Aalto University
Department of Applied Physics
COMP Center of Excellence
Multiscale Statistical Physics

Improved mechanical equilibration algorithms for phase field crystal modelling

Kristjan Eimre

Supervisors: Vili Heinonen
Cristian Vasile Achim
Tapio Ala-Nissilä

September 18, 2016

Contents

1 Introduction

1.1 Motivation

Mechanical and electronic properties of engineered materials depend strongly on the microstructure of the materials. Most civil engineering projects require high strength steels, with a specific microstructure containing the right amount and size of crystal grains and dispersions of hard and soft phases. In aerospace applications, where strength to weight ratio is very important, lighter metals are strengthened by introducing grains of heavier elements into their microstructure. Crystal defects and impurities also play an important role in the electronic properties of materials.

Microstructure develops during solidification, solid state precipitation, and thermo-mechanical processing and depends on the environmental conditions during these processes. Figure 1.1 shows molten steel, which was cooled by a cool wall at the left side. In the left panel, cooling was considerably slower than in the right panel and the secondary dendrites are considerably larger. The mechanical properties depend strongly on the size of the secondary dendrites. Realistic modeling of these processes under different environmental conditions allows to predict and fine-tune material properties to suit a given application. This work considers the phase field and phase field crystal methods, which are used for modelling solidification processes.

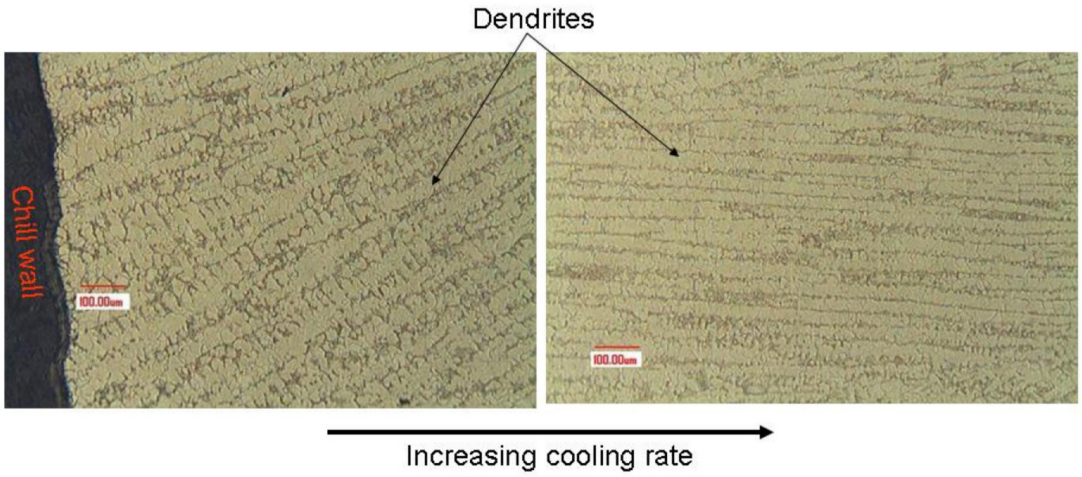


Figure 1.1: Dendrites in a steel alloy. In both cases, the solidification occurred from left to right. The steel was cooled much more rapidly on the right side and the dendritic structure is clearly different. Figure taken from [1].

1.2 Background and purpose of the project

This project was carried out during the author's summer internship at the Multiscale Statistical Physics group of Academy of Finland's Center of Excellence in Computational Nanoscience research (COMP) at the Department of Applied Physics, Aalto University.

The main objective was to improve the numerical performance of an optimization algorithm, namely mechanical equilibration of the phase field crystal model. The performance of the existing numerical algorithms was deemed too weak for practical purposes calling for a significant speedup.

1.3 Acknowledgements

I would like to thank Vili Heinonen and Cristian Achim for introducing me to the theoretical and computational side of phase field modelling. I am especially grateful to Cristian for all the comments and help regarding code development. I would also like to thank Tapio for invaluable comments and discussions about the work.

Additionally, I am grateful to Aalto Science Institute and especially Jaako Järvinen and Pekka Orponen for organizing the summer internship program and enabling me to participate. And finally, I would like to thank other AScI interns, especially Kajetan Stanski for interesting discussions throughout the summer.

2 Overview of the phase field crystal model

2.1 Phase field methods

Phase field methods [2] are used to model systems involving an interface between multiple phases, such as solidification or spinodal decomposition. Phase field systems are described through an order parameter field $\phi(\boldsymbol{r})$, which describes the phase of the system at point $\boldsymbol{r}$. Two different constant values of $\phi(\boldsymbol{r})$ denote two different phases, whose interface is described in a smooth interpolated manner. See Figure 2.1 for an example of a one dimensional liquid-solid system, where the left side is in the solid phase and the right side is in the liquid phase.

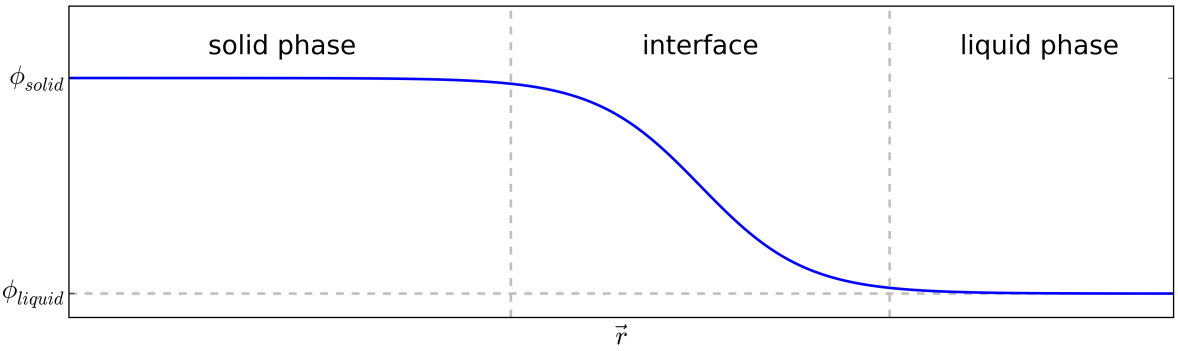


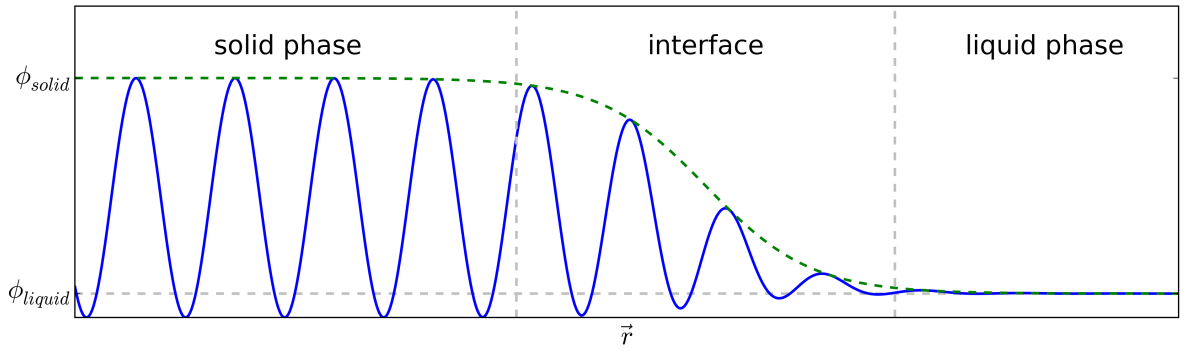
Figure 2.1: A schematic of a one-dimensional phase field of a liquid–solid system. The left side is in the solid and the right side is in the liquid phase.

