High-Order Time-Adaptive Numerical Methods For The Allen-Cahn and Cahn-Hilliard Equations

by

Mark Ryerson Willoughby

B.Sc., Brock University, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2011

Abstract

In some nonlinear reaction-diffusion equations of interest in applications, there are transition layers in solutions that separate two or more materials or phases in a medium when the reaction term is very large. Two well known equations that are of this type: The Allen-Cahn equation and the Cahn-Hillard equation. The transition layers between phases evolve over time and can move very slowly. The models have an order parameter ϵ. Fully developed transition layers have a width that scales linearly with ϵ. As $\epsilon \to 0$, the time scale of evolution can also change and the problem becomes numerically challenging.

We consider several numerical methods to obtain solutions to these equations, in order to build a robust, efficient and accurate numerical strategy. Explicit time stepping methods have severe time step constraints, so we direct our attention to implicit schemes. Second and third order time-adaptive methods are presented using spectral discretization in space. The implicit problem is solved using the conjugate gradient method with a novel preconditioner. The behaviour of the preconditioner is investigated, and the

dependence on ϵ and time step size is identified.

The Allen-Cahn and Cahn-Hilliard equations have been used extensively to model phenomena in materials science. We strongly believe that our high-order adaptive approach is also easily extensible to higher order models with application to pore formation in functionalized polymers and to cancerous tumor growth simulation. This is the subject of ongoing research.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr. Brian Wetton. His help throughout this project has been invaluable, and I appreciate his patience and many observations and directions throughout our meetings. It continues to be an honour to work with Brian and our research group. I also would like to thank Lee Yupitun, the math graduate secretary, for all her administrative help.

I would also like to thank Dr. Scott MacLachlan. His contribution of a linear model preconditioner is the highlight and main contribution of this thesis.

My research colleagues, Iain Moyles, Michael Lindstrom, Kai Rothauge and Ricardo Alves-Martins have also been valuable editors and contributors, and they offer discussion as well as support and friendship. I have watched the development of our research group since I have been involved and I am proud to be in company of such extraordinary people.

My family also has been a pillar of support throughout this project. I want

to thank my mother and sister for always being there for me. I also want to thank my father. While you may not be here with me now, I am here because of you.

Chapter 1

Introduction

Reaction-diffusion equations are one of the most significant tools in material sciences modelling. When the overall effect of reaction kinetics is larger than diffusion, transition layers form which physically represent a separation of two unique states or phases. Phase separation of metals in alloys as well as crystal grains in a metal that grow in competition with each other during annealing are examples of this transition layer behaviour. Two common equations that model the reaction kinetics in reaction-diffusion systems in material science, as well as diffusion-convection models in fluid dynamics are the Allen-Cahn and Cahn-Hilliard equations.

The Allen-Cahn equation was first introduced and named after Allen and Cahn in 1979 [1], and the Cahn-Hilliard by Cahn and Hilliard in 1958 [2]. Discretization and specific solutions to both equations have been studied extensively, see [3, 4, 5, 6, 7]. Analytical methods such as asymptotic analysis will not yield the full dynamics of the solution in general. Therefore an efficient and robust numerical strategy is needed to adequately describe the solution. We expect that the solution will admit transition layers. However,

these layers are not static which means that care will have to be given in the time stepping procedure. Numerical methods for time stepping that are explicit will encounter severe time step constraints for stability, thus we seek implicit schemes. We will consider second and third order time-adaptive methods using spectral discretization in space. Using a novel preconditioner, the implicit problem is solved using the conjugate gradient method. This thesis will investigate these numerical schemes.

Consider annealing, the process by which metal, glass etc. is heated and allowed to cool slowly. In the context of metal, this will harden or strengthen the metal, relieve stresses and improve the structure and ductility. Common examples in practice are the metals: copper, steel, silver and brass. The metal is heated above the recrystallization temperature and kept at that temperature for a length of time. Diffusion of the atoms inside the metal will achieve an equilibrium state. Three stages of annealing consist of recovery, recrystallization and equilibrium. The first stage results in softening of the metal through removal of crystal defects. The second stage exhibits grains that nucleate and grow to replace those deformed by internal stresses. The third stage is the state by which this growth has reached equilibrium [8]. Recrystallization is the process that can be modelled with the Allen-Cahn and Cahn-Hilliard models; deformed grains are replaced by undeformed grains that nucleate and grow until the original grains have been entirely consumed. Consider the Aluminum-Magnesium-Manganese alloy shown in Figure 1.1.

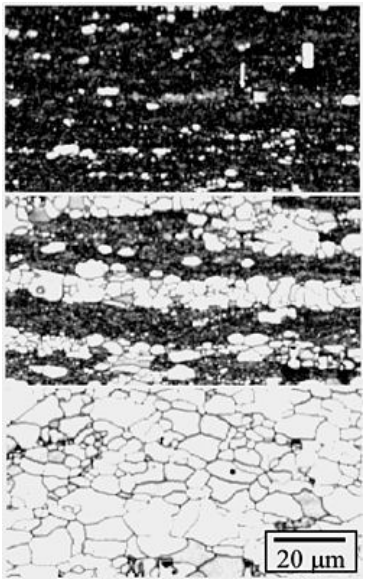

Figure 1.1: Three EBSD maps of the stored energy in an Al-Mg-Mn alloy after exposure to increasing recrystallization temperature. The volume fraction of recrystallized grains (light) increases with temperature for a given time. *Source: Manchester University (2003)*

As the alloy is heated, deformation is removed and a new collection of grains begin to grow, slowing consuming one another until equilibrium. This process is simplified in Figure 1.2.

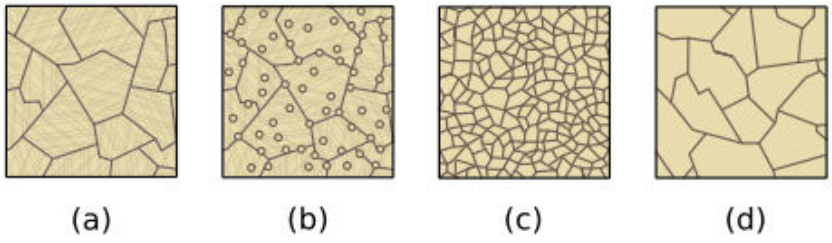

Figure 1.2: Recrystallization of a metallic material (a $\rightarrow$ b) and crystal grains growth (b $\rightarrow$ c $\rightarrow$ d). *Source: Wikipedia Commons.*

As stated, the behaviour of annealing, and many other problems, can be modelled using the Allen-Cahn equation, which is a partial differential equation having the general form

$$\begin{aligned} u_t &= \epsilon^2 \Delta u - f(u), \qquad \mathbf{x} \in \Omega, \epsilon \ll 1, \\ u(\mathbf{x}, 0) &= u_0(x), \end{aligned} \tag{1.1}$$

where we have variables t for time and $\mathbf{x}$ for position, $f(u)$ is nonlinear, and with boundary conditions dependent on the problem for a domain Ω. The variable u can be considered the concentration, volume difference or phase state between materials. The small parameter $\epsilon > 0$ is known as the interaction length, and also describes the thickness of the transition boundary between materials, also called the transition layer. We assume that ϵ is small with respect to the length scale of the domain of the problem. This captures the dominating effect of the reaction kinetics and appears in the equation to represent an effective diffusivity.

Like most material science applications, we can assume a certain periodicity in the lattice structure and so we will impose periodic boundary conditions to mimic a large system [9]. The rationale is that a small square section of the metal can be repeated to simulate the entire plate. With this being said, we consider our domain, $\Omega = [0, 2\pi]^d$ where d is the dimension of the problem. We have chosen this domain due to our application of spectral methods

which we will investigate in chapter 5. The initial value problems that we will then consider in this thesis are the Allen-Cahn equation

$$\begin{cases} u_t = \epsilon^2 \Delta u - w'(u), & \mathbf{x} \in [0, 2\pi]^d, \epsilon \ll 1, \\ \text{Periodic Boundary Conditions}, \\ u(\mathbf{x}, 0) = u_0(x), \\ w(u) = \frac{1}{4}(1 - u^2)^2, \end{cases} \tag{1.2}$$

where our nonlinear term $f(u) = w'(u)$. The term $w(u)$ is known as the Ginzburg-Landau double well potential. We also consider the Cahn Hilliard problem

$$\begin{cases} u_t = -\Delta \left[\epsilon^2 \Delta u - w'(u) \right], & \mathbf{x} \in [0, 2\pi]^d, \epsilon \ll 1, \\ \text{Periodic Boundary Conditions}, \\ u(\mathbf{x}, 0) = u_0(x), \\ w(u) = \frac{1}{4}(1 - u^2)^2, \end{cases} \tag{1.3}$$

which is fourth order in space. Equations (1.2) and (1.3) both have solutions that describe the separation of the medium we desire to model, where there is an abrupt change from one phase $u = 1$ to the other $u = -1$, the transition layers, such as we see in Figure 1.1. The transition layers between these phases will move slowly over time for the problems we are considering. As such, we are interested in the behaviour of long time scales when $\epsilon \to 0$. An important feature of the Allen-Cahn and Cahn-Hilliard equations is that

they can be viewed as the gradient flow of the Liapunov energy functional

$$E(u) = \int_\Omega \left(\frac{\epsilon^2}{2} |\nabla u|^2 + w(u) \right) \, \mathrm{d}x, \tag{1.4}$$

where we intend to minimize $E(u)$ [4]. The rapid changes in the solution of these equations with the time scale of evolution, and the steep gradients in the transition layers, must be resolved. The ideal numerical solution should have high accuracy while computing the solution efficiently. We will show the transition layers developed in the solution are of width $O(\epsilon)$, which gives us the constraint that we require a sufficient number of grid points in the transition layer to compute an accurate solution. Since the solution outside the transition layer is not contributing to the dynamics, an adaptive-in-space numerical method is preferred. However, a significant improvement in efficiency can be found by using adaptive time stepping alone. This thesis will examine the subject of adaptive time stepping methods, which will reduce the number of time steps we will need to solve, as well as an efficient solution for each step.

Using asymptotic analysis in chapter 2, we will develop a first order approximation to the transition layer for the Allen-Cahn equation in 1D. This yields the steady state solution and the approximate dynamics of the transition layer as $\epsilon \to 0$. However, the full dynamics of the solution will require our numerical methods. We then investigate several time stepping techniques,

starting with a first order method in chapter 3 using our most basic methods. The simplest method to implement is Backward Euler. For small ϵ, we will show that an approximation using uniform time steps cannot compute the solution to its steady state in a reasonable time, and adaptivity in time is essential. We will compute the solution using Forward Euler as a predictor, and show that the difference between these solutions will estimate the local error, which then can be used to decide if we increase or decrease the time step. We will see the solution quickly reaches a metastable state, where the transition layer has formed and now moves slowly.

To solve the nonlinear system of equations for each time step, we use Newton Iteration. In chapter 4 we develop a Conjugate Gradient method to solve for each iteration, where a linear system must be solved. We then develop a Preconditioned Conjugate Gradient method for this linear problem with a novel preconditioner that is a constant coefficient version of the system at the pure phase states $u = \pm 1$, suggested to us by Scott MacLachlan of Tufts University [11]. The width of the transition layers in our problem are much smaller than the entire domain for small ϵ. The dynamics of the solution do not change outside of the layer, thus a linear model would be a good candidate as a preconditioner. We will show that indeed a linear model will improve each CG solve for our methods. The development and investigation of this solver is the main contribution of this thesis.

It is our goal to show that adaptivity in time is sufficient for an efficient numerical method. This is achieved by using spectral methods in space. In chapter 5, we develop a Fourier periodic spectral method. As a result, at each time step we will require a solution to a more dense system of discretized equations. However, since this system is composed of N Fourier basis functions, a sufficient N will introduce almost no spatial error as a result [12]. We can approximate the solution with a minimum number of basis functions growing linearly with $1/\epsilon$. We will show that spectral methods give a good efficient solution to our problem.

We then consider higher order time stepping methods to compare to our first order method. In chapter 6, we will look at second and third order singly-diagonal implicit Runge Kutta methods as well as linear multi step methods to obtain second and third order adaptive-in-time methods. Each adaptive time stepping method derives its adaptivity from the local error estimator. There are several estimators that can be used, but an efficient estimator will be a explicit method which is quick to calculate and is of the same order of the method. We have chosen the explicit Adams-Bashforth method for this purpose. At each time step, we then calculate a pair of solutions, implicit and explicit, and use the maximum norm of the difference as our local error estimator.

In chapter 7 several full numerical simulations will be presented for both

Allen-Cahn and Cahn-Hilliard in 1D and 2D for small ϵ. The programming for each method will be done in MATLAB and detailed comparison will be used to determine the advantages and disadvantages of each method. We will show that our second and third order methods are more efficient than the simple Backward Euler method.

Phenomena in material science have often been modelled using the Allen-Cahn and Cahn-Hilliard equations. We strongly believe that the methods proposed in this thesis will be easily extensible to higher order models and applications, such as pore formation in functionalized polymers or Polymer Electrolyte Membranes (PEM) of fuel cells [14], or to cancerous tumor growth simulation (using a sixth order Cahn-Hilliard equation) used in cancer research [15].

Chapter 2

Analytical Methods

Considerable work has been done using analytical methods to understand the behaviour of solutions to the Allen-Cahn and Cahn-Hilliard equations, see [5, 16, 17, 18, 19]. While explicit analytic solutions cannot be found in general, we can use analytical techniques to gain insight into the dynamics. One such way to explore dynamics is to look for steady state solutions through the use of asymptotic analysis. Consider the Allen-Cahn equation in one dimension (1D)

$$u_t = \epsilon^2 u_{xx} + u - u^3. \tag{2.1}$$

Consider situations when the diffusion is negligible because $\epsilon \ll 1$. This occurs in regions where the composition consists almost entirely of one material. The boundary between materials will be the region of interest for diffusion. Neglecting diffusion in (2.1) leads to $u_t = u - u^3$, which is an ordinary differential equation (ODE) which has steady state solutions of $u^* = 0, \pm 1$. We can perform a linear stability analysis in the absence of diffusion of the form $u = u^* + \phi e^{\lambda t}$ for eigenvalues λ. We conclude that the eigenvalues for $u^* = \pm 1$

are negative while the eigenvalue for $u^* = 0$ is positive. Consequently, the states $u^* = \pm 1$ are stable while the state $u^* = 0$ is unstable.

Starting from general initial conditions, values of u will tend to $u = \pm 1$ for most values of x as shown in the example in Figure 2.1. There is a transition layer between each of the adjacent regions which will exhibit a steep profile for small ϵ. Each boundary is not static and thus it may move slowly which is referred to as metastability until they merge or are annihilated. It is these steep transition layers which we would like to investigate.

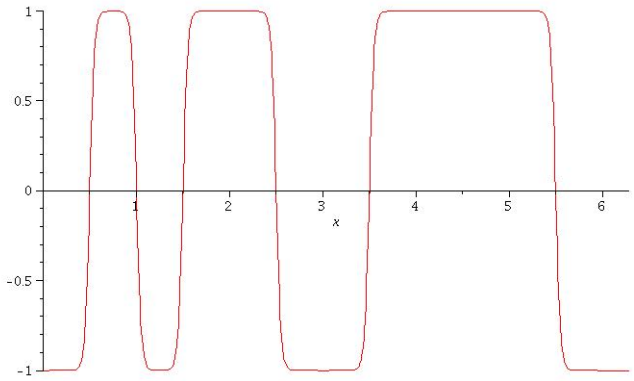

Figure 2.1: Possible initial conditions with transition layers between $u = \pm 1$.

Using asymptotic analysis, we now consider the case of one transition layer when diffusion is not negligible. We look at x far away from the transition layer, the outer problem, where we expect $u \to \pm 1$. We need to construct a transition layer so that it connects our two outer solutions $u = \pm 1$. If we center this layer around $x = x_0$ then we can form a boundary layer near this point, shown in Figure 2.2.

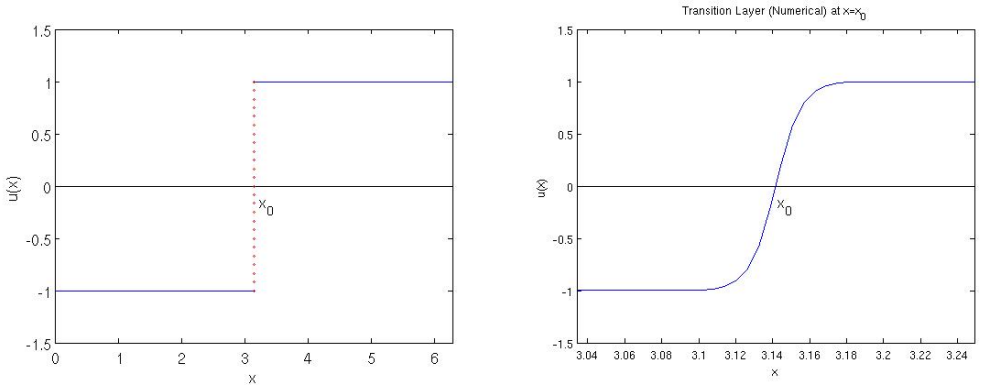

Figure 2.2: (a). Matching requirement for $u = -1$ and $u = +1$ at $x = x_0$. (b). Numerical solution showing transition layer near $x = x_0$.

Focusing on the small region between the outer solution $u = \pm 1$, we can define an inner region variable, $y = \frac{x - x_0}{\epsilon}$, $-\infty < y < \infty$ so that the inner problem becomes

$$u_t = u_{yy} + u - u^3. \tag{2.2}$$

The scaling on y is chosen so that diffusion is not negligible. The power of ϵ that appears in y indicates that the transition layer has a width of $O(\epsilon)$. We can look for steady state solutions of (2.2) by looking at the problem

$$u_{yy} = u^3 - u.$$

Multiplying by u_y on both sides gives

$$\begin{aligned} u_y u_{yy} &= u_y u^3 - u_y u \\ \frac{1}{2}(u_y^2)_y &= \frac{1}{4}(u^4)_y - \frac{1}{2}(u^2)_y. \end{aligned}$$

Integrating both sides with respect to y yields

$$\begin{aligned} \int (u_y^2)_y \, \mathrm{d}x &= \frac{1}{2}\int (u^4)_y \, \mathrm{d}x - \int (u^2)_y \, \mathrm{d}x \\ u_y^2 &= \frac{1}{2}u^4 - u^2 + C, \end{aligned}$$

where C is an arbitrary constant. To determine the constant C we consider the boundary conditions $u_y \to 0$ and $u \to \pm 1$ when $y \to \pm\infty$ which corresponds to the appropriate matching conditions with $x - x_0$, which is $O(1)$ in the outer solution. We have

$$C = u^2 - \frac{1}{2}u^4\bigg|_{u=\pm 1} = \frac{1}{2}$$

Then our problem for u_y is now

$$\begin{aligned} u_y &= \pm\sqrt{\frac{1}{2}u^4 - u^2 + \frac{1}{2}} \\ &= \pm\frac{1}{\sqrt{2}}\sqrt{(u^2-1)^2} \\ &= \pm\frac{1}{\sqrt{2}}(u^2 - 1). \end{aligned}$$

Since we have chosen our matching condition to go from $u = -1$ to $u = +1$ we take the negative square root. Separating variables and integrating, we have

$$\int \frac{1}{u^2 - 1}\, \mathrm{d}u = -\frac{1}{\sqrt{2}} y + D.$$

Upon performing the integration we get

$$\begin{aligned} -\text{arctanh}(u) &= -\frac{1}{\sqrt{2}} y + D \\ u &= \tanh\left(\frac{y}{\sqrt{2}} + D\right). \end{aligned}$$

Returning to the outer variable x, we have

$$u = \tanh\left(\frac{x - x_0}{\sqrt{2}\epsilon} + D\right).$$

The constant D in this case is due to the translational invariance of this function on an infinite domain. Since we chose to center the transition layer at $x = x_0$ then we are imposing that $u = 0$ there. This then determines that the constant D must be zero. Therefore, the solution to the transition layer that describes the transition between the stable states $u = \pm 1$ is

$$u = \tanh\left(\frac{x - x_0}{\sqrt{2}\epsilon}\right), \tag{2.3}$$

as shown in Figure 2.3 which matches our numerical transition layer shown

earlier in Figure 2.2 (b).

