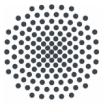


University of Stuttgart
Germany

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

This thesis follows reproducible research philosophy, in that 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. Further details on execution and reading of the original source provided in org-mode format,

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 ϕ, μ are phase-field variable and chemical potential,

$$\begin{aligned} \phi &: \Omega \times [0,T) \to \mathbb{R}^d \\ \mu &: \Omega \times [0,T) \to \mathbb{R}^d \end{aligned} \tag{2.2}$$

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:

$$E^{\text{bulk}}[\phi] = \int_{\Omega} \frac{\varepsilon^2}{2}|\nabla\phi|^2 + W(\phi)\, dx, \tag{2.3}$$

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.

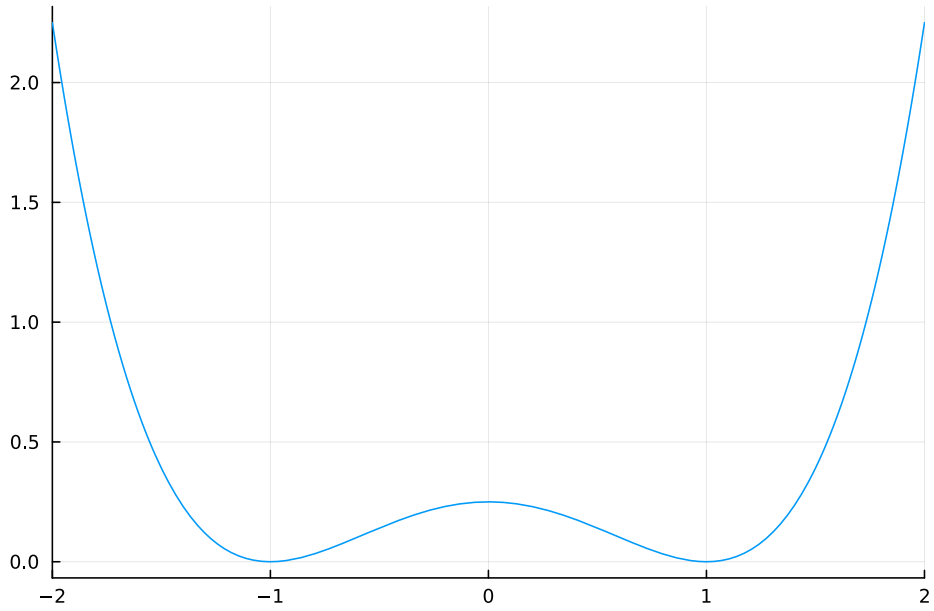


Figure 2.1: Double well potential $W(\phi)$

The chemical potential, μ, then follows as the variational derivation of the free energy 2.3.

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

2.1.2 Derivation of the CH equation from mass balance

The paper [2] motivates us to derive the CH equation as follows:

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

where J is mass flux. The equation 2.4 then ensures continuity of mass Using the no-flux boundary conditions:

$$\begin{aligned} J \cdot n = 0 \quad \partial\Omega \times (0,T) \\ \partial_n \phi = 0 \quad \partial\Omega \times (0,T) \end{aligned} \tag{2.5}$$

where n is the outward normal on $\partial\Omega$. conservation of mass follows see[2].

$$\begin{aligned} \frac{d}{dt}\int_\Omega \phi &= \int_\Omega \frac{\partial \phi}{\partial t} dV \\ &= -\int_\Omega \nabla \cdot J \; dV \\ &= \int_{\partial\Omega} J \cdot n \; dA \\ &= 0 \end{aligned} \tag{2.6}$$

Therefore mass is conserved over time, as shown in 2.6. We define the mass flux, J, as the gradient in chemical potential as follows

$$J = -\nabla \mu \tag{2.7}$$

This results in the CH equation as stated in 2.1.

$$\begin{aligned} -\nabla \mu = 0 \\ \partial_n \phi = 0 \end{aligned} \tag{2.8}$$

i.e. no flow leaves and potential on the border doesn't change. In order to show the CH equation's consistency with thermodynamics we take the time derivation of the free energy 2.3 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:

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

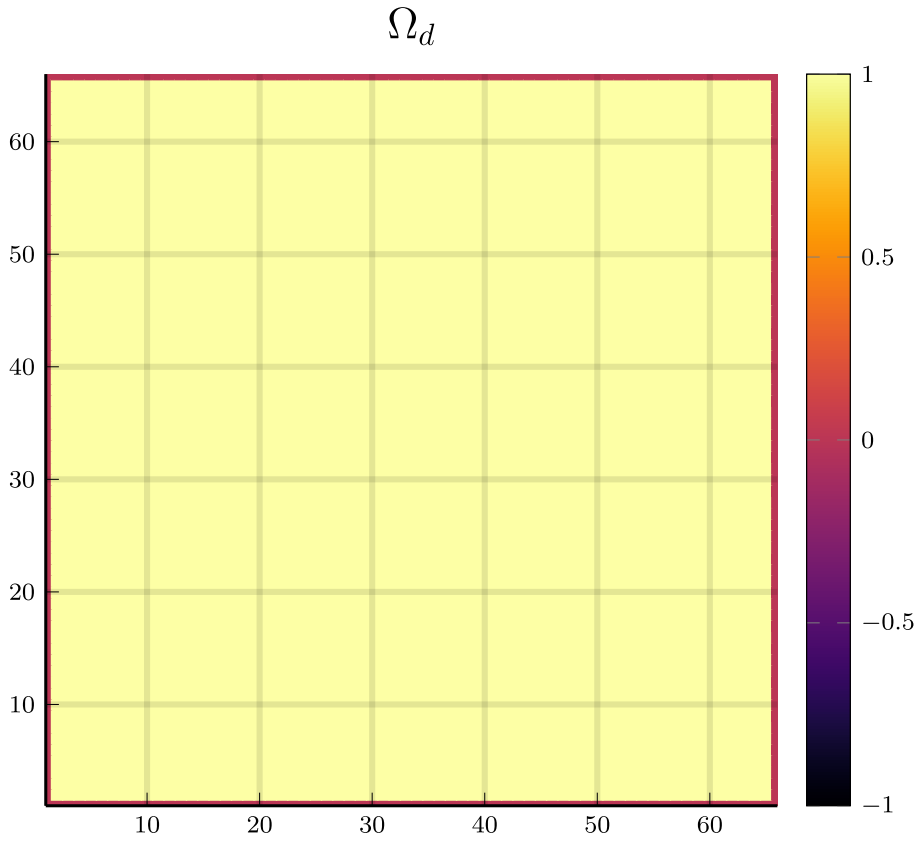


Figure 3.1: Discrete Domain used for most of the experiments in this Thesis

```
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:

$$D_x\phi_{i+\frac{1}{2}j} = \frac{\phi_{i+1j} - \phi_{ij}}{h} \qquad D_y\phi_{ij+\frac{1}{2}} = \frac{\phi_{ij+1} - \phi_{ij}}{h} \tag{3.3}$$

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

$$\nabla_d \phi_{ij} = (D_x \phi_{i+1j},\ D_y \phi_{ij+1}) \tag{3.4}$$

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

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 $\nabla_d \mu_{ij}^{n+\frac{1}{2}}, \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij})$ 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:

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

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:

$$\begin{aligned}
\phi_{ij} &= \begin{cases} 1 & , \|(i,j) - (\frac{N}{2}, \frac{N}{2})\|_p < \frac{N}{3} \\ -1 & , else \end{cases} && \text{where } p \in \{2, \infty\} \\
\phi_{ij} &= \begin{cases} 1 & , i < \frac{N}{2} \\ -1 & , else \end{cases} \\
\phi_{ij} &= \begin{cases} 1 & , \|(i,j) - (\frac{N}{2}, 2)\|_2 < \frac{N}{3} \\ -1 & , else \end{cases} \\
\phi_{ij} &= \begin{cases} 1 & , \|(i,j) - q_k\|_p < \frac{N}{5} \\ -1 & , else \end{cases} && p \in \{1, 2, \infty\}, q_k \in Q
\end{aligned} \tag{3.6}$$

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

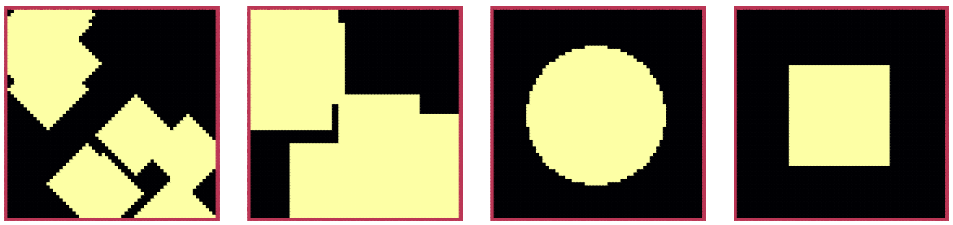


Figure 3.2: Examples of different phase-fields used as the initial condition in this work.

3.3 Numerical ansatz

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

$$\begin{aligned} \frac{\phi_{ij}^{n+1} - \phi_{ij}^{n}}{\Delta t} &= \nabla_d \cdot (G_{ij} \nabla_d \mu_{ij}^{n+\frac{1}{2}}) \\ \mu_{ij}^{n+\frac{1}{2}} &= 2\phi_{ij}^{n+1} - \varepsilon^2 \nabla_d \cdot (G_{ij} \nabla_d \phi_{ij}^{n+1}) + W'(\phi_{ij}^{n}) - 2\phi_{ij}^{n} \end{aligned} \tag{3.7}$$

and derive a numerical scheme from this implicit equation.

3.4 The discrete scheme

The authors in [1] derive their method by separating 3.7 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 3.7 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:

$$L\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix} = \begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} \tag{3.8}$$

SMOOTH consists of point-wise Gauss-Seidel relaxation, by solving 3.8 for all i, j with the initial guess for $\zeta_{ij}^n, \psi_{ij}^n$. Since L is linear we can write 3.8 as

$$\begin{aligned}\begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} &= DL\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\cdot\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix} + \begin{pmatrix}-\frac{1}{h^2}\Sigma_{Gij}\mu_{ij}^{n+\frac{1}{2}}\\ +\frac{\varepsilon^2}{h^2}\Sigma_{Gij}\phi_{ij}^{n+1}\end{pmatrix}\\ \begin{pmatrix}\zeta_{ij}^{n}\\ \psi_{ij}^{n}\end{pmatrix} - \begin{pmatrix}-\frac{1}{h^2}\Sigma_{Gij}\mu_{ij}^{n+\frac{1}{2}}\\ +\frac{\varepsilon^2}{h^2}\Sigma_{Gij}\phi_{ij}^{n+1}\end{pmatrix} &= DL\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\cdot\begin{pmatrix}\phi_{ij}^{n+1}\\ \mu_{ij}^{n+\frac{1}{2}}\end{pmatrix}\end{aligned} \tag{3.9}$$

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 3.9 is of form $b = Ax$ We then and solve 3.9 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
```

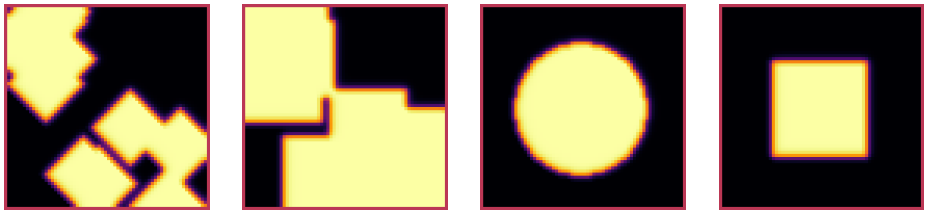


Figure 3.3: inputs from 3.2 after SMOOTH.