Another important quantity in the phase field framework is the free energy – a functional of the order parameter field $\phi(\boldsymbol{r})$. A simple free energy functional, which describes phase transformations can be modified from the Landau-Ginzburg free energy by adding a symmetry breaking term and a surface term.[1] It is given by

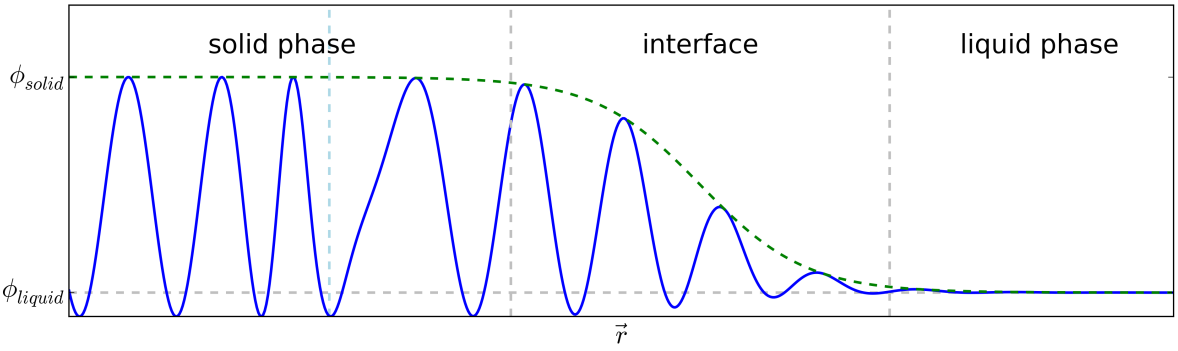
$$F[\phi(\boldsymbol{r})] = \int \left[\frac{1}{4}\phi(\boldsymbol{r})^4 - \frac{\tau}{3}\phi(\boldsymbol{r})^3 + \frac{\Delta B}{2}\phi(\boldsymbol{r})^2 + \frac{B^x}{2}|\nabla\phi(\boldsymbol{r})|^2 \right] \mathrm{d}\boldsymbol{r}, \tag{2.1}$$

where the parameters τ, ΔB and B^x depend on the system characterstics and thermodynamic conditions.

[1] See Eq. (2.50) in Ref. [3].



(a)



(b)

Figure 2.2: Schematic of a one-dimensional PFC system. Each peak in the crystal phase corresponds to one atom. Panel (a) shows a relaxed crystal, while in Panel (b) one atom is displaced to the left from its equilibrium position.

2.2 Phase field crystal model

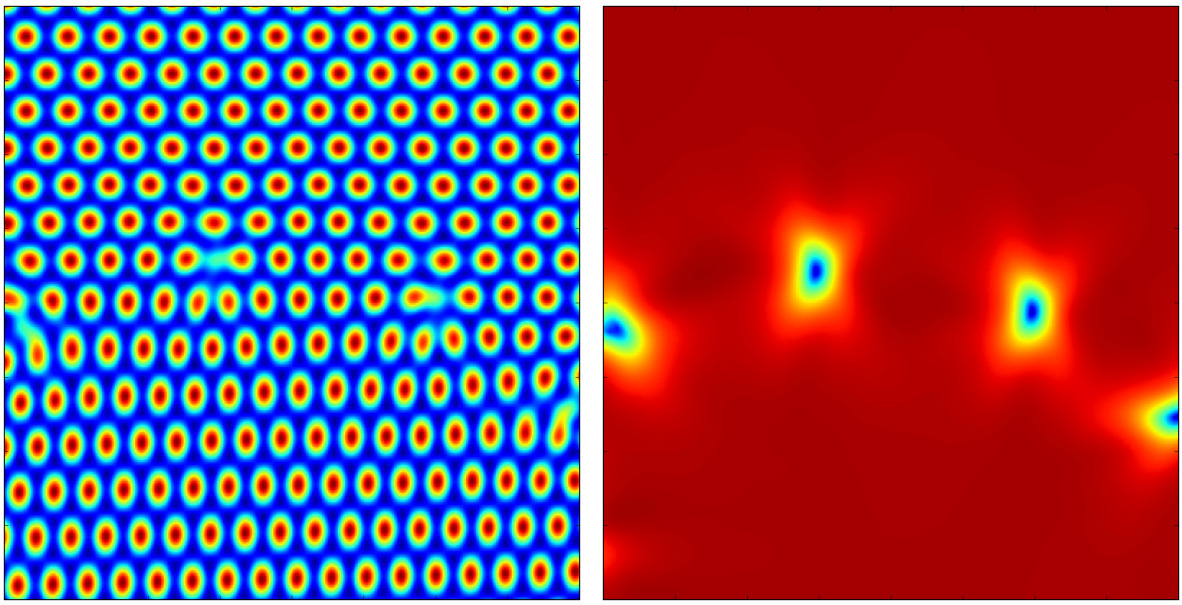
In the phase field crystal (PFC) model [1, 3], the phase field energy functional (Eq. 2.1) is modified such that the energy is minimized by a periodic order parameter field $\psi(\boldsymbol{r})$ in the crystal phase, where each peak of the periodic function ψ corresponds to one atom. This will allow the model to describe elasticity, plasticity and topological defects in the crystal in addition to solid-liquid ordering. The modified energy functional is given by [2]

$$F[\psi(\boldsymbol{r})] = \int \left[\frac{\nu}{4}\psi(\boldsymbol{r})^4 - \frac{\tau}{3}\psi(\boldsymbol{r})^3 + \frac{\Delta B}{2}\psi(\boldsymbol{r})^2 + \frac{B^x}{2}\psi(\boldsymbol{r})(1+\nabla^2)^2\psi(\boldsymbol{r}) \right] d\boldsymbol{r}. \tag{2.2}$$

Figure 2.2 shows the PFC equivalent of the phase field system shown in Figure 2.1. The PFC approach allows to describe the elastic excitation of the system by introducing a (spatially-varying) phase shift of the periodic field ψ. Additionally, in 2D and 3D cases, topological defects, such as line dislocations can be represented[3]. As an example see Figure 2.3 showing multiple line dislocations at a grain boundary.

[2] See Eq. (2.55) in Ref. [3].

[3] Vacancies are present in the phase field crystal framework. They are not confined on lattice sites and hop around until they are diffused throughout the system, which results in a varying average density.



(a) (b)

Figure 2.3: A 2D system with two grains and multiple line dislocation cores at the grain boundary. Panel (a) shows ψ (the 2D equivalent of the blue line in Fig. 2.2) and Panel (b) shows the envelope or the amplitude corresponding to the dashed green line in Fig. 2.2 (more specifically the sum of the magnitudes of η_j, see Section 2.3).

2.3 Amplitude expansion of the PFC model

The idea of the amplitude expansion of the PFC model [1, 3] is to describe the system through the envelope or in other words, the amplitude of the periodic phase field $\psi(\boldsymbol{r})$ (see Figures 2.2 and 2.3 for ψ and its envelope). The advantage is that the envelope function varies spatially much less rapidly than the PFC density ψ, which varies at atomistic length scales allowing, for example, to use a considerably sparser numerical mesh for the numerical calculations.

