij} using a centered difference scheme on G

$$B_{ij} = \max\Big(|G_{i+\frac{1}{2}j} - G_{i-\frac{1}{2}j}|, |G_{ij+\frac{1}{2}} - G_{ij-\frac{1}{2}}|\Big) * C \tag{3.2}$$

We present an example on a 32x32 Domain with $C = 1$ of the Boundary field $\mathbf{B}$ for a square domain 3.1 and an example on a circular domain in 3.2. In this Project we use the following adaptation of the discretization from [1].

$$\begin{aligned} \frac{\phi_{ij}^{n+1} - \phi_{ij}n}{\Delta t} &= \nabla_d \cdot (G_{ij}\nabla_d \mu_{ij}^{n+\frac{1}{2}}) \\ \mu_{ij}^{n+\frac{1}{2}} &= 2\phi_{ij}^{n+1} - \varepsilon^2\nabla_d \cdot (G_{ij}\nabla_d\phi_{ij}^{n+1}) + B_{ij} + W'(\phi_{ij}^n) - 2\phi_{ij}^n \end{aligned} \tag{3.3}$$

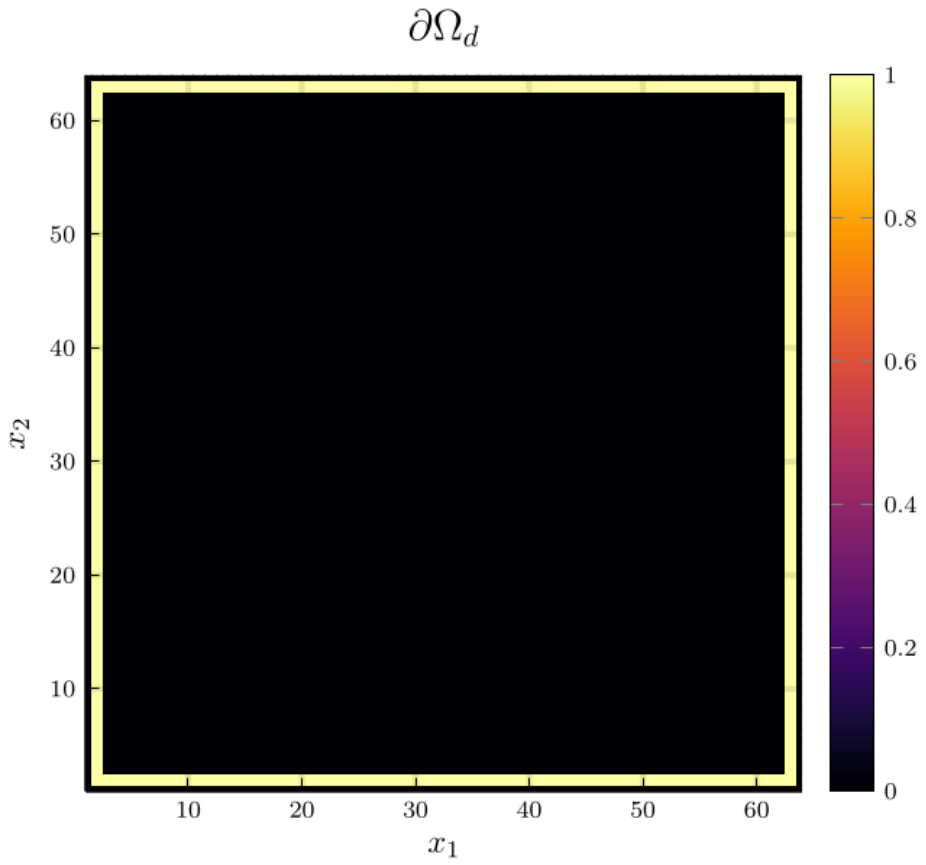


Figure 3.1: visualization of all grid-cells adjacent to the boundary $\partial\Omega_d$ of a square domain