3.6 Multigrid method

Notably the sharp interphase of the initial phase-fields has been smoothed, and the values are between $-1, 1$. 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.

1. a Gauss-Seidel relaxation for smoothing Chapter 5.4.
2. restriction and prolongation methods between grids $h \leftrightarrow H$.
3. 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 5.4 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.

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
```

The iteration of the solver is then done as follows

```
for j in 1:timesteps

    set_xi_and_psi!(solvers[1])

    for i = 1:subiterations
```

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

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

4 Numerical evaluation

The analytical CH equation conserves mass 2.4 and the free energy ,E_{bulk}, 2.3 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:

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

Since the continous Helmholtz energy 2.3.

```
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
```

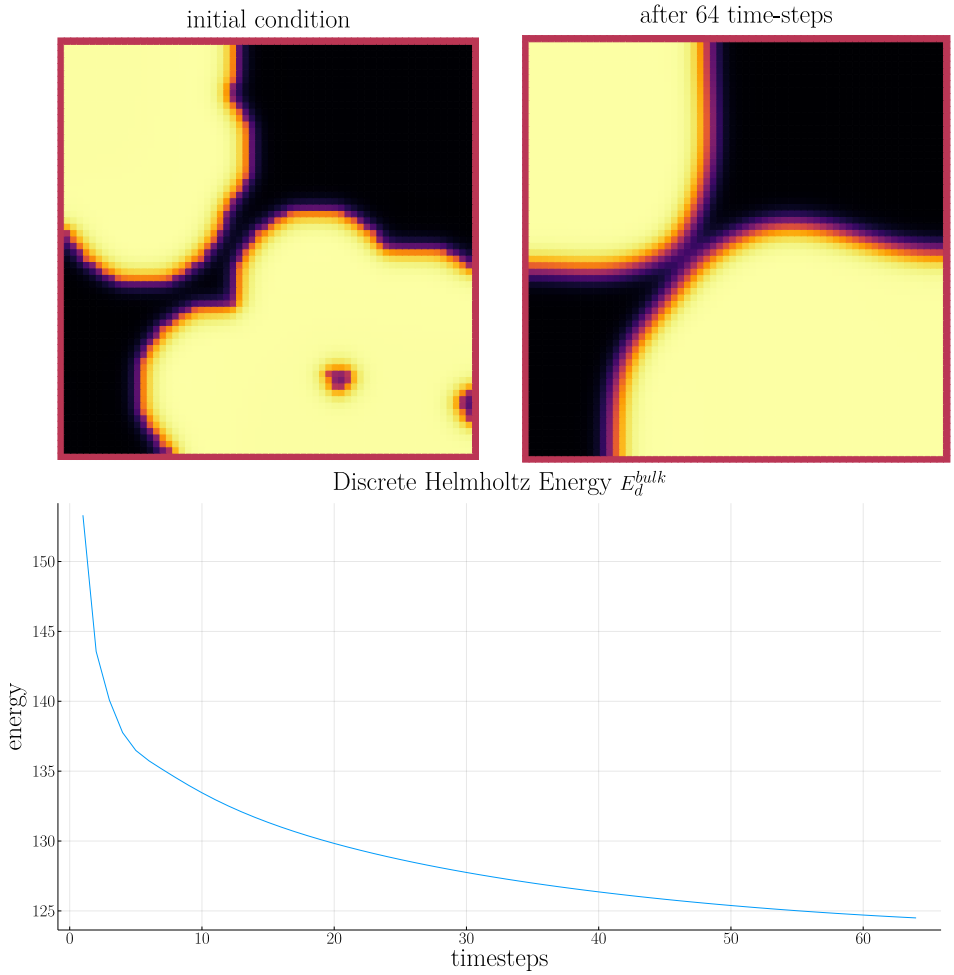


Figure 4.1: behaviour of energy E_{bulk} over time for one initial condition ϕ_0.

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 balance

Instead of a physical mass we use the average of ϕ over the domain Ω. We calculate this balance between *phase 1* and *phase 2* as follows:

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

such that $b = 1$ means there is only phase 1, $\phi \equiv 1$, and $b = -1$ means there is only phase 2, $\phi \equiv -1$.