The PFC density can be approximated as

$$\psi(\boldsymbol{r}) \approx \rho(\boldsymbol{r}) + \sum_j [\eta_j(\boldsymbol{r}) e^{i\boldsymbol{q}_j \cdot \boldsymbol{r}} + \text{C.C.}], \tag{2.3}$$

where $\rho(\boldsymbol{r})$ is a slowly varying "average" density[4], $\eta_j(\boldsymbol{r})$ are complex amplitudes of the first modes of the periodic $\psi(\boldsymbol{r})$ and $\boldsymbol{q}_j$ are the primitive reciprocal lattice vectors. I choose

$$\boldsymbol{q}_1 = (-\sqrt{3}/2, -1/2),\ \boldsymbol{q}_2 = (0, 1),\ \boldsymbol{q}_3 = (\sqrt{3}/2, -1/2), \tag{2.4}$$

for the reciprocal lattice vectors $\boldsymbol{q}_j$, corresponding to 2D hexagonal crystal symmetry.

The system is described by the complex amplitudes

$$\eta_j(\boldsymbol{r}) = \phi_j(\boldsymbol{r}) \exp(i\theta_j(\boldsymbol{r})). \tag{2.5}$$

The magnitudes of the fields ϕ_j determine the magnitude of the oscillating mode corresponding to $\boldsymbol{q}_j$. The fields ϕ_j will be constant in perfect crystalline state and will be

[4] In this brief overview it is assumed that this is constantly zero, see [3] for complete description.

reduced in case of crystal defects (e.g. dislocation cores). In the liquid state the fields ϕ_j become zero. The complex phases $\theta_j(\boldsymbol{r})$ describe[5] the displacement field $\boldsymbol{u}(\boldsymbol{r})$ as

$$\theta_j(\boldsymbol{r}) = -\boldsymbol{q}_j \cdot \boldsymbol{u}(\boldsymbol{r}) + \frac{1}{3}\Delta\theta, \tag{2.6}$$

where $\Delta\theta = \sum_j \theta$ and is nonzero only near crystal defects.

The energy functional of PFC defined by Eq. (2.2) in terms of the complex amplitudes η_j is given by[6]

$$F[\eta_j(\boldsymbol{r})] = \int \left\{ \frac{\Delta B}{2} A^2 + \sum_{j=1}^{3} B^x |\mathcal{G}_j \eta_j|^2 - 4\,\mathrm{Re}\left(\prod_{j=1}^{3} \eta_j \right) + \frac{3\nu}{4} A^4 - \frac{3\nu}{2} \sum_{j=1}^{3} |\eta_j|^4 \right\} d\boldsymbol{r}, \tag{2.7}$$

where $A^2 = 2\sum_j |\eta_j|^2$ and $\mathcal{G}_j = \nabla^2 + 2i\boldsymbol{q}_j \cdot \nabla$.

2.4 Dynamics and mechanical equilibrium

The dynamics of the statistical fields describing the system is assumed to be first order dissipative driving the system to its minimum energy configuration. For the PFC density $\psi(\boldsymbol{r}, t)$, the time evolution is given by[7]

$$\frac{\partial \psi(\boldsymbol{r}, t)}{\partial t} = \nabla^2 \frac{\delta F[\psi(\boldsymbol{r}, t)]}{\delta \psi(\boldsymbol{r}, t)}. \tag{2.8}$$

The corresponding time evolution for the complex amplitudes η_j can be written as[8]

$$\frac{\partial \eta_j(\boldsymbol{r}, t)}{\partial t} = -\frac{\delta F[\eta_j(\boldsymbol{r}, t)]}{\delta \eta_j^*(\boldsymbol{r}, t)}, \tag{2.9}$$

where

$$\frac{\delta F[\eta_j(\boldsymbol{r}, t)]}{\delta \eta_j^*(\boldsymbol{r}, t)} = (\Delta B + B^x \mathcal{G}_j^2)\eta_j - 2\tau \prod_{i \neq j} \eta_j^* + 3\nu(A^2 - |\eta_j|^2)\eta_j. \tag{2.10}$$

One big advantage of the PFC model is that it can describe elastic excitations of the crystal. Elastic excitations relax through emitting phonons (vibrations of the crystal lattice), but it is impossible to describe this relaxation with first order diffusive dynamics (given by Eq. (2.9)). One way to take this into account is to assume that the system is in mechanical equilibrium at all times. This is a good assumption, as mechanical relaxation happens much faster than diffusive processes. This translates into a mechanical equilibrium constraint

$$\frac{\delta F(t)}{\delta \boldsymbol{u}} = 0, \tag{2.11}$$

which must be satisfied at all times. The field $\boldsymbol{u}(\boldsymbol{r}, t)$ is the displacement field, which can be expressed through the complex phases θ_j, transforming the mechanical equilibrium

[5] Remember, as illustrated in Figure 2.2, the displacement of an atom was described by a local phase shift in the periodic function ψ.

[6] Eq. (2.89) in [3].

[7] ∇^2 is needed to ensure that the density is conserved locally.

[8] Eq. (3.36) in [3].

condition into the following form[9]

$$\boldsymbol{q}_j \cdot \sum_{k=1}^{3} \boldsymbol{q}_k \,\mathrm{Im}\left(\eta_k^*(\boldsymbol{r},t)\frac{\delta F}{\delta \eta_k^*(\boldsymbol{r},t)}\right) = 0. \tag{2.12}$$

[9]This formulation also takes care of keeping $\Delta\theta = \sum_j \theta$ constant during the equilibration procedure.

3 Numerical methods and simulations

3.1 Time-stepping for PFC dynamics

Solving Eq. (2.9) requires the use of a time-stepping scheme. Explicit time-stepping schemes find the solution at the following moment directly by using the present state. Implicit schemes require preconditioning, i.e. manipulation of the discrete equation before iteration. Implicit methods are often preferred for their better numerical stability. I introduce a semi-implicit time-stepping scheme previously used in the literature [3].

The partial differential equation (2.9) can be expressed as

$$\frac{\partial \eta_j}{\partial t} = -\frac{\delta F}{\delta \eta_j^*} = -(\mathcal{L}\eta_j + \mathcal{N}(\eta_j)), \tag{3.1}$$

where the right-hand side is divided into a linear operator $\mathcal{L} = \Delta B + B^x G_j^2$ and a nonlinear operator $\mathcal{N}(\eta_j) = -2\tau \prod_{i \neq j} \eta_i^* + 3\nu(A^2 - |\eta_j|^2)\eta_j$.

To numerically solve the differential equation, the linear part is treated implicitly and the nonlinear term explicitly giving

$$\frac{\eta_j(t+\Delta t) - \eta_j(t)}{\Delta t} = -(\mathcal{L}\eta_j(t+\Delta t) + \mathcal{N}(t)). \tag{3.2}$$

Using the Fourier transform of the operator G_j

$$\mathcal{F}[G_j] = \mathcal{F}[\nabla^2 + 2i\boldsymbol{q}_j \cdot \nabla] = -\boldsymbol{k}^2 - 2\boldsymbol{q}_j \cdot \boldsymbol{k} \tag{3.3}$$