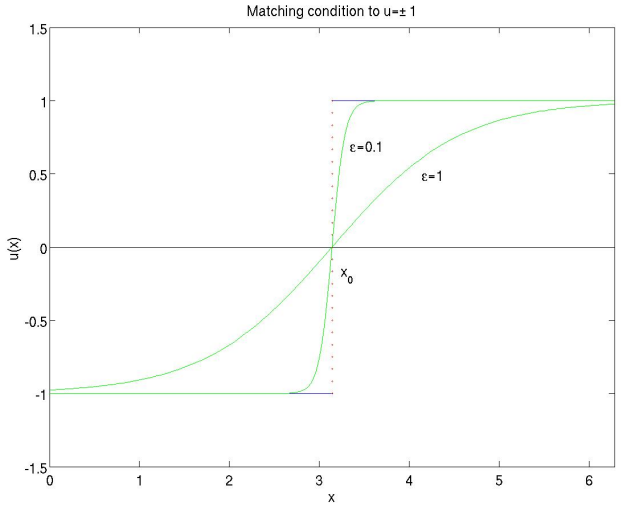

Figure 2.3: Transition layer solution to match stable solutions $u = \pm 1$.

This solution describes one transition layer, while there could be several layers that develop depending on initial conditions. For the methodology discussed so far, we are only considering steady-states and thus we are ignoring any transient periods that may occur. The transients can involve the slow movement and interaction of transition layers. Some aspects of these metastable dynamics can be described with more elaborate asymptotic methods [5], but in general, numerical approximation is required. This will be the approach for the remainder of this thesis.

Chapter 3

Basic Numerical Methods

Due to the complexity associated with many partial differential equations, analytical techniques are often unable to provide us with explicit solutions. Though there are many approaches to obtain insight into a partial differential equation (PDE), it is our goal to obtain an accurate solution using a numerical method. To do this, we restrict the domain of the problem to only a finite number of points $(x_i, t_n), i = 0, 1, ..., I, n = 0, 1, ..., N$, called the computational domain. The scope of this thesis will be a periodic domain $[0, 2\pi]^d$, where d is the dimension of the problem. At each of the finite points $x_0, x_1, ..., x_I$, which we call a mesh, we numerically evaluate a solution u to the PDE for a time interval $[0, T]$. We first consider how the time derivative is computed in the context of a numerical scheme. By partitioning the given time interval $[0, T]$ into N subintervals $t_0, t_1, ..., t_N$, and time step k, we can compute a numerical approximation to the time derivative. If we consider a uniform time step $k = t_{n+1} - t_n$, then we would call this a regular interval. Consider first the scalar initial value problem

$$\frac{du}{dt} = f(t, u(t)), \qquad u(t_0) = u_0. \tag{3.1}$$

Using the computational mesh, we construct an algebraic approximation to (3.1). We use Taylor expansion to approximate the derivatives at each of the points of the mesh for each time t_n. We know that if a function has continuous derivatives, i.e. $f(x) \in C^\infty$, it can be approximated by its power series or Taylor series. When this series is truncated to a finite number of terms, we have a Taylor polynomial, which we use to numerically approximate functions. If a function $f(x) \in C^{N+1}$, then we can write $f(x) = P_N(x) + R_N(x)$ where $P_N(x)$ is a polynomial and $R_N(x)$ is the remainder. Using the mean value theorem, we can express the remainder in Lagrange form. We write

$$f(x) = \sum_{n=0}^{N} \frac{f^{(n)}(a)}{n!}(x-a)^n + \frac{f^{N+1}(c)}{(N+1)!}(x-a)^{N+1},$$

for some c between x and a. If we expand this polynomial, we have

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + O\left((x-a)^3\right). \tag{3.2}$$

To approximate the first derivative of $f(x)$, we solve (3.2) for $f'(a)$ to get

$$f'(a) = \frac{f(x) - f(a)}{x-a} + O(x-a),$$

where the higher order terms would be considered a truncation error of $O(x-a)$. This is the definition of the derivative of $f(x)$ in the limit of a small distance $x-a$. With this framework, we now consider the simplest numerical approximation to the time derivative, the Forward Euler method.

3.1 Forward Euler Method

We want to approximate the time derivative u_t in our PDE (1.2) for a solution $u = u(x,t)$ at a time t_n for time step k. If we want to find the solution at the next time step $u(t_n + k)$, we can approximate it with the Taylor expansion of u. In this case, x in (3.2) will be our next time step t_{n+1} and a will be the previous time step t_n. With the notation of $k = t_{n+1} - t_n$, we write

$$u(t_n + k) = u(t_n) + k\frac{du}{dt}(t_n) + \frac{k^2}{2}\frac{d^2u}{dt^2}(t_n) + O(k^3). \tag{3.3}$$

From (3.1) we can substitute $\frac{du}{dt}(t_n) = f(t_n, u(t_n))$ and by the chain rule the second derivative is given by

$$\begin{aligned}\frac{d^2u}{dt^2}(t_n) &= \frac{df}{dt}(t_n, u(t_n)) + \frac{df}{du}(t_n, u(t_n))f(t_n, u(t_n)) \\ &= \frac{df}{dt} + \frac{df}{du}f,\end{aligned}$$

for $f = f(t_n, u(t_n))$. Then (3.3) becomes

$$u(t_n + k) = u(t_n) + kf + \frac{k^2}{2}\left(\frac{df}{dt} + \frac{df}{du}f\right) + O(k^3). \tag{3.4}$$

This is the method employed by Leonhard Euler in 1768 in his book Institutiones Calculi Integralis [21]. His idea was to take the first two terms of this

expansion, where we arrive at the Forward Euler (FE) approximation

$$u(t_n + k) \approx u(t_n) + kf(t_n, u(t_n)). \tag{3.5}$$

In compact notation, we would write the numerical approximation as $u^{n+1} = u^n + kf(t_n, u^n)$. This method is first order, which we can see by returning to (3.3). Rearranging, the approximation for $u_t(t_n)$ is given by

$$\frac{du}{dt}(t_n) = \frac{u(t_n + k) - u(t_n)}{k} - \frac{k}{2}\frac{d^2u}{dt^2}(t_n) + O(k^2) \tag{3.6}$$

which is first order error in time. Namely, the error we expect is a constant C times k. We see that the error term from (3.6) contains a constant $C = \frac{1}{2}$, called the error constant. While this is the simplest method to implement, we will see in this thesis that constructing higher order methods in this fashion will allow us to reduce the error to higher order terms.

Notice how the right hand side (rhs) of (3.5) consists entirely of the previous time step. The process allows us to approximate the next time step for each mesh point as an explicit formula provided that we only know the previous time step. One of the computational problems with explicit methods is that they suffer from time step restrictions on this type of problem. That is, excessively small time steps must be taken to make the explicit time stepping methods stable or bounded. These time steps are much smaller than needed for accuracy, and this leads to inefficiencies in the method. This leads us to

implicit methods, where the simplest of these is the Backward Euler method.

3.2 Backward Euler Method

Now suppose we step forward in time, but arrange the formula so that we step from t_n to t_n+k but evaluate the derivative at time t_n+k. Our expansion (3.4) then becomes

$$u(t_n) = u(t_n + k) - kf(t_n + k, u(t_n + k)) + \frac{k^2}{2}(f_t + f_u f) - O(k^3). \quad (3.7)$$

Again taking the first two terms, we arrive at the so called Backward Euler (BE) approximation

$$u(t_n + k) = u(t_n) + kf(t_n + k, u(t_n + k)), \quad (3.8)$$

which now has the truncation error of $-\frac{k}{2}\frac{d^2u}{dt^2}$, similar to FE but with the opposite sign, which is again first order $O(k)$ in time. The error constant for the BE method is then $C = -\frac{1}{2}$. Now the rhs of (3.8) consists of the next time step and thus we are no longer able to solve explicitly for $u(t_n+k)$ and hence we have an implicit method. Implicit methods are more desirable, since they have better stability properties, with BE having unconditional stability [20]. Since both the FE and BE methods have the same dominant error term with opposite signs, we can use FE to generate a predicted solution, named the predictor, and use that as an initial condition in BE, the corrector, in order

to find the correct solution and efficiently estimate the error made in a given time step.

The estimated error associated with this predictor-corrector method can be constructed from our Taylor series (3.6) and (3.7). We want to estimate the listed term as an approximation of δ, the magnitude of the local truncation error for BE. Taking the difference of the FE and BE approximations, which are consistent so the $O(1)$ terms cancel, we are left with the error term

$$u_{BE} - u_{FE} \approx (C_{BE} - C_{FE})k^2 \frac{d^2u}{dt^2}.$$

Our approximation to the local error for BE is then

$$\delta \approx \left| -\frac{1}{2}k^2 \frac{d^2u}{dt^2} \right| \approx \left| \frac{C_{BE}}{C_{BE} - C_{FE}} \right| ||u_{BE} - u_{FE}||, \tag{3.9}$$

where $C_{BE} = -\frac{1}{2}$ and $C_{FE} = \frac{1}{2}$ calculated above. We have looked at the time derivative, now we will look at how the space derivative is computed in the context of a numerical scheme.

3.3 Finite Differences for Spatial Derivatives

The Allen-Cahn equation contains the spatial derivative Δu. If we consider a uniform mesh in 1D then our spatial grid is regular with a distance $h = x_{i+1} - x_i$ between each mesh point. We can again use Taylor expansion in

space to approximate $\Delta u(x_i, t_n)$. To achieve this we use center differencing that uses both the left and right neighbour x_{i-1} and x_{i+1} of a point x_i. The Taylor expansion in 1D for the left and right neighbouring points are

$$u(x_{i+1}, t_n) = u(x_i, t_n) + hu_x|_{x_i,t_n} + \frac{h^2}{2!}u_{xx}|_{x_i,t_n} + \frac{h^3}{3!}u_{xxx}|_{x_i,t_n} + O(h^4)$$
$$u(x_{i-1}, t_n) = u(x_i, t_n) - hu_x|_{x_i,t_n} + \frac{h^2}{2!}u_{xx}|_{x_i,t_n} - \frac{h^3}{3!}u_{xxx}|_{x_i,t_n} + O(h^4)$$

where a negative sign appears in the left neighbour since $x_i - x_{i-1} = -h$. Adding these two equations will eliminate the odd order derivatives. Rearranging, we can approximate $u_{xx}|_{x_i,t_n}$ with

$$u_{xx}|_{x_i,t_n} = \frac{u(x_{i+1}, t_n) - 2u(x_i, t_n) + u(x_{i-1}, t_n)}{h^2} + \frac{2h^2}{4!}u_{xxxx}|_{x_i,t_n} + O(h^4). \tag{3.10}$$

Taking the first two terms, we arrive at the standard finite difference (FD) formula

$$u_{xx}|_{x_i,t_n} = \frac{u(x_{i+1}, t_n) - 2u(x_i, t_n) + u(x_{i-1}, t_n)}{h^2} + O(h^2)$$

which is second order with $O(h^2)$ error in space, with error term $\frac{2h^2}{4!}u_{xxxx}|_{x_i,t_n} + O(h^4)$. Since the FD formula applies at all mesh points, let us introduce the notation $U = U(x_i, t_n)$ to denote the discretized $u(x, t)$ at each point. The

second derivative approximation would read

$$U_{xx} \approx \Delta_h U,$$

where Δ_h, the discretized Laplacian, is the matrix

$$\Delta_h = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & -2 \end{bmatrix}$$

where our periodic boundary conditions are incorporated. Now that we have considered how the derivatives are computed, we can construct a full discretization of the Allen-Cahn equation.

3.4 Discretization

Using a BE approximation to u_t and the discretized Laplacian for Δu, the discretization of the Allen-Cahn equation (1.2) is

$$\frac{U^{n+1} - U^n}{k} = \epsilon^2 \Delta_h U^{n+1} - w'(U^{n+1}).$$

Rearranging to have U^{n+1} on the left hand side and in matrix notation we have

$$(I - k\epsilon^2\Delta_h)U^{n+1} + kw'(U^{n+1}) = U^n$$

where I is the identity. Since we only know previous information U^n we need to solve this vector equation for U^{n+1}. The equation would read

$$F(U^{n+1}) = \mathbf{0}, \tag{3.11}$$

where F is defined as

$$F(Y) = (I - k\epsilon^2\Delta_h)Y + kw'(Y) - U^n, \tag{3.12}$$

where by $w'(Y)$ we mean the vector with components $w'(Y_j)$. $F(Y)$ is a nonlinear equation which we will solve using Newton iterations. At each iteration, a system of linear equations must be solved. Solving this type of system efficiently is the main goal of this thesis. We give a brief overview of Newton Iteration.

3.5 Newton Iteration

Consider the scalar equation

$$f(x) = 0. \tag{3.13}$$

Newton iteration, attributed to Sir Issac Newton, is an iterative method to approximate the roots or zeros of (3.13) starting with an initial guess x_0. Assuming that $f(x)$ is differentiable, we can approximate the root by calculating the tangent line at x_0. The x-intercept of this tangent line will usually be a better approximation to the current guess. If this is the case, then successive iterations will converge to the true root, as shown in Figure 3.1.

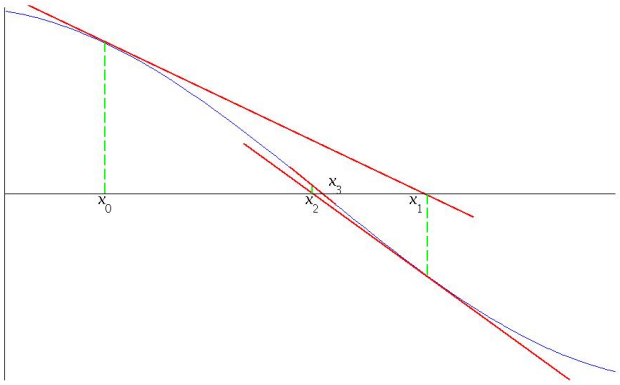

Figure 3.1: Graphical interpretation for Newton Iteration.

The tangent line from $f(x_m)$ to the x-intercept at x_{m+1} is given by

$$f'(x_m) = \frac{f(x_m) - 0}{x_m - x_{m+1}}$$

which leads to the iterative formula

$$x_{m+1} = x_m - \frac{f(x_m)}{f'(x_m)}.$$

Returning to (3.11), a matrix equation, let $V^m \approx U^{n+1}$ denote the mth iterate that will be converging to U^{n+1}. This iterative formula would now be written as

$$V^{m+1} = V^m - \eta,$$

where η is the solution to the matrix equation $J(V^m)\eta = F(V^m)$. J is the Jacobian, the first derivative of F, namely $J_{ij} = \frac{\partial F_i}{\partial U_j}$. We can calculate J from (3.12) by taking the derivative with respect to U to give

$$J(U) = (I - k\epsilon^2\Delta_j) + kW''(U) \tag{3.14}$$

where $W''(U)$ denotes the diagonal matrix with entries $w''(U_j)$. Newton iteration can fail to converge or find a different root to the one sought after if the function has many inflection points or if the initial guess is not close enough. For our purposes, we will not run into these cases as our initial guess computed using FE will be sufficiently close for sufficiently small time step k in our BE solution.

3.6 Backward Euler Solution

We now have a Backward Euler approximation to u_t, a standard Lapacian for Δu and the means to solve the resulting nonlinear discrete equations. Using the Forward Euler predictor as an initial guess, we use Newton iteration to solve for the next time step. Numerically, Newton iteration will continue until we are satisfied with the accuracy to the actual root. Given a tolerance `tol_newton` specified by the user, we calculate the residual $R = ||F(\mathbf{u}^{n+1})||$, until it is less than the tolerance, where we are using the maximum norm. Because of the condition number of J, machine precision is not achievable. Approximation of the Allen-Cahn equation is now possible with Algorithm 1.

Algorithm 1 *Backward Euler Approximation to Allen-Cahn*

Require: h, T, k, ϵ, mesh X, `tol_newton`, F, J and U_0
  for time steps $n = 0, 1, ..., T-1$ **do**
    $Y = U^n$
    while `residual`$<$ `tol_newton` **do**
      Solve $J(Y)\eta = F(Y)$ for η, then $Y = U - \eta$
      `residual`$= ||F(Y)||$
    end while
    $U^{n+1} = Y$
  end for

end

In section 3.2 we calculated using Taylor series that the BE method is first order in time, and in section 3.3 that FD is second order in space. We now can confirm these results by explicitly calculating the order of convergence.

3.7 Convergence

Suppose we had an exact solution u to our PDE and our approximation to that exact solution $\tilde{u}$. The global error E_i is the absolute value of the difference between the true solution and the exact solution at each mesh point. Our method is said to converge if this error vanishes as $h \to 0$ and $k \to 0$. The order of the error is then the order of the method. In this case, we have $O(k)$ error from Backward Euler and $O(h^2)$ error from finite difference. Then the total error in our current method is $E = O(k) + O(h^2)$. For one step in this method, the local truncation error is $O(k^2) + O(kh^2)$. This is the error associated with the approximation to the derivatives as we calculated earlier in this chapter. The local truncation error is different than the global error, that is, each time step introduces a local error. The first $O(k^2)$ error comes from the time derivative approximation from (3.7) where the truncated Taylor series has the error term $\frac{k^2}{2}u_{tt}(t_n) \approx O(k^2)$ for one step. The second $O(kh^2)$ term comes from the spatial second derivative approximation from (3.10) where the truncated Taylor series has the error term $\frac{h^2}{4!}u_{xxxx}(x_i, t_n)$ for one step and is multiplied by k in the discrete equation.