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

The baseline solver manages a massbalance close to machine precision for our test cases. Mass loss and phase change is therefore negligible.

4.3 stability of a multigrid sub iteration

in order to evaluate convergence we observe the change in phase

$$\|\phi^n - \phi^{n+1,m}\|_{Fr} \tag{4.1}$$

where $\|\cdot\|_{Fr}$ is the Frobenious norm defined by

$$\|f\| = \sqrt{\sum_{i,j\in\Omega_d} f_{ij}^2} \tag{4.2}$$

over the tensors representing $\phi.^n, \phi^{n+1,m}$. we expect our solver to converge if we do more sub-iterations. To test this we compare the phase-field $\phi_{ij}^{n+1,m-1}$ after $m-1$ sub-iterations with the phase-field $\phi_{ij}^{n+1,m}$ after m sub-iterations. As sub-iterations increase , $m \to \infty$ we expect the difference between both phase-fields to go to zero $\|\phi^{n+1,m} - \phi^{n+1,m-1}\|_{Fr} \to 0$

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.

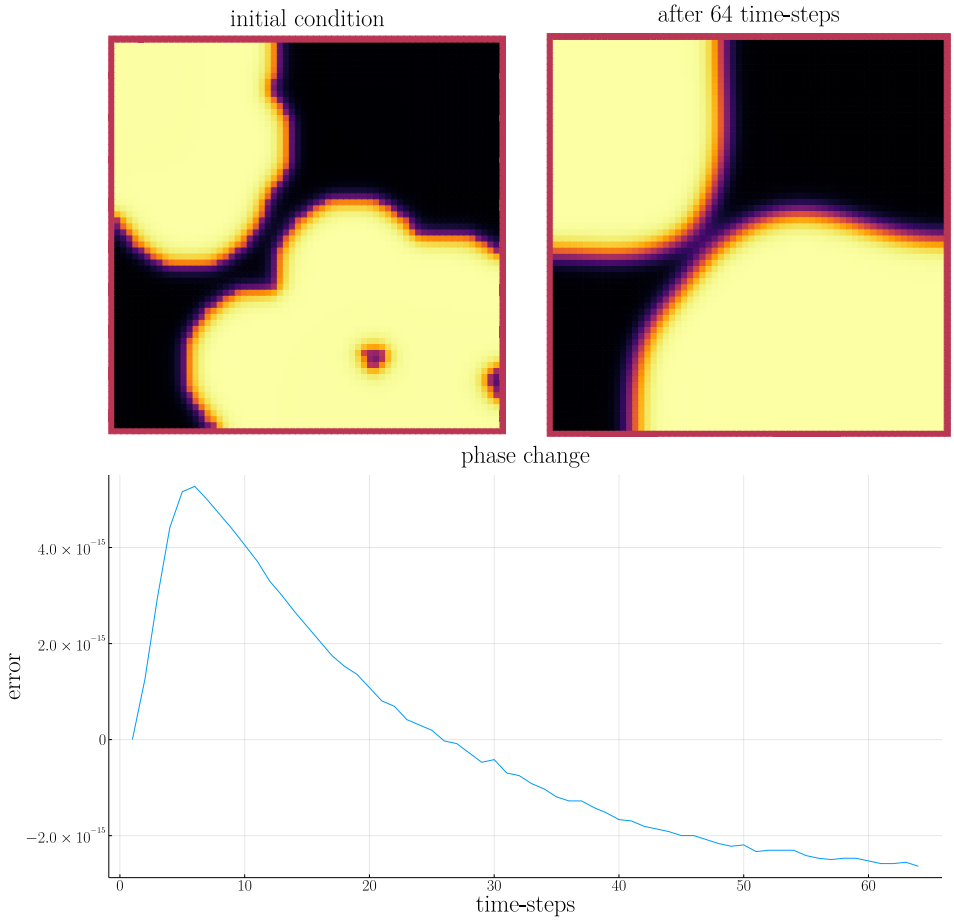


Figure 4.2: behaviour of phase change over time for one initial condition ϕ_0.

4.4 STABILITY UNDER REFINMENT IN TIME

we test the bahaviour unter refinement in time by succesivly subdeviding the original time interval $[0, T]$ in finer parts

In this experiment we ran our solver on a fixed length time intervall ie. we ran our solver for one time.step with $\Delta t = 10^{-2}$, for two time-steps with $\Delta t = \frac{10^{-2}}{2}$ and so on until 64 time-steps with $\Delta t = \frac{10^{-2}}{64}$.