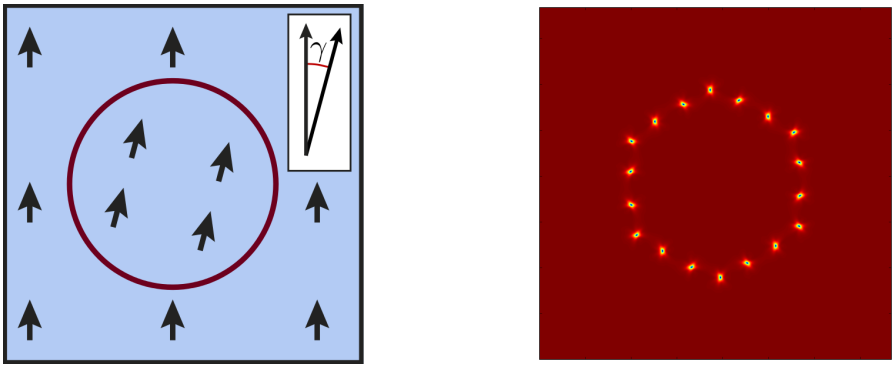
the time stepping scheme given by Eq. (3.2) can be simplified as

$$\begin{aligned} \frac{\hat{\eta_j}(t+\Delta t) - \hat{\eta_j}(t)}{\Delta t} &= -\hat{\mathcal{L}}\hat{\eta_j}(t+\Delta t) - \hat{\mathcal{N}}(t); \\ (1 + \Delta t\hat{\mathcal{L}})\hat{\eta_j}(t+\Delta t) &= \hat{\eta_j}(t) - \Delta t\hat{\mathcal{N}}(t); \\ \hat{\eta_j}(t+\Delta t) &= \frac{\hat{\eta_j}(t) - \Delta t\hat{\mathcal{N}}(t)}{1 + \Delta t\hat{\mathcal{L}}}, \end{aligned} \tag{3.4}$$

transforming the problem to a first order recurrence relation.

3.2 Mechanical equilibrium constraint as a numerical optimization problem

When numerically evolving the phase field crystal system, the mechanical equilibration constraint given by Eq. (2.12) can be formulated as a numerical optimization problem. A numerical optimization algorithm is used to modify the phases $\theta_j(\boldsymbol{r})$ such that the constraint holds. In practice, this optimization can be done with an interval of a number of time steps without a significant loss of accuracy. See Chapter 4 for more details about the numerical optimization algorithms.



(a) (b)

Figure 3.1: Initial condition for the grain rotation calculation. (a) System schematic for the grain rotation. A circle is rotated by an angle γ inside a perfect lattice. Figure taken from [3]. (b) The magnitude $\sum |\eta_j|$ for the initial condition after 10000 PFC time steps (given by Eq. 3.4).

3.3 Evaluating energy and mechanical equilibrium constraint

For the numerical optimization algorithms and analysis, energy F given by Eq. (2.7) and the mechanical equilibrium constraint defined by Eq. (2.12) and Eq. (2.10) need to be numerically evaluated. In both of the expressions, most of the terms can be evaluated directly from $\eta_j(\boldsymbol{r})$ with the exception of the terms containing $\mathcal{G}_j = \nabla^2 + 2i\boldsymbol{q}_j \cdot \nabla$. These terms are evaluated in the Fourier space as

$$\mathcal{F}[\mathcal{G}_j \eta_j] = (-\boldsymbol{k}^2 - 2\boldsymbol{q}_j \cdot \boldsymbol{k})\hat{\eta}_j, \tag{3.5}$$

$$\mathcal{F}[\mathcal{G}_j^2 \eta_j] = (-\boldsymbol{k}^2 - 2\boldsymbol{q}_j \cdot \boldsymbol{k})^2 \hat{\eta}_j. \tag{3.6}$$

In principle, the terms containing $\mathcal{G}_j$ can also be evaluated by finite difference formulas directly in real space, which can be computationally faster, but the drawback is reduced accuracy. This was investigated in Appendix A and the reduced precision turned out to be detrimental to the optimization algorithms. For the model system, spectral methods were superior to real space methods in all regards.

3.4 Simulation of a rotated grain

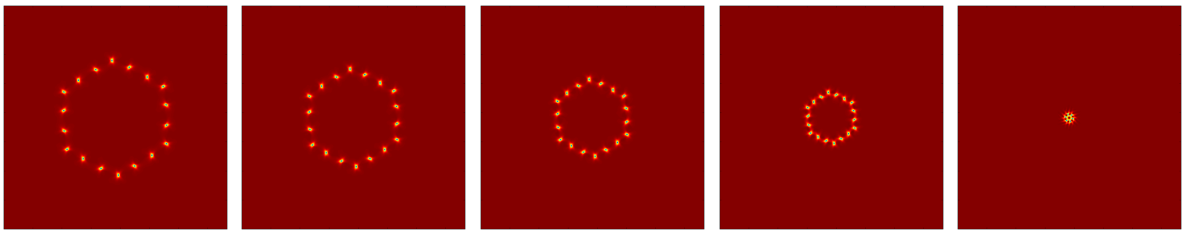
I tested the computer codes and the optimization algorithms by calculating the time evolution of a rotated crystalline grain. The schematic for the initial system can be seen in Figure 3.1. The grain rotation angle was $\gamma = 5.0°$. The complex amplitudes magnitude was set as the perfect lattice equilibrium value at $\phi = 0.10867304595992146$ and the parameters used are shown in Table 3.1. After the initial state, the system was run for 10000[1] time steps without mechanical equilibration in order to form the grain boundary.

After the initital setup, the system was evolved by repeatedly taking 80 time steps and mechanically equilibrating the system. The evolution of the system at different times can be seen in Figure 3.2.

[1] Much less steps could have been used here.

Table 3.1: Parameters for the grain rotation simulation.

parameter	value	description
nx	384	grid size in x direction
ny	384	grid size in y direction
dx	2.0	space step in x dir.
dy	2.0	space step in y dir.
dt	0.125	time step
bx	1.0	B^x in Eq. (2.7)
bl	0.995	$B^l = B^x - \Delta B$
tt	0.585	τ
vv	1.0	ν
eq_freq	80	mech. equilibration frequency
eq_tol	7.5×10^{-9}	mech. equilibration tolerance



(a) Time: 2150 (b) Time: 5050 (c) Time: 8050 (d) Time: 11050 (e) Time: 14050

Figure 3.2: Evolution of the rotated grain at different times.

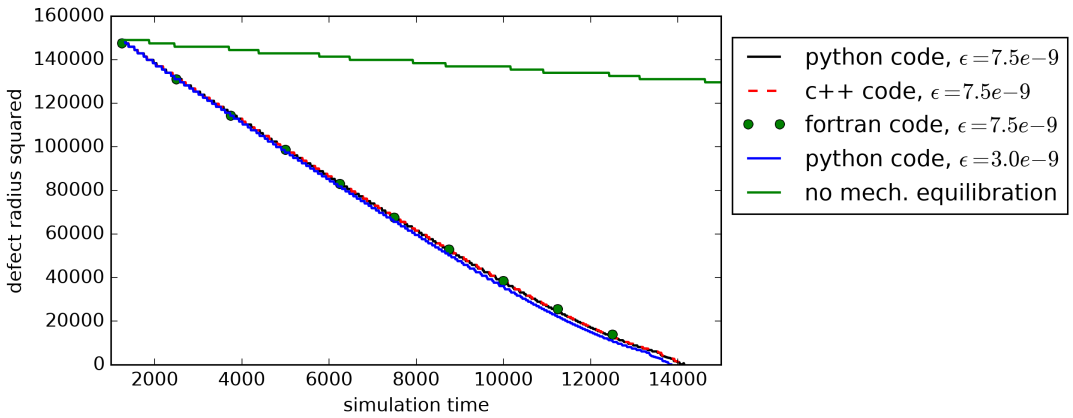


Figure 3.3: Time evolution of the square of the grain radius.