We now use our code to show that our Backward Euler method is indeed first order in time and second order in space. To check first order in time, we fix the number of mesh points. This is done so that the spatial error $O(h^2)$ remains constant and will not affect convergence. We then divide the time step successively by 2. The error is calculated by taking the maximum norm of each successive approximations and comparing the ratio. The ratio for the estimated convergence rate can be written as

$$C_R = \log_2 \left(\frac{||U_{k/4} - U_{k/2}||_\infty}{||U_{k/2} - U_k||_\infty} \right),$$

where we take the ratio of successive differences in the solution for each mesh. We take the log of this quantity for convenience to obtain convergence to 1 for first order and 2 for second order. First order convergence in time is clearly seen Table 3.1.

Time Step k	Error	C_R
3.1494e-05	5.1651e-03	1.0804
1.5747e-05	2.5396e-03	1.0242
7.8735e-06	1.2566e-03	1.0150
3.9368e-06	6.2356e-04	1.0109
1.9684e-06	3.1039e-04	1.0064
9.8419e-07	1.5479e-04	1.0038

Table 3.1: First order convergence in time for Backward Euler Method.

To check second order in space, we now fix the time step as $k = h^2$ so our error is now $E = O(h^2)$. We then double the number of mesh points

successively. Similarly, the error is calculated by taking the maximum norm of each successive approximations and comparing the ratio. The estimate for the convergence rate can be written as

$$C_R = \log_2 \left(\frac{||U_{h/4} - U_{h/2}||_\infty}{||U_{h/2} - U_h||_\infty} \right).$$

The results are shown in Table 3.2, and we see that indeed we achieve second order convergence.

Mesh Size (I)	Error	C_R
125	3.1309e-02	3.3054
249	6.6459e-03	2.2361
497	1.5605e-03	2.0904
993	3.5452e-04	2.1381

Table 3.2: Second order convergence in space for Backward Euler Method.

Now that we have established a first order method, we now consider the case where our solution remains in a metastable state for a long period of time. Increasing the time step would be an advantage in this case, which we explore in the next section.

3.8 Adaptive Time Stepping

The previous method that we have used so far has used a uniform time step. As we stated, a transition layer forms for small ϵ in $O(1)$ time then moves in a very slow time scale, possibly asymptotically slow as $\epsilon \to 0$. Thus, a

uniform time step method will be inefficient in this part of the computation. For small enough ϵ the transition layer will move slowly enough that an inordinate number of time steps are required to resolve the dynamics.

To reduce the amount of work required, adaptivity in time should be used for these problems. This is enabled by computing a local error estimate (LEE) for each time step. If this LEE is within a tolerance that we have chosen, we accept the solution for the current time step and increase the next time step. And if this tolerance is violated, we decrease the time step. This method is described in the local error method.

3.9 Local Error Method

We estimate the LEE for each time step to justify whether the solution should be accepted and then determine the next time step. The time step will then evolve with the dynamics of the solution, allowing us to compute the solution efficiently far in time through the slow dynamics of the transition. We will use the uniform time step solution as our basis of comparison to the methods outlined in this chapter. The general algorithm of an adaptive time step method is:

1. Calculate the next time step.

2. Compute the LEE.

3. Accept/Reject the solution and adjust the time step.

An estimate of the LEE is found by the local error method [22]. The LEE, denoted as δ, is given by the difference between two solutions, a predictor and a corrector. The corrector solution will be our BE method, our intended solution to the next time step. The predictor solution should be a best guess, with the most efficient and quickest method to calculate the next time step. This will be given by the Forward Euler method, an explicit and easy to calculate solution. We then calculate a BE-FE pair for each time step. Then the LEE is given by (3.9).

We can now adjust the time step to make δ smaller than the user specified tolerance δ_{tol} =`tol_delta`. At each time step, we allow an increase in the time step k by a factor of ξ or decrease by a factor of $\frac{1}{\xi}$, for ξ to be determined. Suppose δ is smaller than δ_{tol}, we would like to increase k by the highest amount without violating the tolerance inequality that we are currently satisfying for the next time step. We need to determine ξ such that

$$\delta_{n+1} \to \xi^{p+1}\delta_n,$$

where p is the order of the method, $p = 1$ for the BE time stepping described so far. The reason that ξ has the power $p+1$ is because the local truncation error is of order k^{p+1}.

If we require that $\delta < \frac{1}{\gamma}\delta_{tol}$, $\gamma > 1$, for a user defined γ, which then warrants an increase in k, then by multiplying $k_{n+1} = \gamma^{1/(p+1)}k_n$ will result in the next δ to be near δ_{tol}. To make sure we are not too close to δ_{tol} we introduce the safety factor $\theta = 0.8$. Then ξ will be

$$\xi = \sqrt[p+1]{\theta\gamma}.$$

We determine the action to take after each time step with the following algorithm similar to [22].

- if $\frac{\delta}{\delta_{tol}} < \frac{1}{\gamma}$. Accept u, $k_{n+1} = \xi k_n$ (Increase time step).
- else if $\frac{\delta}{\delta_{tol}} \in \left[\frac{1}{\gamma}, 1\right)$. Accept u, $k_{n+1} = k_n$ (Keep the same time step).
- else, Reject u, $k_n = \frac{1}{\xi}k_n$ (Reduce time step).

We take this approach rather than smoothly varying k because we will consider higher order multi step methods in later sections which require fixed time steps to be efficient. When γ is increased, there is a decrease in the number of instances we adjust the time step, however more time steps would be required. For example, if we take $\gamma = 2$ and apply it to our BE-FE solution for Allen-Cahn, the evolution of the time step is shown in Figure 3.2 compared to $\gamma = 4$.

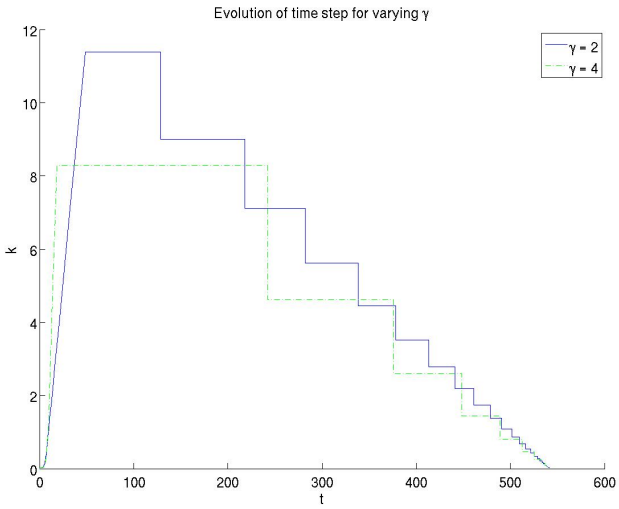

Figure 3.2: Backward Euler Solution with $\gamma = 2$ vs. $\gamma = 4$ in 1D.

In the context of a first order method, we only require one previous time step. Thus there is no consequence to the previous data in increasing or decreasing the time step. Figure 3.3 shows the number of time steps required and number of times we increase or decrease k as we vary γ.

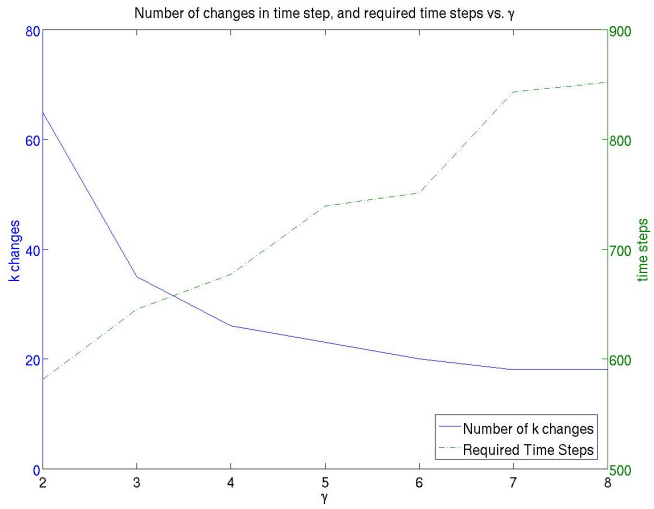

Figure 3.3: Number of changes of time step with the time steps required with varying γ in the BE-FE solution.

We see that while increasing γ reduces the number of changes in k, the number of time steps required increases. However, we shall see that in higher order methods, each increase or decrease in the time step does affect previous data and larger γ will allow us to reduce the number of times we need to restart or recalculate this previous data. This may be more beneficial if a moderate increase in time steps are required. In Figure 3.3, by setting $\gamma = 3$ or 4, we get a decrease in the number of times we change k with only a moderate increase in the total number of time steps required.

While the optimal γ in this case is 3 or 4, it is problem specific and should be determined depending on the problem. We now want to assess the confidence we have in our method, and we introduce the idea of a ripening time.

3.10 Ripening Time

The ripening time T_r is a chosen moment in the dynamics of the solution that will allow us to compare to other methods. In the case of the Allen-Cahn or Cahn-Hilliard equations, with the appropriate initial condition, the solution will have a region that quickly forms transition layers from the $u = +1$ to $u = -1$ regions. As the transition layers move through time, the regions will start to move toward the dominate region. In our cases, smaller regions in the $u = -1$ regime eventually cross the zero axis into the $u = +1$ regime. The moment this happens, we call the ripening time. This can be monitored by looking at the minimum of our current solution. As we let $\delta_{tol} \to 0$ we expect these ripening time estimates to converge, corresponding to our solution converging to a accurate solution. We also expect the number of time steps $M(\delta_{tol})$ required to follow linearly as $\frac{M(\delta_{tol}/10)}{M(\delta_{tol})} = \sqrt[p+1]{10}$, where p is the order of the method, in this case $p = 1$. This is because as we decrease δ_{tol} by a factor of 10, the error we had would be $\delta_{old} \approx Ck^{p+1}$ and the next error will be $\delta_{new} \approx C(\nu k)^{p+1} = \nu^{p+1}Ck^{p+1} = \nu^{p+1}\delta_{old}$, for constant ν. Thus to reduce the local error by a factor of 10, the time step size should be reduced by a factor of about $\sqrt[p+1]{10}$. The results of this numerical test for the Allen-Cahn problem to be presented in the next section are shown in Table 3.3.

δ_{tol}	T_r	$M(\delta_{tol})$	$\frac{M(\delta_{tol}^n)}{M(\delta_{tol}^{n-1})}$
1.0000e-04	541.8487	645	3.0861
1.0000e-05	545.2975	2,000	3.1008
1.0000e-06	546.2953	6,275	3.1375
1.0000e-07	546.6366	20,169	3.2142
1.0000e-08	546.7448	62,480	3.0978
1.0000e-09	546.7788	199,630	3.1951

Table 3.3: Convergence of the ripening time for Allen-Cahn using BE-FE time stepping method.

We see that our solution converges to $T_r = 546.8$ and as we expected, the ratio of $M(\delta_{tol})$ also converges close to $\sqrt[2]{10} = 3.1622$, and we thus confident in to application of our method to this problem. We will use these results to compare to our higher-order methods.

3.11 Numerical Experiment - Allen-Cahn

For Allen-Cahn, after we have passed the ripening time, the solution will then evolve to a steady state, and in our case it will be $u = 1$. Our initial condition is $u_0(x) = \sin x + 0.8$ on $[0, 2\pi]$. This is chosen so that the region above $u = 0$ is slightly larger than below $u = 0$ which avoids symmetry. We are interested in the case where we have a dominant region, and the subdominant region is eventually absorbed. The parameters we have chosen

are

$$\begin{aligned} \epsilon &= 0.16, & n &= 128, \\ T &= 537, & \texttt{tol\_delta} &= 10^{-4}, \\ \theta &= 0.8, & \texttt{tol\_newton} &= 10^{-8}, \\ \gamma &= 3, & u_0(x,0) &= \sin(x) + 0.8. \end{aligned}$$

As we begin the simulation, we expect the initial dynamics to require small time steps as the transition layer is formed. Figure 3.4 shows the evolution of the solution as we form the transition layer, metastable state and then reach the stable state.

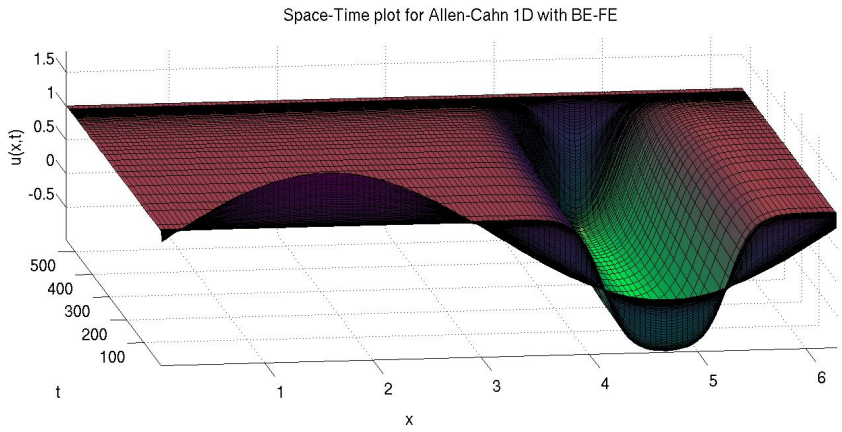

Figure 3.4: BE-FE Solution with Adaptive Time Stepping for Allen-Cahn in 1D for $f(x) = \sin(x) + 0.8$ on $[0, 2\pi]$.

We see the fast dynamics from the initial condition to the metastable state, where two transition layers are formed. Then, during the metastable state the time step k is rapidly increased as shown in Figure 3.5.

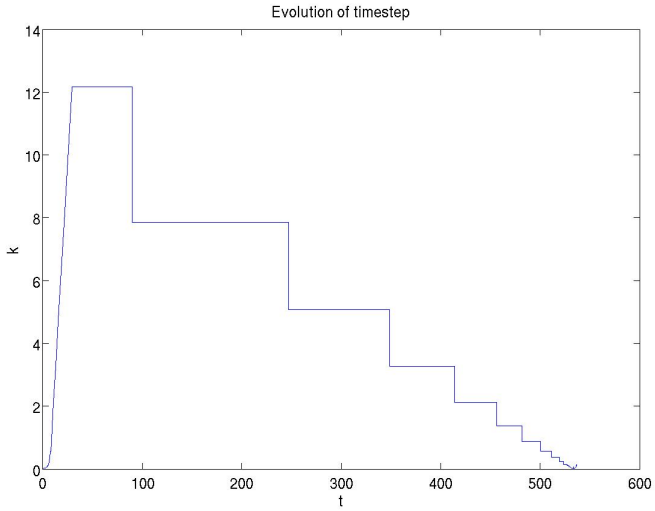

Figure 3.5: BE-FE Solution - Evolution of time step k vs time t for Allen-Cahn.

We see rapid growth initially and gradual decline as we leave the metastable state. The total number of time steps for this experiment was 811. We will show a comparison to the uniform method in the next section. We also plot the Liapunov energy (1.4) to show that we indeed are reducing the energy in time, and the number of Newton iterations, shown in Figure 3.6, where the average number of iterations is 3.39, with a maximum of 11.

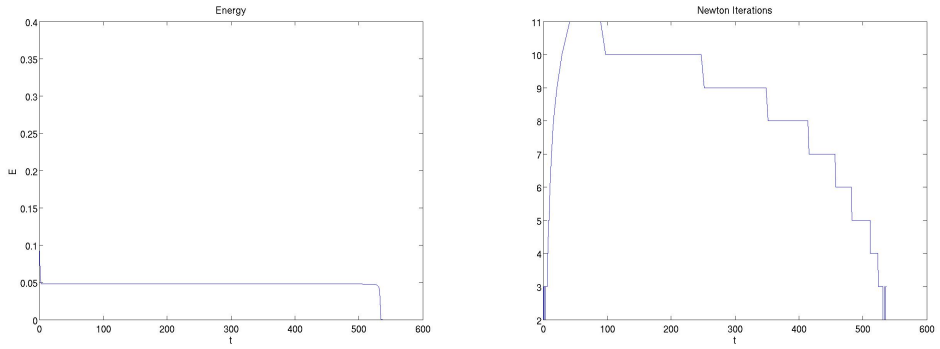

Figure 3.6: BE-FE Solution for Allen-Cahn - Liapunov energy vs time step k and Number of Newton Iterations.

Now that we have established an adaptive-in-time method, we can compare our method to the standard uniform time step method.

3.11.1 Comparison to Uniform Method

To compare our BE-FE adaptive method to uniform time steps we first compute a very accurate uniform time step solution which ends at the ripening time $T = 546.8$. We will be using $k = h^3$, which will require 1,149,820 time steps. Using this solution as our approximation to the exact solution for comparison, we show the number of time steps required for the BE-FE solution for various values of δ_{tol}. Each solution produces an error, decreasing as $\delta_{tol} \to 0$. To compare to the uniform solution, we find the number of uniform time steps which produces the same error when compared to our highly accurate "exact" solution. The results are shown in Table 3.4 where we see that our BE-FE method using our simple adaptive strategy is more

efficient than uniform time steps as we let $\delta_{tol} \to 0$.

δ_{tol}	Error $\|\|u_{\text{adap}} - u_{\text{exact}}\|\|$	Time steps (Adaptive)	Error $\|\|u_{\text{uni}} - u_{\text{exact}}\|\|$	Time steps (Uniform)
1.0000e-04	7.9679e-01	657	7.9691e-01	516
1.0000e-05	1.0668e-01	1,700	1.0657e-01	6,775
1.0000e-06	3.3542e-02	5,239	3.3525e-02	21,270
1.0000e-07	1.2659e-02	16,627	1.2667e-02	49,415

Table 3.4: Required number of time steps for BE-FE vs uniform for decreasing δ_{tol}.

As we expected, our problem contains a significant length of time where the solution is in the metastable state and no dynamics are occurring, and we see the reduction in time steps required for the adaptive method which increases the step rapidly during this time. This is the most prominent gain over the uniform method. We can also apply this method to the other equation we are investigating, the Cahn-Hillard equation.

