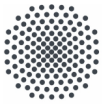


University of Stuttgart
Germany

ians

Bachelor's Thesis

Numerical methods

on the Cahn-Hilliard Equation

Jonathan Ulmer
Matriculation Number: 3545737

Examiner: Prof Rohde i believe
Advisor: Hasel

Institute of Applied Analysis and Numerical Simulation

Completed 01.01.2022

Abstract

This Thesis gives a short overview and derivation for the Cahn-Hilliard Equation. It uses a discretization by the authors [1] as baseline, and expands upon this dicretisation with an elliptical relaxation approach. It introduces evaluation metrics in terms of time , space and subiteration stability and compares the elliptical approach against the baseline. It shows a qualitative succes of the elliptical solver, however it also highlights challanges in numerical stability.

Contents

1 Introduction

The Cahn Hilliard equation is a well known fourth order PDE used in multiphase flow.It is used to couple different phases with a diffuse interface approach, as compared to a sharp interface approach. Therefore it has a smooth transition between phases. The CH equation serves the same purpose, as the second order Allen Cahn equation. However the Allen Cahn equation is not mass conservative. Hence the cahn hilliard equation is used if mass conservation is required. In this thesis we implement numerical solvers for the cahn hilliard equation in the julia programming language. We initially have tried to implement them in python, however the switch to julia has quickly proofed to be valuable since it provided faster runtime , better and easier plots as well as support for matrix algorithms[1]. We did experiment with additional tools such as org-mode that allow for scientific note taking and literate programming. We began writing this thesis with a reproducible research philosophy in mind. Hence we provide all relevant code in the same file as the writing itself. We then use this file to generate exports to html and PDF, as well as extract the code to be used independently. This file is available on our github repository as `Thesis_jl.org`.

[1] Julia provides iteration utilities over n dimensional matricies. Therefore it would technically be possible to write dimension agnostic algorithms. While we used some of this functionality, we did not implement a full n-dimensional algorithm, and only provide a 2D implementation

2 The Cahn-Hilliard equation

The Cahn-Hilliard(CH) equation is a partial differential equation (PDE) that governs the dynamics of a two-phase fluid[2]. The form of the CH equation used in this thesis in the domain $\Omega \times (0,T)\,, \Omega \subset \mathbb{R}^d\,, d \in \mathbb{N}\,, T > 0$.

$$\begin{aligned} \partial_t \phi(x,t) &= \nabla \cdot (M(\phi)\nabla\mu), \\ \mu &= -\varepsilon^2 \Delta\phi + W'(\phi), \end{aligned} \tag{2.1}$$

where the variables $\phi, \mu : \Omega \times (0,T) \to \mathbb{R}^d$ are phase-field variable and chemical potential, ε is a positive constant correlated with interface thickness, $W(\phi)$ is a double well potential and $M(\phi) > 0$ is a mobility coefficient [2]. ϕ is defined in an interval $I = [-1,1]$ and represent the different phases.

$$\phi = \begin{cases} 1 & ,\phi \in \text{phase 1} \\ -1 & ,\phi \in \text{phase 2} \end{cases}$$

In this thesis we assume $M(\phi) \equiv 1$, simplifying the CH equation.

The advantages of the CH approach, as compared to traditional boundary coupling, are for example: "explicit tracking of the interface" [2], as well as "evolution of complex geometries and topological changes [...] in a natural way" [2]. In practice it enables linear interpolation between different formulas on different phases.

2.1 Physical derivation of the CH equation 2.1

2.1.1 The free energy

The authors in [2] define the CH equation using the **Ginzburg-Landau** free energy equation:

$$E^{\text{bulk}}[\phi] = \int_\Omega \frac{\varepsilon^2}{2}|\nabla\phi|^2 + W(\phi)\,dx, \tag{2.2}$$

where $W(\phi)$ denotes the Helmholtz free energy density of mixing [2] that we approximate it in further calculations with $W(\phi) = \frac{(1-\phi^2)^2}{4}$ as in [1] shown in Fig. 2.1.

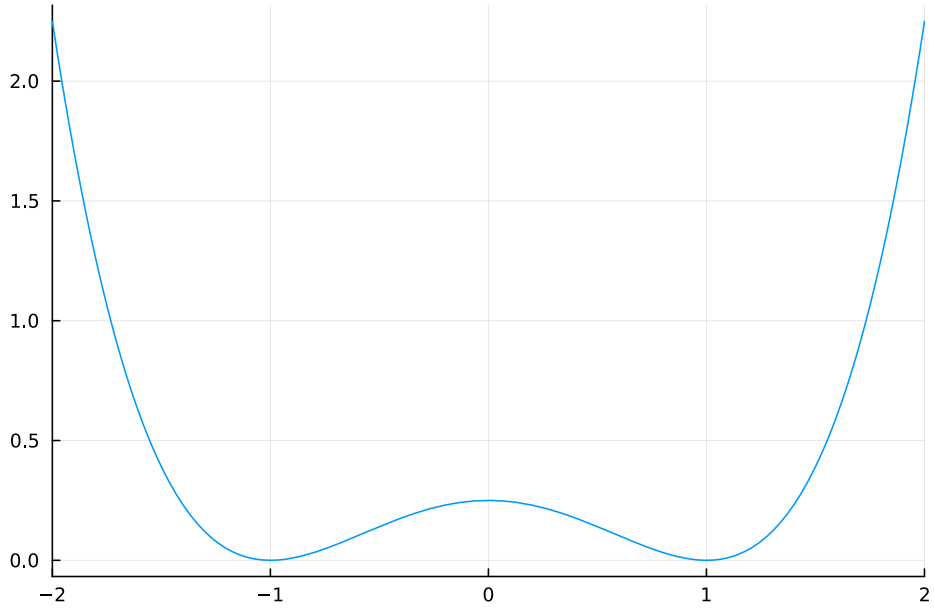


Figure 2.1: Double well potential $W(\phi)$

The chemical potential, μ, then follows as the variational derivation of the free energy 2.2.

$$\mu = \frac{\delta E_{bulk}(\phi)}{\delta \phi} = -\varepsilon^2 \Delta \phi + W'(\phi) \tag{2.3}$$

2.1.2 Derivation of the CH equation from mass balance

The paper [2] states that the observable phase seperation is driven by a diffusion resulting from the gradient in chemical potential μ. The emergent conservative dynamics motivate the following diffusion equation

$$\partial_t \phi + \nabla \cdot \mathbf{J} = 0, \tag{2.4}$$

where $\mathbf{J} = -\nabla\mu$ represents mass-flux . We follow the authors [2] in deriving the CH equation by combining Eq.2.3 and Eq.2.4.

$$\begin{aligned} \implies \partial_t\phi &= -\nabla\cdot\mathbf{J} = \Delta\mu, \\ \mu &= -\varepsilon^2\Delta\phi + W'(\phi) \end{aligned} \tag{2.5}$$

Furthermore the CH equation is mass conservative under homogeneous Neumann boundary conditions, defined as:

$$\begin{aligned} \mathbf{J}\cdot n &= 0 \quad \text{on}\,\partial\Omega\times(0,T), \\ \partial_n\phi &= 0 \quad \text{on}\,\partial\Omega\times(0,T), \end{aligned} \tag{2.6}$$

where n is the outward normal on $\partial\Omega$. To show the conservation of mass we analy se the change in total mass in the domain Ω over time.

$$\begin{aligned} \frac{d}{dt}\int_\Omega \phi\, d\mathbf{x} &= \int_\Omega \frac{\partial\phi}{\partial t}\, d\mathbf{x} \\ &= -\int_\Omega \nabla\cdot\mathbf{J}\, d\mathbf{x} \\ &= \int_{\partial\Omega} \mathbf{J}\cdot n\, d\mathbf{s} \\ &= 0 \qquad\qquad \forall t\in(0,T) \end{aligned} \tag{2.7}$$

In order to show the CH equation's consistency with thermodynamics we take the time derivation of the free energy 2.2 and we show that it decreases in time.

$$\begin{aligned} \frac{d}{dt}E^{bulk}(\phi(t)) &= \int_\Omega (\varepsilon^2\nabla\phi\cdot\nabla\partial_t\phi + W'(\phi)\partial_t\phi)\, dx \\ &= \int_\Omega (\varepsilon^2\nabla\phi + W'(\phi))\partial_t\phi\, dx \\ &= \int_\Omega \mu\partial_t\phi\, dx \\ &= \int_\Omega \mu\cdot\Delta\mu\, dx \\ &= -\int_\Omega \nabla\mu\cdot\nabla\mu\, dx + \int_{\partial\Omega}\mu\nabla\phi_t\cdot n\, dS \\ &\overset{\partial_n\phi=0}{=} -\int_\Omega |\nabla\mu|^2\, dx, \qquad\qquad \forall t\in[0,T) \end{aligned}$$

3 Baseline multi-grid solver

3.1 The discretization of the CH equation:

As baseline for numerical experiments we use a two-grid method based on the finite difference method defined in [1]. Our discretization follows the one taken by the authors in [1]. We discretize our domain Ω to be a Cartesian-grid Ω_d on a square with side-length $N \cdot h$, where N is the number of grid-points in one direction, and h is the distance between grid-points. In all our initial data h is $3 \cdot 10^{-3}$ and $N = 64$. However for stability tests we change h and N.

$$\Omega_d = \{i, j \mid i, j \in \mathbb{N}, i, j \in [2, N+1]\} \tag{3.1}$$

where Ω_d is the discrete version or our domain as shown in 3.1.