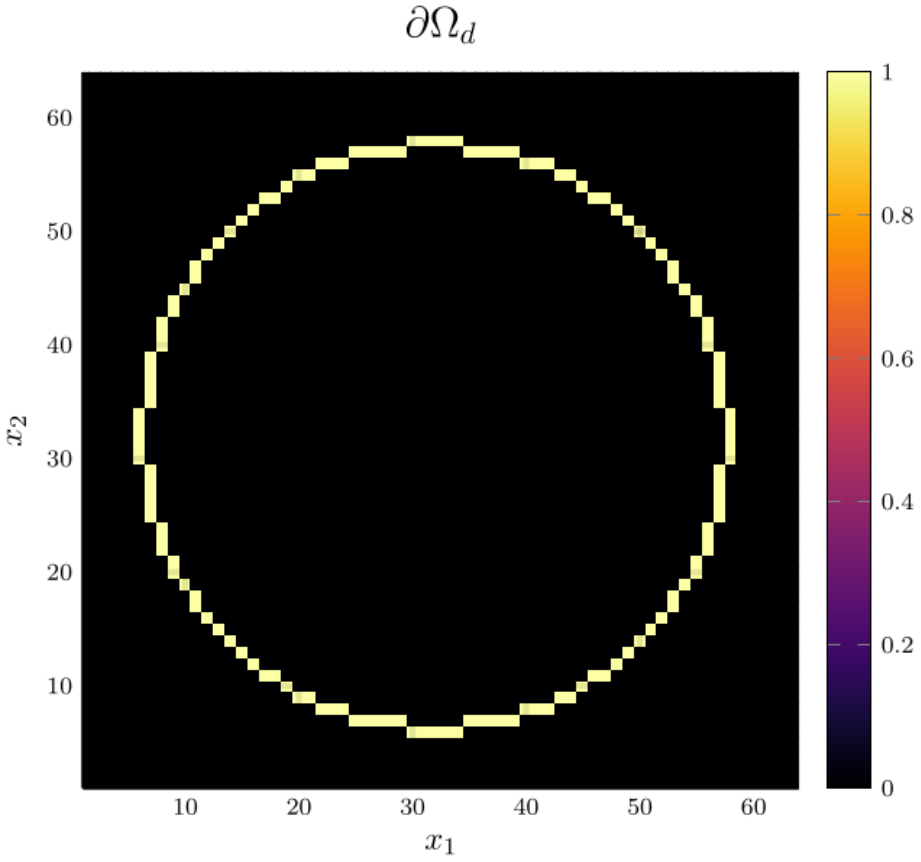


Figure 3.2: visualization of all grid-cells adjacent to the boundary $\partial\Omega_d$ of a circular domain

4 Numerical solver

Contrary to the solver proposed in[2] we do not use a multi-grid Gauss-Seidel Solver to solve the linear system, and use a Jacoby solver instead, since it is easier to paralyze. Similar to [2] we linearize (3.3) to

$$\begin{aligned} \frac{\phi_{ij}^{n+1}}{\Delta t} - \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2}}) &= \frac{\phi_{ij}^{n}}{\Delta t} \\ \mu_{ij}^{n+\frac{1}{2}} - 2\phi_{ij}^{n+1} + \varepsilon^2 \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1}) + B_{ij} &= 2\phi_{ij}^{n} - W'(\phi_{ij}^{n}) \end{aligned} \tag{4.1}$$

After some rearranging we note, that the left-hand side is linear and, the right-hand side is solely dependent on the previous time step. Therefore, this constitutes a linear system, which we solve with a Jacoby method, the element wise formula of which is given as follows: Provided the mth Jacoby iteration has been computed, the $m+1th$ iteration is computed by solving

$$\begin{aligned} \frac{\phi_{ij}^{n+1,m+1}}{\Delta t} - \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2},m+\frac{1}{2}}) &= \frac{\phi_{ij}^{n}}{\Delta t} \\ \mu_{ij}^{n+\frac{1}{2},m} - 2\phi_{ij}^{n+1,m} + \varepsilon^2 \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1,m+\frac{1}{2}}) + B_{ij} &= 2\phi_{ij}^{n} - W'(\phi_{ij}^{n}) \end{aligned} \tag{4.2}$$

For $\phi_{ij}^{n+1,m+1}, \mu_{ij}^{n+\frac{1}{2},m+1}$, where $\nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2},m+\frac{1}{2}})$ and $\nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1,m+\frac{1}{2}})$. Use the results from the previous jacoby step for values off the center. eg.

$$\begin{aligned} \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1,m+\frac{1}{2}}) = &\frac{1}{h^2} (G_{i+\frac{1}{2}j} \phi_{i+1j}^{n+1,m} + G_{i-\frac{1}{2}j} \phi_{i-1j}^{n+1,m} \\ &+ \quad G_{ij+\frac{1}{2}} \phi_{ij+1}^{n+1,m} + G_{ij-\frac{1}{2}} \phi_{ij-1}^{n+1,m}) \\ &- \left(G_{i+\frac{1}{2}j} + G_{i-\frac{1}{2}j} + G_{ij+\frac{1}{2}} + G_{ij-\frac{1}{2}}\right) \phi_{ij}^{n+1,m+1} \end{aligned} \tag{4.3}$$