3.12 Numerical Experiment - Cahn-Hilliard

The Cahn-Hilliard equation is volume or mass preserving, and we expect qualitatively different dynamics. We look at problem (1.3), which is fourth-order in space, where the discretization is now

$$\frac{U^{n+1} - U^n}{k} = -\epsilon^2 \Delta_h^2 U^{n+1} + \Delta_h w'(U^{n+1}).$$

with Δ_h^2 the Biharmonic operator. Our method that we applied to the Allen-Cahn problem will extend to this problem in the same manner. Our initial condition is $u_0(x,t) = \cos(2x) + 0.01e^{\cos(x+0.1)}$ on $[0, 2\pi]$. This is chosen to have two similar regions with one slightly smaller than the other. We are interested in the case where the two regions will eventually absorb into one region. The parameters we have chosen are

$$\begin{aligned} \epsilon &= 0.22, & n &= 128, \\ T &= 1,033, & \texttt{tol\_delta} &= 10^{-4}, \\ \theta &= 0.8, & \texttt{tol\_newton} &= 10^{-8}, \\ \gamma &= 3, & u_0(x,0) &= \cos(2x) + 0.01e^{\cos(x+0.1)}. \end{aligned}$$

This configuration demonstrates the two separate regions, as we see that both regions will eventually be absorbed together as we observe in Figure 3.7.

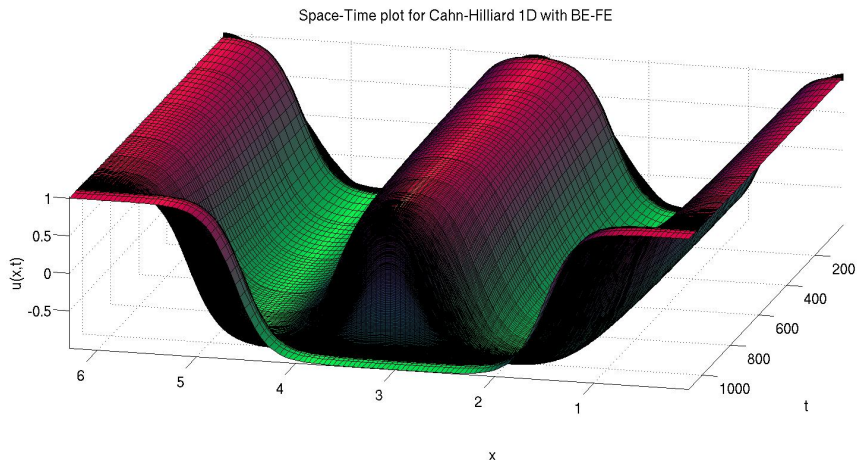

Figure 3.7: BE-FE Solution with Adaptive Time Stepping for Cahn-Hilliard in 1D for $f(x) = \cos(2x) + 0.01e^{\cos(x+0.1)}$ on $[0, 2\pi]$.

As we have seen in the Allen-Cahn case, we also see the fast dynamics from the initial condition to the metastable state, where transition layers are formed. Since one region is slightly smaller than the other, we expect the time step k to be increased rapidly as shown in Figure 3.8.

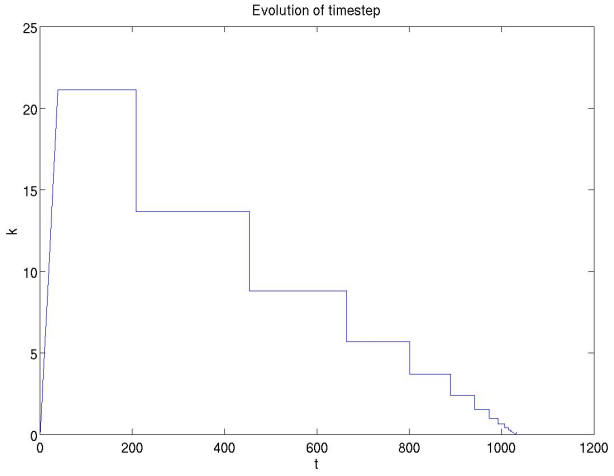

Figure 3.8: BE-FE Solution - Evolution of time step k vs time t for Cahn-Hilliard.

The final result occurs when the center transition layer is eventually absorbed, shown in Figure 3.9 where both regions have joined together.

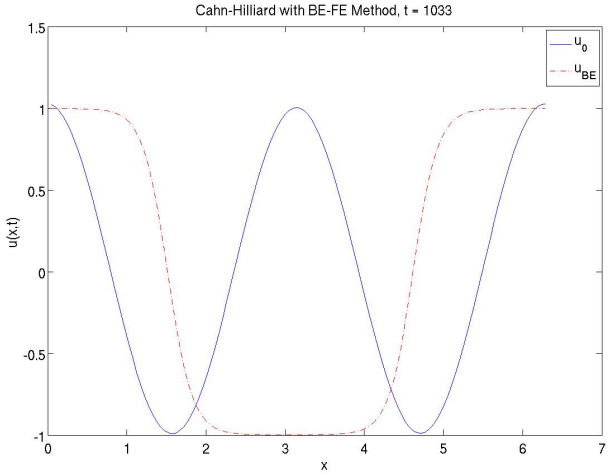

Figure 3.9: BE-FE Solution - Initial condition and final solution for Cahn-Hilliard.

We also plot the Liapunov energy (1.4) to show that we indeed are reducing the energy in time, and we also plot the number of Newton iterations, shown in Figure 3.6, where we have an average 4.34 and a maximum of 12.

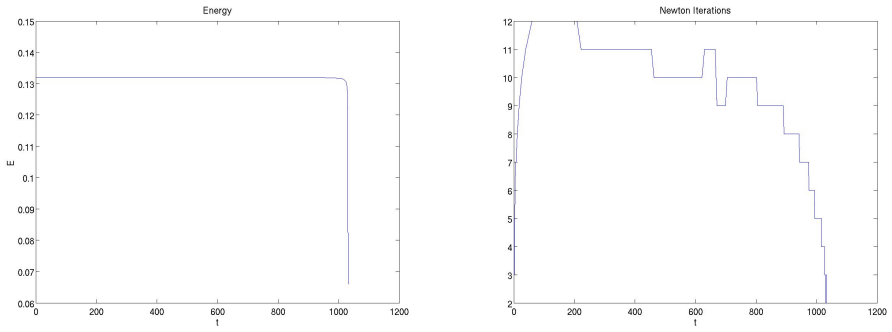

Figure 3.10: BE-FE Solution for Cahn-Hilliard - Liapunov energy vs time step k and Number of Newton Iterations.

The method of solving our equations during Newton Iteration up to this

point include the standard Laplacian and standard techniques such as `A/b` in MATLAB with A defined as a sparse matrix. What we are now interested in is a more efficient method to solve these equations. This comes in the form of a Conjugate Gradient method and Spectral Methods, which we will now investigate.

Chapter 4

Solution Techniques

In this chapter we look at the Conjugate Gradient (CG) method and employ our preconditioner for Preconditioned CG (PCG). In Section 3.5, we solve a linear system of equations for each time step. The CG method is an iterative method that is potentially an efficient method to solve the system

$$A\mathbf{x} = \mathbf{b},$$

where A is symmetric positive definite (spd), where all the entries are real. That is, for an $n \times n$ matrix A, it is symmetric ($A^T = A$), positive definite ($\mathbf{x}^T A\mathbf{x} > 0$) $\forall$ nonzero vectors $\mathbf{x} \in \mathbb{R}^n$. The specific system we will be solving is given in (3.12). The CG method is in a class called Krylov subspace methods. We will give brief background and then present the CG algorithm as this method plays a central role in the methods presented in this thesis.

4.1 Conjugate Gradient

For a small and dense linear system, direct methods would be the most efficient solution to the system. However, in the case of simulating solid cancer

tumor growth [15], for example, the system is very large and sparse where iterative methods will be the most efficient [23].

Krylov subspace methods, [23, 24, 25], are intended to iteratively converge to the solution to the system $A\mathbf{x} = \mathbf{b}$ when $A \in \mathbb{R}^{n\times n}$. The approximations are in a Krylov subspace, which is a subspace spanned by vectors of the form $p(A)\mathbf{v}$ where p is a polynomial of degree up to $k-1$. That is, for any real, $n \times n$ nonsingular matrix A and vector $\mathbf{b}$ of the same length, the k-dimensional Krylov Subspace of A with respect to $\mathbf{b}$ is defined by the space spanned by powers of A, written as $\mathcal{K}_k(A, \mathbf{b}) = \text{span}\left\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, ..., A^{k-1}\mathbf{b}\right\}$ [26]. Then, the solution to our system $\mathbf{x} = A^{-1}\mathbf{b}$ can be solved in a more efficient manner by approximating $A^{-1}\mathbf{b}$ with $p(A)\mathbf{b}$, an element of this subspace.

The CG method is in a class called gradient descent where we can write the iteration as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

where $\mathbf{p}_k$ is known as the search direction and

$$\alpha_k = \frac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{p}_k, A\mathbf{p}_k \rangle}$$

is the step size, where $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$ is the inner product [26]. In terms of a Krylov subspace, we have $\mathbf{x}_k \in \mathcal{K}_k$ and $\mathbf{p}_k \in \mathcal{K}_{k+1}$. The idea behind gradient descent is that the search direction is found by taking the gradient or find the minimizer of

$$\phi(x) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T \mathbf{x}.$$

If we take the gradient of $\phi(x)$ we get $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ which is known as the residual [26]. If our initial guess was $\mathbf{x}_0 = 0$, then our first search direction will be $\mathbf{p}_1 = \mathbf{b}$. In the CG method, we require that the remaining $\mathbf{p}_k$ are each orthogonal, or conjugate to this gradient, which gives its name. The main feature of $\mathbf{p}_k$ is that they are A-conjugate, namely that $\mathbf{p}_k^T A \mathbf{p}_j = 0$ for $k \neq j$ [26]. The next search direction is a linear combination of the previous search direction and current residual as

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

for β_k calculated in [26], given by

$$\beta_k = \frac{\langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle}{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}.$$

Full details into the derivation and discussion about CG can be found in [26, 23]. The conjugate gradient method is given in Algorithm 2.

Algorithm 2 *Conjugate Gradient Method*

Require: $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$

for $k = 0, 1, ...,$ until convergence **do**

$\alpha_k = \langle \mathbf{r}_k, \mathbf{r}_k \rangle / \langle \mathbf{p}_k, A\mathbf{p}_k \rangle$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$

$\beta_k = \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle / \langle \mathbf{r}_k, \mathbf{r}_k \rangle$

$\mathbf{p}_{k+1} = r_{k+1} - \beta_k \mathbf{p}_k$

end for

end

4.2 Preconditioning

Preconditioning is the method by which a matrix P is applied as a transformation to our problem $A\mathbf{x} = \mathbf{b}$ which gives a more suitable problem to solve. This is related to the condition number of the matrix A. The condition number $\kappa(A)$ of A, is the ratio of the maximum and minimum eigenvalues of A. If $\kappa(A)$ is large, the number of iterations required to converge to the solution can also be large. In general, the number of CG iterations to reach a specific residual tolerance is $O\left(\sqrt{\kappa(A)}\right)$. Even if the condition number is large, convergence better than this generic result can be obtained if the eigenvalues are well clustered. A preconditioner P used in the CG algorithm will be a spd matrix such that P^{-1} approximates the inverse of A. Then our CG method would solve a problem where the matrix $P^{-1}A$ has a lower con-

dition number, preferably with eigenvalues clustered near 1. Our CG method with preconditioning is now shown in Algorithm 3.

Algorithm 3 *Preconditioned Conjugate Gradient Method*

Require: $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $P\mathbf{p}_0 = \mathbf{r}_0$

for $k = 0, 1, ...,$ until convergence **do**

$\alpha_k = \langle \mathbf{r}_k, \mathbf{r}_k \rangle / \langle \mathbf{p}_k, A\mathbf{p}_k \rangle$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$

$\beta_k = \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle / \langle \mathbf{r}_k, \mathbf{r}_k \rangle$

Solve $Py = r_{k+1}$ for y

$\mathbf{p}_{k+1} = y - \beta_k \mathbf{p}_k$

end for

end

If we want to employ a standard preconditioner to our iterative method, such as Incomplete LU (ILU), it would have to be calculated for each Newton step. This is because the Jacobian changes for each iteration. This is very inefficient, and we would like to use a preconditioner that avoids calculation at every iteration for each implicit time step. We can employ a linear model preconditioner as suggested to us by Scott MacLachlan. Consider (3.12), the implicit problem to be solved at each time step for the discretization of the Allen-Cahn Equation (1.2). For each Newton iteration, the next iterate

requires a solve with the Jacobian coefficient matrix given in (3.14).

The matrix J is dense but symmetric and multiplication by J can be done efficiently using the fast Fourier transform and diagonal multiplication, which leads to our use of CG as a solution technique. We use the preconditioner P^{-1} where P is the discretization of the constant coefficient problem

$$P = I - k\epsilon^2\Delta_h + 2kI, \tag{4.1}$$

which can be inverted efficiently. This is motivated heuristically by the observation that during ripening, the solution will have values approximated ± 1 at most grid points, as shown in Figure 4.1.

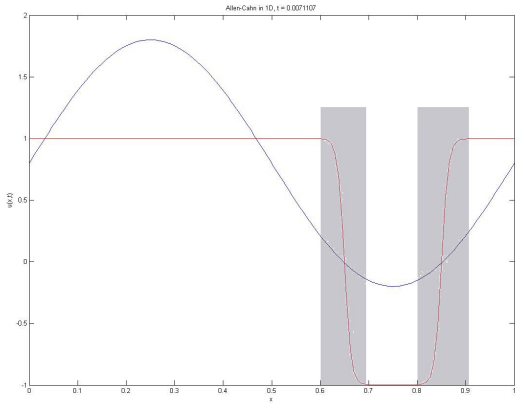

Figure 4.1: The solution in 1D where most is ± 1.

In our CG method used with spectral methods, this matrix can be integrated efficiently into the CG algorithm since it is diagonalized by the dis-

crete Fourier transform. In Figure 4.2 we plot the eigenvalues of the Jacobian J used in our method versus the preconditioned matrix $P^{-1}J$ for the Cahn-Hilliard problem in section 3.12.

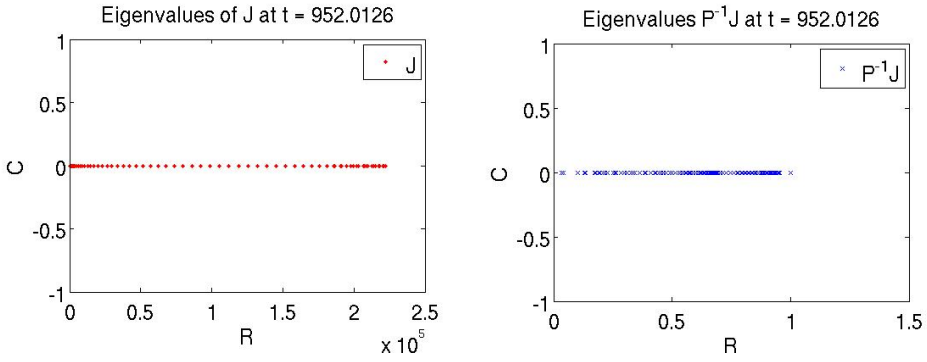

Figure 4.2: (a) Eigenvalues of the Jacobian J. (b) Clustering of Eigenvalues by $P^{-1}J$ near 1.

The desired effects of clustering the the eigenvalues near 1 and reducing the condition number are achieved. For the condition numbers for this problem, for a chosen time during the metastable state are $\kappa(J) = 2.23 \times 10^5$ and $\kappa(P^{-1}J) = 28.23$. In practice computing the inverse of P explicitly would defeat any gains in efficiency since inverting a matrix is computationally expensive. However, we will use a fast method based on the fast Fourier transform to do the solve with coefficient matrix P.

We now compare our PCG with our preconditioner P versus straight CG for the number of iterations required for each time step, as shown in Figure 4.3, for the problem we considered in chapter 3.

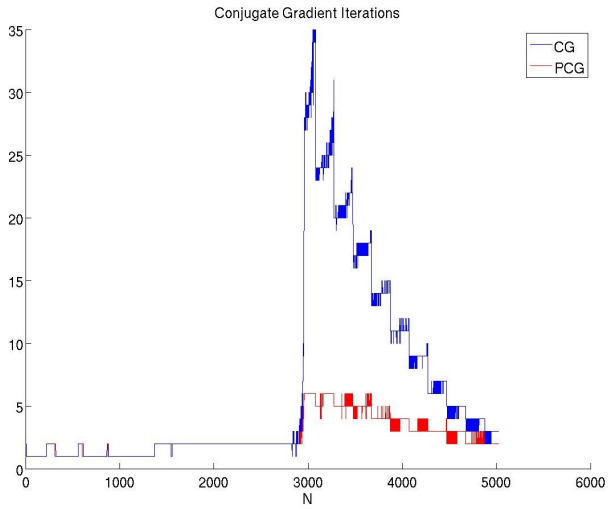

Figure 4.3: Number of iterations vs. time steps for Allen-Cahn using CG vs. PCG.

We see that both methods are relatively the same through the metastable state. As we leave the metastable state our preconditioner reduces the number of iterations required from 35 to 6 and continues to improve as we approach the ripening time. A PCG iteration is roughly twice the computation cost of a CG one. This improvement can also be seen more dramatically as we look at our Cahn-Hilliard problem considered in 3, which is shown in Figure 4.4.

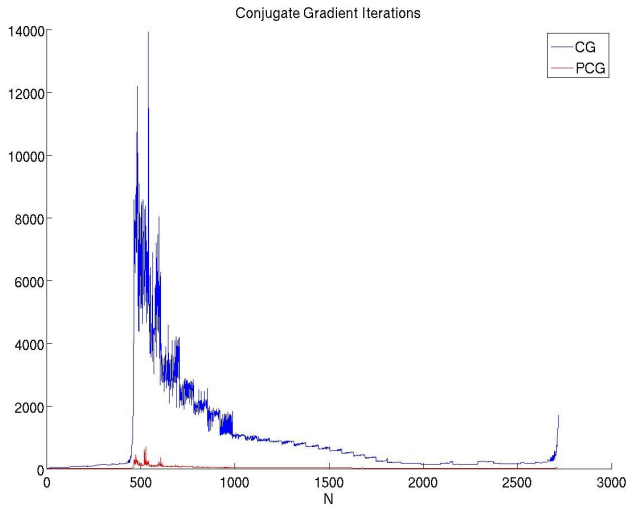

Figure 4.4: Number of CG iterations vs. Time steps for Cahn-Hilliard using CG vs. PCG.

Here we see a significant reduction of iterations required, from 13922 to 689, and continues to improve during the metastable state. The linear model preconditioner suggested to us by Scott MacLachlan has allowed us to improve our CG by reducing the number of iterations to converge, which for the thousands of time steps required for both problems, presents an advantage as we notice a significant speed up when we run our programs. As we stated, our preconditioner is diagonalized through the fast Fourier transform, therefore we now investigate spectral methods to improve our solution and provide an efficient spatial discretization to our problem.

Chapter 5

Spectral Methods in Space

Spectral methods are a class of numerical methods to solve differential equations, usually including the Fast Fourier Transform. They were introduced by Steven Orszag in 1969 [10]. While they are fast to compute, they also have the best possible error properties for our application, specifically called spectral accuracy. We will give a simplified introduction to Spectral Methods below. The material in this chapter is largely adapted from [12].