We discretize the phase-field ,ϕ, and chemical potential ,μ, into grid-wise functions ϕ_{ij}, μ_{ij}

$$\begin{aligned} \phi_{ij}^n &: \Omega_d \times \{0, \dots\} \to \mathbb{R} \\ \mu_{ij}^n &: \Omega_d \times \{0, \dots\} \to \mathbb{R} \end{aligned} \tag{3.2}$$

Here n denotes the nth time-step , and (i, j) are cartesian indicies on the discrete domain Ω_d. The authors in [1] then use the characteristic function G of the domain Ω to enforce no-flux boundary conditions 2.8.

$$G(x, y) = \begin{cases} 1, & (x, y) \in \Omega \\ 0, & (x, y) \notin \Omega \end{cases}$$

We implement the discrete version of G on Ω_d as follows:

$$G_{ij} = \begin{cases} 1, & (i, j) \in \Omega_d \\ 0, & \text{else} \end{cases}$$

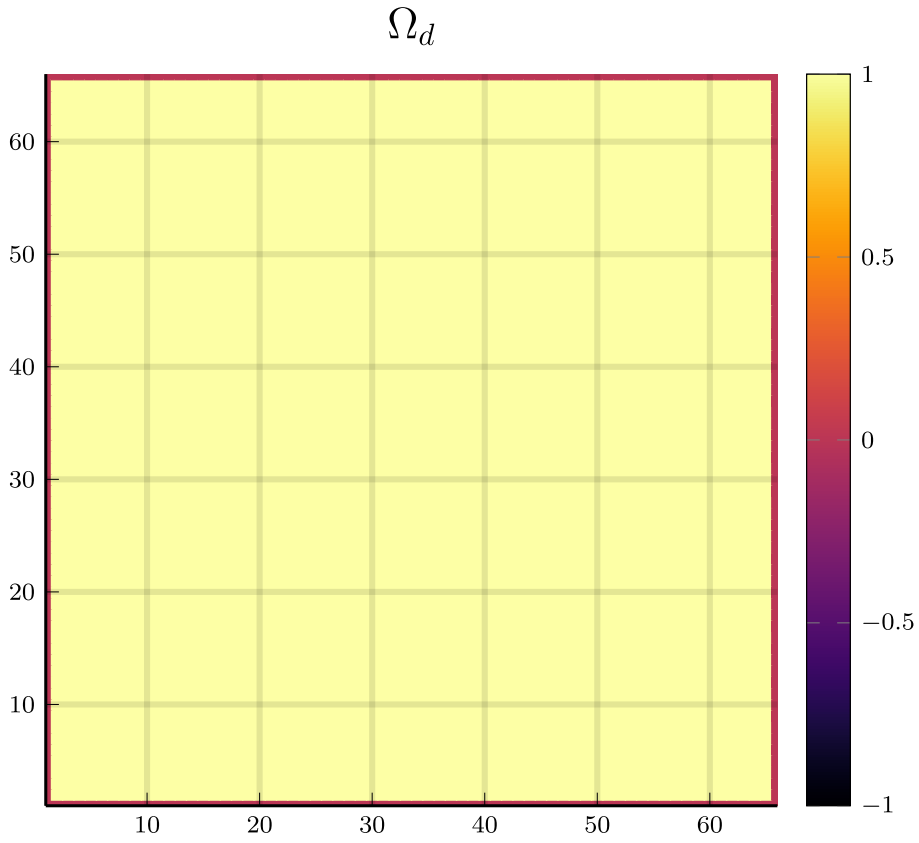


Figure 3.1: Discrete Domain used for most of the experiments in this Thesis

```
function G(i, j, len, width)
    if 2 <= i <= len + 1 && 2 <= j <= width + 1
        return 1.0
    else
        return 0.0
    end
end
```

We then define the discrete derivatives $D_x\phi_{ij},\ D_y\phi_{ij}$ using centred differences:

$$D_x\phi^{n+1,m}_{i+\frac{1}{2}j} = \frac{\phi^{n+1,m}_{i+1j} - \phi^{n+1,m}_{ij}}{h} \qquad D_y\phi^{n+1,m}_{ij+\frac{1}{2}} = \frac{\phi^{n+1,m}_{ij+1} - \phi^{n+1,m}_{ij}}{h} \tag{3.3}$$

We define $D_x\mu^{n+\frac{1}{2},m}_{ij}, D_y\mu^{n+\frac{1}{2},m}_{ij}$ in the same way. Next we define the discrete gradient $\nabla_d\phi^{n+1,m}_{ij}$, as well as a modified Laplacian $\nabla_d \cdot (G_{ij}\nabla_d\phi^{n+1,m}_{ij})$:

$$\nabla_d \phi_{ij}^{n+1,m} = \left(D_x \phi_{i+1j}^{n+1,m},\ D_y \phi_{ij+1}^{n+1,m} \right) \tag{3.4}$$

$$\nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1,m}) = \frac{G_{i+\frac{1}{2}j} D_x \phi_{i+\frac{1}{2}j}^{n+1,m} - G_{i-172} D_x \phi_{i-\frac{1}{2}j}^{n+1,m} + D_y \phi_{ij+\frac{1}{2}}^{n+1,m} - D_y \phi_{ij-\frac{1}{2}}^{n+1,m}}{h} \tag{3.5}$$

$$= \frac{G_{i+\frac{1}{2}j} \phi_{i+1j}^{n+1,m} + G_{i-\frac{1}{2}j} \phi_{i-1j}^{n+,m} + G_{ij+\frac{1}{2}} \phi_{ij+1}^{n+,m} + G_{ij-\frac{1}{2}} \phi_{ij-1}^{n+,m}}{h^2} \tag{3.6}$$

$$- \frac{\left(G_{i+\frac{1}{2}j} + G_{i-\frac{1}{2}j} + G_{ij+\frac{1}{2}} + G_{ij-\frac{1}{2}} \cdot \phi_{ij} \right)}{h^2} \tag{3.7}$$

We define $\nabla_d \cdot (G_{ij} \nabla_d \phi_{ij})$ instead of a discrete laplacian Δ_d to ensure a discrete version of boundary conditions 2.8. the discretizations for $\nabla_d \mu_{ij}^{n+\frac{1}{2},m}, \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2},m})$ are done the same as for ϕ_{ij}^{n+1}

The authors in [1] show this to be the case by expanding $\nabla_d \cdot (G_{ij} \nabla_d \phi_{ij})$.

notably, when one point lies outside the domain, then $G_{i\pm\frac{1}{2}} = 0$ and therefore the corresponding discrete gradient $\frac{\phi_{i\pm1} - \phi_i}{h}$ is weighted by 0. This corresponds the discrete version of $\partial_n \phi = 0$. The authors in [1]

To simplify the notation for discretized derivatives we use the following abbreviations:

- $\Sigma_G \phi_{ij} = G_{i+\frac{1}{2}j} \phi_{i+1j}^{n+1,m} + G_{i-\frac{1}{2}j} \phi_{i-1j}^{n+1,m} + G_{ij+\frac{1}{2}} \phi_{ij+1}^{n+1,m} + G_{ij-\frac{1}{2}} \phi_{ij-1}^{n+1,m}$
- $\Sigma_{Gij} = G_{i+\frac{1}{2}j} + G_{i-\frac{1}{2}j} + G_{ij+\frac{1}{2}} + G_{ij-\frac{1}{2}}$

Code:

```
function neighbours_in_domain(i, j, G, len, width)
    (
        G(i + 0.5, j, len, width)
        + G(i - 0.5, j, len, width)
        + G(i, j + 0.5, len, width)
        + G(i, j - 0.5, len, width)
    )

end
function discrete_G_weigted_neigbour_sum(i, j, arr, G, len, width)
    (
        G(i + 0.5, j, len, width) * arr[i+1, j]
```

```
        + G(i - 0.5, j, len, width) * arr[i-1, j]
        + G(i, j + 0.5, len, width) * arr[i, j+1]
        + G(i, j - 0.5, len, width) * arr[i, j-1]
    )
end
```

We can then write the modified Laplacian $\nabla_d(G\nabla_d\phi_{ij})$ as:

$$\nabla_d \cdot (G\nabla_d\phi_{ij}) = \frac{\Sigma_G\phi_{ij} - \Sigma_G \cdot \phi_{ij}}{h^2}$$

We use this modified Laplacian to deal with boundary conditions. Our abbreviations simplify separating implicit and explicit terms in the discretization.

3.2 Initial data

For testing we use initial phase-fields defined by the following equations:

$$\begin{aligned}
\phi_{ij} &= \begin{cases} 1 & ,\|(i,j) - (\frac{N}{2}, \frac{N}{2})\|_p < \frac{N}{3} \\ -1 & ,else \end{cases} && \text{where } p \in \{2, \infty\} \\
\phi_{ij} &= \begin{cases} 1 & ,i < \frac{N}{2} \\ -1 & ,else \end{cases} \\
\phi_{ij} &= \begin{cases} 1 & ,\|(i,j) - (\frac{N}{2}, 2)\|_2 < \frac{N}{3} \\ -1 & ,else \end{cases} \\
\phi_{ij} &= \begin{cases} 1 & ,\|(i,j) - q_k\|_p < \frac{N}{5} \\ -1 & ,else \end{cases} && p \in \{1, 2, \infty\}, q_k \in Q
\end{aligned} \tag{3.8}$$

where q_k are random points inside my domain. Those we generate those using the following rng setup in julia

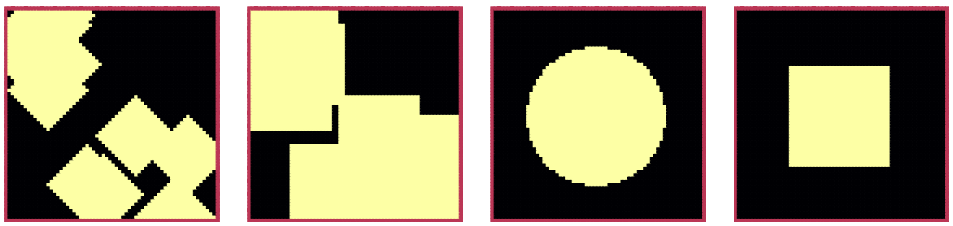


Figure 3.2: Examples of different phase-fields used as the initial condition in this work.

3.3 Numerical ansatz

The authors in [1] then define the discrete CH equation adapted for the domain as:

$$\begin{aligned} \frac{\phi_{ij}^{n+1} - \phi_{ij}^{n}}{\Delta t} &= \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2}}) \\ \mu_{ij}^{n+\frac{1}{2}} &= 2\phi_{ij}^{n+1} - \varepsilon^2 \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1}) + W'(\phi_{ij}^{n}) - 2\phi_{ij}^{n} \end{aligned} \tag{3.9}$$

and derive a numerical scheme from this implicit equation.

3.4 The discrete scheme

The authors in [1] derive their method by separating 3.9 into implicit and linear terms, and explicit non-linear terms. We write the implicit terms in form of a function $L : \mathbb{R}^2 \to \mathbb{R}^2$ and the explicit terms in $(\zeta_{ij}^n, \psi_{ij}^n)^T$.

$$L \begin{pmatrix} \phi_{ij}^{n+1} \\ \mu_{ij}^{n+\frac{1}{2}} \end{pmatrix} = \begin{pmatrix} \frac{\phi_{ij}^{n+1}}{\Delta t} - \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2}}) \\ \varepsilon^2 \nabla_d \cdot (G \nabla_d \phi_{ij}^{n+1}) - 2\phi_{ij}^{n+1} + \mu_{ij}^{n+\frac{1}{2}} \end{pmatrix}$$

This operator follows from 3.9 by separating implicit and explicit terms L and $(\zeta_{ij}^n, \psi_{ij}^n)^T$, respectively.

$$\begin{pmatrix} \zeta_{ij}^n \\ \psi_{ij}^n \end{pmatrix} = \begin{pmatrix} \frac{\phi_{ij}^n}{\Delta t} \\ W'(\phi_{ij}^n) - 2\phi_{ij}^n \end{pmatrix}$$

Due to being explicit, we know everything needed to calculate $(\zeta_{ij}^n, \psi_{ij}^n)^T$ at the beginning of each time step. We compute those values once and store them in the solver.

Furthermore, as it is needed later on, we derive its Jacobian with respect to the current grid point $(\phi_{ij}^{n+1}, \mu_{ij}^{n+\frac{1}{2}})^T$:

$$DL \begin{pmatrix} \phi_{ij} \\ \mu_{ij} \end{pmatrix} = \begin{pmatrix} \frac{1}{\Delta t} & \frac{1}{h^2} \Sigma_{Gij} \\ -\frac{\varepsilon^2}{h^2} \Sigma_{Gij} - 2 & 1 \end{pmatrix}$$

Implementation details can be found in the Apendix under baseline.