4.5 STABILITY UNDER REFINEMENT 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

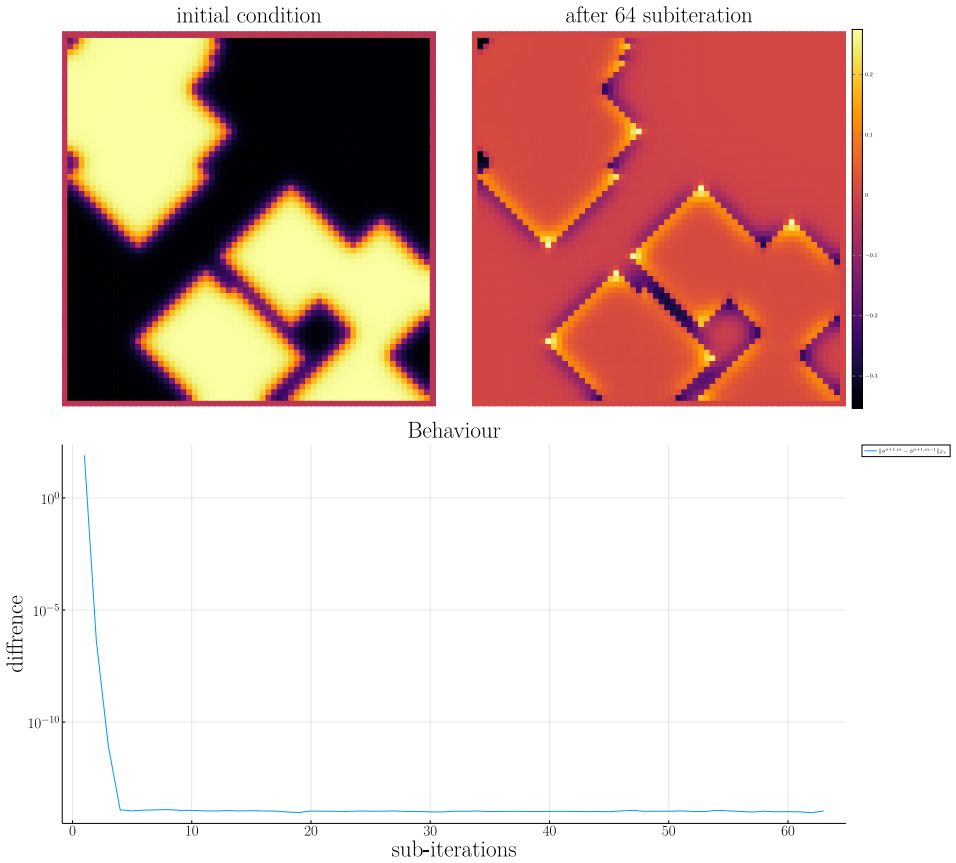


Figure 4.3: stability of the original CH solver for increasing sub-iterations

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 $Nh = 10^{-3} \cdot 1024$

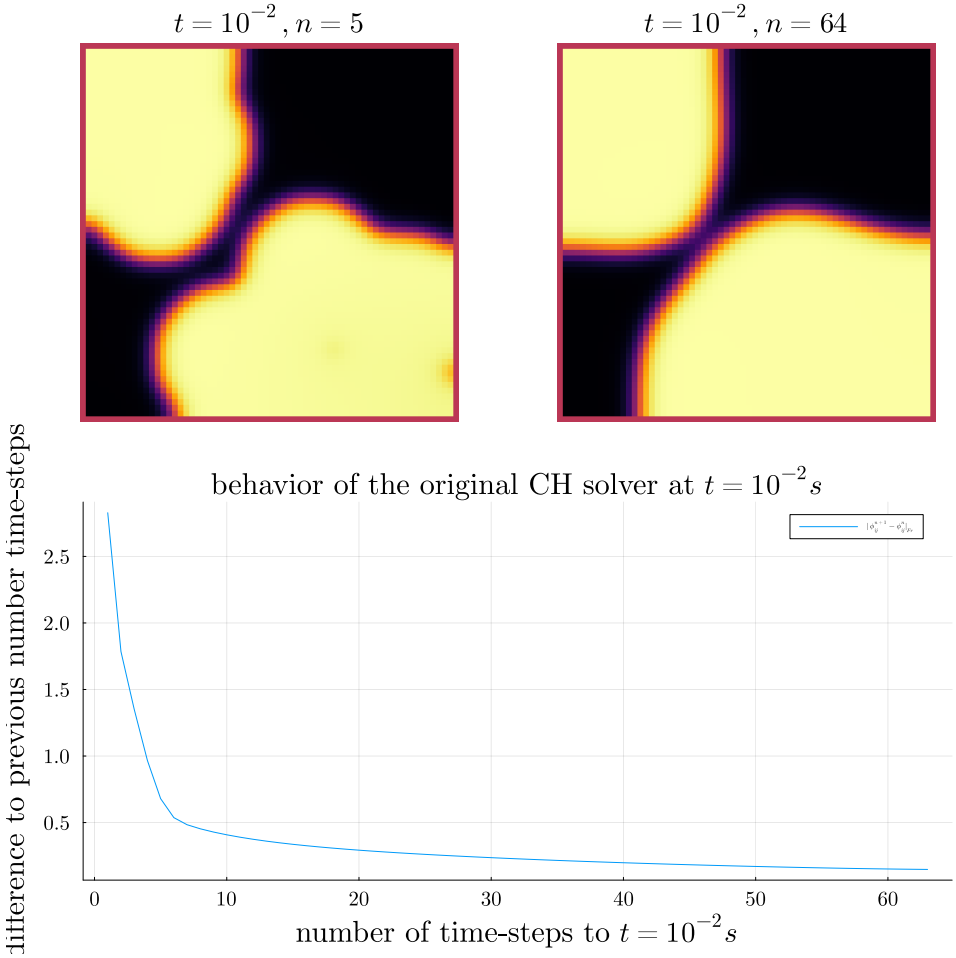


Figure 4.4: behavior of the baseline solver while solving the time interval $T = \left[0, 10^{-2}\right]$ with increasing number of time-steps.

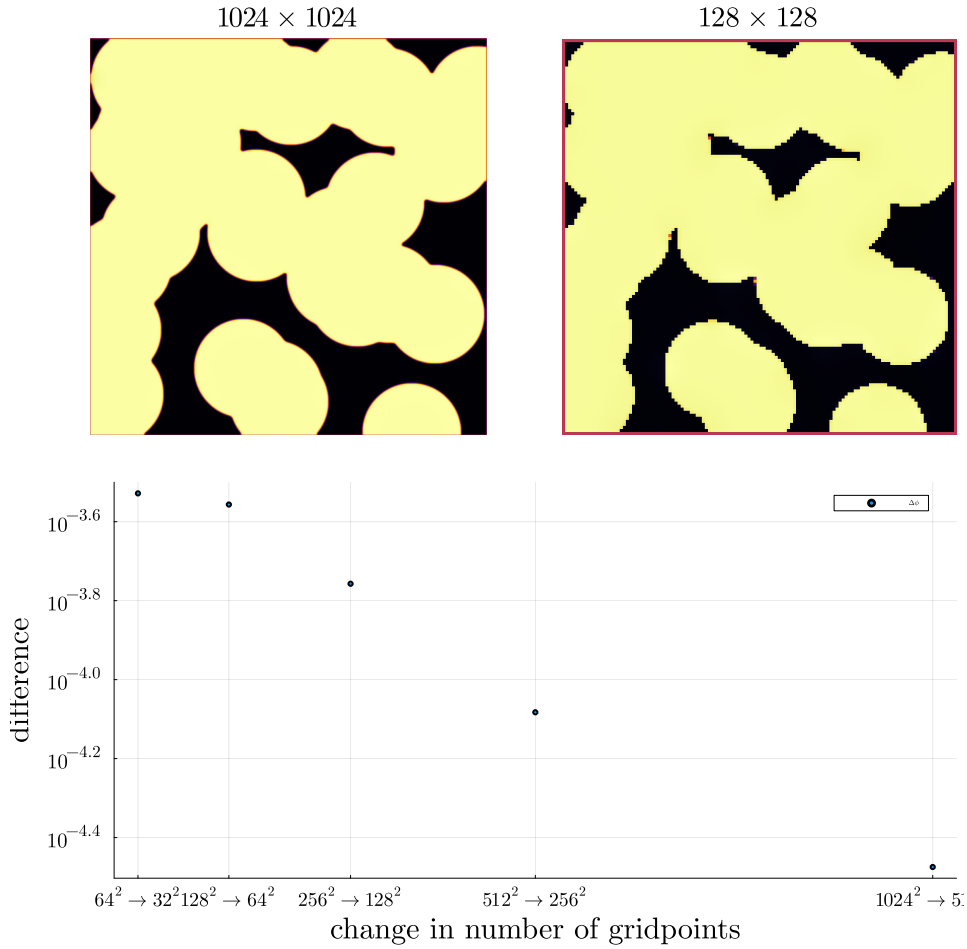


Figure 4.5: behavior of the baseline solver while solving on successively finer grids

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 2.1 as $\alpha \to \infty$. This results in the following relaxation for the classical CH equation 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 3.7 adapted to the relaxed CH equation 5.3. We then adapt the multi-grid solver proposed in 3 to the relaxed problem by replacing the differential operators by their discrete counterparts as defined in 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 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 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 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 3.6. 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.

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

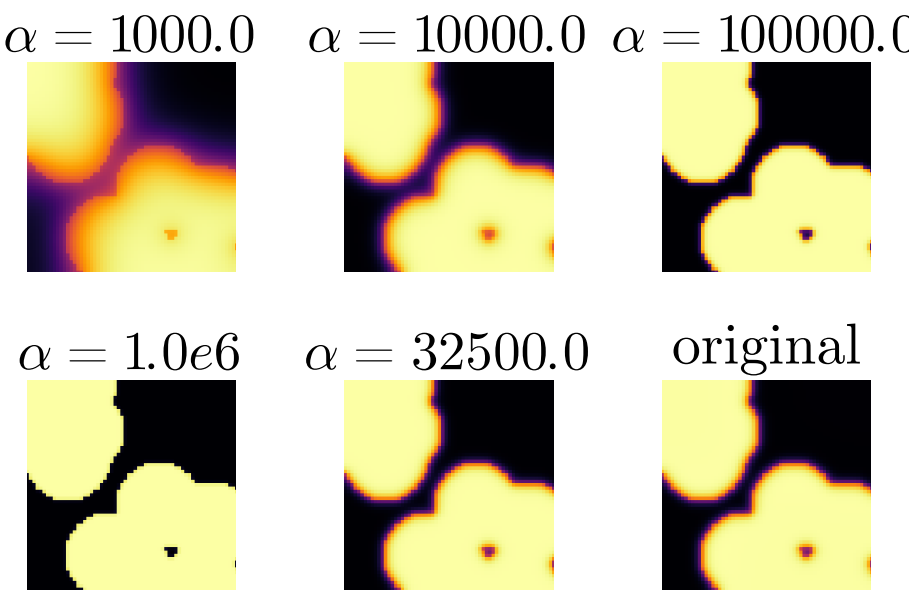


Figure 5.1: effect of the relaxed SMOOTH operator, and additional solving of the elliptical problem, for different values of alpha