As we have seen in Chapter 3, the finite difference approximations we make of the derivatives in a differential equation at a point x_j are found using the neighbouring points in the mesh. If we are given initial data $U = \{u_1, ...u_I\}$ at each point $\{x_j\} = \{x_1, ..., x_I\}$ in the domain, then we can approximate the derivatives in this fashion. For example, our familiar second derivative approximation of u at x_j, $u''_j = u''_j(x_j)$ is

$$u''_j \approx \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}$$

We can write this in matrix form as

$$U'' \approx DU,$$

where D is differentiation matrix given by

$$D = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & & & & 1 \\ 1 & -2 & 1 & \ddots & & & \\ & & & \ddots & & & \\ & & & \ddots & 1 & -2 & 1 \\ 1 & & & & 0 & 1 & -2 \end{bmatrix} \tag{5.1}$$

where we have assumed a periodic domain to coincide with our problem. We can also define a fourth order approximation to the second derivative as

$$u_j'' \approx \frac{-u_{j+2} + 16u_{j+1} - 30u_j + 16u_{j-1} - u_{j-2}}{12h^2}$$

where the fourth order differentiation matrix D is

$$D = \frac{1}{12h^2}\begin{bmatrix} & & \ddots & & & -1 & 16 \\ & & \ddots & -1 & & & -1 \\ & & \ddots & 16 & \ddots & & \\ & & \ddots & -30 & \ddots & & \\ & & \ddots & 16 & \ddots & & \\ -1 & & & -1 & \ddots & & \\ 16 & -1 & & & \ddots & & \end{bmatrix} \tag{5.2}$$

The error we expect from using (5.1) and (5.2) will be second and fourth order respectively. We see in Figure 5.3 at the end of this section that we indeed have second and fourth order accuracy, but we require a large number of grid points I for very high accuracy. As we increase the order of accuracy, the more terms D will contain. The number of diagonals of D is called the bandwidth. The spectral approximation results when this bandwidth is extended in a limit. Then we have that D will be a dense matrix.

To introduce the spectral differentiation matrix, we first consider the Discrete Fourier Transform (DFT) and its inverse (IDFT), and we also introduce the phenomenon known as aliasing. Consider the finite grid shown in Figure 5.

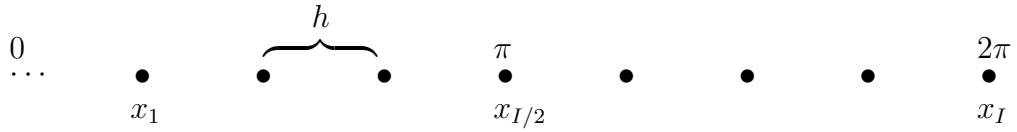

Figure 5.1: Finite grid $h\mathbb{Z}$ on $[0, 2\pi]$.

The spacing of the grid is $h = 2\pi/I$ or

$$\frac{\pi}{h} = \frac{I}{2}.$$

with the convention that the number of grid points is even. Consider a function $v(x), x \in \mathbb{Z}$, evaluated at grid points $v_j = v(x_j)$, the DFT and IDFT are defined as

$$\hat{v}_\alpha = h \sum_{j=1}^{I} e^{-i\alpha x_j} v_j, \qquad \alpha = -\frac{I}{2} + 1, ..., \frac{I}{2}, \tag{5.3a}$$

$$v_j = \frac{1}{2\pi} \sum_{\alpha=-I/2}^{I/2} e^{i\alpha x_j} \hat{v}_k, \qquad j = 1, ..., I. \tag{5.3b}$$

Since this is discrete, we are dealing with a finite problem, where the DFT and IDFT act on vectors. To introduce the phenomenon of aliasing, consider the two complex functions $f(x) = e^{i\alpha_1 x}$ and $g(x) = e^{i\alpha_2 x}$ where $f \neq g$ if $\alpha_1 \neq \alpha_2$. On our finite grid we only consider the value of these two functions at the discrete points x_j. The values are equal on the finite grid if $\alpha_1 - \alpha_2$ is an integer multiple of $2\pi/h$. This is shown in Figure 5.2 for functions $\cos(2\pi x)$ and $\cos(8\pi x)$, where they are equal on $\frac{\pi}{5}\mathbb{Z}$ on $[0, 2\pi]$. We can not

distinguish between the wave modes, which is what is referred to as aliasing. High frequency and low frequency modes are seen as the same.

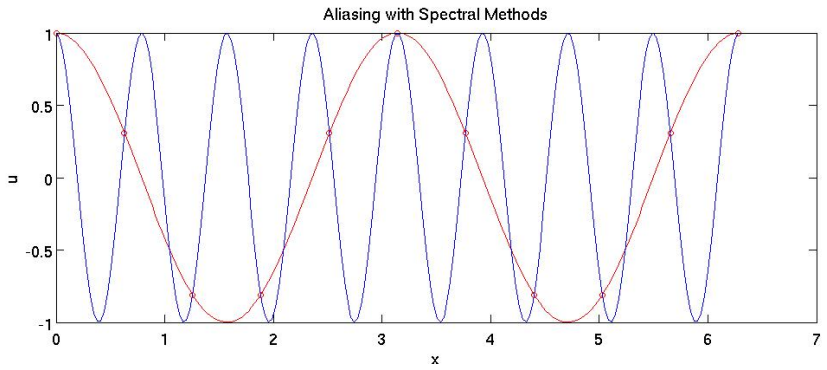

Figure 5.2: Spectral aliasing. Both $\cos(2\pi x)$ and $\cos(8\pi x)$ are equal on $\frac{\pi}{5}\mathbb{Z}$.

To construct the spectral derivative, we introduce an interpolant using (5.3b). Given a $\hat{v}$, define the interpolant p as

$$p(x) = \frac{1}{2\pi} \sum_{\alpha=-I/2}^{I/2}{}' e^{i\alpha x} \hat{v}(\alpha)\, \mathrm{d}k, \qquad x \in [0, 2\pi] \tag{5.4}$$

where $p(x_j) = v_j \ \forall \ j$. The prime indicates that the terms $k = \pm I/2$ are multiplied by $1/2$, which is a special case $\hat{v}_{-I/2} = \hat{v}_{I/2}$ to impose a zero derivative when we take the inverse transform [12]. The interpolant can be constructed explicitly by considering first v_j as the Kroneker delta function. In this case, $\hat{v}_\alpha$ takes on a constant value of h. From [12], computing (5.4) with the delta function yields the periodic Sinc function

$$S_I(x) = \frac{\sin(\pi x/h)}{(2\pi/h)\tan(x/2)}.$$

This interpolation is a translation-invariant process in the sense that for any m, the interpolant of δ_{j-m} is $S_I(x - x_m)$ [12]. Then a linear combination of translated Sinc functions can also be written to describe the interpolant

$$p(x) = \frac{1}{2\pi} \sum_{j=1}^{I} v_j S_I(x - x_j).$$

To find the spectral second derivative matrix, we simply compute the second derivative of $S_I(x)$.

$$S_I''(x_j) = \begin{cases} -\frac{\pi^2}{3h^2} - \frac{1}{6}, & j \equiv 0 \pmod{I}, \\ \frac{1}{2}(-1)^j \csc^2(jh/2), & j \neq 0 \pmod{I}. \end{cases}$$

And the second derivative matrix D can be written as

$$D_I^{(2)} = \frac{1}{h} \begin{bmatrix} & \vdots & \\ & -\frac{1}{2}\csc^2\frac{3h}{2} & \\ \ddots & \frac{1}{2}\csc^2 h & \\ \ddots & \frac{1}{2}\csc^2\frac{h}{2} & \\ \ddots & -\frac{\pi^2}{3h^2} - \frac{1}{6} & \ddots \\ & \frac{1}{2}\csc^2\frac{h}{2} & \ddots \\ & -\frac{1}{2}\csc^2 h & \ddots \\ & \frac{1}{2}\csc^2\frac{3h}{2} & \\ & \vdots & \end{bmatrix} \tag{5.5}$$

Now using (5.5) as our approximation to the second derivative, we see in

Figure 5.3 that the approximation to the second derivative of $f(x) = e^{\sin x}$ achieves very high accuracy with a significantly less number of grid points compared to (5.1) and (5.2) up to 10^{12}. What we observe once $N > 30$ is an accuracy that has reached machine precision. Therefore we can choose a much smaller N than our standard methods, and any larger N will not achieve any better accuracy, as shown in the cloud of data points in Figure 5.3.

Aliasing is the only source of error in this approach, small for C^∞ functions because $\hat{v}(\alpha)$ decays like $1/\alpha^p$ for every $v_j \in C^\infty$. Such a high accurate approximation is limited by machine precision and no further improvement is possible. It can be shown that the approximation is asymptotically better than any fixed convergence order when applied to smooth (C^∞) functions [12]. This is what is referred to as spectral accuracy. In contrast to finite differences, where the error decreases like $O(h^p)$ for a p-order approximation, we achieve $O(h^p)$ convergence for any p assuming the solution is infinitely differentiable. The spectral differentiation matrix is diagonalized by the DFT, which is evaluated efficiently using the FFT. And using Newton iteration, we can solve our problem as we have before now with the spectral differentiation matrix.

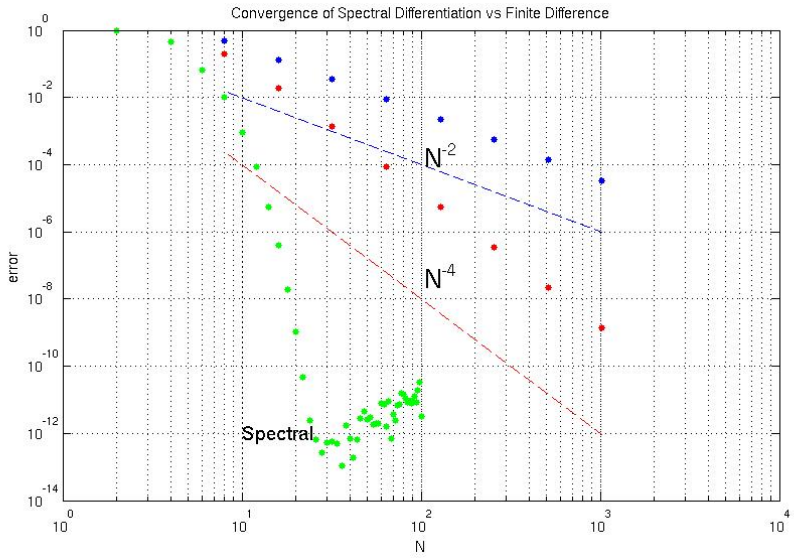

Figure 5.3: Spectral order accuracy versus Finite Difference, on a test problem for the second derivative of $f(x) = e^{\sin x}$ taken as a periodic function on $[0, 2\pi]$.

5.1 Using the Fast Fourier Transform

Instead of using the full spectral differentiation matrix (5.5), we can use the FFT and IFFT to compute approximations of second derivatives of u in a faster way on the uniform grid with I grid points [13]. Recall the DFT in (5.3a) and IFDT in (5.3b). We first obtain I discrete Fourier coefficients $\hat{v}_\alpha$ by taking the FFT of u_j on grid points x_j. However, as stated, because of aliasing the spectral method provides us with $\hat{u}_j$ for $|\alpha_j| \leq I/2$ and all other

modes of $\alpha_j + I$ are indistinguishable on the grid. We write the vector $\boldsymbol{\alpha}$ as

$$\boldsymbol{\alpha} = \left(0, 1, ..., \frac{I}{2} - 1, \frac{I}{2}, -\frac{I}{2} + 1, ..., -2, -1\right).$$

In Fourier representation, differentiation is diagonal multiplication by $i\boldsymbol{\alpha}$. Second derivatives are achieved by multiplication by $-\alpha_j^2$. Second derivative values are then obtained using the IFFT. To summarize,

$$D_I^{(2)} = F^{-1}\Lambda_2 F,$$

where F is the matrix representation of the DFT and Λ_2 is a diagonal matrix with entries $-\alpha_j^2$. Note that although $D_I^{(2)}$ is a full matrix, we can multiply by $D_I^{(2)}$ efficiently with this representation. This allows for efficient PCG solution of the problems from implicit time stepping for our models.

Chapter 6

High-Order Time step Techniques

In this chapter, we investigate higher order time stepping methods. Backward Differentiation Formula (BDF) methods are often used for efficient solution of stiff problems. For example, the MATLAB routine `ode15s` optionally uses BDF techniques. We implement BDF time stepping of order 2 and 3 on our problems, with order 2 and 3 Adams Bashforth explicit methods as our predictor, and we will use implicit Runge-Kutta (RK) methods for initial time steps and restarts when the time step is changed.

6.1 Runge Kutta Methods

Consider the initial value problem posed in (3.1). Following [28], Runge considered the midpoint rule in 1895 to improve on the $O(k)$ error of the Euler Method. An intermediate Euler step between t_n and $t_n + k$ with step

size $\frac{k}{2}$ gives the first Runge method as

$$\begin{aligned} K_1 &= f(t_n, u(t_n)), \\ K_2 &= f\left(t_n + \frac{k}{2}, u(t_n) + \frac{k}{2}K_1\right), \\ u(t_n + k) &= u(t_n) + kK_2. \end{aligned}$$

It is shown in [28] that by adding this intermediate step, we now have a method that is of $O(k^2)$, which is a improvement to the Euler method. We can then construct higher order Runge Kutta methods by using more intermediate stages in its calculation.

The general form of a Runge Kutta method to solve the equation $u_t = f(t, u), u = u(x, t)$ is as follows. Let a_{ij}, b_i and $c_i = \sum_{j=1}^{i-1} a_{ij}, i = 1, ..., s$ be real coefficients. The method

$$\begin{aligned} K_i &= f\left(t^n + c_i k, u^n + k\sum_{j=1}^{p} a_{ij}K_j\right), \qquad i = 1, ..., p \\ u^{n+1} &= u^n + k\sum_{i=1}^{p} b_i K_i, \end{aligned} \tag{6.1}$$

is called an p-stage Runge Kutta Method. From Butcher (1964b), we can express a RK method in a table called a Butcher tableau, a general form shown in Table 6.1.

c_1	a_{11}	a_{12}	$\cdots$	a_{1p}
c_2	a_{21}	a_{22}	$\cdots$	a_{2p}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_p	a_{p1}	a_{p2}	$\cdots$	a_{pp}
	b_1	b_2	$\cdots$	b_p

Table 6.1: Runge Kutta Butcher Tableau with p stages.

When $a_{ij} = 0$ for $i \leq j$, a lower triangular matrix, then we have a explicit RK method. If $a_{ij} = 0$ for $i < j$ and at least one $a_{ii} \neq 0$, then we have a diagonal implicit RK method (DIRK). Further, if all diagonal elements are equal, we have a singly diagonal implicit method (SDIRK) [29]. An SDIRK method also has the property that the last step is similar to the last intermediate step, or the last row is equal to the b_i in the Tableau. For example, the Backward Euler scheme can be expressed in a Butcher Tableau as shown in Table 6.2.

1	1
	1

Table 6.2: Backward Euler Butcher Tableau.

From this table we substitute the values into (6.1) with $p = 1$ to get

$$K_1 = f\left(u^n + kK_1\right),$$
$$u^{n+1} = u^n + kK_1,$$

or, more simply as

$$u^{n+1} = u^n + kf(u^{n+1}).$$

In our problem we will consider the SDIRK methods of order two and three for use in our second and third order methods.

6.1.1 Singly Diagonal Implicit Runge Kutta 2 (SDIRK2)

For higher order multi-step methods described in section 6.2 it is usually required to have p previous values for a p order method. Runge Kutta methods do not have this requirement. Therefore these methods are desirable, but are computationally expensive due to the intermediate steps for only one time step. Thus, we will use them to compute previous values needed for multi-step methods initially and when the time step size changes. The Butcher Tableau for SDIRK2 is given by

1	β	0
1	$1-\beta$	β
	$1-\beta$	β

Table 6.3: SDIRK2 Butcher Tableau.

From (6.1), this gives the formula

$$\begin{aligned} K_1 &= f(u^n + k\beta K_1), \\ K_2 &= f(u^n + k(1-\beta)K_1 + k\beta K_2), \qquad \beta = 1 - \frac{\sqrt{2}}{2}, \\ u^{n+1} &= u^n + k[(1-\beta)K_1 + \beta K_2], \end{aligned} \tag{6.2}$$

where β is found as the root of the polynomial $\frac{1}{2} - 2\beta + \beta^2$. This polynomial arises from the order conditions derived in [29], where each order of SDIRK method consists of a p order polynomial to obtain β . Similar to Section 3.7 where we computed the convergence of the BE solution for the Allen-Cahn equation, we can check that SDIRK2 is indeed second order in time of $O(k^2)$. We divide the time step k by 2 successively with a fixed mesh, shown in Table 6.4.

k	Error	C_R
6.0239e-04	1.6938e-06	2.0030
3.0120e-04	4.2301e-07	2.0015
1.5060e-04	1.0570e-07	2.0007
7.5299e-05	2.6419e-08	2.0004
3.7650e-05	6.6038e-09	2.0002

Table 6.4: Second order convergence in time for SDIRK2.

SDIRK2 will become very useful as a second order method in our adaptive time stepping techniques. It allows us to compute initial conditions and restart our methods while maintaining second order accuracy so that the entire method is second order.

6.1.2 Singly Diagonal Implicit Runge Kutta 3 (SDIRK3)

The Butcher Tableau for SDIRK3 is given by

1	β	0	
1	$T_2 - \beta$	β	
1	b_1	b_2	β
	b_1	b_2	β

Table 6.5: SDIRK3 Butcher Tableau.

where

$$\begin{aligned} T_2 &= \frac{1+\beta}{2}, \\ b_1 &= -\frac{6\beta^2 - 16\beta + 1}{4}, \\ b_2 &= \frac{6\beta^2 - 20\beta + 5}{4}. \end{aligned}$$

From (6.1), this gives the formula

$$\begin{aligned} K_1 &= f(u^n + k\beta K_1), \\ K_2 &= f[u^n + k(T_2 - \beta)K_1 + k\beta K_2], \\ K_3 &= f[u^n + k(b_1 K_1 + b_2 K_2) + k\beta K_3], \\ u^{n+1} &= u^n + k[b_1 K_1 + b_2 K_2 + \beta K_3], \end{aligned} \tag{6.3}$$

where β is found as the root of the polynomial $\frac{1}{6} - \frac{3}{2}\beta + 3\beta^2 - \beta^3 = 0$, The

derivation of this formula is found in [29]. The root β to this equation will be irrational, and up to 20 digits we have

$$\beta = 0.43586652150845899942.$$

Now we can check that this method is indeed third order in time of $O(k^3)$. Again similar to Section 3.7 where we computed the convergence of the BE solution for the Allen-Cahn equation, we look at the convergence of SDIRK3 for time step k by dividing k by 2 successively with a fixed mesh, shown in Table 6.6.

k	Error	C_R
6.0239e-04	4.7413e-08	2.9531
3.0120e-04	6.0273e-09	2.9757
1.5060e-04	7.5990e-10	2.9876
7.5299e-05	9.5400e-11	2.9937
3.7650e-05	1.1952e-11	2.9967
1.8825e-05	1.4957e-12	2.9984

Table 6.6: Third order convergence in time for SDIRK3.