3.5 SMOOTH operator

The authors [1]derived Gauss-Seidel Smoothing from:

$$L\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix} = \begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} \tag{3.10}$$

SMOOTH consists of point-wise Gauss-Seidel relaxation, by solving Eq.3.10 for all i, j with the initial guess for $\zeta_{ij}^n, \psi_{ij}^n$. Since L is linear we can write Eq.3.10 as

$$\begin{aligned}\begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} &= DL\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\cdot\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix} + \begin{pmatrix}-\frac{1}{h^2}\Sigma_{Gij}\mu_{ij}^{n+\frac{1}{2}}\\ +\frac{\varepsilon^2}{h^2}\Sigma_{Gij}\phi_{ij}^{n+1}\end{pmatrix}\\ \begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} - \begin{pmatrix}-\frac{1}{h^2}\Sigma_{Gij}\mu_{ij}^{n+\frac{1}{2}}\\ +\frac{\varepsilon^2}{h^2}\Sigma_{Gij}\phi_{ij}^{n+1}\end{pmatrix} &= DL\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\cdot\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\end{aligned} \tag{3.11}$$

where

- $\Sigma_G\phi_{ij}^{n+1} = G_{i+\frac{1}{2}j}\phi_{i+1j}^{n+1,m} + G_{i-\frac{1}{2}j}\phi_{i-1j}^{n+1,m} + G_{ij+\frac{1}{2}}\phi_{ij+1}^{n+1,m} + G_{ij-\frac{1}{2}}\phi_{ij-1}^{n+1,m}$,
- $\Sigma_G\mu_{ij} = G_{i+\frac{1}{2}j}\mu_{i+1j}^{n+\frac{1}{2},m} + G_{i-\frac{1}{2}j}\mu_{i-1j}^{n+\frac{1}{2},m} + G_{ij+\frac{1}{2}}\mu_{ij+1}^{n+\frac{1}{2},m} + G_{ij-\frac{1}{2}}\mu_{ij-1}^{n+\frac{1}{2},m}$,

In order to compute $\left(\phi_{ij}^{n+1}, \mu_{ij}^{n+\frac{1}{2}}\right)$ we have to evaluate those grid-wise functions on at neighbouring indicies k, l eg. $k = i+1, l = j-1$. since values for $\phi_{kl}^{n+1,m}, \mu_{kl}^{n+\frac{1}{2},m}$ are unknown,if $k > i, l > j$, the authors in [1] and we use initial approximations, and the values of the current smooth iteration else. As initial approximation we use the values of $\phi_{kl}^{n+1,m}, \mu_{kl}^{n+\frac{1}{2},m}$ from the last smoothing iteration. The equation Eq.3.11 is of form $b = Ax$ We then and solve Eq.3.11 for $\left(\phi_{ij}^{n+1}, \mu_{ij}^{n+\frac{1}{2}}\right)$.

```
function SMOOTH!(
    solver::T,
    iterations,
    adaptive
) where T <: Union{multi_solver, adapted_multi_solver , gradient_boundary_solver}
    for k = 1:iterations
        # old_phase = copy(solver.phase)
        for I in CartesianIndices(solver.phase)[2:end-1, 2:end-1]
            i, j = I.I

            <<calculate-left-hand-side-b>>
```

```
            res = dL(solver, i,j ) \ b
            solver.phase[i, j] = res[1]
            solver.potential[i, j] = res[2]
        end
    end
end
```

In Fig.3.3 we show 4 of the 7 initial data after one 200 iterations of smoothing. It is apparent that the sharp interface from the initial Data has diffused.

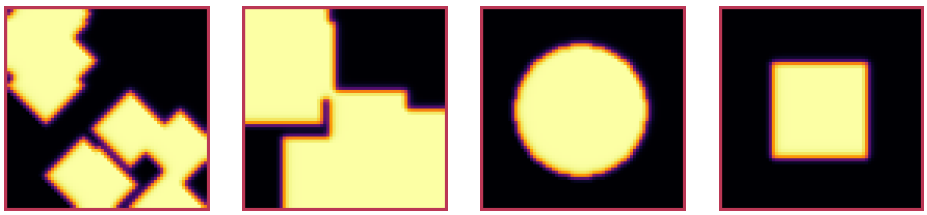


Figure 3.3: inputs from **??** after SMOOTH.

3.6 Multigrid method

The numerical method proposed in [1] consists of a V-cycle multi-grid method derived from previously stated operators. Specificly we use a two-grid implementation consisting of.

```
for j in 1:timesteps

    set_xi_and_psi!(solvers[1])

    for i = 1:subiterations

        v_cycle!(solvers, 1)
    end
end
```

The V-cycle is a

1. a Gauss-Seidel relaxation for smoothing Chapter **??**.
2. calculate the residual error in phase and potential $d_{ij,H}^{n+1,m}, r_{ij,H}^{n+1,m}$.

3. restriction and prolongation methods between grids $h \leftrightarrow H$.

4. a Newton iteration to solve $L(\phi_{ij,H}^{n+1,m}, \mu_{ij,H}^{n+\frac{1}{2},m})_H = L(\bar{\phi}_{ij,H}^{n+1,m}, \bar{\mu}_{ij,H}^{n+\frac{1}{2},m}) + (d_{ij,H}^{n+1,m}, r_{ij,H}^{n+1,m})$. we solve using the same iteration as in Chapter **??** however we replace $(\zeta_{ij}^n, \psi_{ij}^n)$ with $L(\bar{\phi}_{ij,H}^{n+1,m}, \bar{\mu}_{ij,H}^{n+\frac{1}{2},m}) + (d_{ij,H}^{n+1,m}, r_{ij,H}^{n+1,m})$. in the iteration, where $\bar{\phi}_{ij,H}^{n+1,m}, \bar{\mu}_{ij,H}^{n+\frac{1}{2},m}$ are the values after the smooth restricted to the coarser grid and $d_{ij,H}^{n+1,m}, r_{ij,H}^{n+1,m}$ is the residual from the smooth iteration on the fine grid restricted onto the coarse grid.

5. post smoothing

The V-cycle of a two-grid method using pre and post smoothing is then stated by:

```
function v_cycle!(grid::Array{T}, level) where T <: solver
    solver = grid[level]
    #pre SMOOTHing:
    SMOOTH!(solver, 400, false)

    d = zeros(size(solver.phase))
    r = zeros(size(solver.phase))

    # calculate error between L and expected values
    for I in CartesianIndices(solver.phase)[2:end-1, 2:end-1]
        d[I], r[I] = [solver.xi[I], solver.psi[I]] .- L(solver, I.I...,
        ↪  solver.phase[I], solver.potential[I])
    end

    <<restrict-to-coarse-grid>>

    #Newton Iteration for solving smallgrid
    for i = 1:300
        for I in CartesianIndices(solver.phase)[2:end-1, 2:end-1]

            diffrence = L(solution, I.I..., solution.phase[I],
            ↪  solution.potential[I])
                        .- [d_large[I], r_large[I]]
                        .- L(solver, I.I..., solver.phase[I], solver.potential[I])

            local ret = dL(solution, I.I...) \ diffrence

            u_large[I] = ret[1]
            v_large[I] = ret[2]
```

```
        end
        solution.phase .-= u_large
        solution.potential .-= v_large
    end

    <<prolong-to-fine-grid>>

    SMOOTH!(solver, 800, false)
end
```

After a few iterations, V-cycle exhibits the following behavior:
images/iteration.gif

4 Numerical experiments

The analytical CH equation conserves mass Eq.2.4 and the free energy ,E_{bulk}, Eq.2.2 decreases in time, i.e. consistence with the second law of thermodynamics. Therefore, we use discrete variants of those concepts as necessary conditions for a "good" solution. Furthermore, since E_{bulk} is closely correlated with chemical potential, μ, we evaluate this difference as quality of convergence.

4.1 Energy evaluations

As discrete energy measure we use:

$$\begin{aligned} E_d^{\text{bulk}}(\phi_{ij}) &= \sum_{i,j\in\Omega} \frac{\varepsilon^2}{2} |G\nabla_d \phi_{ij}|^2 + W(\phi_{ij}) \\ &= \sum_{i,j\in\Omega} \frac{\varepsilon^2}{2} G_{i+\frac{1}{2}j}(D_x\phi_{i+\frac{1}{2}j})^2 + G_{ij+\frac{1}{2}}(D_y\phi_{ij+\frac{1}{2}})^2 + W(\phi_{ij}) \end{aligned} \tag{4.1}$$

Since the continous Helmholtz energy Eq.2.2.

```
function bulk_energy(solver::T) where T <: Union{multi_solver ,
↪  relaxed_multi_solver}
    energy = 0
    dx = CartesianIndex(1,0)
    dy = CartesianIndex(0,1)
    W(x) = 1/4 * (1-x^2)^2
    for I in CartesianIndices(solver.phase)[2:end-1,2:end-1]
        i,j = I.I
        energy += solver.epsilon^2 / 2 * G(i+ 0.5,j ,solver.len, solver.width) *
        ↪  (solver.phase[I+dx] - solver.phase[I])^2 + G(i,j+0.5,solver.len
        ↪  ,solver.width) * (solver.phase[I+dy] - solver.phase[I])^2 +
        ↪  W(solver.phase[I])
        end
   return energy
end
```