Our implementation makes use of the julia programming language, to dispatch the solution for each element in paralell on the GPU. The full implementation of the jacoby iteration is given as:

```
@kernel function jacoby!(
    Φ,
    M,
    @Const(Ξ),
    @Const(Ψ),
    @Const(h),
    @Const(ε),
    @Const(Δt),
    @Const(iterations)
)
    I   = @index(Global, Cartesian)
    Id  = oneunit(I)
    Ids = CartesianIndices(M)
    Ix = CartesianIndex(1, 0)
    Iy = CartesianIndex(0, 1)
    if I in (Ids[begin]+Id:Ids[end]-Id)
        g = G(2 * I + Ix, Ids) + G(2 * I + Iy, Ids) + G(2 * I - Ix, Ids) + G(2 * I - Iy,
        ↪  Ids)
        a1 = 1/Δt
        a2 = -1* ε^2/h^2 * g  - 2
        b1 = 1/h^2 * g
        b2 = 1
        for _ = 1:iterations

            Σμ = G(2 * I + Ix, Ids) * M[I+Ix] + G(2 * I + Iy, Ids) * M[I+Iy] + G(2 * I -
            ↪  Ix, Ids) * M[I-Ix] + G(2 * I - Iy, Ids) * M[I-Iy]

            Σϕ = G(2 * I + Ix, Ids) * Φ[I+Ix] + G(2 * I + Iy, Ids) * Φ[I+Iy] +G(2 * I -
            ↪  Ix, Ids) * Φ[I-Ix] +G(2 * I - Iy, Ids) * Φ[I-Iy]

            c1 = Ξ[I] + 1/h^2   * Σμ
            c2 = Ψ[I] - ε^2/h^2 * Σϕ

            # stupid matrix solve
            @inline Φ[I] = (c1*b2 - c2*b1) / (a1*b2 - a2*b1)
            @inline M[I] = (a1*c2 - a2*c1) / (a1*b2 - a2*b1)
            #
            @synchronize()
        end

    end
end
```

5 Numerical evaluation

We set constant values for B_{ij} on the boundary to begin with our evaluations. One may note that C = 0 is equivalent to the no-flux condition of the original solver introduced in the Bachelor thesis. Now, as a preliminary verification step we set C = 0 as our first choice. Consequently, for $C = 0$, the interface lies orthogonal on the boundary (see Fig. 5.1), which we expect for a CH solver with no-flux boundary conditions. For $B_{ij} \in \{-1, 1\}$ we observed behavior connected to hydrophobic / hydrophilic substances on the boundary, where $B_{ij} = 1$ resulted in the one phase pearling off the boundary, while the other seemed attracted. These certainly leads to the apparent contact angles of 180° and 0° respectively. Using $B_{ij} = -1$ results in the opposite behavior.

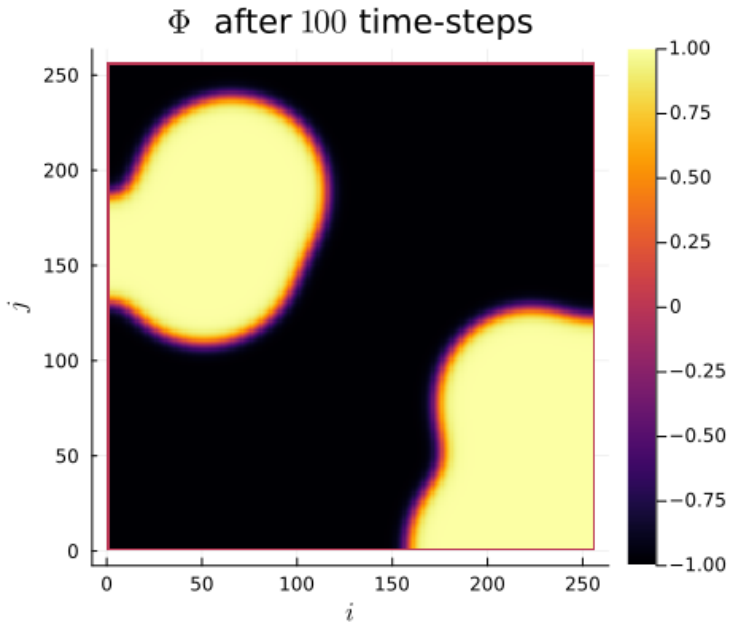


Figure 5.1: phase-field ϕ after 100 time-steps with $C = 0$ emmulating no-flux boundary.

We show, that our solver is stable for values $C \neq 0$. In 5.2 we employ a constant value of $C = 1$ and observe the phase corresponding to $\phi = 1$ puling away from the

boundary. The contact angle between phase 1 and the boundary approaches 180° i.e. the interface runs parallel to the boundary.