SDIRK3 will be used in our third order method for restarts to maintain a fully third order method. Now we will consider our predictor, where we require a second and third order explicit formulas to coincide with our second and third order methods. We have chosen Adams Bashforth methods, which are linear multi step methods.

6.2 Linear Multi Step Methods

A linear multi step method (LMM) of n steps requires the use of previous information in the last n time steps. Using more than one previous time step is different than RK methods, and we need a method to compute this additional information. We will use implicit SDIRK methods to compute the previous time steps needed for use with LMM's. The general form of LMM's are

$$\sum_{j=0}^{n} a_j u_{n-j} = k \sum_{j=0}^{n} b_j f_{n-j}$$

where n is the number of steps in the method, and a_j, b_j are coefficients specific to the method.

6.2.1 Adams-Bashforth 2 (AB2)

Explicit LMM's such as Adams-Bashforth (AB) solve the equation $u_t = f(u)$ with

$$u^{n+1} = u^n + \int_{t_{n-1}}^{t_n} f(t, u(t))dt,$$

where $f(t, u(t))$ is approximated by an explicit interpolation polynomial, which does not involve the value of u^{n+1}. Hence, these are explicit methods that will only be used in the estimation our solution, as we will see in our adaptive time stepping methods. For our second order method we will

be using the second order AB2 which has formula

$$u^{n+1} = u^n + \frac{k}{2}\left[3f(u^n) - \frac{1}{2}f(u^{n-1})\right]. \tag{6.4}$$

Similar to our BE-FE adaptive in time solution in Chapter 3, we use the difference between this explicit predicted value and the solution from the implicit time step described below to estimate the local error δ made in the time step. As a first step, we calculate the error constant C for the AB2 method.

Error Constant

For a second order method, we expect the local error to be approximately a constant C times k^3 times u_{ttt}. We can explicitly calculate this constant for AB2 using Taylor series. Assuming we have continuity and differentiability up to the third derivative, expanding a Taylor series of u^{n+1} and u^n with respect to the next time step t_{n+1}, where $t_{n+1} - t_n = k, t_{n+1} - t_{n-1} = 2k$ and $f(u) = u_t$, we have

$$\begin{aligned} u^{n+1} &= u \\ u^n &= u - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{3!}u_{ttt} + ... \\ f(u^n) &= u_t - ku_{tt} + \frac{k^2}{2}u_{ttt} + ... \\ f(u^{n-1}) &= u_t - 2ku_{tt} + 2k^2u_{ttt} + ... \end{aligned} \tag{6.5}$$

Substituting (6.5) into (6.4), we have

$$u = u - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{3!}u_{ttt} + \frac{k}{2}\left(3u_t - 3ku_{tt} + 3\frac{k^2}{2}u_{ttt} + ... - \frac{1}{2}u_t + ku_{tt} - k^2u_{ttt} - ...\right).$$

Looking at each term in orders of k, we have

$$\begin{aligned} O(1): &\quad (1-1)u = 0 \\ O(k): &\quad \left(-1 + \frac{3}{2} - \frac{1}{2}\right)ku_t = 0 \\ O(k^2): &\quad \left(\frac{1}{2} - \frac{3}{2} + 1\right)k^2u_{tt} = 0 \\ O(k^3): &\quad \left(-\frac{1}{6} - \frac{3}{4} - \frac{1}{4}\right)k^3u_{ttt} = \frac{5}{12}k^3u_{ttt} \end{aligned}$$

This leaves one term of $\frac{5}{12}k^3u_{ttt}$. Thus the error constant for AB2 is $C = \frac{5}{12}$.

6.2.2 Adams-Bashforth 3 (AB3)

The third order AB3 is given by

$$u^{n+1} = u^n + \frac{k}{12}(23f(u^n) - 16f(u^{n-1}) + 5f(u^{n-2})), \tag{6.6}$$

which will be used as our explicit predictor in our third order method. Similar to our calculation in Section 6.2.1 for the error constant, using Taylor series we calculate the constant C as a first step to correctly estimate δ. The term that does not vanish is $\frac{3}{8}k^4u_{tttt}$. Thus the error constant for AB3 is $C = \frac{3}{8}$.

We now look at our corrector schemes to be used in our methods.

6.2.3 Backward Difference Formula 2 (BDF2)

Backward Difference Formulas (BDF) are implicit LMM's, and are commonly used for stiff problems. Also, BDF formulas interpolate values of u instead of the rhs f. In the case of the second order BDF (BDF2), our discretization of the equation $u_t = f(u)$ is

$$\frac{3}{2}u^{n+1} - 2u^n + \frac{1}{2}u^{n-1} = kf(u^{n+1}), \tag{6.7}$$

where we now require two previous time steps. Given the initial condition, we will use the SDIRK2 method to calculate these previous time steps, although it is more work than Crank Nicholson or BE because it preserves the local truncation order of BDF2 and has good stability properties.

Error Constant

Similar to AB, we can compute the error constant for BDF2. expanding a Taylor series of u^{n+1} and u^n with respect to the next time step t_{n+1}, where

$u^{n+1} - u^n = k, u^{n+1} - u^{n-1} = 2k$ and $f(u) = u_t$, we have

$$\begin{aligned} u^{n+1} &= u \\ u^n &= u - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{3!}u_{ttt} + \frac{k^4}{4!}u_{ttt} - ... \\ u^{n-1} &= u - 2ku_t + 2k^2u_{tt} - \frac{4k^3}{3}u_{ttt} + \frac{2k^4}{3}u_{ttt} - ... \end{aligned} \tag{6.8}$$

Substituting (6.8) into (6.7), we have

$$\begin{aligned} &\frac{3}{2}u - 2\left(u - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{3!}u_{ttt} + ...\right) \\ &+\frac{1}{2}\left(u - 2ku_t + 2k^2u_{tt} - \frac{4k^3}{3}u_{ttt} + ...\right) = ku_t. \end{aligned}$$

Looking at each term in orders of k, we have

$$\begin{aligned} O(1): &\quad \left(\frac{3}{2} - 2 + \frac{1}{2}\right)u = 0 \\ O(k): &\quad (2 - 1 - 1)ku_t = 0 \\ O(k^2): &\quad (-1 + 1)k^2u_{tt} = 0 \\ O(k^3): &\quad \left(\frac{1}{3} - \frac{2}{3}\right)k^3u_{ttt} = -\frac{1}{3}k^3u_{ttt} \end{aligned}$$

This leaves one term of $-\frac{1}{3}k^3u_{ttt}$. Thus the error constant for BDF2 is $C = -\frac{1}{3}$.

6.2.4 Backward Difference Formula 3 (BDF3)

In the case of the third order BDF (BDF3), our discretization of the equation $u_t = f(u)$ is

$$\frac{11}{6}u^{n+1} - 3u^n + \frac{3}{2}u^{n-1} - \frac{1}{3}u^{n-2} = kf(u^{n+1}). \tag{6.9}$$

Similar to Section 6.2.3, we can calculate the error constant C for BDF3 using Taylor series. The term that does not vanish is $-\frac{1}{4}k^4u_{tttt}$. Thus the error constant for BDF3 is $C = -\frac{1}{4}$.

6.3 BDF2 with AB2 Local Error Estimation and SDIRK2 Restart

We have seen in Chapter 3 the performance of the BE-FE solution. The local solution can be predicted more accurately using the second order Adams-Bashforth method (AB2) [30]. Then we compute a BDF2-AB2 pair for each time step in the same manner as we did for BE-FE. Since BDF2 is a second order method, we would prefer a second order method to compute the first time step, as well as the restart when we adjust the time step. We will use the SDIRK2 method given in section 6.1.1. The local error estimate is carried out by calculating the explicit AB2 method, given by (6.4). We use Newton iteration for our solution, both in the SDIRK2 method and BDF2 method. At each time step we will compare our AB2 explicit solution to our BDF2

solution to determine how we adjust the time step. For each time step in BDF2, we employ the following steps:

1. Compute the AB2 solution u_{AB2}.
2. Compute the BDF2 solution u_{BDF2} using Newton iteration and the AB2 solution as an initial guess.
3. Compute the relative local error estimate $\delta = C||u_{BDF2} - u_{AB2}||$.
4. Accept or Reject this new solution based on this error and update the time step accordingly.

The constant C is given by $C = \left|\frac{C_{BDF2}}{C_{BDF2}-C_{AB2}}\right|$, which we have calculated, so $C = \frac{1/3}{1/3+5/12} = \frac{4}{9}$.

Once we have calculated the AB2 and BDF2 solution, we use the same δ requirements as the BE-FE method. The requirements were:

- if $\frac{\delta}{\delta_{tol}} < \frac{1}{\gamma}$. Accept u, $k_{n+1} = \xi k_n$ (Increase time step).
- else if $\frac{\delta}{\delta_{tol}} \in \left[\frac{1}{\gamma}, 1\right)$. Accept u, $k_{n+1} = k_n$ (Keep the same time step).
- else, Reject u, $k_n = \frac{1}{\xi} k_n$ (Reduce time step).

where $\xi = \sqrt[p+1]{\theta\gamma}$, our computed time step factor developed in section 3.9, for user defined values θ and γ. In this case, $p = 2$ for a second order method.

The first case is if $\frac{\delta}{\delta_{tol}} < \frac{1}{\gamma}$. We accept the solution and increase the time

step. Unlike our first order method, we now are required to restart our time stepping method. As illustrated in Figure 6.1, we take our computed solution u^{n+1} as our new u_0^* initial condition and calculate the next time step u_1^* using SDIRK2 that will be used in the next iteration.

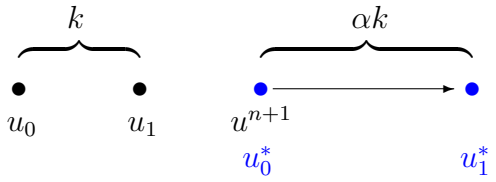

Figure 6.1: Restart: Increasing time step for BDF2-AB2.

We then continue our computation with the next BDF2 solution and check our local error again. The second case of our method is when our solution is acceptable and $\frac{\delta}{\delta_{tol}} \in \left[\frac{1}{\gamma}, 1\right)$. Then we are satisfied with the current state of the solution and will continue on to the next step with no change.

The third case, if $\frac{\delta}{\delta_{tol}} > 1$, we reject the time step because the local error is too large. We restart the computation, now with u_1 as our new u_0^* and compute the next time step u_1^* using SDIRK2 with the reduced time step. We then continue the computation with the next BDF2 solution. This is illustrated in Figure 6.2.

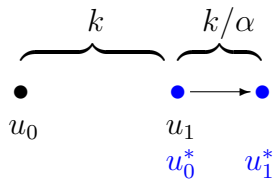

Figure 6.2: Restart: Decreasing time step for BDF2-AB2.

Now we have computed previous time steps for the new time step, and we continue the iteration. In the case where we have rejected a time step, reduced k, and the next time step fails again, we go back and restart again from the last accepted solution. This requires us to retain the last accepted solution as a separate variable. The act of rejecting a time step is also used when our Newton iteration fails to converge, or there are too many CG steps taken. Tolerances `tol_its_newton` and `tol_its_cg` are checked during each Newton iteration. If these counters exceed the given tolerance, we decrease the time step in the same manner. In practice, we do not see any violation of our tolerance for Newton iteration.

Our computation continues to increase or decrease the time step throughout the computation until the time when our given final time is reached. The last time step is found when our next time step exceeds our user defined end time T. When this occurs, we calculate the last time step using a SDIRK2 step. We now want to again assess the confidence we have in our method using the ripening time.

6.3.1 Ripening Time

The ripening time T_r for BDF2-AB2 should be consistent with our BE-FE solution for us to be confident in both methods. We again compute the ripening time by looking at the minimum of our current solution. As we let $\delta_{tol} \to 0$ we expect our solution to converge to this ripening time as we did in Chapter 3. We also expect the number of time steps $M(\delta_{tol})$ to increase as δ_{tol} is reduced by a factor of 10 by a factor of $\sqrt[p+1]{10}$, where p is the order of the method, in this case $p = 2$. Using the same problem for Allen-Cahn as we did in section 3.11, the results are shown in Table 6.7.

δ_{tol}	T_r	$M(\delta_{tol})$	$\frac{M(\delta_{tol}^n)}{M(\delta_{tol}^{n-1})}$
1.0000e-04	546.0982	154	2.3692
1.0000e-05	546.6408	361	2.3442
1.0000e-06	546.8019	787	2.1801
1.0000e-07	546.7919	1,696	2.1550
1.0000e-08	546.7929	3,796	2.2382
1.0000e-09	546.7970	8,288	2.1834

Table 6.7: Convergence of the ripening time for BDF2-AB2.

We see that our solution converges to $T_r = 546.8$ which matches our first order method, and as we expected, the ratio of $M(\delta_{tol})$ also converges to $\sqrt[3]{10} = 2.15443$. We will use these results to compare to our third order method. We also see a reduction in required time steps as $\delta_{tol} \to 0$ compared to our first order method.

To compare our BDF2-AB2 adaptive method to uniform time steps we carry

out the same method in chapter 3. We first compute a very accurate solution using BDF2 with $k = h^3$ as the approximated exact solution and compute the number of time steps required for the BDF2-AB2 solution for various values of δ_{tol}. To compare to the uniform solution, we find the number of uniform time steps which produces the same error using BDF2 time stepping when compared to our highly accurate exact solution. The results are shown in Table 6.8 where we see that our BDF2-AB2 method using our spectral method adaptive strategy is more efficient than uniform time steps as we let $\delta_{tol} \to 0$.

δ_{tol}	Error $\|\|u-u_{\text{exact}}\|\|$	Time steps (Adaptive)	Error $\|\|u_{\text{uni}}-u_{\text{exact}}\|\|$	Time steps (Uniform)(BDF2)
1.0000e-03	4.2091e-01	64	4.1955e-01	549
1.0000e-04	1.3908e-01	140	1.1478e-01	1,079
1.0000e-05	4.2383e-02	332	4.2127e-02	2,450
1.0000e-06	1.0426e-02	730	1.0365e-02	5,150
1.0000e-07	1.7655e-03	1,598	1.7250e-03	14,800

Table 6.8: Required number of time steps for BDF2-AB2 vs uniform for decreasing δ_{tol}.

We see a significant decrease in time steps required using our second order adaptive method. Now we will look at our third order method.

6.4 BDF3 with AB3 Local Error Estimation and SDIRK3 Restart

We consider here a third order scheme similar to the second order scheme described above. BDF3 time stepping is used with an AB3 predictor and using an SDIRK3 scheme for restarts.

We now require two initial time steps to start BDF3 initially and three previous time steps kept throughout the computation. This leads to the use of SDIRK3 for the initial time steps and restarts. Our BDF3-AB3 time stepping method follows the same construct as our BDF2-AB2 method. The local error is now given by

$$\delta = \frac{2}{5}||u_{BDF3} - u_{AB3}||.$$

The acceptance of our solution based on this error follows the same method as our second order method, however now that we have three time steps, we now do two SDIRK3 steps for each restart as illustrated in Figure 6.3.

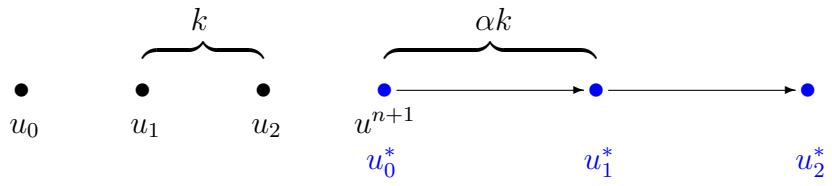

Figure 6.3: Restart: Increasing time step for BDF3-AB3.

Similarly, when we reject a time step, we follow the same method in our second order method computing the two new initial conditions from u_2 required as illustrated in Figure 6.4.

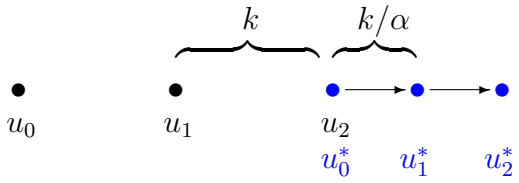

Figure 6.4: Restart: Decreasing time step for BDF3-AB3.

Similarly, we retain the last acceptable solution, in this case u_2. For repeated rejection of a time step, we restart from this last solution. We will see in chapter 7 that this occurs several times for our third order methods since the time step is increased much larger than our lower order methods. To convince ourselves that our third order method matches our lower order methods, we again compute the ripening time for the same problem.

6.4.1 Ripening Time

The ripening time T_r for BDF3-AB3 should be consistent with both our BE-FE and BDF2-AB2 solutions for us to be confident in both methods. We again compute the ripening time by looking at the minimum of our current solution. We also expect the number of time steps $M(\delta_{tol})$ to increase as δ_{tol} is reduced by a factor of 10 by a factor of $\sqrt[p+1]{10}$, where $p = 3$. Using the same problem for Allen-Cahn as we did in chapter 3, the results are shown

in Table 6.9.

δ_{tol}	T_r	$M(\delta_{tol})$	$\frac{M(\delta_{tol}^n)}{M(\delta_{tol}^{n-1})}$
1.0000e-04	546.9734	72	1.3585
1.0000e-05	546.7535	130	1.8056
1.0000e-06	546.8286	270	2.0769
1.0000e-07	546.8152	551	2.0407
1.0000e-08	546.7992	1,055	1.9147
1.0000e-09	546.7976	2,087	1.9782

Table 6.9: Convergence of the ripening time for BDF3-AB3 time stepping method.

We see that our solution converges to $T_r = 546.8$ which matches our first order method in section 3.10 and second order method in section 6.3.1. As we expected, the ratio of $M(\delta_{tol})$ also converges close to $\sqrt[4]{10} = 1.778279$. We have shown that all three methods converge to the same ripening time, which gives us confidence in these methods of adaptive time stepping. We also see a reduction in required time steps as $\delta_{tol} \to 0$ compared to our second order method.

To compare our BDF3-AB3 adaptive method to uniform time steps we carry out the same method in section 6.3.1. We use BDF3 time stepping for our uniform solutions. The results are shown in Table 6.10 where we see that our BDF3-AB3 method using our spectral method adaptive strategy is more efficient than uniform time steps as we let $\delta_{tol} \to 0$.