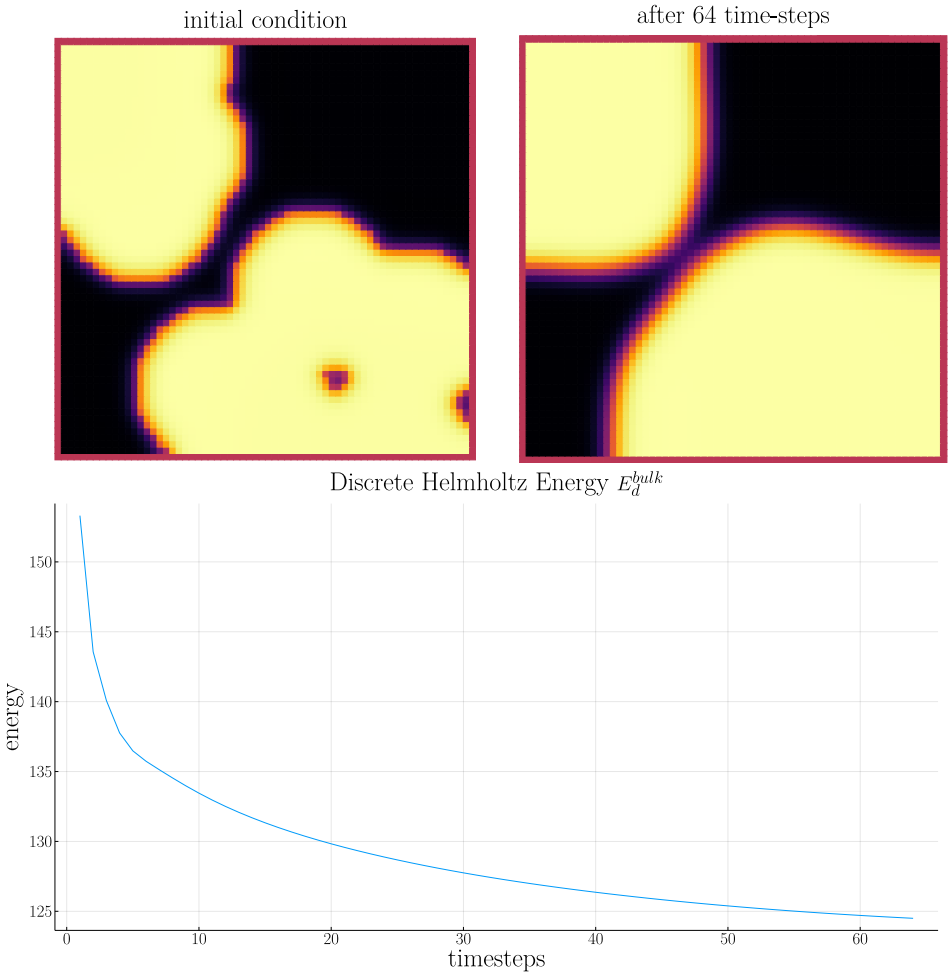


Figure 4.1: behaviour of energy E_{bulk} over time for one initial condition ϕ_0.

here we observe the discrete Helmholtz energy going down with increasing number of timesteps, as we expect from a cahn hilliard based solver.

4.2 Numerical mass conservation

Instead of a physical mass we use the average of ϕ over the domain Ω. This yields a balance between both phases. Since the analytical CH equation Eq.2.1 is mass conserving we require a good numerical implementation to exhibit as few loss in mass

as possible. Since our implementation uses no-flow boundary conditions the balance between *phase 1* and *phase 2* stays the same. We therefore calculate a balanace

$$b = \frac{\sum_{i,j\in\Omega} \phi_{ij}}{N^2}$$

such that $b = 1$ means there is only phase 1, $\phi \equiv 1$, and $b = -1$ means there is only phase 2, $\phi \equiv -1$. Ideally this value stays constant over time. In practice we observe slight fluctuations in Figure 4.2. Those however are close to machine precision and can therefore be ignored.

```
function massbal(arr)
    num_cells= *((size(arr).-2)...)
    return sum(arr[2:end-1, 2:end-1])/num_cells
    end
```

4.3 Stability of a multigrid sub iteration

We expect our solver to stay stable when increasing the number of multigrid sub-iterations. To validate this assumption we compare the phase-field of the current sub-iteration $\phi_{ij}^{n+1,m}$ with the phse-field of the previous sub-iteration $\phi_{ij}^{n+1,m-1}$.

$$\|\phi^{n+1,m-1} - \phi^{n+1,m}\|_{Fr} = \sqrt{\sum_{i,j\in\Omega_d} \left|\phi_{ij}^{n+1,m-1} - \phi_{ij}^{n+1,m}\right|} \tag{4.2}$$

As sub-iterations increase , $m \to \infty$, we expect the difference between both phase-fields to go to zero $\|\phi^{n+1,m} - \phi^{n+1,m-1}\|_{Fr} \to 0$. We observe this behaviour in Figure 4.3

in practise we observe the behaviour we expect, where an increasing number of sub-iterations leads to decreasing change compared to the previous sub-iteration.

4.4 Stability in time

We expect our numerical error to decrease when calculating with smaller time steps. To test this, we succesivly subdivide the original time interval $[0, T]$ in finer parts. We fix $\Delta t \cdot n = T$ for $T = 10^{-2}$ and test different values of n. In Figure 4.4 we compare the phase-field ϕ_{ij}^{n} and ϕ_{ij}^{n-1} at $T = 10^{-2}$. and observe the decrease we expect.

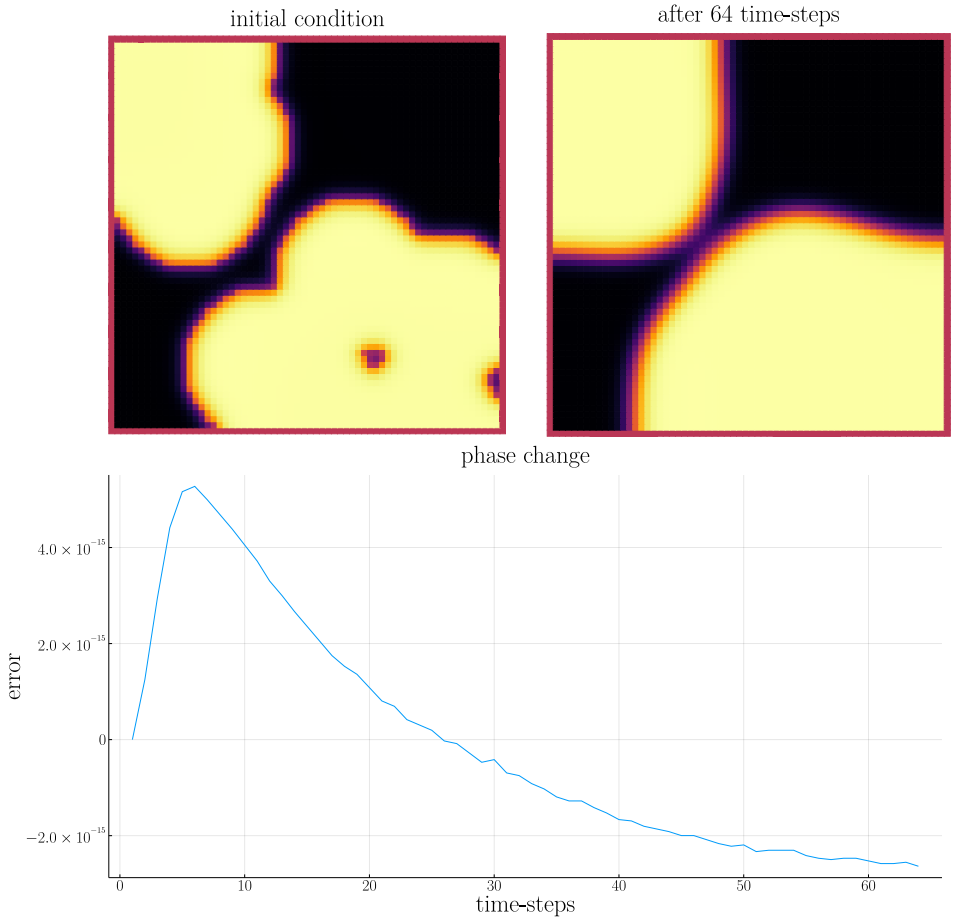


Figure 4.2: behaviour of phase change over time for one initial condition ϕ_0.

4.5 Stability in space

We expect our methods to be stable under different grid-sizes h and gridpoints N. Therefore we expect the difference after one time-step between eg. a 512×512 grid and a 1024×1024 grid to be smaller than the difference between a 64×64 grid and a 128×128 grid. In order to keep the problem the same , we fix $Nh = 10^{-3} \cdot 1024$ and test for $N \in \{1024, 512, 256, 128, 64, 32\}$