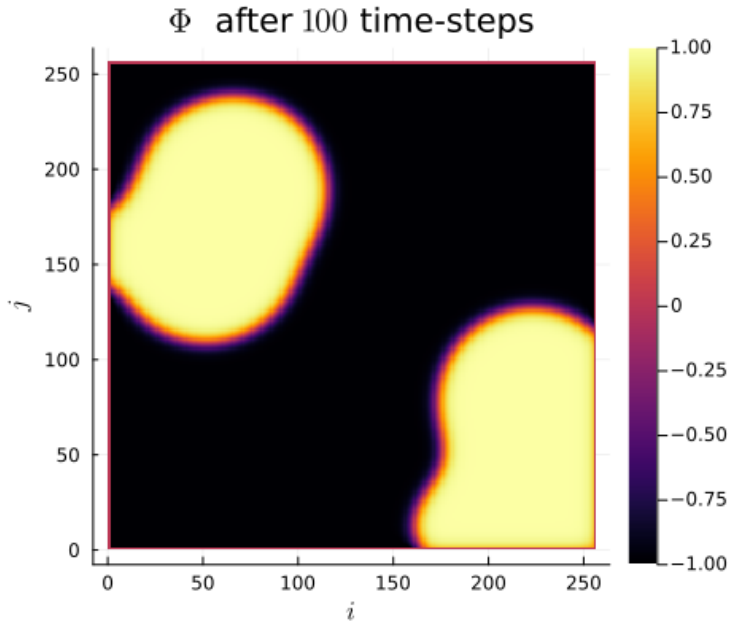


Figure 5.2: phase-field ϕ after 100 time steps with $C = 1$

In 5.3 we try the reverse situation. And we observe corresponding behavior. When using a value of $C = -1$ we observe opposite behavior relative to the case in Fig. 5.2. Where the contact angle on the boundary lies at 0°, the interface runs parallel to the boundary again.

The most interesting behavior are noted for values between $(-1, 1)$, where we observe the contact angle of the interface at the boundary changes from parallel 0° to parallel 180°.

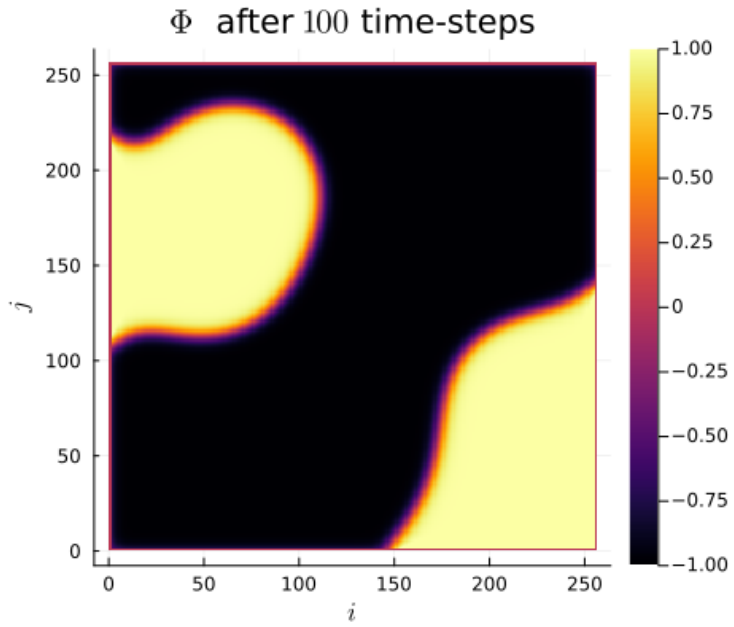
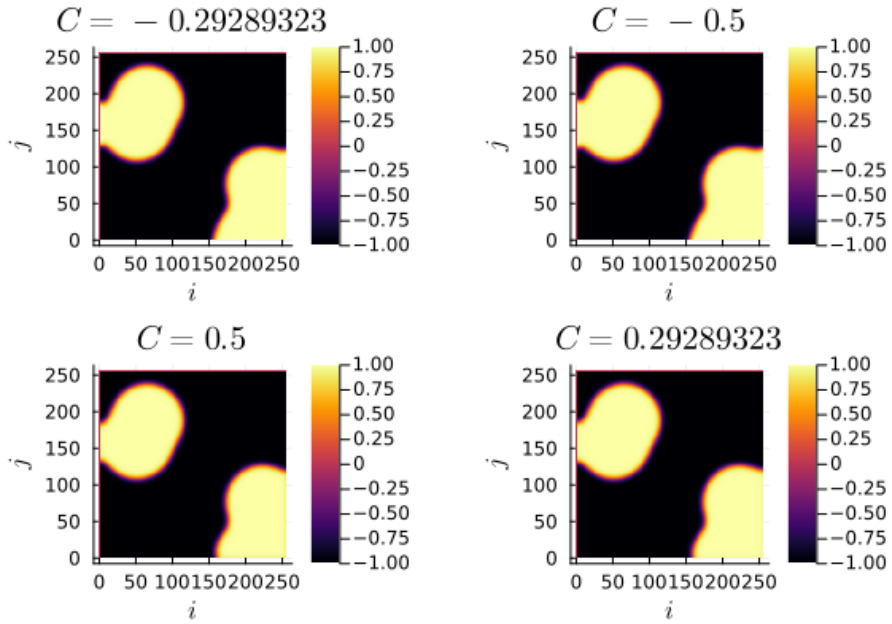
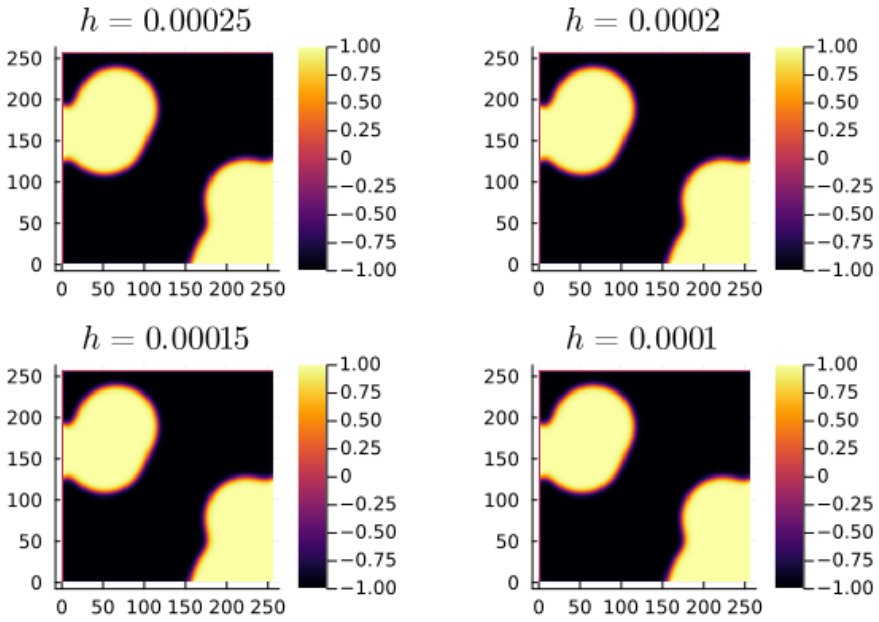