The grain area decreases linearly [3]. Figure 3.3 shows that including mechanical equilibration considerably increases the contraction rate of the grain. It also tests the Python and C++ codes against a previously used Fortran code. Additionally, it was checked if decreasing the mechanical equilibration tolerance from 7.5×10^{-9} to 3.0×10^{-9} would considerably change the results. Figure 3.3 also shows that changing the tolerance does not affect rate of contraction in a major way.

4 Mechanical equilibration algorithms

The mechanical equilibrium condition given by Eq. (2.12) can be formulated as an unconstrained optimization problem, for which a lot of different algorithms exist.

In optimization, the goal is to find the configuration x, which (globally or locally[1]) minimizes a scalar function $J(x)$, where x is a vector of dimension n. In the minimum point $x^{\checkmark}$

$$\nabla J(x^{\checkmark}) = 0. \tag{4.1}$$

The numerical task is to iterate from an initial state x_0 towards $x^{\checkmark}$. Most algorithms need to be able to evaluate $J(x)$ and $\nabla J(x)$.

Condition in Eq. (2.12) can be thought of as a gradient that is zero at the configuration $\theta_j^{\checkmark}$ we are looking for, expressed as

$$\boldsymbol{q}_j \cdot \sum_{k=1}^{3} \boldsymbol{q}_k \operatorname{Im}\left(\eta_k^* \frac{\delta F}{\delta \eta_k^*}\right) \equiv \nabla G(\theta_j^{\checkmark}) = 0. \tag{4.2}$$

The function $G(\theta_j)$, however, is not needed to be known, as we can evaluate the energy functional $F[\theta_j]$ of Eq. (2.7) to check for sufficient decrease or increase in $G(\theta_j)$.

4.1 Benchmark equilibration

To compare the performance of different optimization algorithms, I use the rotated grain system introduced in Section 3.4. The initial setup of the rotated grain consists of 10000 PFC time steps followed by an initial mechanical equilibration. This initial equilibration was not used as the benchmark equilibration because of the atypically long relaxation time. Instead, the next equilibration, which was done after 80 PFC time steps, was chosen to be the benchmark and most of the following results in this chapter show the performance of the optimization algorithms for this benchmark optimization problem.

All of the algorithms were run on a single core of a laptop with an i5-3320M @ 2.60GHz processor. The testing and benchmarking was done with the Python code (see Section 5.1).

The error of the mechanical equilibration algorithms was estimated by evaluating the 1-norm of the mechanical equilibration condition[2] (Eq. (4.2)) divided by the number of components

$$\epsilon = \frac{1}{3n_x n_y} \sum_{c=1}^{3} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} |\nabla G_{c,i,j}|, \tag{4.3}$$

where n_x and n_y describe the number of spatial points in x and y directions. This gives the average absolute value of the components of the gradient. Division in necessary to

[1] For mechanical equilibrium, we are searching for the local minimum.

[2] Or in other words, the norm of the "gradient".

obtain similar error values for different system sizes. Another option would be to take the maximum absolute value of the components of gradient. The tolerance, i.e. the value of the error, when the optimization algorithm is terminated, was chosen to be 7.5×10^{-9}, by observing the speed of the rotated grain convergence (see Figure 3.3), but this does not guarantee the best energy convergence for most algorithms (see Section 4.6).

4.2 Steepest descent

The method of steepest descent is the simplest optimization method. It takes steps in the direction of the negative gradient and is given by the following recurrence relation

$$x_{k+1} = x_k - \lambda_k \nabla J(x_k), \tag{4.4}$$

where k is the iteration number and λ_k is the step length chosen such that F is minimized in the gradient direction. This is done by a sub-algorithm called line search[3]. Line search, however, requires multiple function and gradient evaluations and in the case of computationally expensive evaluations it is often more efficient to use a predetermined fixed step λ_0. This is a common practice for example with machine learning algorithms. Fast convergence favours a large step size but too large step sizes can make the algorithm unstable.

For the benchmark equilibration, the step size was chosen to be $\lambda_0 = 1.0$. Note that for different parameters of the system (see Table 3.1; mainly depending on dx and dy), the step size needs to be decreased for the algorithm to be stable. Perhaps the best way to determine the step size is to run a line search steepest descent and then choose either the most frequent step size or the minimum step size.

In addition, I tested the steepest descent algorithm with a line search. The line search algorithm used takes exponentially increasing steps until a step with minimal energy is found[4]. This search is fairly cheap compared to more robust searches and in general does not work very well, as it can jump to different local minima. In our case, the dimensionality and the physics of the problem prevent this sort of unwanted jumps.

The results are shown in Figure 4.1. The fixed step steepest descent took rougly 19000 seconds to reach the tolerance, while line search SD took considerably less, about 7000 seconds.

4.3 Nesterov's accelerated gradient descent

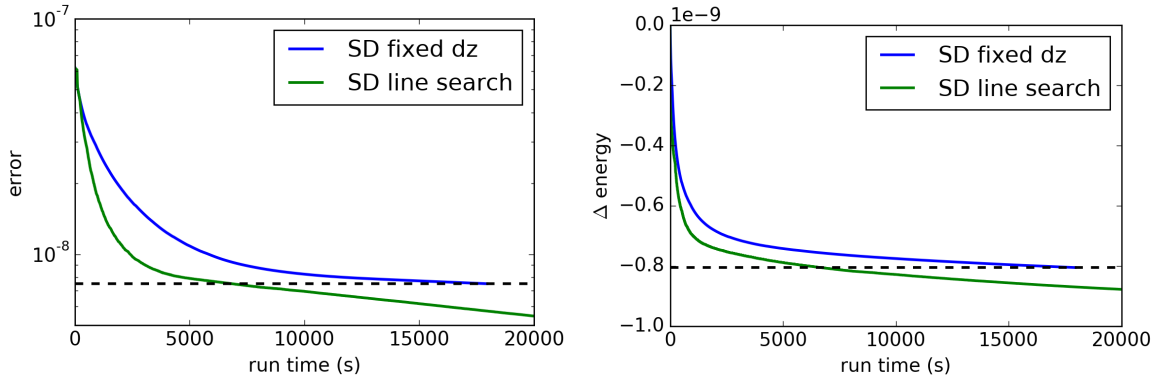
Steepest descent algorithms have trouble navigating through ravines, i.e. areas where the objective function varies considerably faster in some directions than the others. One approach for alleviating this shortcoming of the steepest descent algorithm is to add momentum to the descent. This is given by the following scheme:

$$v_k = \gamma v_{k-1} - \lambda \nabla J(x_k), \tag{4.5}$$

$$x_{k+1} = x_k + v_k. \tag{4.6}$$

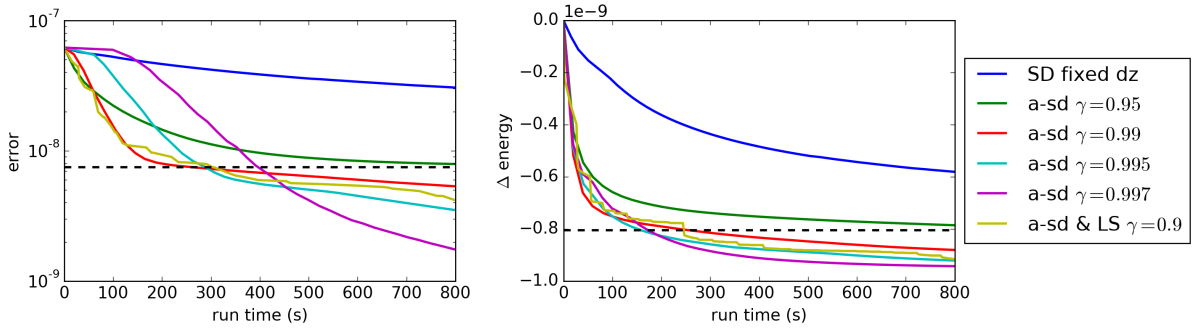
[3]Line search minimizes a function on a one dimensional line. To this end, there are multiple methods [4].