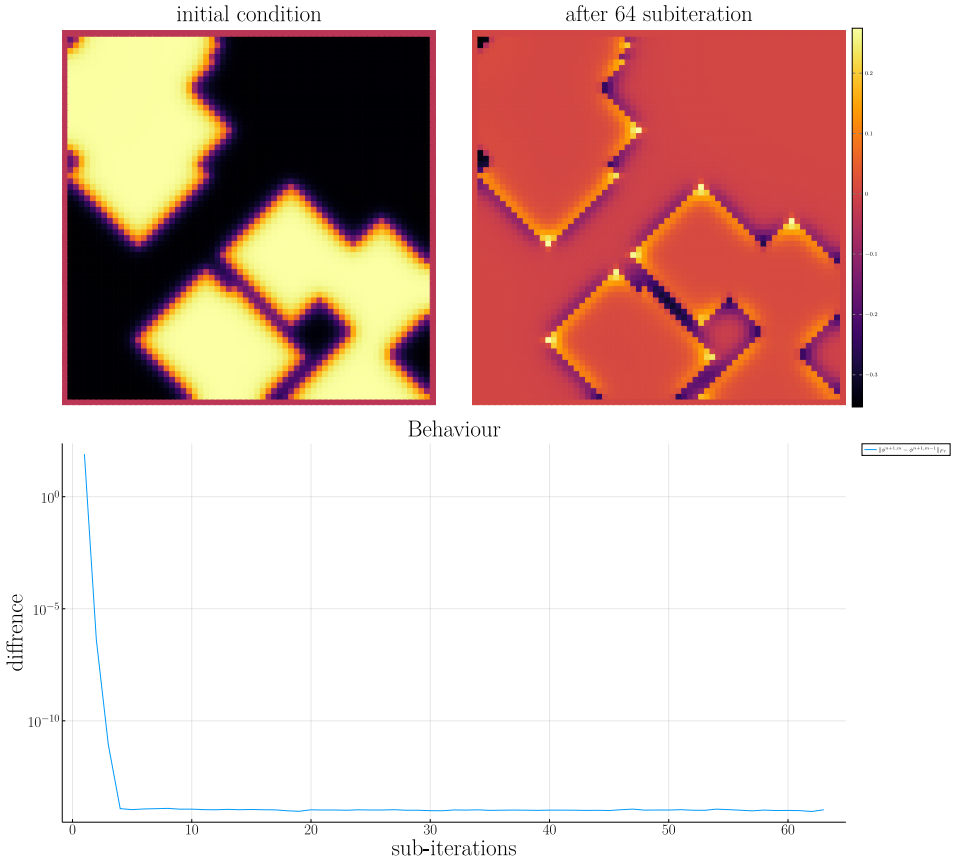


Figure 4.3: stability of the original CH solver for increasing sub-iterations

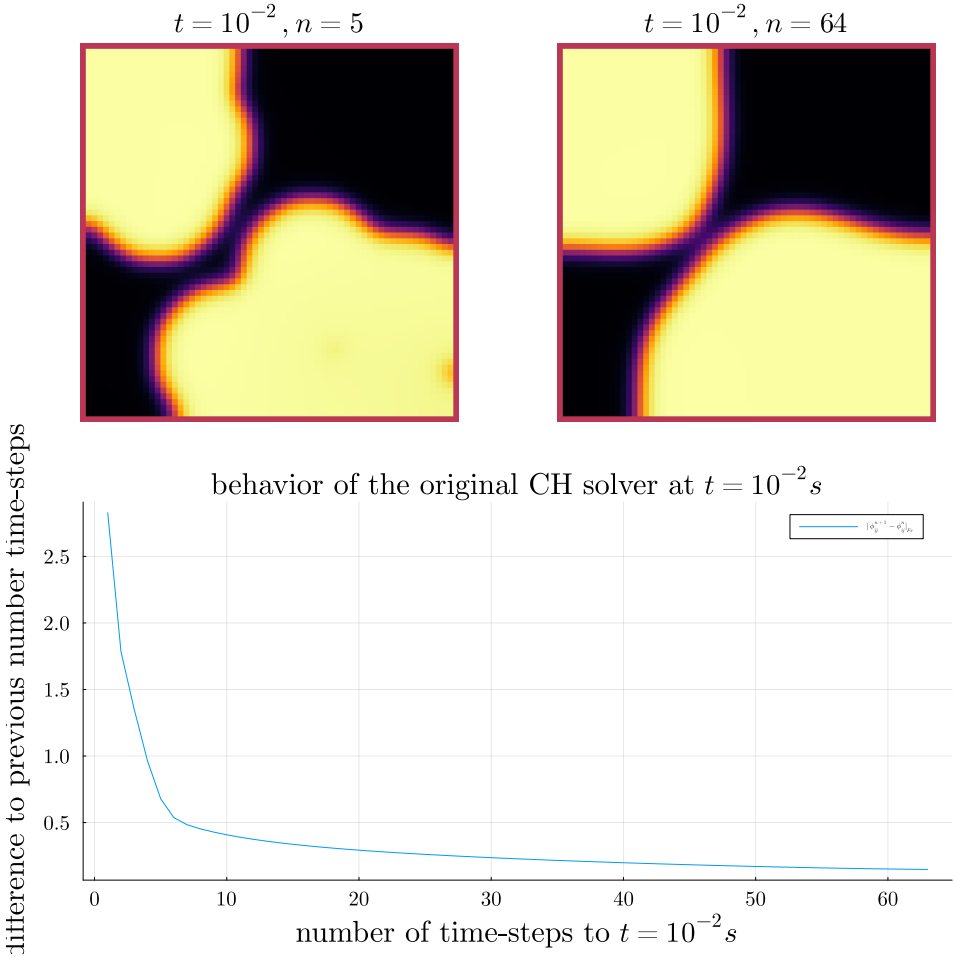


Figure 4.4: behavior of the baseline solver while solving the time interval $T = \left[0, 10^{-2}\right]$ with increasing number of time-steps.

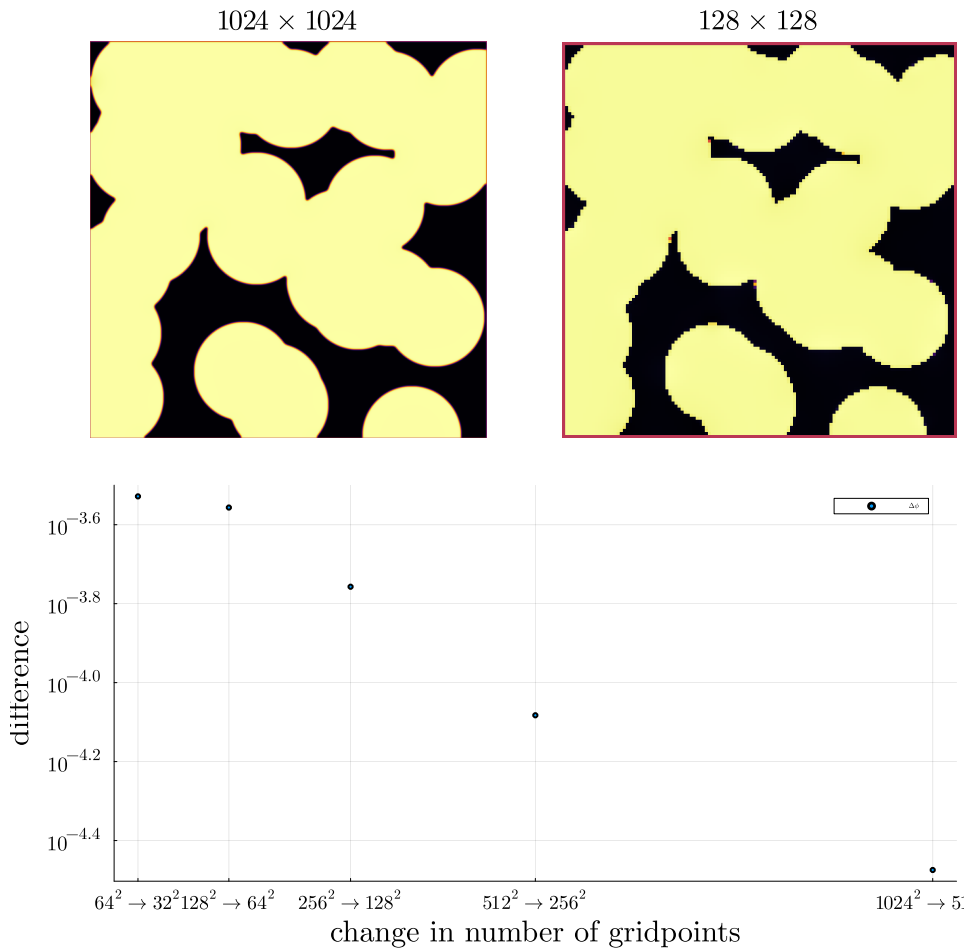


Figure 4.5: behavior of the baseline solver while solving on successively finer grids

5 Relaxed problem

In effort to decrease the order of complexity, from fourth order derivative to second order, we propose an elliptical relaxation approach, where the relaxation variable c is the solution of the following elliptical PDE:

$$-\Delta c^\alpha + \alpha c^a = \alpha \phi^\alpha, \tag{5.1}$$

where α is a relaxation parameter. We expect to approach the original solution of the CH equation Eq.2.1 as $\alpha \to \infty$. This results in the following relaxation for the classical CH equation Eq.2.1:

$$\begin{aligned} \partial_t \phi^\alpha &= \Delta \mu \\ \mu &= \varepsilon^2 \alpha (c^\alpha - \phi^\alpha) + W'(\phi) \end{aligned} \tag{5.2}$$

It requires solving the elliptical PDE each time-step to calculate c.

As ansatz for the numerical solver we propose:

$$\begin{aligned} \frac{\phi_{ij}^{n+1,\alpha} - \phi_{ij}^{n,\alpha}}{\Delta t} &= \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2},\alpha}) \\ \mu_{ij}^{n+\frac{1}{2},\alpha} &= 2\phi_{ij}^{n+1,\alpha} - \varepsilon^2 a (c_{ij}^{n+1,\alpha} - \phi_{ij}^{n+1,\alpha}) + W'(\phi_{ij}^{n,\alpha}) - 2\phi_{ij}^{n,\alpha} \end{aligned} \tag{5.3}$$

This approach is inspired by Eq.3.9 adapted to the relaxed CH equation Eq.5.3. We then adapt the multi-grid solver proposed in **??** to the relaxed problem by replacing the differential operators by their discrete counterparts as defined in Eq.3.4, and expand them.

5.1 Elliptical PDE

In order to solve the relaxed CH equation we solve the following PDE in each time step:

$$-\nabla \cdot (G \nabla c^\alpha) + \alpha c^\alpha = \alpha \phi^\alpha$$

Similarly to the first solver we solve this PDE with a finite difference scheme using the same discretization as before.

5.1.1 Discretization

The discretization of the PDE expands the differential operators in the same way and proposes an equivalent scheme for solving the elliptical equation Eq.5.1.

$$-\nabla_d \cdot (G_{ij} \nabla_d c_{ij}^\alpha) + \alpha c_{ij}^\alpha = \alpha \phi_{ij}^\alpha$$

$$\Longrightarrow$$

$$\begin{aligned} -(\frac{1}{h}(G_{i+\frac{1}{2}j} \nabla c_{i+\frac{1}{2}j}^\alpha + G_{ij+\frac{1}{2}} \nabla c_{ij+\frac{1}{2}}^\alpha) \\ -(G_{i-\frac{1}{2}j} \nabla c_{i-\frac{1}{2}j}^\alpha + G_{ij-\frac{1}{2}} \nabla c_{ij-\frac{1}{2}}^\alpha)) + \alpha c_{ij}^\alpha = \alpha \phi_{ij}^\alpha \end{aligned}$$

$$\Longrightarrow$$

$$\begin{aligned} -\frac{1}{h^2}(G_{i+\frac{1}{2}j}(c_{i+1j}^\alpha - c_{ij}^\alpha) \\ +G_{ij+\frac{1}{2}}(c_{ij+1}^\alpha - c_{ij}^\alpha) \\ +G_{i-\frac{1}{2}j}(c_{i-1j}^\alpha - c_{ij}^\alpha) \\ +G_{ij-\frac{1}{2}}(c_{ij-1}^\alpha - c_{ij}^\alpha)) + \alpha c_{ij}^\alpha = \alpha \phi_{ij}^\alpha \end{aligned}$$

As before we abbreviate $\Sigma_G c_{ij}^\alpha = G_{i+\frac{1}{2}j} c_{i+1j}^\alpha + G_{i-\frac{1}{2}j} c_{i-1j}^\alpha + G_{ij+\frac{1}{2}} c_{ij+1}^\alpha + G_{ij-\frac{1}{2}} c_{ij-1}^\alpha$ and $\Sigma_{Gij} = G_{i+\frac{1}{2}j} + G_{i-\frac{1}{2}j} + G_{ij+\frac{1}{2}} + G_{ij-\frac{1}{2}}$. Then the discrete elliptical PDE can be stated as:

$$-\frac{\Sigma_G c_{ij}^\alpha}{h^2} + \frac{\Sigma_G}{h^2} c_{ij}^\alpha + \alpha c_{ij}^\alpha = \alpha \phi_{ij}^\alpha \tag{5.4}$$

solving Eq.5.4 for c_{ij}^α then results in.

$$\begin{aligned} \left(\frac{\Sigma_{Gij}}{h^2} + \alpha\right) c_{ij}^\alpha &= \alpha \phi_{ij}^\alpha + \frac{\Sigma_G c_{ij}^\alpha}{h^2} \\ c_{ij}^\alpha &= \frac{\alpha \phi_{ij}^\alpha + \frac{\Sigma_G c_{ij}^\alpha}{h^2}}{\frac{\Sigma_G}{h^2} + \alpha} \\ c_{ij}^\alpha &= \frac{\alpha h^2 \phi_{ij}^\alpha}{\Sigma_{Gij} + \alpha h^2} + \frac{\Sigma_G c_{ij}^\alpha}{\Sigma_{Gij} + \alpha h^2} \end{aligned}$$

and can be translated to code as follows

```
function elyps_solver!(solver::T, n) where T  <: Union{relaxed_multi_solver ,
↪  adapted_relaxed_multi_solver}
    for k in 1:n
        for i = 2:(solver.len+1)
            for j = 2:(solver.width+1)
                bordernumber = neighbours_in_domain(i, j,G, solver.len,
                ↪  solver.width)
                solver.c[i, j] =
                    (
                        solver.alpha * solver.phase[i, j] +
                        discrete_G_weigted_neigbour_sum(i, j, solver.c, G,
                        ↪  solver.len, solver.width) / solver.h^2
                    ) / (bordernumber / solver.h^2 + solver.alpha)

            end
        end
    end
end
```