Figure 5.3: phase-field ϕ after 100 time-steps with $C = -1$




```
include("src/solvers.jl")
θ = -5f-1
n = 100
arr = _init()
```

```
d = domain(get_backend(arr) , 256 , size(arr))
d(arr)
h = 25e-5
solution = solve(arr , n , θ=0)
h1 = heatmap(Array(solution) , aspect_ratio=:equal , clims=(-1,1),
↪  lims=(0,size(solution,1)), widen=1.06 , title=L"h=%$h")
h = 20e-5
solution = solve(arr , n , θ=0)
h2 = heatmap(Array(solution) , aspect_ratio=:equal , clims=(-1,1),
↪  lims=(0,size(solution,1)), widen=1.06 , title=L"h=%$h")
h = 15e-5
solution = solve(arr , n , θ=0)
h3 = heatmap(Array(solution) , aspect_ratio=:equal , clims=(-1,1),
↪  lims=(0,size(solution,1)), widen=1.06 , title=L"h=%$h")
h = 10e-5
solution = solve(arr , n , θ=0)
h4 = heatmap(Array(solution) , aspect_ratio=:equal , clims=(-1,1),
↪  lims=(0,size(solution,1)), widen=1.06 , title=L"h=%$h")
plot(h1,h2,h3,h4)
```



6 Numerical evaluation on a circle

The original solver presented in [1] was able to solve the CH equation on arbitrary domains. Since the addition of our boundary function depends solely on the characteristic function of the discrete domain, we are able to use our approach on different Domains, by providing a different characteristic function. We present the results of which in this chapter. To show the behavior of the CH solver in 6.1, we first employ no-flux boundary conditions on a circular domain. We observe the interface perpendicular on the boundary, as we expect.

```
W′ (generic function with 1 method)
```

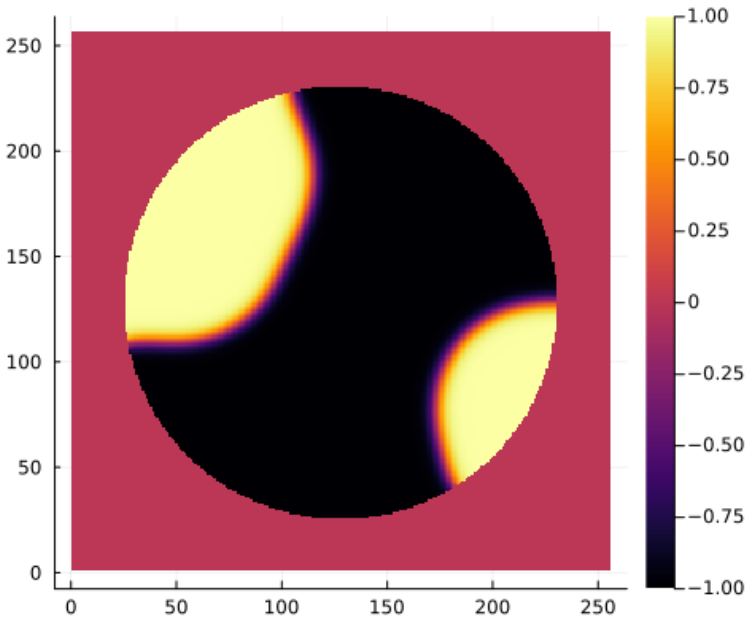


Figure 6.1: ϕ after 100 time steps on a circular domain with no-flux boundary-conditions after 100 time steps on a circular domain with no-flux

The results we observe in 6.2 are similar to the results on a square domain in 5.2. The contact angle is 180° i.e. the interface does not touch the boundary and runs parallel to it.