[4]The initial step was chosen to be 1.0 and if this turned out to increase energy, the algorithm exponentially searched for a shorter step.



(a) Error convergence (b) Energy convergence

Figure 4.1: Steepest descent method. The dashed lines show the goal tolerance and the corresponding energy at the goal tolerance (this energy can be different for different algorithms). (The error for SD line search and the following algorithms fluctuate a lot, short time scale fluctuations of the error are smoothed out in order to better present the long time scale evolution of the error.)



(a) Error convergence (b) Energy convergence

Figure 4.2: Nesterov's accelerated gradient descent. For the most suitable values of γ, the algorithm reaches goal tolerance in 300 s. The algorithm "a-sd & LS" has similar performance suggesting the use of the plain Nesterov's algorithm.

An improvement to the simple momentum based method is the Nesterov's accelerated gradient descent [5] (simple overview is given in [6]), which is described by

$$v_k = \gamma v_{k-1} - \lambda \nabla J(x_k + \gamma v_{k-1}), \tag{4.7}$$

$$x_{k+1} = x_k + v_k. \tag{4.8}$$

The results for mechanical equilibration for different values of γ are shown in Figure 4.2. Additionally, I tested a Nesterov's algorithm with an occasional line search ("a-sd & LS $\gamma = 0.9$"), a method with higher code complexity but similar performance. The step length was chosen to be $\lambda = 1.0$[5]. Figure 4.2 shows that the error reaches the tolerance in about 300 seconds, about $19000/300 \approx 63$ times faster than steepest descent.

[5]But similarly to SD, it might need to be changed for different system parameters.

4.4 L-BFGS

The limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm [4, 7, 8] is a quasi-Newton method defined by

$$x_{k+1} = x_k - \lambda_k H_k \nabla J(x_k), \tag{4.9}$$

where H_k is an approximation of the inverse Hessian based on previous gradient evaluations. The step size λ_k is generally found by a line search. In case of Newton's method[6], the step size of $\lambda = 1.0$ will jump directly to the minimum of a quadratic function and in theory this should also hold for the L-BFGS method, but in practice the algorithm might not be stable and the step size needs to be reduced. Step sizes similar to steepest descent fixed step can be used and can be determined by running a line search steepest descent algorithm for a number of steps, for example.

The limited memory BFGS approximation to H_k needs to store m last steps of the vectors $s_k = x_{k+1} - x_k$ and $y_k = \nabla J_{k+1} - \nabla J_k$. The approximate H_k is calculated using Algorithm 4.1, where $\rho_k = 1/(y_k^T s_k)$, H_k^0 is an initial approximation to the Hessian (identity matrix for our case) and the final result is $r = H_k \nabla J_k$.

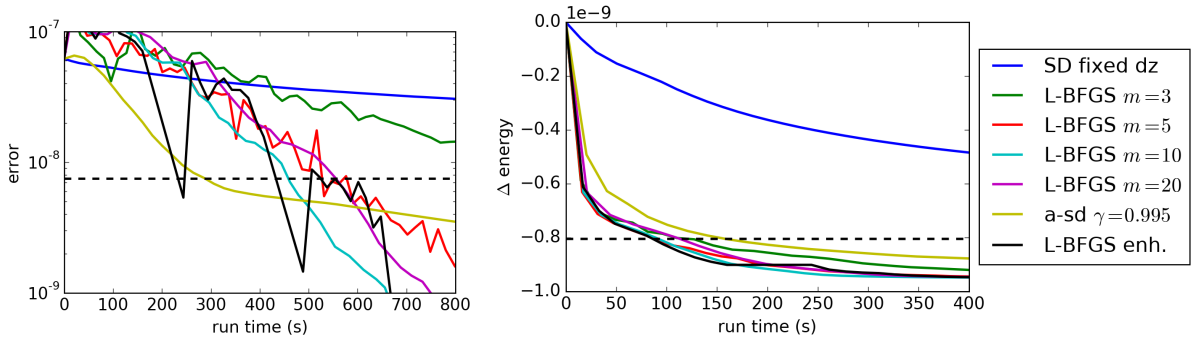
$q = \nabla J_k$
for $i = k-1, k-2, \ldots, k-m$ **do**
  $\alpha_i = \rho_i s_i^T q$
  $q = q - \alpha_i y_i$
end
$r = H_k^0 q$
for $i = k-m, k-m+1, \ldots, k-1$ **do**
  $\beta = \rho_i y_i^T r$
  $r = r + s_i(\alpha_i - \beta)$
end

Algorithm 4.1: L-BFGS approximation evaluation. The final result is in variable $r = H_k \nabla J_k$.

The equilibration results can be seen in Figure 4.3, which shows that the algorithm with $m = 3$ performs worst, while others are on par. On the other hand, increasing m requires additional memory suggesting using $m = 5$ for the computations. The energy convergence for the L-BFGS method is considerably better than for the accelerated descent. The L-BFGS method, however, converges slower than the accelerated descent in terms of error for low tolerance values like the dashed line ($\epsilon = 7.5 \times 10^{-9}$). Error convergence is used as the stopping condition for the algorithms, so this determines the computational performance.

In order to get a better error convergence for the L-BFGS, Nesterov's accelerated descent was used with a specified interval to reduce the error. The best performance is achieved when L-BFGS steps are taken until the accelerated descent will reduce the error below tolerance. In practice this cannot be guaranteed, but in case of a PFC calculation, similar mechanical optimization problems are solved repeatedly and the same number of

[6]Completing a step of Newton's method requires a solution of a linear system and for our problem, where we seek to solve systems with up to $3 \times 5000 \times 5000$ variables, this is infeasible unless the system can be made sparse by for example finite difference equations, but this would introduce additional restrictions.



(a) Error convergence (b) Energy convergence

Figure 4.3: L-BFGS method for the benchmark equilibration. The error convergence of the plain L-BFGS is slower than accelerated gradient descent for the given tolerance, whereas the energy is lowered considerably faster. The enhanced L-BFGS converges also fast in error.