5.2 Relaxed PDE as operator L

We reformulate the discretization Eq.5.3 in terms of the relaxed operator L as follows:

$$L_r\begin{pmatrix}\phi^{n+1,\alpha}\\ \mu^{n+\frac{1}{2},\alpha}\end{pmatrix} = \begin{pmatrix}\frac{\phi_{ij}^{n+1,m,\alpha}}{\Delta t} - \nabla_d\cdot(G_{ji}\nabla_d\mu_{ji}^{n+\frac{1}{2},m,\alpha})\\ \varepsilon^2\alpha(c^\alpha-\phi_{ij}^{n+1,m,\alpha}) - 2\phi_{ij}^{n+1,m,\alpha} - \mu_{ji}^{n+\frac{1}{2},m,\alpha}\end{pmatrix}$$

and its Jacobian:

$$DL_r\begin{pmatrix}\phi\\ \mu\end{pmatrix} = \begin{pmatrix}\frac{1}{\Delta t} & \frac{1}{h^2}\Sigma_G\\ -\varepsilon^2\alpha-2 & 1\end{pmatrix}$$

5.3 The relaxed multigrid method

As the difference between both methods is abstracted away in the operators, the relaxed V-cycle the replaces the original operators with their relaxed counterparts. Due to julias multiple dispatch features this changes nothing in the implementation Therefore we reuse the original V-cycle in the **??**. In the executions for each time step, we add the elliptic solver in the subiteration.

```
for j in 1:timesteps

    set_xi_and_psi!(solvers[1])

    for i = 1:subiterations

        elyps_solver!(solvers[1] , 1000)
        v_cycle!(solvers, 1)
    end
end
```

images/relaxed-anim.gif

5.4 SMOOTH operator

The relaxed solver uses the same approach as the original solver, where we solve $L_r(\phi_{ij}^{n+1,m,\alpha}, \mu_{ij}^{n+\frac{1}{2},m,\alpha}) = (\zeta_{ij}^n, \psi_{ij}^n)^T$ for each grid-point $\phi_{ij}^{n+1,m,\alpha}$. Notably $(\zeta_{ij}^n, \psi_{ij}^n)^T$ is the same as in the original part. As in the original smoothing, evalations of $\mu_{kl}^{n+\frac{1}{2},m,\alpha}$ for $k, l > i, j$ are replaced with their values from the previous SMOOTH iteration.

Correspondingly the SMOOTH operation expands to:

$$\begin{aligned} -\frac{\Sigma_{Gij}}{h^2}\overline{\mu_{ji}^{n+\frac{1}{2},m,\alpha}} &= \frac{\phi_{ij}^{n+1,m,\alpha}}{\Delta t} - \zeta_{ij}^{n,\alpha} - \frac{\Sigma_G \mu_{ij}}{h^2} \\ \varepsilon^2\alpha\overline{\phi_{ij}^{n+1,m,\alpha}} + 2\phi_{ij}^{n+1,m,\alpha} &= \varepsilon^2\alpha c_{ij}^{n,\alpha} - \overline{\mu_{ji}^{n+\frac{1}{2},m,\alpha}} - \psi_{ij}^{n,\alpha} \end{aligned} \tag{5.5}$$

where

- $\Sigma_G \mu_{ij} = G_{i+\frac{1}{2}j}\mu_{i+1j}^{n+\frac{1}{2},m} + G_{i-\frac{1}{2}j}\mu_{i-1j}^{n+\frac{1}{2},m} + G_{ij+\frac{1}{2}}\mu_{ij+1}^{n+\frac{1}{2},m} + G_{ij-\frac{1}{2}}\mu_{ij-1}^{n+\frac{1}{2},m}$,

We then solve directly for the smoothed variables, $\overline{\mu_{ij}^{n+1,m,\alpha}}$ and $\overline{\phi_{ij}^{n+1,m,\alpha}}$. This was not done in the original paper [1] because the required system of linear equations in the paper [1] was solved numerically.

$$\varepsilon^2\alpha(\phi_{ij}^{n+1,m,\alpha}) + 2\phi_{ij}^{n+1,m,\alpha} = \varepsilon^2\alpha c^\alpha - \frac{h^2}{\Sigma_G}(\frac{\phi_{ij}^{n+1,m,\alpha}}{\Delta t} - \zeta_{ij}^n - \frac{1}{h^2}\Sigma_G\mu_{ij}) - \psi_{ij}$$

$\Longrightarrow$

$$\varepsilon^2\alpha(\phi_{ij}^{n+1,m,\alpha}) + 2\phi_{ij}^{n+1,m,\alpha} + \frac{h^2}{\Sigma_{Gij}}\frac{\phi_{ij}^{n+1,m,\alpha}}{\Delta t} = \varepsilon^2\alpha c^\alpha - \frac{h^2}{\Sigma_G}(-\zeta_{ij}^n - \frac{1}{h^2}\Sigma_G\mu_{ij}) - \psi_{ij}$$

$$\implies$$

$$(\varepsilon^2\alpha + 2 + \frac{h^2}{\Sigma_G \Delta t})\phi_{ij}^{n+1,m,\alpha} = \varepsilon^2\alpha c^{\alpha} - \frac{h^2}{\Sigma_G}(-\zeta_{ij}^{n} - \frac{\Sigma_G \mu_{ij}}{h^2}) - \psi_{ij}$$

$$\implies$$

$$\phi_{ij}^{n+1,m,\alpha} = \left(\varepsilon^2\alpha c^{\alpha} - \frac{h^2}{\Sigma_G}(-\zeta_{ij}^{n} - \frac{\Sigma_G \mu_{ij}}{h^2}) - \psi_{ij}\right)\left(\varepsilon^2\alpha + 2 + \frac{h^2}{\Sigma_G \Delta t}\right)^{-1}$$

```
function SMOOTH!(
    solver::T,
    iterations,
    adaptive
) where T <: Union{relaxed_multi_solver , adapted_relaxed_multi_solver}
    for k = 1:iterations
        # old_phase = copy(solver.phase)
        for I in CartesianIndices(solver.phase)[2:end-1, 2:end-1]
            i, j = I.I
            <<solve-for-phi>>
            <<update-potential>>
        end

        #if adaptive && LinearAlgebra.norm(old_phase - solver.phase) < 1e-10
            ##println("SMOOTH terminated at $(k) succesfully")
            #break
        #end
    end
end
```

Furthermore, experimentation shows that alpha alone is insufficient to get a relaxed method consistent with the original solver, since alpha had an effect similar to epsilon, where it changed the boundary thickness in the phase-field ϕ. Therefore epsilon and alpha cannot be chosen independently. Hence we use a simple MCMC optimizer for α, ε in order to give the relaxed solver the best chance we can. Monte Carlo Optimizer For ε, α.

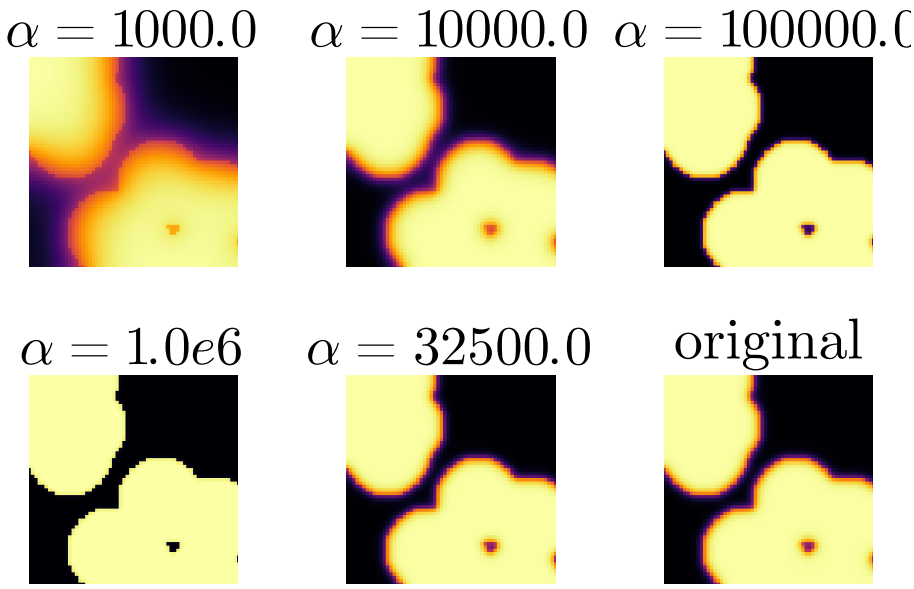


Figure 5.1: effect of the relaxed SMOOTH operator, and additional solving of the elliptical problem, for different values of alpha

6 Relaxed Experiments

We expect the relaxed solver to behave the same as the baseline method for all test cases that we have introduced in Chapter **??**. Therefore we run the same experiments for our relaxed solver.

6.1 Relaxed energy evaluations

we do evaluate our relaxed method using the discrete Helmoltz energy defined in Eq.4.1. On the same initial data, and with the same values for ε, h, dt as in the Chapter.**??**. In Figure.6.1 we then observe the energy decay we expected. Our relaxed approach closely follows the baseline, although it consistently decayed slightly faster. This is within our expectations.

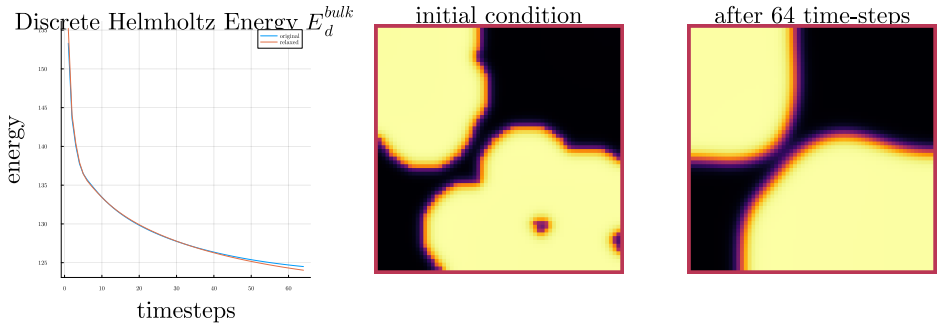


Figure 6.1: energy decay of the relaxed solver compared to the original solver.

We observe the discrete Helmoltz energy decrease is the same manner as with the original solver.

6.2 Relaxed numerical mass balance

since both the CH equation Eq.2.1 and the baseline solver from Fig.4.2 are mass conservative, the relaxed solver should be as well, to be competitive with the baseline approach. Our relaxed solver shows mass loss around 2% as seen in Fig.6.2. This

is nowhere near the machine precision, we reached in Fig.4.2. However it is still tolerable.