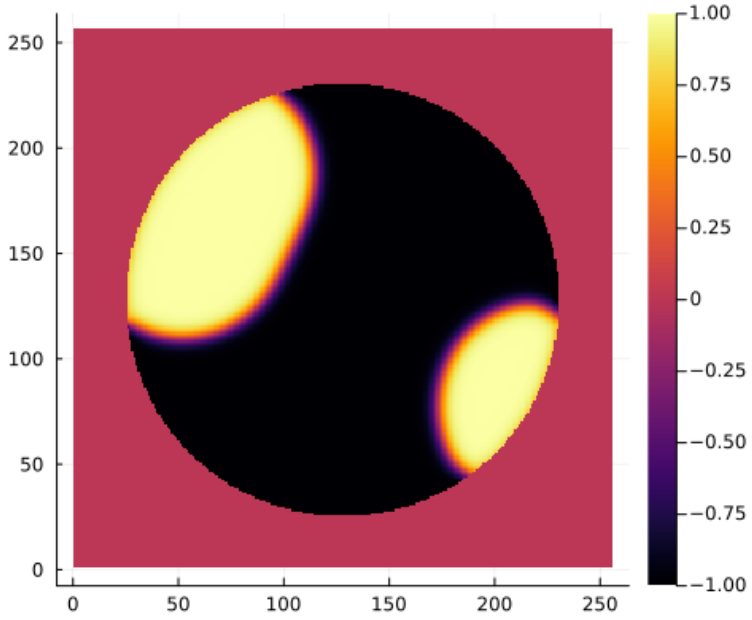


Figure 6.2: phase-field ϕ after 100 time-steps with $C = 1$

The results for $C = -1$ in 6.3 on the circular domain, are similar to the results in 5.3 on the square domain as well, where the interface touches the boundary and runs parallel with a contact angle of 0°.

When evaluating intermediate contact angles in 6.4, the results are similar to the square domain again, however, especially for shallow angles, we observe some artifacts of one phase appearing in places where previously was none. We observe similar behavior on square domains only in the corners, i.e. points where the boundary has high curvature.

When using random initial phase-fields, the results look the comparable to the square domain, and exhibit the for the CH equation expected behavior, whereas time goes on, the many small parts coalesce into larger parts.