```
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)
```

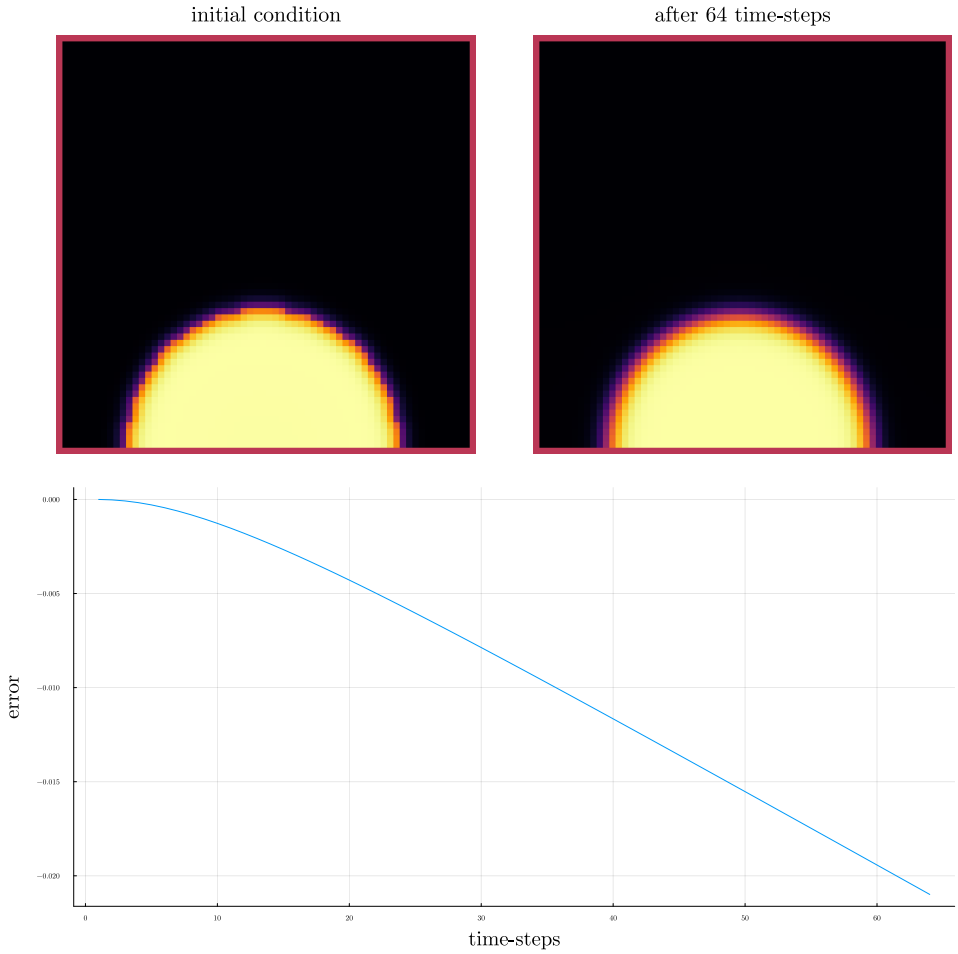
```
    eps = max(rand(epsilon_distribution)  ,1e-10)
    relaxed_solver = testgrid(relaxed_multi_solver, M, 2; alpha=alpha,
    ↪  epsilon=eps)
    set_xi_and_psi!(relaxed_solver[1])
    #SMOOTH!(relaxed_solver[1], 100, false)
    for j=1:64
    elyps_solver!(relaxed_solver[1], 2000)
    v_cycle!(relaxed_solver , 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")
```

sadly the MCMC didn't yield results consistent with the original solver after a few Iterations

6 RATE OF STABILITY

6.1 MASSBAL



6.2 ENERGY

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

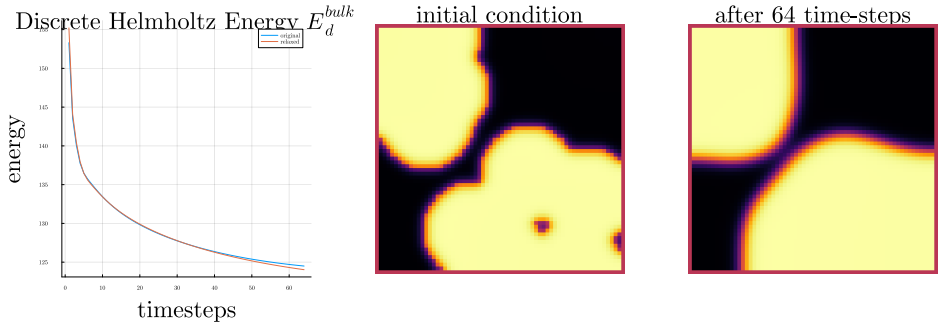
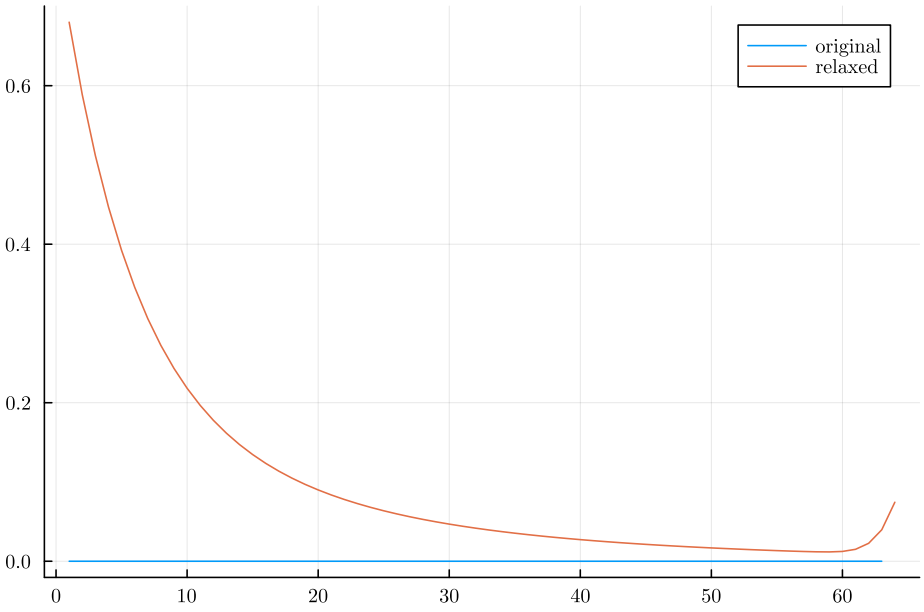