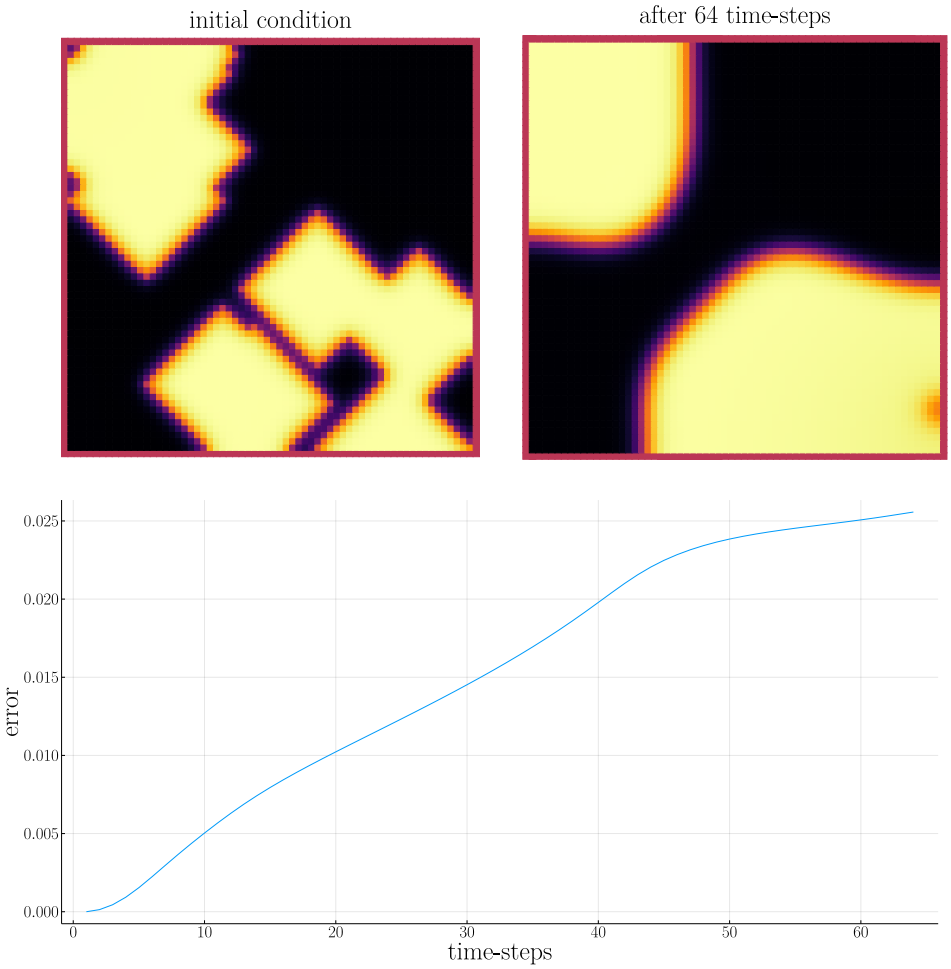
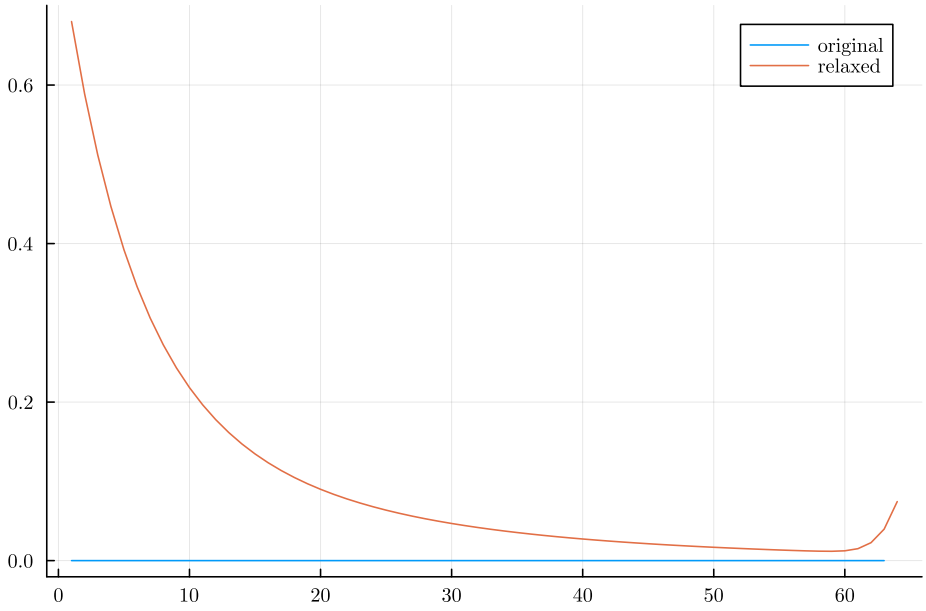


Figure 6.2: Mass los in the relaxed solver

6.3 Stability of a relaxed multigrid sub-iteration

We also compare the subiteration behaviour of the relaxed solver to the original we therefore plot $\|\phi_{ij}^{n+1,m} - \phi_{ij}^{n+2,m-1}\|_{Fr}$ against $\|\phi_{ij}^{n+1,m,\alpha} - \phi_{ij}^{n+1,m-1,\alpha}\|$ for $m \in \{2, \ldots, 64\}$. Here we observe instablility at about 60 sub-iterations in Fig.6.3. We are uncertain, as to why.



6.4 Relaxed stability in time

we test the behaviour under refinement in time by succesivly subdividing the original time interval $[0, T]$ in finer parts. We use the same meassure as in Chaper.**??** and directly compare. We observe simmilar behaviour to the original solver in Fig.6.3. The relaxed solver has consisten lower difference than the original solver. This might suggest a more consistent method over time. However since the sub-iteration showed problematic behaviour, this micht also be a side-effect of this.

6.5 Relaxed stability in space

we test convergence in space by succesivly subdividing our grid into finer meshes

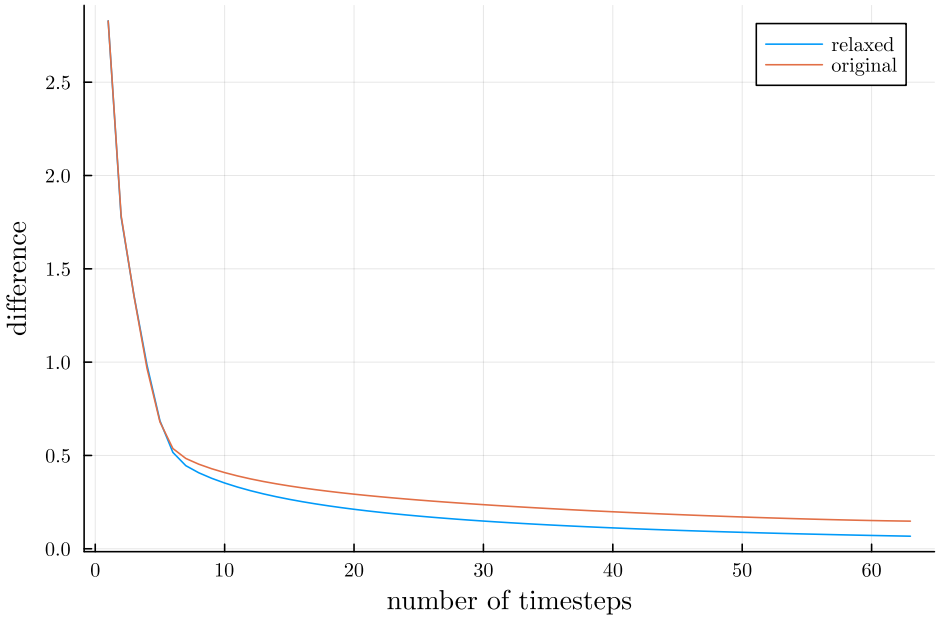
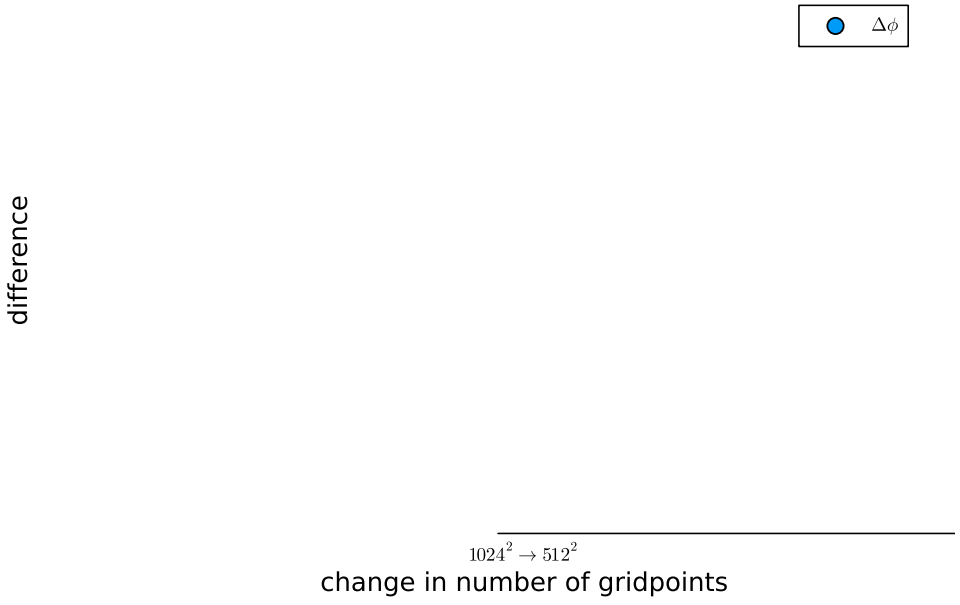


Figure 6.3: behavior of the relaxed and baseline solvers while solving the time interval $T = [0, 10^{-2}]$ with increasing number of time-steps.