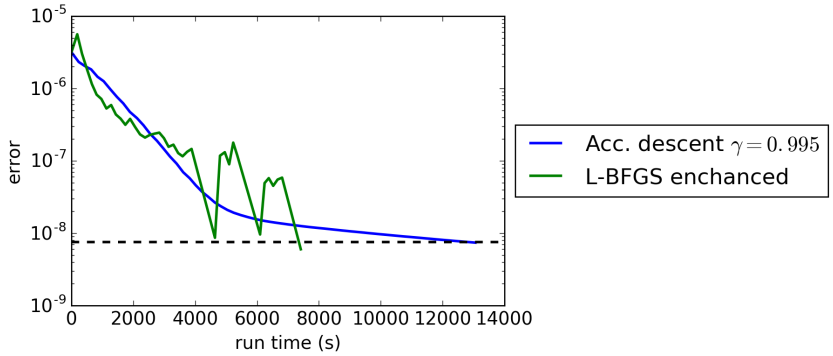
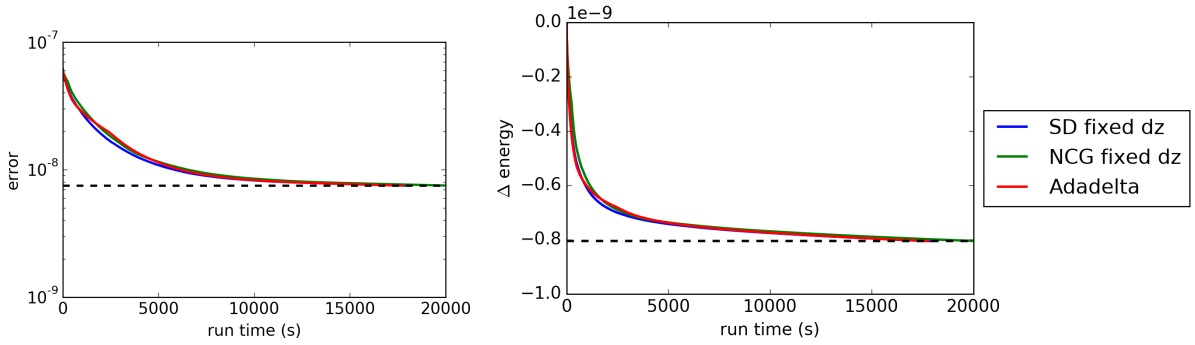


Figure 4.4: Error convergence for L-BFGS and accelerated gradient descent for a system size of 1024×1024.

L-BFGS steps can be used as was needed in the previous equilibration[7]. Additionally a line search was done at the start of the equilibration and also when accelerated descent changed back to L-BFGS steps. The performance is seen in the same figure under the label "L-BFGS enh" showing that the error convergence is on par with accelerated descent but energy is much closer to the converged value.

The enchanced L-BFGS was compared with the Nesterov's accelerated descent also for a bigger system. All parameters are the same as for the benchmark equilibration, except the system size was scaled up to 1024×1024 grid points. Additionally, this equilibration was the first one after setting the initial condition and taking 80 PFC time steps (as opposed to the second equilibration after initial setup for the benchmark). The error convergence can be seen in Figure 4.4. The goal tolerance of L-BFGS was reached in the third error reduction phase, which is not optimal, but it was still roughly twice as fast as the accelerated gradient descent. This shows that L-BFGS scales considerably better than the accelerated descent (and probably also the simple steepest descent) with an increasing system size.

[7]For the first and second equilibration, a more specific search should be employed to find the best interval of L-BFGS and Nesterov's steps.



(a) Error convergence

(b) Energy convergence

Figure 4.5: Nonlinear conjugate gradients and Adadelta.

4.5 Other methods: NCG and Adadelta

This section will contain a brief overview of methods that were tested, but did not work very well.

Nonlinear conjugate gradients (NCG) [7] is a popular optimization method, which was implemented and tested with a fixed step size $\lambda = 1.0$.

Adadelta [9] is an optimization method that is used in machine learning to optimize neural networks with a lot of different degrees of freedom. Due to expensive function and gradient evaluations, it is designed to not need a line search and dynamically choose the best step size based on previous changes in parameters and function values. The parameters used for this run were $\epsilon = 1\text{e}{-2}$, $\rho = 0.9$.

The results are shown in Figure 4.5. The performance of both algorithms is similar to the steepest descent. One reason for the bad performance of NCG can be that it relies on a line search and a fixed step size just does not work. Or perhaps neither of the methods are suitable for the given problem.

4.6 Energy convergence of the algorithms

The energy should reach a converged value after an optimal equilibration. In the previous energy convergence figures (e.g. Figure 4.1), the energy value goes below the dashed line (which was the final energy of the steepest descent at tolerance 7.5×10^{-9}). Figure 4.6 shows the energy dependence as a function of the error for the benchmark equilibration for a selection of algorithms. The converged energy value is clearly lower (by roughly 0.15×10^{-9}) than the value steepest descent reaches at tolerance 7.5×10^{-9}. The L-BFGS algorithm reaches the converged energy value very quickly, but the error is large. The benchmark equilibration might be special (and for most other quilibrations the energy has converged at tolerance 7.5×10^{-9}) or this energy difference does not really affect the evolution of the system, because the grain contraction time dependence was practically the same for tolerances 7.5×10^{-9} and 4.0×10^{-9} (see Sec. 3.4).

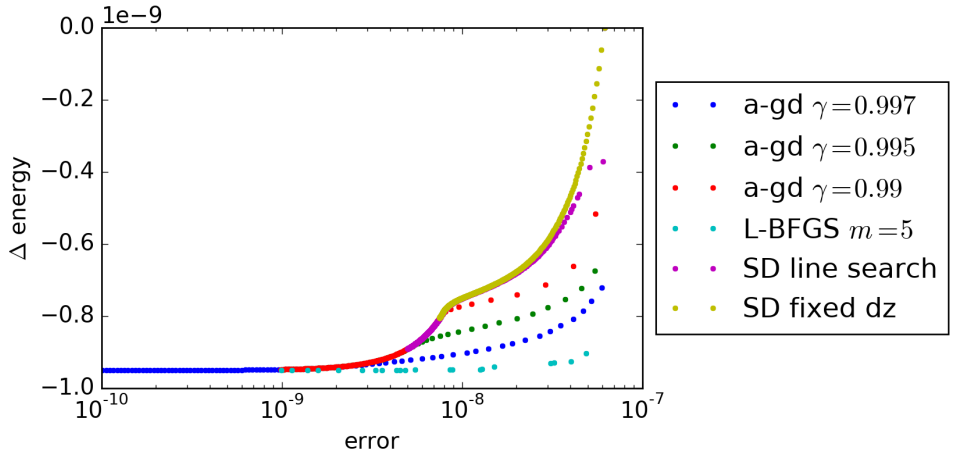


Figure 4.6: Energy convergence as a function of the error. All methods converge to a fixed energy value at low error values. At smaller error values, the energies are considerably different.

5 Code overview and usage

5.1 Python code

The full phase field crystal (PFC) code was written in Python 3 for convenient prototyping and testing. All of the mechanical equilibration algorithms mentioned in this work were implemented in the Python code. The code was tested to work with Python 3.4.3 and 3.5.1. Prerequisite libraries are *numpy* and *matplotlib*.

The code consists of three python files: `main.py`, `equilibrium_algorithms.py` and `equilibrium_algorithms_2.py`. `main.py` contains the main PFC code with PFC time stepping, energy calculation and mechanical equilibration constraint evaluation. This file also contains the `main` function, which will call mechanical equilibration algorithms. Files `equilibrium_algorithms.py` and `equilibrium_algorithms_2.py` contain functions for all of the mechanical equilibrium algorithms mentioned in this work.

To use the code, the `main` function[1] should be modified to suit the problem. An example `main` function is presented in Listing 1. Parameters of the code can be changed at the top of the `main.py` file. The parameters for mechanical equilibration algorithms, such as the tolerance, can be changed in their respective function definitions.

5.2 C++ MPI code

Programs written in Python are relatively slow and the language does not have a convenient and fast message passing interface (MPI) fast Fourier transform (FFT) library. Therefore, the full PFC simulation code was also written in C++ with MPI by utilizing FFTW, a popular FFT library with MPI support. The basic and more successful mechanical equilibration algorithms were implemented, namely

- steepest descent (with fixed time step and with line search);
- accelerated gradient descent (with and without occasional line search);
- L-BFGS (and the enhanced version, with ACG error reduction and occasional line search).

In addition to FFTW, OpenMPI is a prerequisite. The code was tested with OpenMPI 1.10.0, 1.10.2, 2.0.0 and FFTW 3.4.3. A `Makefile` is included to build and run the program.