δ_{tol}	Error $\|\|u-u_{\text{exact}}\|\|$	Time steps (Adaptive)	Error $\|\|u_{\text{uni}}-u_{\text{exact}}\|\|$	Time steps (Uniform)(BDF3)
1.0000e-03	1.5457e-02	47	1.5368e-02	692
1.0000e-04	1.0360e-02	65	1.0587e-02	741
1.0000e-05	8.6347e-03	115	8.6618e-03	765
1.0000e-06	1.8725e-03	254	1.8257e-03	915

Table 6.10: Required number of time steps for BDF3-AB3 vs uniform for decreasing δ_{tol}.

We have now established a time stepping technique which has a first, second and third order versions. We would like to establish the amount of work required for each method, which can be compared using the total number of PCG iterations for the problems we investigate. We now present our methods on problems with small ϵ in both 1D and 2D to show the differences of our three methods.

Chapter 7

Numerical Experiments

In this chapter, we focus on the solution to the Allen-Cahn and Cahn-Hilliard problems with small ϵ. As we have shown in chapter 3, our adaptive-in-time strategy is more efficient than uniform time steps in this case. We will now employ our time stepping methods with spectral discretization and using PCG to solve for each Newton iteration at each time step. We begin in one space dimension with the Allen-Cahn equation, where the initial condition is chosen to be periodic and having the property that only two transitions from $u = \pm 1$ result as the solution progresses. In the case of the Allen-Cahn equation, this one hump solution will be eventually consumed into one region of $u = 1$, while in the Cahn-Hilliard case we consider initial conditions in which four transition layers appear in early dynamics. Since this model is volume preserving, the regions of $u = -1$ cannot disappear but the dynamics will result in both regions joining together.

7.1 Allen-Cahn in 1D

For the Allen-Cahn equation in 1D, we apply our method to the same problem in chapter 3, but now with $\epsilon = 0.12$. We compute the solution for all three methods to show how our second and third order methods perform versus our first order method. In this problem however, the time scale is very large, with $T = 31,055$. Our initial condition is again $u_0(x) = \sin x + 0.8$ on $[0, 2\pi]$. The parameters we have chosen are

$$\begin{aligned} \epsilon &= 0.12, & n &= 128, \\ T &= 31,055, & \texttt{tol\_delta} &= 10^{-4}, \\ \theta &= 0.8, & \texttt{tol\_newton} &= 10^{-8}, \\ \gamma &= 3, & \texttt{tol\_cg} &= 10^{-9}. \end{aligned}$$

What we observe is three phases in the solution. A fast transition from the initial condition to the metastable state. We then experience a long time scale in this state, and then eventually we pass the ripening time, which again is fast as the solution arrives at stable state of $u = +1$. The behaviour where the activity between phases is confined to the start and end of the computation with a long metastable state can be seen in Figure 7.1.

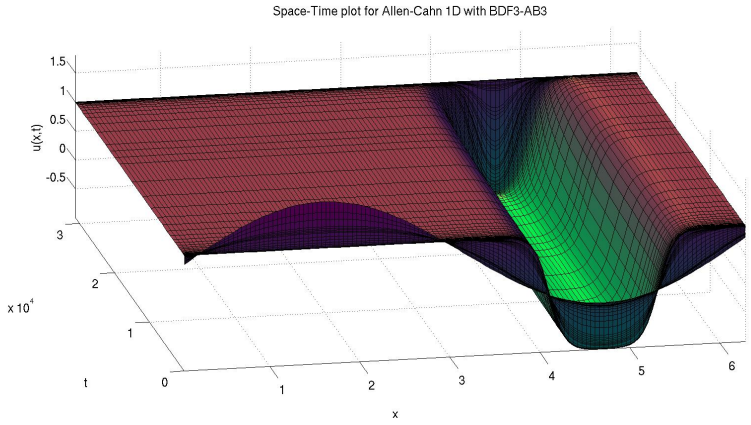

Figure 7.1: Space-Time plot for Allen-Cahn in 1D for $\epsilon = 0.12, \gamma = 3$ for the BDF3-AB3 method.

In this figure we show the BDF3-AB3 solution that exhibits large time steps during the metastable state, seen by increasing distance of grid lines, and a similar situation occurs with our BE-FE and BDF2-AB2 methods. This is seen more clearly in Figure 7.2.

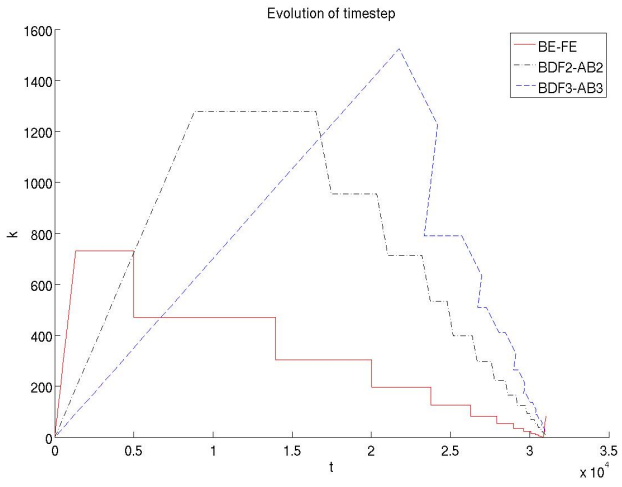

Figure 7.2: Evolution of time step k vs. t for Allen-Cahn in 1D for $\epsilon = 0.12, \gamma = 3$.

We see larger growth in the time step for our higher order methods as we expect. The time step as it is increased reaches very high numbers, which could not be used in a uniform method. Our PCG method, shown in Figure 7.3, displays the number of PCG iterations vs. time for all three methods.

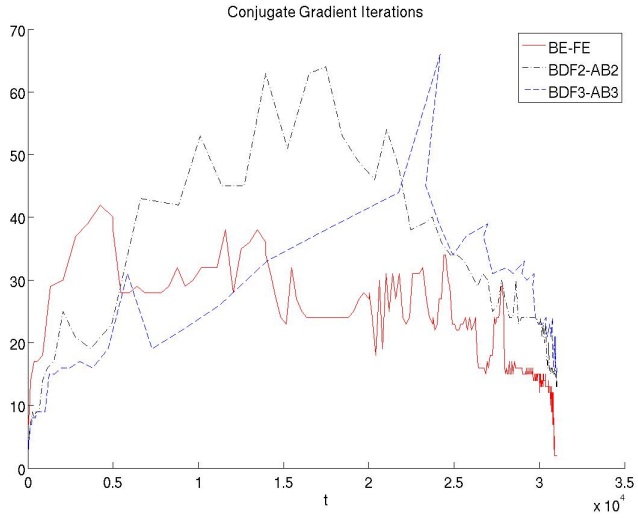

Figure 7.3: Evolution of PCG iterations vs. time for Allen-Cahn in 1D for $\epsilon = 0.12, \gamma = 3$.

There are significantly less time steps required as we move to higher order methods, with only a small increase in the number of CG steps. In Table 7.1 we show the statistics found for each method.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	1,027	83	1.3645	4	8.2924	42	8,508
BDF2-AB2	174	72	1.8023	5	15.9186	64	2,738
BDF3-AB3	89	83	1.6163	5	18.3140	66	1,575

Table 7.1: Statistics for the Allen-Cahn equation in 1D for each method for $\delta_{tol} = 10^{-4}$.

For this problem, we again see an improvement in the number of time steps for higher order methods with maintaining similar results for both Newton solves and PCG iterations for each time step. While an increase in PCG iterations per time step is experienced for higher order, the reduction of time steps is more dominant. This problem could not have been calculated in a realistic time using uniform time steps. As we have seen in chapters 3 and 6, for our three methods to agree on a solution, which we have determined to be at the ripening time, we require δ_{tol} to be smaller. Computing the same problem with $\delta_{tol} = 10^{-8}$ we have the following statistics, shown in Table 7.2.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	55,754	68	1.0000	1	1.8402	7	102,599
BDF2-AB2	4,059	1,375	1.0000	1	3.1627	8	12,831
BDF3-AB3	2,148	714	1.0000	1	4.9855	8	10,694

Table 7.2: Statistics for the Allen-Cahn equation in 1D for each method for $\delta_{tol} = 10^{-8}$.

Here we observe that the Newton scheme converges within one iteration. This is because the solution does not change significantly over one time step.

However, computing our methods for small δ_{tol} has a cost of increased time steps but less PCG iterations per time step. We see a large reduction in the number of time steps required for our higher order methods versus our first order method. This is again a significant advantage of our high order methods. We now apply our three methods to the Cahn-Hilliard equation, which has the property that it is volume preserving. While the Allen-Cahn solutions always approach the stable states $u = \pm 1$, the Cahn-Hilliard equation will result in one dominant region.

7.2 Cahn-Hilliard in 1D

In the Cahn-Hilliard case, we now have two separate regions where our initial condition is $u_0(x,t) = \cos(2x) + 0.01e^{\cos(x+0.1)}$. This is chosen to have two similar regions with one slightly smaller than the other similar to chapter 3. However, in this case we choose $\epsilon = 0.22$, where we expect this to result a longer time scale than our previous problem in chapter 3. The parameters we have chosen are

$$\begin{aligned} \epsilon &= 0.22, & n &= 128, \\ T &= 1,062, & \texttt{tol\_delta} &= 10^{-5}, \\ \theta &= 0.8, & \texttt{tol\_newton} &= 10^{-8}, \\ \gamma &= 3, & \texttt{tol\_cg} &= 10^{-9}. \end{aligned}.$$

Similarly to Allen-Cahn, in early dynamics the solution tends to $u = \pm 1$ in regions with transition layers of width $O(\epsilon)$ between them. This metastable state evolves slowly in time as the center between the two $u = -1$ regions will eventually decrease in size and becomes absorbed. This type of dynamic can be explained as exponential tails for each region interacting with each other [5]. We observe the entire solution in a space-time plot shown in Figure 7.4.

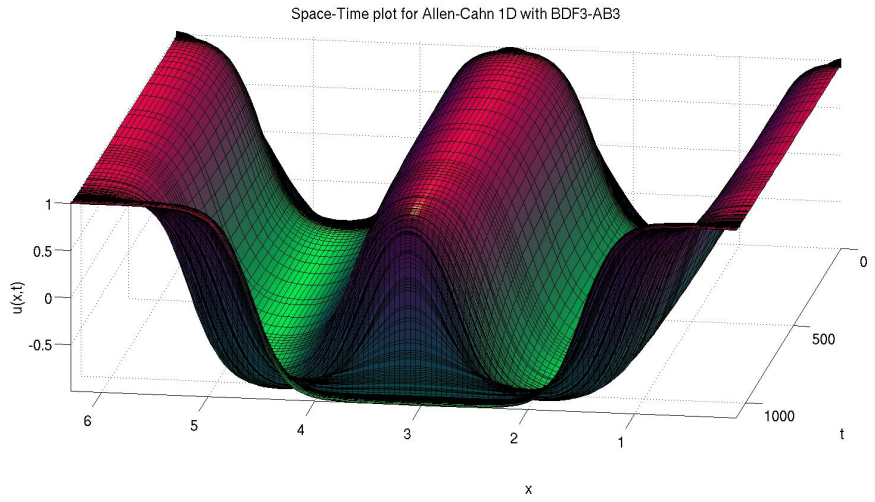

Figure 7.4: Space-Time plot for the Cahn-Hillard in 1D for $\epsilon = 0.22, \gamma = 3$ with the BDF3-AB3 Method.

Figure 7.4 is situated with the end in front to show that indeed both regions were absorbed into one. Similar to the Allen-Cahn model in the previous section, the evolution of the time step has the same type of peak from increasing to decreasing as we pass through the metastable state, shown in Figure 7.5.

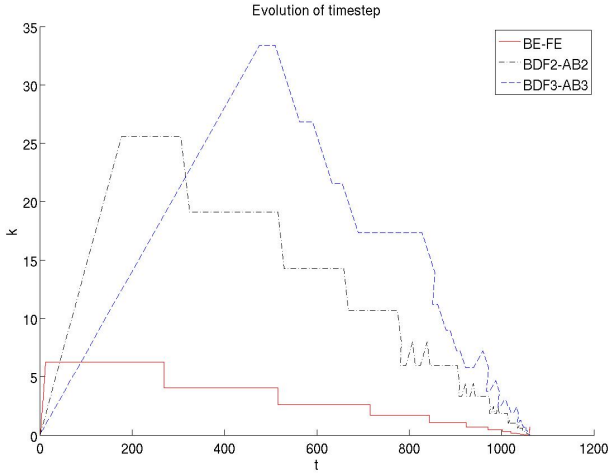

Figure 7.5: Evolution of time step for Cahn-Hillard in 1D for $\epsilon = 0.22, \gamma = 3$.

In this case we observe a similar increase in time step as in our Allen-Cahn problem. The number of PCG iterations vs. time are also similar, shown in Figure 7.6.

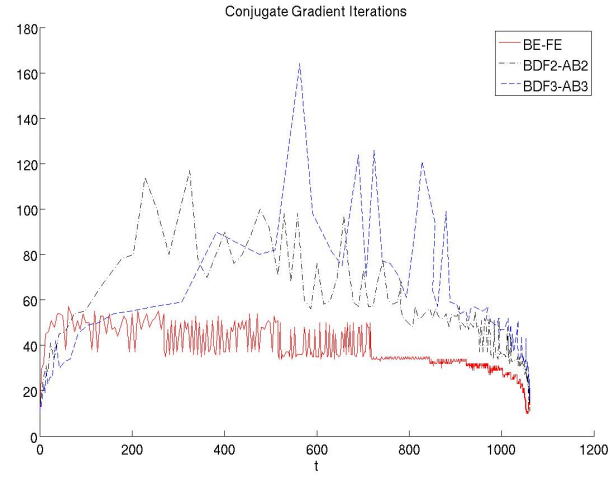

Figure 7.6: Evolution of PCG Iterations vs. time for Cahn-Hillard in 1D for $\epsilon = 0.22, \gamma = 3$.

If we reduce δ_{tol}, we will see a reduction in PCG steps as we saw in the Allen-Cahn case. The statistics for this problem for $\delta_{tol} = 10^{-5}$ are shown in Table 7.3.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	2,244	68	1.4900	3	18.4383	57	41,357
BDF2-AB2	412	102	2.0512	6	30.2195	117	12,390
BDF3-AB3	154	118	2.1457	6	42.8146	164	6,465

Table 7.3: Statistics for the Cahn-Hilliard equation in 1D for each method for $\delta_{tol} = 10^{-5}$.

Again, we see an improvement with our higher order methods while maintaining similar Newton and PCG iterations. Our 1D solution computes very fast even on a laptop and using our BDF3-AB3 method we see a substantial increase in efficiency. To obtain agreement between methods, we again compute the solution for small δ_{tol}, shown in Table 7.4.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	20,243	52	1.1733	2	9.3623	28	189,511
BDF2-AB2	1,814	428	1.3604	3	14.3819	57	26,060
BDF3-AB3	716	359	1.5975	3	18.4404	55	13,148

Table 7.4: Statistics for the Cahn-Hilliard equation in 1D for each method for $\delta_{tol} = 10^{-7}$.

Agreement between our methods for small δ_{tol} is achieved with the cost of increased time steps but for less PCG iterations per time step. Again, we see a large reduction in the number of time steps needed for our higher order

methods versus our first order method. We now present problems in 2D for both the Allen-Cahn and Cahn-Hilliard equations.

7.3 Allen-Cahn in 2D

In two space dimensions, we now have the domain $\Omega = [0, 2\pi]^2$, and our methods are easily extended to this case. Our discretization is now

$$\frac{U^{n+1} - U^n}{k} = \epsilon^2 D U^{n+1} + w'(U^{n+1}),$$

where U is a matrix corresponding to each mesh point in the domain. The matrix D is now the spectral approximation of the 2D Laplacian. The only difference from the method that we have been using in one space dimension is the mesh and the Liapunov energy. In our spectral method, we also use the 2D version of the FFT. We consider the initial condition $u_0(x, y, t) = 2e^{(\sin x + \sin y - 2)} + 2.2e^{(-\sin x - \sin y - 2)} - 1$. The parameters we have chosen are

$$\begin{aligned} \epsilon &= 0.18, & n &= 64, \\ T &= 33, & \texttt{tol\_delta} &= 10^{-5}, \\ \theta &= 0.8, & \texttt{tol\_newton} &= 10^{-8}, \\ \gamma &= 3, & \texttt{tol\_cg} &= 10^{-9}. \end{aligned}$$

We choose $\epsilon = 0.18$, and we expect that the smaller region will be annihilated first followed by the second larger region which occurs at $T = 33$. This is

shown in Figure 7.7, that display the solution as a contour plot for chosen times through the computation.

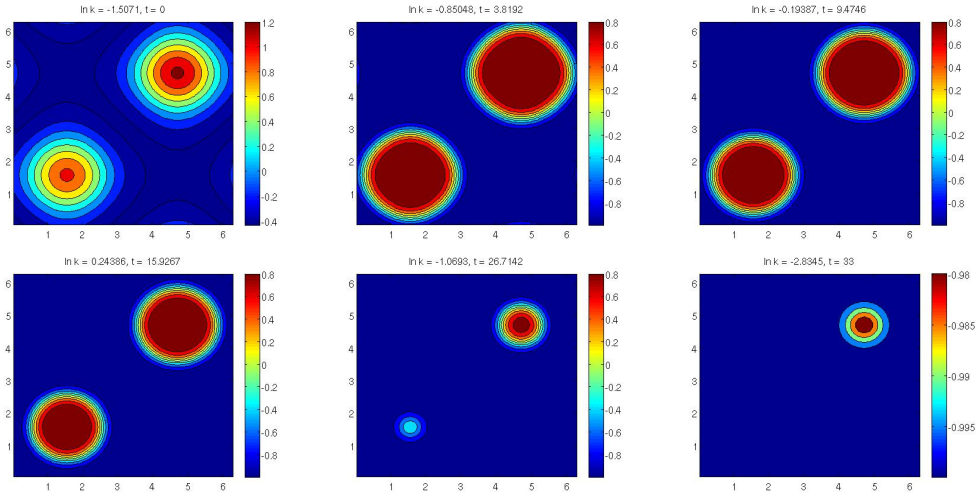

Figure 7.7: Time evolution plots for Allen-Cahn in 2D for $\epsilon = 0.18, \gamma = 3$.

We observe that indeed it is the case that the smaller region is annihilated prior to the larger region. Both eventually reach the stable state of $u = -1$, as we expect. We observe similar results for the evolution of the time step in Figure 7.8.