7 Comparison

In the previous chapter we have shown stability compared to the original solver. However we have not yet show a direct comparison between both methods. Since the relaxed solver is dependant on the relaxation variable α We are interested in finding an optimal value for it. Furthemore to see the effect α has on our solver, we evaluate both solvers after one time-step , and then calculate the difference between ϕ_{ij}^{n+1} and $\phi_{ij}^{n+1,\alpha}$, for various values of α. Should the relaxed solver approach the original, we would expect

$$||\phi_{ij}^{n+1} - \phi_{ij}^{n+1,\alpha}||_{Fr} \to 0 \tag{7.1}$$

In Fig.7.1 we observe the following behaviour where in all cases the difference to the original solver is apparent. Furthermore we observe a optimal value of α at approximately $7.5*10^5$ we explain this with our observations done for the Smoothing operator, where for small and large values of α the relaxed approach ironically results in restricted behaviour. Empirical this is to be expected as. for large values of alpha the elliptical equation approaches ϕ and for small values the elliptical solver from chapter **??** does not converge.

images/relaxed-comp.gif

although we can observe slight differences between the original solver and the relaxed approach they are barely noticeable by eye. Therefore we run our solver for a fixed value of $\alpha = 7700$, as this was one of the best values from Fig.7.1, We then show the numerical difference between ϕ_{ij}^{n} and $\phi_{ij}^{n,\alpha}$ in Fig.7. We observe a a small difference between both methods, especially in areas with high curvature and inclusions of small segments of one phase in the other. images/relaxed-comparison.gif

images/relaxed-comparison.gif

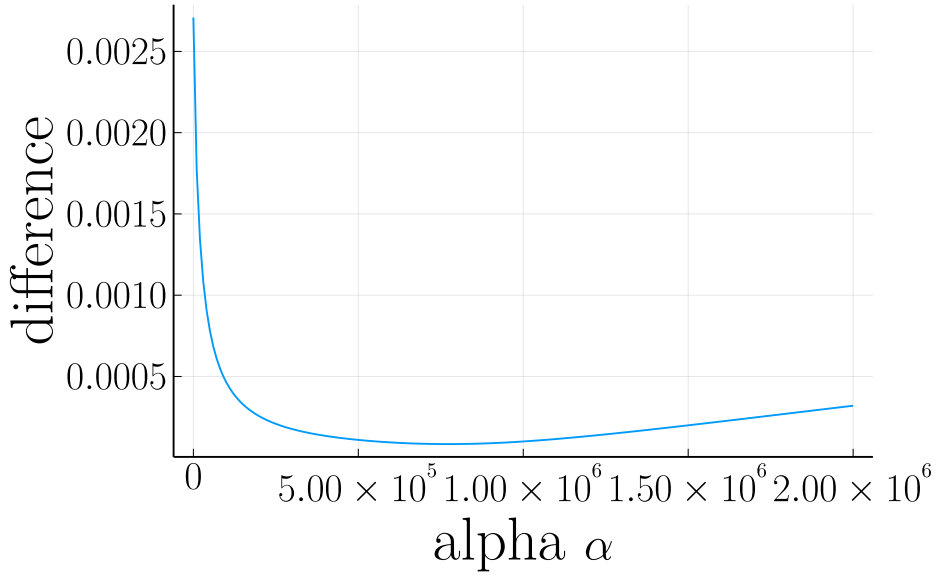
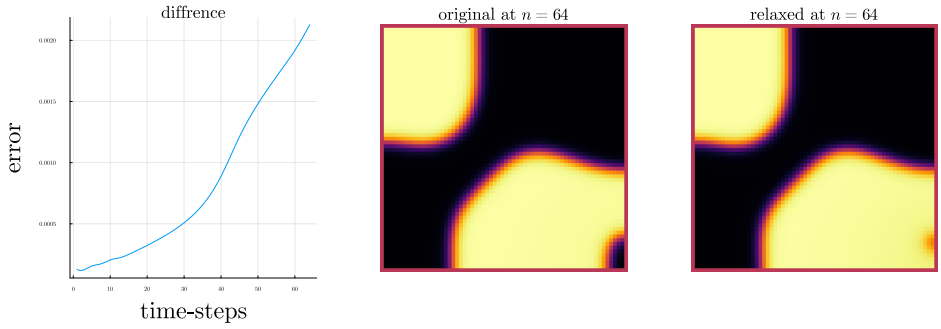


Figure 7.1: Difference between the original solver ϕ_{ij}^{1} and the relaxed solver $\phi_{ij}^{1,\alpha}$



In addition to the experiments in Fig.7.1 we have experimented with a Monte Carlo Optimizer to opmimize α in conjuvtion with ε to best approximate the baseline solver after one time-step. This resulted in a optimal ε found that was very close to the actual ε used. (9e-3 compared to 8e-3).

```
using Distributions
using DataFrames
using JLD2
include(pwd() * "/src/solvers.jl")
include(pwd() * "/src/adapted_solvers.jl")
include(pwd() * "/src/utils.jl")
include(pwd() * "/src/multisolver.jl")
include(pwd() * "/src/multi_relaxed.jl")
```

```
include(pwd() * "/src/testgrids.jl")
include(pwd() * "/src/elypssolver.jl")
using Plots
using LaTeXStrings
using LinearAlgebra
using Printf
using ProgressBars
default(fontfamily="computer modern" , titlefontsize=32 , guidefontsize=32 ,
↪  tickfontsize = 22 )
pgfplotsx()
layout2x2 = grid(2,2)
layout3x1 = @layout [ b  c ; a]
size3x1 = (1600,1600)
SIZE = 64
M = testdata(SIZE, SIZE ÷ 5, SIZE /5 , 2)


function test_values(alpha_distribution::Distribution ,
↪  epsilon_distribution::Distribution , M)
    alpha = rand(alpha_distribution)
    eps = max(rand(epsilon_distribution)  ,1e-10)
    relaxed_solver = testgrid(relaxed_multi_solver, M, 2; alpha=alpha,
    ↪  epsilon=eps)
    set_xi_and_psi!(relaxed_solver[1])
    #SMOOTH!(relaxed_solver[1], 100, false)
    for j=1:64
    elyps_solver!(relaxed_solver[1], 2000)
    v_cycle!(relaxed_solver , 1)
    end
    error = norm(relaxed_solver[1].phase .- original_solver[1].phase) /
    ↪  *(size(relaxed_solver[1].phase)...)
    return (;alpha=alpha , epsilon=eps , error=error)
end

original_solver = testgrid(multi_solver, M, 2)
set_xi_and_psi!(original_solver[1])
for j=1:64
v_cycle!(original_solver , 1)
end
#SMOOTH!(original_solver[1], 100, false);
eps = 3e-3
#M = testdata(64, div(64,3), 64/5 , 2)
alpha0 = 10000
epsilon0 = 1e-2
```

```
best_alpha = alpha0 / 10
best_epsilon = epsilon0 / 10
best_error  = Inf
results = DataFrame()
for n=1:1000
    searchradius = 1
    alpha_distribution = Normal(best_alpha , searchradius * alpha0)
    epsilon_distribution = Normal(best_epsilon , searchradius * epsilon0)
    result = test_values(alpha_distribution , epsilon_distribution , M)
    if result.error < best_error
        global best_error = result.error
        global best_alpha = result.alpha
        global best_epsilon = result.epsilon
        println(result)
    end
push!(results , result)
end
jldsave("experiments/alpha-epsilon.jld2"; result=results)
println("Best alpha: $best_alpha , Best epsilon: $best_epsilon")
```

8 Conclusion

In this thesis we have presented a simple introduction to the CH equation and have shown 2 numerical solvers for it. We have presented a baseline method implemented from the authors [1], and have Shown how to derive it from their initial approach. We have done the derivations in a way, that enables a simple adaptation to a modified version of the discrete CH equation Eq.3.9, as introduced in [1]. We have introduced messures to evaluate both solvers in space , time and mass conservation as well as their sub-iteration behaviour. We have shown the baseline to be mass conservative, in a numerical sense, and we have shown it to be stable in all tested measures. We have shown our relaxed solver to approach the baseline, however we have also highlighted instability with subiterations, and massloss. We intentionally didn't evaluate runtime since numerical experiments have shown both solvers to be dependant on the amount of sub-iterations, hyperparameters such as ε as well as the number off smoothing iterations. It would therefore be unfair to evaluate one solver on a set of parameters tweaked for the other. As example for this dilemma we recall runs where the relaxed solver was around 10x faster than the baseline with the same parameters. The baseline solver was able to run with 10x less smoothing iterations than the relaxed one. A fair comparison would hence require to find the optimal number of smoothing for each solver.

For the sake of compleetness we include runtime benchmarks Of both methods. Those should be taken with a pinch of salt because of the reasons above. Both examples are run with the same parameters.

```
Executing... 1a392143
```

BenchmarkTools.Trial: 1 sample with 1 evaluation. Single result which took 8.938 s (3.95% GC) to evaluate, with a memory estimate of 3.36 GiB, over 63995963 allocations.

BenchmarkTools.Trial: 5 samples with 1 evaluation. Range (min … max): 1.030 s … 1.304 s GC (min … max): 3.26% … 2.74% Time (median): 1.068 s GC (median): 3.14% Time (mean ±): 1.128 s ± 111.612 ms GC (mean ±): 2.87% ± 0.34%

1.03 s Histogram: frequency by time

1.3 s <

Memory estimate: 293.88 MiB, allocs estimate: 5013565.

9 Apendix

9.1 Operator implementation

```
function set_xi_and_psi!(solver::T) where T <: Union{multi_solver ,
↪  relaxed_multi_solver}
    xi_init(x) = x / solver.dt
    psi_init(x) = solver.W_prime(x) - 2 * x
    solver.xi[2:end-1, 2:end-1] = xi_init.(solver.phase[2:end-1,2:end-1])
    solver.psi[2:end-1, 2:end-1] = psi_init.(solver.phase[2:end-1,2:end-1])
    return nothing
end
```

9.1.1 baseline

```
function L(solver::multi_solver,i,j , phi , mu)
    xi = solver.phase[i, j] / solver.dt -
         (discrete_G_weigted_neigbour_sum(i, j, solver.potential, G, solver.len,
         ↪  solver.width)
          -
          neighbours_in_domain(i, j, G, solver.len, solver.width) * mu
          ↪  )/solver.h^2
    psi = solver.epsilon^2/solver.h^2 *
          (discrete_G_weigted_neigbour_sum(i, j, solver.phase, G, solver.len,
          ↪  solver.width)
           -
           neighbours_in_domain(i, j, G, solver.len, solver.width) * phi) - 2 *
           ↪  phi + mu
    return [xi, psi]
end
```

```
function dL(solver::multi_solver , i , j)
    return [ (1/solver.dt) (1/solver.h^2*neighbours_in_domain(i,j,G,solver.len ,
    ↪  solver.width));
             (-1*solver.epsilon^2/solver.h^2 *
             ↪  neighbours_in_domain(i,j,G,solver.len , solver.width) - 2) 1]
    end
```

9.1.2 RELAXED

```
function L(solver::relaxed_multi_solver,i,j , phi , mu)
    xi = solver.phase[i, j] / solver.dt -
         (discrete_G_weigted_neigbour_sum(i, j, solver.potential, G, solver.len,
         ↪  solver.width)
          -
          neighbours_in_domain(i, j, G, solver.len, solver.width) * mu
          ↪  )/solver.h^2
    psi = solver.epsilon^2 * solver.alpha*(solver.c[i,j] - phi) -
    ↪  solver.potential[i,j] - 2 * solver.phase[i,j]
    return [xi, psi]
end
```

```
function dL(solver::relaxed_multi_solver , i , j)
    return [ (1/solver.dt) (1/solver.h^2*neighbours_in_domain(i,j,G,solver.len ,
    ↪  solver.width));
             (-1*solver.epsilon^2 * solver.alpha  - 2) 1]
    end
```

9.2 RNG GENERATION

for random point generation we use the folowing Function and seed.

```
using Random
rng = MersenneTwister(42)
gridsize = 64
radius = gridsize /5
blobs = gridsize ÷ 5
rngpoints = rand(rng,1:gridsize, 2, blobs)
```

Executing... 0a3c1295

the random testdata is then generated as follows

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] 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.