The code is written in an object oriented manner, to separate the mechanical equilibration code from the rest of the PFC code. The class `PhaseField` (declared in `pfc.h` and defined in `pfc.cpp`) contains the parameters, initial system setup, energy

[1] The entry point of the program.

```python
def main():
  # Allocate memory for the complex amplitudes
  eta = np.zeros((3, nx, ny), dtype=np.complex128)

  # Initialize state to rotated grain and calculate Fourier transform
  init_state_circle(eta)
  eta_k = np.fft.fft2(eta)

  # Calculate derivative operators in k-space
  calculate_coefficients_tile()

  # Take 80 PFC time steps
  for ts in range(80):
      time_step(eta, eta_k)

  # Run LBFGS algorithm for mechanical equilibration
  equilibrium_algorithms_2.lbfgs_enh(
          eta, calculate_energy, calc_grad_theta, nx, ny)
  eta_k = np.fft.fft2(eta)

  # Save state to file
  np.save("./data/test_run/test", eta)
```

Listing 1: Example `main` function of the Python code. It takes 80 PFC time steps and does one mechanical equilibration.

and gradient calculations, input-output and the main loop of the program (methods `start_calculations` and `test`). The other class `MechanicalEquilibrium` contains the mechanical equilibration algorithms.

6 Summary and conclusions

The main objective of the project was to improve the performance of the existing mechanical equilibration algorithm. A numerical speedup by almost a factor of 100 was achieved for the benchmark equilibration with a system size of 384×384 using the Nesterov's accelerated gradient descent and the L-BFGS method. Due to superlinear scaling for the L-BFGS, I expect the speedup to be more significant for larger system sizes. The energy convergence of L-BFGS is roughly 200 times faster for the benchmark equilibration.

Below is a brief summary of the work I did during this project:

- Implemented and tested the following numerical optimization algorithms (the number shows the approximate speedup compared to the simple steepest descent for the benchmark equilibration; for other systems, the speedup can be very different)
  - Steepest descent 1.0
  - Steepest descent with line search ~ 3.0
  - Nonlinear conjugate gradient ~ 1.0
  - Adadelta ~ 1.0
  - Nesterov's accelerated gradient descent ~ 63
  - L-BFGS ~ 40
  - L-BFGS + accelerated gradient descent + line search ~ 75
- Implemented the whole PFC code and all of the optimization methods in python.
- Implemented the whole PFC code and the successful optimization methods in C++ with MPI support utilizing FFTW library.
- Tested the codes using the grain rotation setting.
- Tested using real space finite difference approximations for derivatives for calculating energy and the mechanical equilibrium constraint, concluding that this does not work for the benchmark system.

References

[1] Nikolas Provatas and Ken Elder. *Phase-Field Methods in Materials Science and Engineering*. 1st. Wiley-VCH, 2010. ISBN: 9783527407477.

[2] Ingo Steinbach. "Phase-field models in materials science". In: *Modelling and Simulation in Materials Science and Engineering* 17.7 (2009), p. 073001. URL: http://stacks.iop.org/0965-0393/17/i=7/a=073001.

[3] Vili Heinonen. "Phase field crystal models and fast dynamics". en. G5 Artikkeliväitöskirja. 2016, 96 + app. 86. ISBN: 978-952-60-6862-6 (electronic); 978-952-60-6861-9 (printed). URL: http://urn.fi/URN:ISBN:978-952-60-6862-6.

[4] Jorge Nocedal. "Updating Quasi-Newton Matrices with Limited Storage". In: *Mathematics of Computation* 35.151 (July 1980), pp. 773–782. ISSN: 0025-5718 (print), 1088-6842 (electronic).

[5] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/k2)". In: *Doklady an SSSR*. Vol. 269. 3. 1983, pp. 543–547.

[6] Sebastian Ruder. *Optimizing Gradient Descent - blog post*. URL: http://sebastianruder.com/optimizing-gradient-descent/ (visited on 07/29/2016).

[7] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN: 9780387303031. URL: https://books.google.fi/books?id=eNlPAAAAMAAJ.

[8] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45.1 (1989), pp. 503–528. ISSN: 1436-4646. DOI: 10.1007/BF01589116. URL: http://dx.doi.org/10.1007/BF01589116.

[9] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). URL: http://arxiv.org/abs/1212.5701.

A Finite difference investigation

In Section 3.3, I described how to evaluate the energy and the mechanical equilibration constraint in Fourier space. A benchmark mechanical equilibration (see Sec. 4.1) was done with a steepest descent algorithm that used the 5-point stencil finite difference formulas to calculate the mechanical equilibrium constraint. The formulas for first and second derivatives are the following

$$u'|_n \approx \frac{1}{12dx}\left(-u_{n+2} + 8u_{n+1} - 8u_{n-1} + u_{n-2}\right); \tag{A.1}$$

$$u''|_n \approx \frac{1}{12dx^2}\left(-u_{n+2} + 16u_{n+1} - 30u_n + 16u_{n-1} - u_{n-2}\right). \tag{A.2}$$

Expressing these formulas through k-space gives

$$u'|_n \approx \sum_{k=1}^{N} \hat{u}_k \exp\left(2\pi i \frac{kn}{N}\right)\left[\frac{1}{12dx}\left(8\sin(k_x dx) - \sin(2k_x dx)\right)\right]; \tag{A.3}$$

$$u''|_n \approx \sum_{k=1}^{N} \hat{u}_k \exp\left(2\pi i \frac{kn}{N}\right)\left[\frac{1}{6dx^2}\left(16\cos(k_x dx) - \cos(2k_x dx) - 15\right)\right]. \tag{A.4}$$

The results can be seen in Figure A.1, where the energy was calculated in Fourier space and should give a good estimate of the deviation from the converged solution shown by the bottom dashed line. The FD algorithm jumps to higher energies at the start and then slowly decreases. The convergence is much slower than for the k-space version and it is unlikely that it will eventually converge to the correct solution. This test suggests that it is not worth to use the FD formulas (even not as a preconditioner).

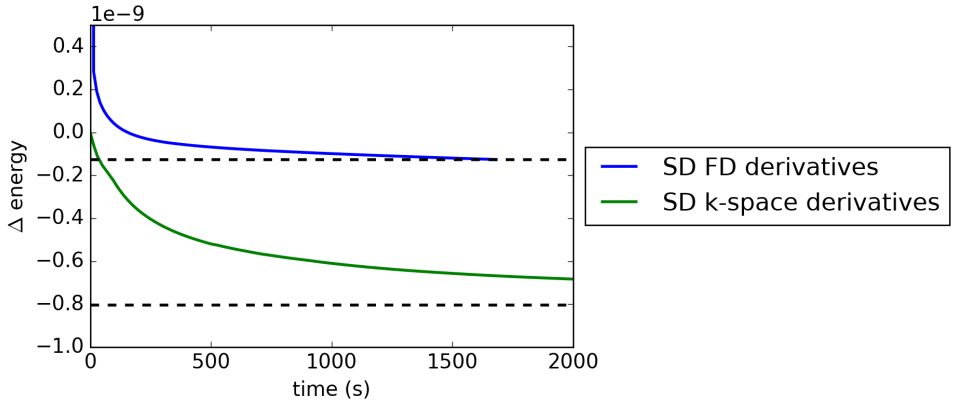


Figure A.1: The energy (calculated via Fourier space) dependence on the run time for the steepest descent (SD) using finite difference formulas and steepest descent using k-space formulas.