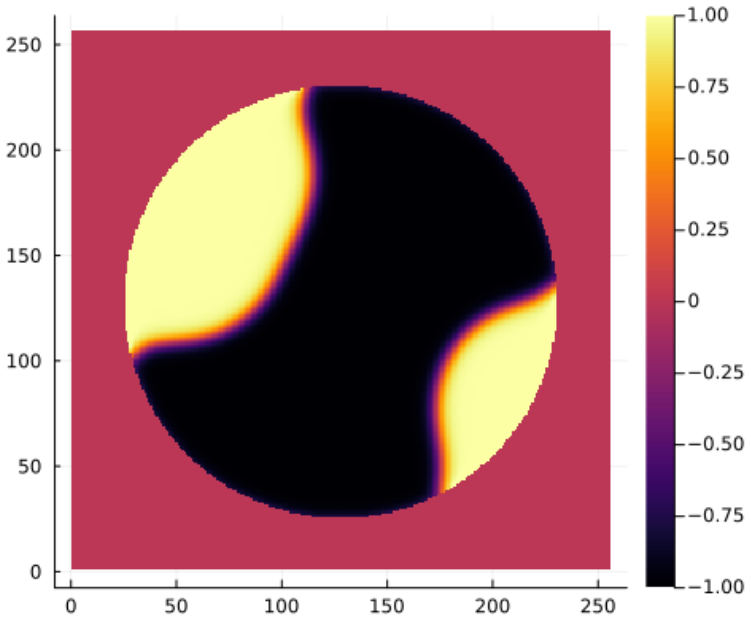


Figure 6.3: phase-field ϕ after 100 time-steps with $C = -1$

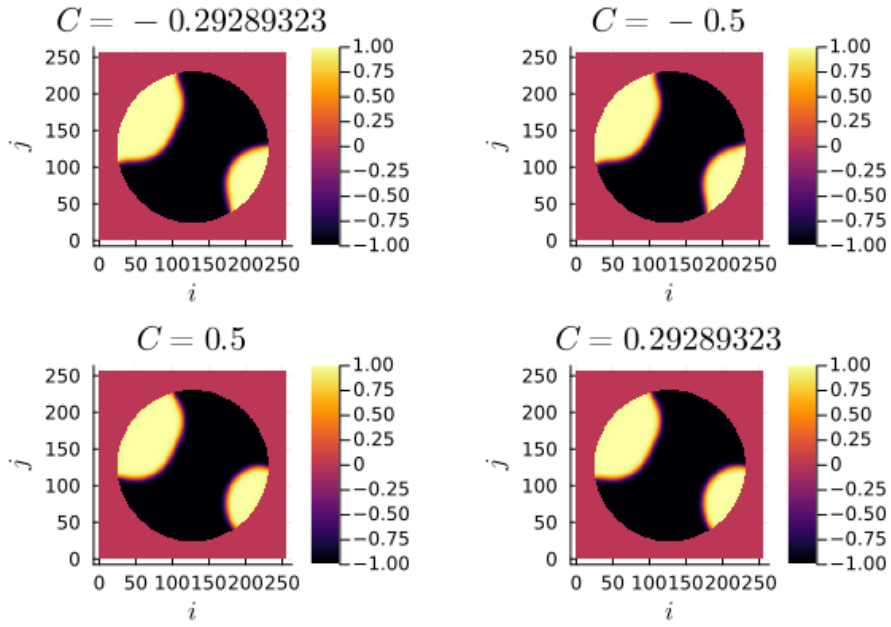


Figure 6.4: phase-field ϕ after 500 time-steps with $C \in \{-1 + \frac{\sqrt{2}}{2}, -0.5, 0.5, 1 - \frac{\sqrt{2}}{2}\}$ on a circular domain.

7 ANGLE

In previous experiments we noted that the angle of the interface changes with different input parameters. While we do not have a mathematical derivation of this relation, we aim to provide numerical insight in this chapter. We calculate this angle using the gradient of the phase-field $\nabla\phi_{ij}$ and the normal of our domains' boundary.

$$\frac{\nabla_d\phi_{ij}\cdot\mathbf{n}_{ij}}{\|\nabla_d\phi_{ij}\|}=\cos(\theta) \qquad \phi_{ij}\in\partial\Omega_d \tag{7.1}$$

For a single point $\vec{x}_{ij}$ on the interface and near the boundary. Since we need a finite difference to evaluate 7.1, we do not select a point directly on the boundary and since we need a point on the interface, where $\nabla\phi_{ij}$ is large, we calculate the angle at

$$P_{ij}=arg\max_{\vec{x}_{ij}}\nabla\phi_{ij} \qquad \phi_{ij}\in\partial\Omega \tag{7.2}$$

```
angle (generic function with 1 method)
```

7.1 CIRCLE

The normal the circular domain in our second example is

$$\mathbf{n}_{ij}:=\mathbf{n}(\vec{x}_{ij})=\frac{\vec{c}-\vec{x}_{ij}}{\|\vec{c}-\vec{x}_{ij}\|} \tag{7.3}$$

Where $\vec{c}$ is the center of the domain.