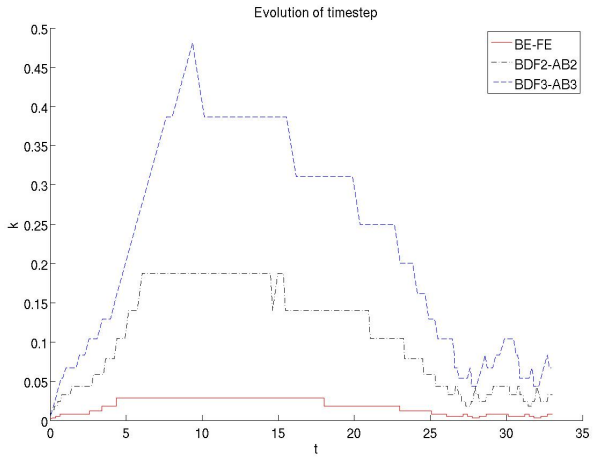

Figure 7.8: Evolution of time step for Allen-Cahn in 2D for $\epsilon = 0.18, \gamma = 3$.

Again, this problem is also difficult to solve using uniform time steps. Our higher order methods increase the time step rapidly and require far less than we would with a uniform method. The number of PCG iterations required is shown in Figure 7.9.

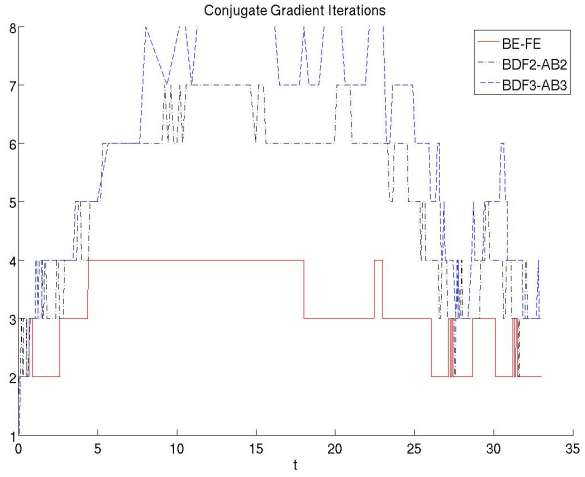

Figure 7.9: Evolution of PCG iterations vs. time for Allen-Cahn in 2D for $\epsilon = 0.18, \gamma = 3$.

While we see an increase in PCG steps for our higher order methods, in Table 7.5 we present statistics for $\delta_{tol} = 10^{-4}$.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	2,970	28	1.0000	1	2.6639	4	7,909
BDF2-AB2	492	48	1.0000	1	4.2898	7	2,102
BDF3-AB3	175	52	1.0000	1	5.4302	8	934

Table 7.5: Statistics for the Allen-Cahn equation in 2D for each method for $\delta_{tol} = 10^{-4}$.

Again, we see a substantial improvement with our higher order methods. To obtain agreement between methods, we again compute the solution for small δ_{tol}, shown in Table 7.6.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	29,099	25	1.0000	1	1.0000	1	29,098
BDF2-AB2	2,384	41	1.0000	1	1.6495	3	3,929
BDF3-AB3	699	51	1.0000	1	2.2773	4	1,585

Table 7.6: Statistics for the Allen-Cahn equation in 2D for each method for $\delta_{tol} = 10^{-7}$.

For 2D, we see that our higher order methods are far more efficient than first order.

7.4 Cahn-Hilliard in 2D

Using the same initial condition as in the Allen-Cahn equation, now with $T = 81$, the parameters we have chosen are

$$\begin{aligned} \epsilon &= 0.18, & n &= 128, \\ T &= 81, & \texttt{tol\_delta} &= 10^{-3}, \\ \theta &= 0.9, & \texttt{tol\_newton} &= 10^{-7}, \\ \gamma &= 2, & \texttt{tol\_cg} &= 10^{-8}. \end{aligned}$$

We show the evolution of the solution for the Cahn-Hilliard problem in 2D in Figure 7.10.

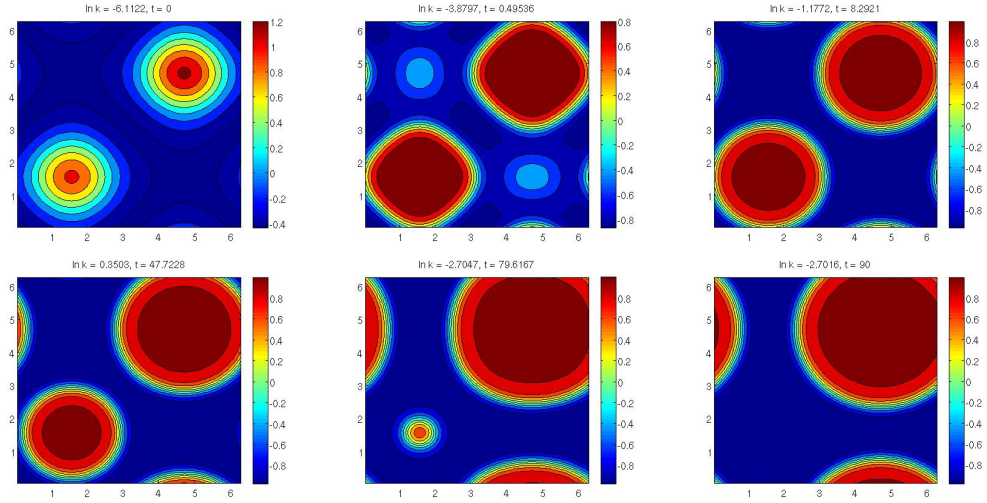

Figure 7.10: Time evolution plots for Cahn-Hilliard in 2D for $\epsilon = 0.18, \gamma = 2$.

In this case, since Cahn-Hilliard is volume preserving, we observe the same behaviour as in 1D, where the larger region consumes the smaller. The evolution of this problem is longer than the Allen-Cahn case with the same initial condition. As shown in Figure 7.11, we observe the increase in time step between our methods.

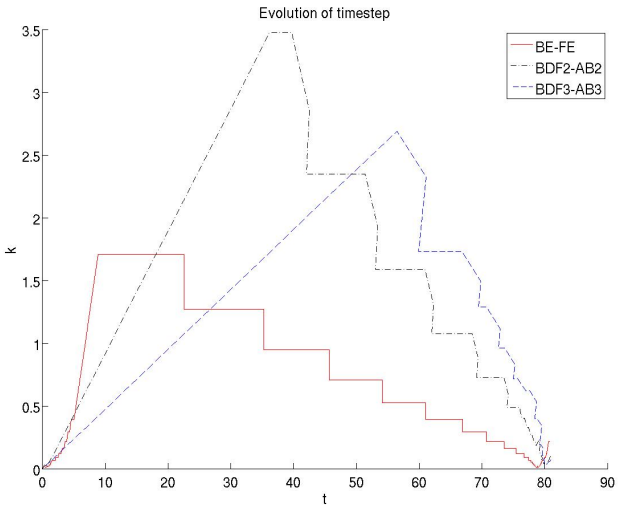

Figure 7.11: Evolution of time step for Cahn-Hilliard in 2D for $\epsilon = 0.18, \gamma = 2$.

In Figure 7.12 we observe the PCG iterations required versus time.

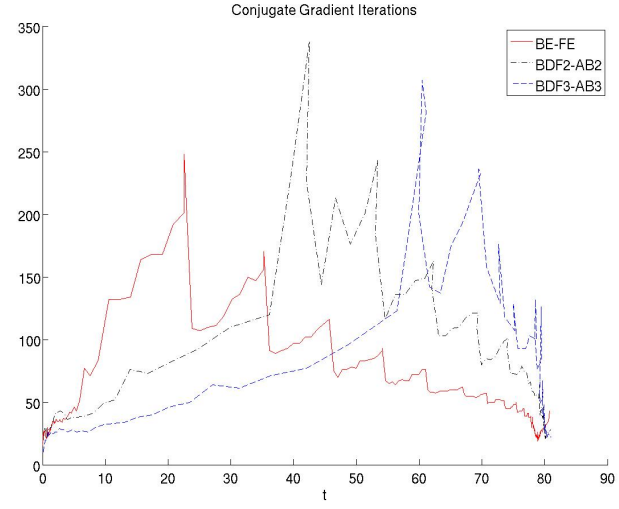

Figure 7.12: Evolution of PCG iterations vs. time for Cahn-Hilliard in 2D for $\epsilon = 0.18, \gamma = 2$.

In Table 7.7 we again see the improvement our higher order methods have on the the number of time steps required to compute the solution.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	380	77	2.4617	5	44.9763	248	17,046
BDF2-AB2	125	102	2.6667	8	70.7642	338	8,704
BDF3-AB3	90	99	2.2414	7	73.0920	307	6,359

Table 7.7: Statistics for the Cahn-Hilliard equation in 2D for each method for $\delta_{tol} = 10^{-3}$.

Again, to obtain agreement between methods, we again compute the solution for small δ_{tol}, shown in Table 7.8, we we see our most improvement in the problems presented.

Method	Time steps	k changes	Newton Its		PCG Its		
			Average	Max	Average	Max	Total
BE-FE	11,162	69	1.1992	2	9.8847	22	110,323
BDF2-AB2	1,309	461	1.3940	2	17.2510	34	22,547
BDF3-AB3	516	274	1.6589	2	22.7018	40	11,646

Table 7.8: Statistics for the Cahn-Hilliard equation in 2D for each method for $\delta_{tol} = 10^{-7}$.

The problems presented here show cases where adaptive time stepping methods are far superior to uniform methods. In addition, we have shown the efficiency gains of using higher order BDF time stepping methods. For very small ϵ, we have shown that these problems exhibit behaviour that can only be calculated in this manner, and we have increased efficiency while not losing accuracy. We have also shown that as we request more accuracy through

δ_{tol}, we have a relationship between an increase of time steps and a decrease in PCG steps. The code for these problems can be easily extended to other problems of this type, such as higher order or vector gradient flows coming from material science models.

Chapter 8

Conclusions and Future Work

We have presented a numerical method for solving the Allen-Cahn and Cahn-Hilliard equations as it relates to material science. Steep transition layers between materials that we see in applications such as annealing, can be resolved and a fast and efficient solution is found using adaptive time stepping techniques. The method presented addresses the challenges in a numerical solution for these problems and offers increased efficiency and the ability to compute problems can can not be solved using traditional methods.

Using asymptotic analysis, we have shown that we expect the transition layers to behave as a tanh function between $u = \pm 1$, with width $O(\epsilon)$ as they move slowly in time. To build a numerical model, we first develop the first order Forward Euler explicit scheme as our predictor and the Backward Euler implicit scheme as our corrector. For our discretization in space we use standard finite difference methods. What results is a nonlinear system of equations to solve for each time step, and we employ Newton iteration to solve this system. Our adaptive strategy is to use the difference between the predictor and corrector as the local error δ. Once δ is within a preset

tolerance, the time step k is then increased. We increase the time step by a factor of ξ, or decrease by a factor of $1/\xi$, for preset values given by the user. The parameter ξ is set up so that our increase will not violate our tolerances for the local error in the next time step. This method is applied to a problem that presents challenges for a uniform method and we show that it is indeed more efficient than the uniform method.

We then utilize spectral methods for the spatial discretization that make the spatial error negligible for a sufficient number of grid points I, and presents a substantial increase in efficiency. We now solve for each Newton iteration using preconditioned conjugate gradient method, where we have developed a linear model preconditioner. It is a constant coefficient version of the system, suggested to us by Scott MacLachlan. The preconditioner for this problem is one of the major contributions of this work. The preconditioner reduces the CG iterations by a significant factor and can be implemented efficiently using the fast Fourier transform.

As the Backward Euler method is a first order time stepping technique, it is natural to consider second and third order methods. Using backward difference formulas of second and third order, coupled with Adams Bashforth methods as our predictor of the same orders, we show that these methods are even more efficient than our first order model. In our numerical experiments, we show that our third order method is the most efficient.

There are many applications of the Allen-Cahn and Cahn-Hilliard equations, with higher order and vector relatives of these equations where our method can be applied. It is our intention to present code that can be easily extensible to these other problems.

8.1 Future Work

While we have shown the power of adaptivity in time for 1D and 2D, for large 3D problems such as cancerous tumor growth the domain may be largely one material and relying on a uniform spatial grid to calculate the solution may no longer be feasible. The computation, as was presented in [15], can be greatly improved with adaptivity in space and time. Since the most activity that occurs during the computation is in the transition layer, it is natural to include more grid points in this area, while less grid points are required far away from the transition layer. In future work, we plan to consider spatially adaptive discretization coupled with multigrid solvers.

There are three general approaches to compute a solution with a non-uniform grid. Consider the layered grid in Figure 8.1.

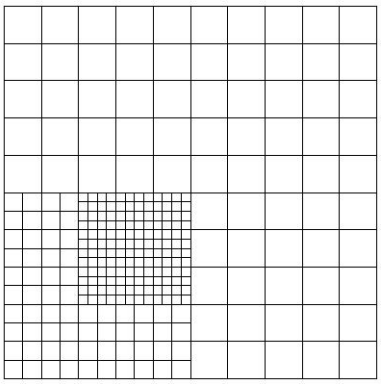

Figure 8.1: Composite grid used in adaptive-in-space methods.

The first method is the Fast Adaptive Composite (FAC) Method by McCormick and Thomas [33]. This method utilizes the entire grid and solves the problem using multigrid. The operators used in the equations are calculated using linear or quadratic interpolation.

The second and third methods are the Multi Level Adaptive Technique (MLAT) and the Fast Approximation Scheme (FAS), both by Brandt [34]. These method uses the same type of composite grid as FAC, but computes each sub grid independently, building the solution for each level again using multigrid. These three methods can be used in conjunction with our adaptive in time method to provide an even more efficient model, especially for large problems.

There are many problems where both adaptive in space and adaptive in time will be most beneficial. This thesis has investigated the first step in

this direction, where we find that a substantial increase in efficiency can be found in adaptive-in-time methods.

We are also interested in more efficient use of CG, such as variable tolerances depending on the performance of the Newton scheme. Since there are a relatively small number of Newton steps in our problems for each time step, it may be beneficial to explore these avenues to gain more efficiency.

Bibliography

[1] S.M. Allen and J.W. Cahn. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta Metall. Mater.* 27 (1979), 1085-1095.

[2] J.W. Cahn and J.E. Hilliard. Free energy of a nonuniform system. I. interfacial free energy. *J. Chem. Phys.* 28 (1958), 258-267.

[3] L. Bronsard, D. Hilhorst. On the slow dynamics for the Cahn-Hilliard equation in one space dimension. *Proceedings: Mathematical and Physical Sciences* 439, 1907 (1992), 669-682.

[4] J. Shen, X. Yang. Numerical approximations of Allen-Cahn and Cahn-Hilliard equations [Preprint]

[5] M. Ward. Metastable bubble solutions for the Allen-Cahn equation with mass conservation. *SIAM Journal on Applied Mathematics* 56, 5 (1996).

[6] C. Grantt and E. Vleckts. Slowly-migrating transition layers for the discrete Allen-Cahn and Cahn-Hilliard equations. *Nonlinearity* 8 (1995), 861.

[7] P. Bates, S. Brown, J. Han, Numerical analysis for a nonlocal Allen-Cahn equation, *Numerical Analysis and Modeling* 6, 1 (2009), 33-49.

[8] J.D. Verhoeven, Fundamentals of Physical Metallurgy, (1975) Wiley, New York.

[9] E. V. L. de Mello, O. T. S. Filho, Numerical study of the Cahn-Hilliard equation in one, two and three dimensions, *Physica A* 347 (2005), 429-443.

[10] S. A. Orszag, Numerical methods for the simulation of turbulence, *Phys. Fluids Supp. II*, 12 (1969) 250-257

[11] MacLachlan, S. (Tufts University), Personal Communication, October 5, 2010

[12] L. Trefethen, *Spectral Methods in MATLAB*, (2000) Society for Industrial and Applied Mathematics.

[13] Canuto, C. et. al., *Spectral Methods in Fluid Dynamics*, (1988) Springer-Verlag.

[14] K. Promislow, B. Wetton, PEM fuel cells: a mathematical overview. *Siam J. Appl. Math* 70, 2 (2009), 359-409.

[15] S.M. Wise, J.S. Lowengrub, V. Cristini, An adaptive multigrid algorithm for simulating solid tumor growth using mixture models, *Mathematical and Computer Modelling* 53 (2011), 120.

[16] P.W. Bates, E.N. Dancer, J. Shi, Multi-spike stationary solutions of the Cahn-Hilliard equation in higher-dimension and instability, *Advances in Diff. Eqts.* 4 (1999), 1-69.

[17] P. Howard, Asymptotic behavior near planar transition fronts for the Cahn-Hilliard equation, *Physica D: Nonlinear Phenomena* 229, 2 (2007) 123-165.

[18] A. Novick-Cohen, Triple-junction motion for an Allen-Cahn/Cahn-Hilliard system, *Physica D: Nonlinear Phenomena* 137, 1-2 (2000) 1-24.

[19] S. Liu, F. Wang, H. Zhao, Global existence and asymptotics of solutions of the CahnHilliard equation, *Journal of Differential Equations* 238, 2 (2007) 426-469.

[20] K.W. Morton, D.F. Mayers, *Numerical Solution of Partial Differential Equations*, (2005) Cambridge University Press.

[21] Euler, L. *Institutiones Calculi Integralis* (1768)

[22] S. Minkoff, N. Kridler. A comparison of adaptive time stepping methods for coupled flow and deformation modeling. *Applied Mathematical Modelling* (2005).

[23] Y. Saad, *Iterative Methods for Sparse Linear Systems*, (2003) Society for Industrial and Applied Mathematics.

[24] J. Demmel, *Applied Numerical Linear Algebra*, (1997) SIAM.

[25] L. Trefethen, D. Bau, *Numerical Linear Algebra*, (1997) SIAM.

[26] U. Ascher, C. Grief, *A First Course on Numerical Methods*, (2011) Society for Industrial and Applied.

[27] J. R. Higgins, Five Short Stories About The Cardinal Series, *Bull. Amer. Math. Soc.* 12 (1985), 45-89.

[28] E. Hairer, S.P. Nørsett, G. Wanner. *Solving Ordinary Differential Equations I - Nonstiff Problems.* (1993) Springer-Verlag Berlin Heidelberg

[29] E. Hairer, S.P. Nørsett, G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems.* (1991) Springer-Verlag Berlin Heidelberg

[30] P. Gresho, D. Griffiths, D. Silvester. Adaptive time-stepping for incompressible flow part I: scalar advection-diffusion. *SIAM J. Sci. Comput.* 30, 4 (2008), 2018-2054.

[31] J. Evans, *Course notes for mathematical modelling and industrial mathematics*, University of Bath.

[32] Fasshauer, G. E. Newton iteration with multiquadrics for the solution of nonlinear PDEs. *Journal of Computers and Mathematics with Applications*, 43 (2002), 423-438.

[33] S. McCormick, J. Thomas, The fast adaptive composite grid (FAC) method for elliptic equations. *Mathematics of Computation*, 46:174 (1986) 439-456

[34] A. Brandt, Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. (1973)