Figure 6.1: energy decay of the relaxed solver compared to the original solver.

6.3 CONVERGENCE OF A SUB ITERATION V-CYCLE



6.4 CONVERGENCE UNTER REFINMENT IN TIME

we test the bahaviour unter refinement in time by succesivly subdeviding the original time interval $[0, T]$ in finer parts

6.5 CONVERGENCE UNDER REFINEMENT IN SPACE

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

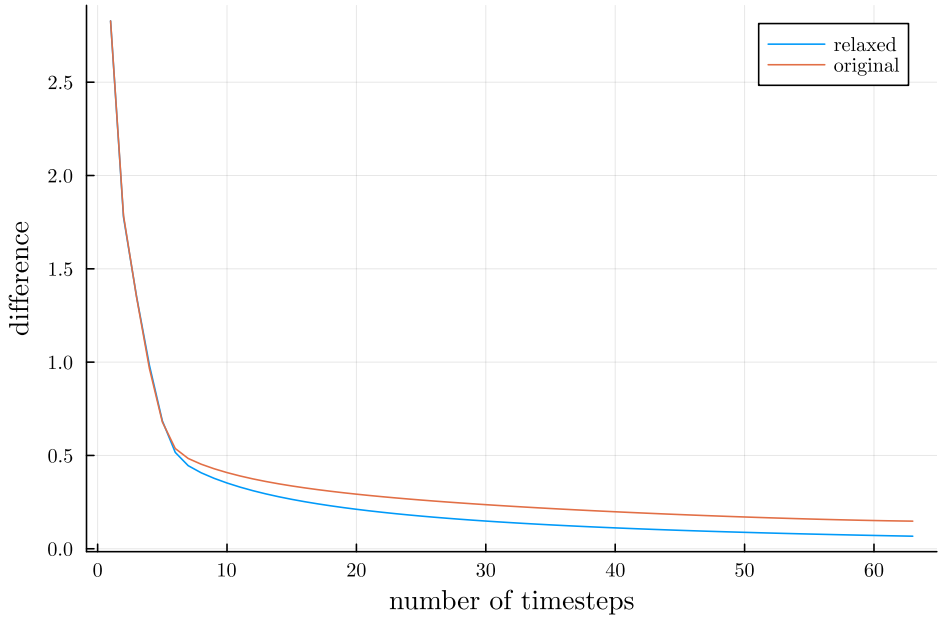
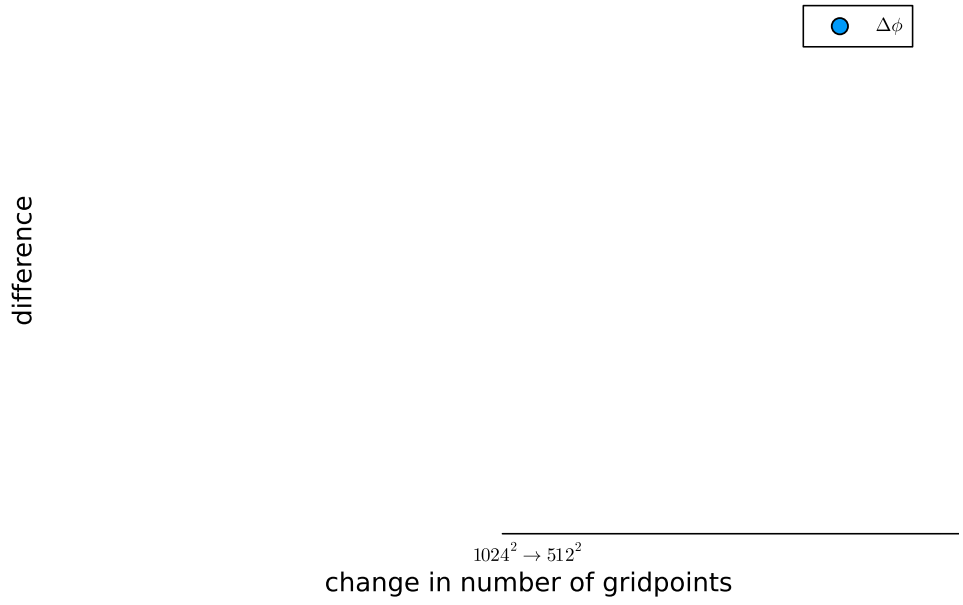


Figure 6.2: behavior of both solvers while solving the time interval $T = \left[0, 10^{-2}\right]$ with increasing number of timesteps



7 Comparison

Furthermore we expect the approximation for ϕ_{ij}^{n+1} to converge.

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

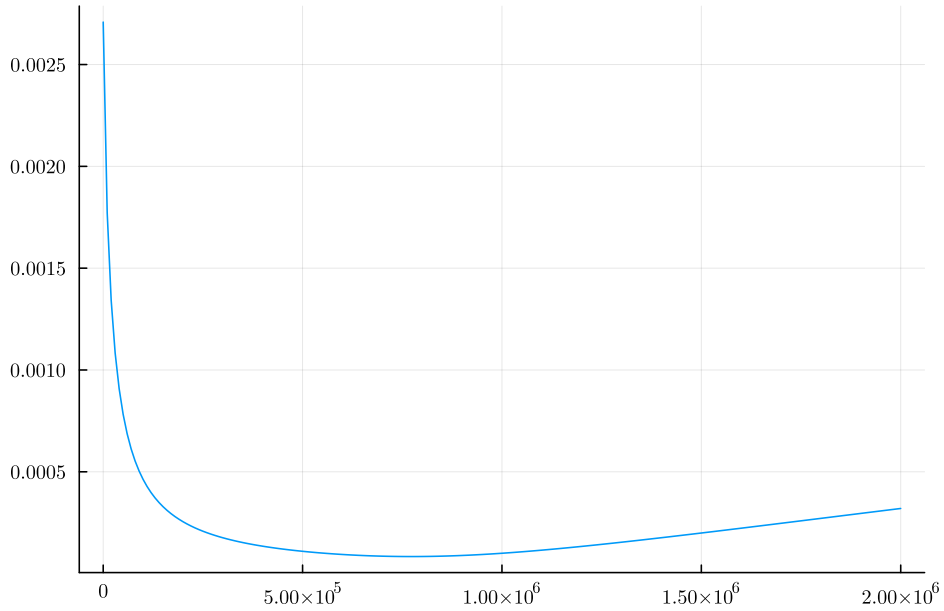
In practice we observe the following behaviour:

```
<<init>>
using JLD2
using Distributed
using ProgressBars
using DataFrames

original_grid = testgrid(multi_solver, M, 2)
alphas = 0:1e4:2e6

function alpha_error(alpha::Number , solution::Array )
    test_solver  = testgrid(relaxed_multi_solver, M, 2, alpha=alpha)
    set_xi_and_psi!(test_solver[1])
    for j in 1:64
        elyps_solver!(test_solver[1], 1000)
        v_cycle!(test_solver , 1)
    end
return [(;alpha=alpha , error=norm(test_solver[1].phase - solution))]
end
set_xi_and_psi!(original_grid[1])
for j in 1:64
    v_cycle!(original_grid, 1)
end
print("finished original v_cycle")
tasks = []
for alpha in alphas
    t = Threads.@spawn alpha_error(alpha , original_grid[1].phase)
    push!(tasks , (alpha=alpha , task = t))
end
result = DataFrame()
```

```
for task in ProgressBar(tasks)
    append!(result , fetch(task.task) )
    end
jldsave("experiments/alpha.jld2"; result)
```



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 does not converge.

images/relaxed-comp.gif

although we can observe slight diffrences between the original solver and the relaxed approach they are barely noticalble by eye. Therefore we also show the numerical difference between both. images/relaxed-comparison.gif

Comparing both solvers visually yields little to no difference. Numerically we observe small discrepancies. However with values around 10^{-3} after 64 time-steps we are still far from the maximum error of 4, which would correspond to the inverse of the original phase.

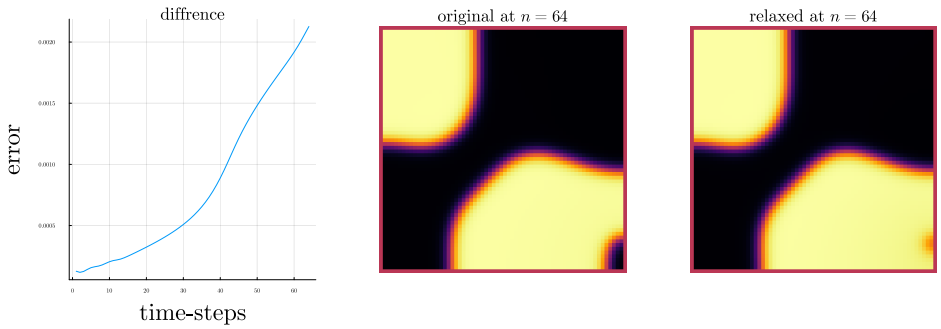


Figure 7.1: comparison between the original and the relaxed CH solvers.

8 Apendix

8.1 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
```

8.1.1 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
```

8.1.2 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
```

8.2 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... 3d1c840c

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.