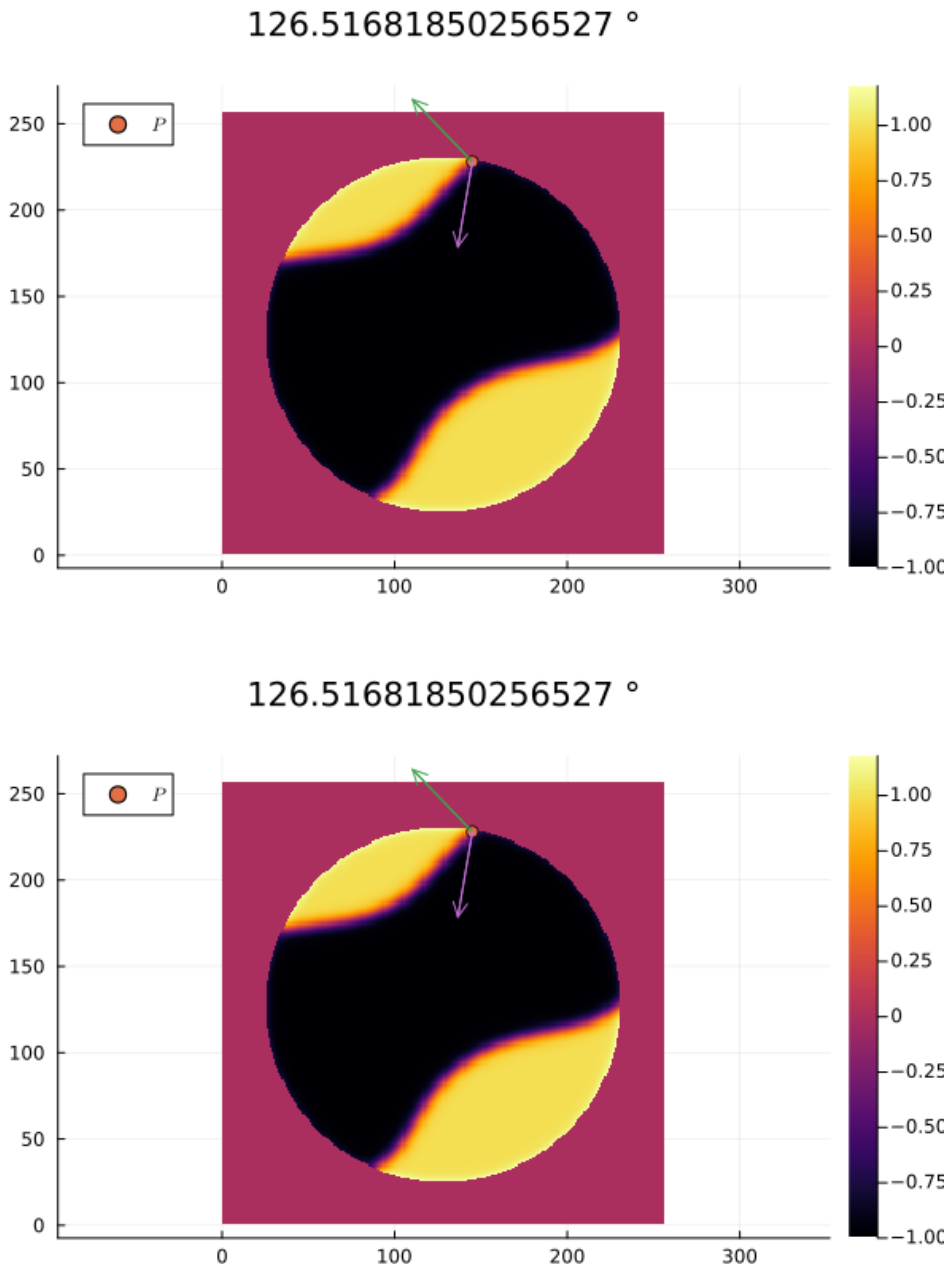


Table 7.1: value for θ and corresponding angle α after 200 time-steps

-0.1	173.49096591056502
-0.095	173.10715739345923
-0.09	172.18364087939332

-0.085	171.54740091859054
-0.08	171.3054040677464
-0.075	171.1455632002332
-0.07	171.02869693204397
-0.065	170.3901810227686
-0.06	170.0449796355949
-0.055	173.27274052589075
-0.05	170.3373892767722
-0.045	168.11953739721892
-0.04	167.41386769034298
-0.035	166.62088559081457
-0.03	164.9014365935728
-0.025	162.8061312020723
-0.02	159.92337650959868
-0.015	155.82320048245077
-0.01	147.4707481361878
-0.005	129.77836444929315
0.0	91.28977210940522
0.005	47.27538237804684
0.01	26.60911004838421
0.015	6.306468865037136
0.02	11.495581754132852
0.025	8.059259459078769
0.03	2.997826637980469
0.035	2.442790881259583
0.04	2.314200756133827
0.045	1.883610279597664
0.05	1.3567468712125557
0.055	0.8024311153759808
0.06	0.5869880299417852
0.065	0.4356076759230446
0.07	0.32719257485287145
0.075	0.03099970458170946
0.08	0.37685133141547533
0.085	0.4151229191583983
0.09	0.7049376111739059

0.095	0.8671639875701463
0.1	1.0282690721714873

8 Summary and outlook

In this project we examined a numerical model o the CH equation, with simple Neumann boundary conditions. We introduced a simplified version of the solver used in [2] and derived from [1]. Which due to GPU acceleration is significantly faster on our available hardware. We have shown a simple Neumann boundary approach that runs stable on both tested domains. The approach introduced by us is able to freely affect the angle of the phase interface on the boundary,

9 References

Bibliography

[1] Jaemin Shin, Darae Jeong, and Junseok Kim. "A conservative numerical method for the Cahn–Hilliard equation in complex domains". In: *Journal of Computational Physics* 230.19 (2011), pp. 7441–7455. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2011.06.009. URL: https://www.sciencedirect.com/science/article/pii/S0021999111003585.

[2] Jonathan Ulmer. "Untitled1". Bachelor Thesis. University of Stuttgart, 3000.

[3] Hao Wu. "A review on the Cahn–Hilliard equation: classical results and recent advances in dynamic boundary conditions". In: *Electronic Research Archive* 30.8 (2022), pp. 2788–2832. DOI: 10.3934/era.2022143. URL: https://doi.org/10.3934%2Fera.2022143.