Solving the Ohta-Kawasaki Equations with FEniCS

Ilan Price

July 18, 2017

MSc Special Topic
Course: Python in Scientific Computing
Lecturer: Professor Patrick Farrell
Mathematical Institute
The University of Oxford

1 Introduction

The Ohta-Kawasaki energy functional [11][12] models the behaviour over time of diblock copolymer melts, which are large groups of diblock copolymer molecules, each of which is comprised of two subchains of monomers of different types (to which Ohta and Kawasaki refer as monomers A and B respectively [11]). At low temperatures the combination of the repulsive forces between monomers A and B, and the covalent bonds between them, cause the copolymers to undergo a process of rearrangement into ordered, periodic structures and patterns, whose form vary with interfacial thickness at the intersections of regions of monomers A and B, as well as with changes in a number of other parameters.

In this paper we use FEniCS, a finite element package in Python, to solve the Ohta-Kawasaki Dynamical Equation. Once we have confirmed that our results corroborate those previously obtained by researchers on this topic, we go on to investigate two possible approaches to maximising the computational efficiency of such simulations, which is important given their potential to be very computationally costly.

Section 2 describes the strong form of the equations to be solved, and in Section 3 we derive their variational form. In Section 4 we explain our FEniCS implementation used to solve these equations, and Section 5 contains our primary simulation results. Next, in Section 6, we use the method of manufactured solutions to verify our numerical simulations. Section 7 then investigates a potential alternative to the fully implicit time-discretisation used elsewhere in this paper, in an attempt to transform the nonlinear system into a linear system at each step. Finally, Section 8 proposes an adaptive time stepping algorithm to better balance efficiency and accuracy in simulations of the Ohta-Kawasaki system in certain cases.

2 Strong Form of the Ohta-Kawasaki System

The molecules in a copolymer melt rearrange to minimise their free energy, which is described by the Ohta-Kawasaki functional [12]

$$E(u) = \frac{1}{2}\int_{\Omega}\left(\varepsilon^2|\nabla u|^2 + \frac{1}{2}(1-u^2)^2 + \sigma|(-\Delta_N)^{-1/2}(u-m)|^2\right)\,\mathrm{d}x, \tag{1}$$

where $u(x,t)$ describes the difference between monomer type densities at a given point in the domain Ω, at a given point in time. σ is the non-local energy coefficient, $\varepsilon << 1$ is the scaled interfacial thickness between regions of monomers A and B, and m is the average value of u over the domain, to which we will refer as the mass, and which

is conserved. $u = \pm 1$ represents the pure phases, that is, local presence of only either type A or B monomers.

The subscript N on the Laplacian operator (Δ_N) specifies zero Neumann boundary conditions, such that if

$$(-\Delta_N)^{-1} f = g, \qquad \text{and therefore} \qquad -\Delta_N g = f, \quad \text{with} \quad f \in H^1_*(\Omega), \tag{2}$$

then[1]

$$\left.\frac{\partial g}{\partial n}\right|_{\partial\Omega} = 0. \tag{3}$$

The dynamics are governed by a $H^{-1}(\Omega)$ gradient flow,

$$\langle u_t, \phi \rangle_{H^{-1}(\Omega)} = -dE[u; \phi], \tag{4}$$

where $dE[u; \phi]$ is the Gateaux derivative of E in the direction ϕ. The resulting Ohta-Kawasaki Dynamic Equation (OKDE) is given by [12]

$$u_t + \Delta_N(\varepsilon^2 \Delta_N u - \mathcal{N}(u)) + \sigma(u - m)) = 0, \tag{5}$$

where $\mathcal{N}(u) = u(u^2 - 1) = \Phi'(u)$, and $\Phi(u) = \frac{1}{4}(1 - u^2)^2$.

We eliminate the biharmonic operator by defining

$$w = -\varepsilon^2 \Delta_N u + \mathcal{N}(u) \tag{6}$$

and hence writing the OKDE as a coupled system,

$$u_t - \Delta_N w + \sigma(u - m)) = 0, \tag{7}$$

$$w + \varepsilon^2 \Delta_N u - \mathcal{N}(u) = 0, \tag{8}$$

with the boundary conditions

$$\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = 0, \qquad \left.\frac{\partial w}{\partial n}\right|_{\partial\Omega} = 0, \qquad \forall t \in (0, T], \tag{9}$$

and some initial condition $u = u_0(x, 0)$.

The total (conserved) mass is given by

$$m = \frac{1}{|\Omega|} \int_\Omega u_0 \, \mathrm{d}x. \tag{10}$$

[1] $H^1_*(\Omega) = \{u \in H^1(\Omega) : \int_\Omega u \, \mathrm{d}x = 0\}$

Over and above solving for the evolution of u over time, we also want to track the total free energy $E(u)$ over time. The form of $E(u)$ given in (2) can be equivalently written as

$$E(u) = \frac{1}{2}\int_\Omega \left(\varepsilon^2|\nabla u|^2 + \frac{1}{2}(1-u)^2 + \sigma\langle r, u-m\rangle\right)\,\mathrm{d}x, \tag{11}$$

where

$$r = (-\Delta_n)^{-1}(u-m). \tag{12}$$

Therefore in order to compute this integral we first need to solve Poisson's equation

$$-\Delta r = u - m, \tag{13}$$

with purely Neumann boundary conditions.

Given that the Neumann boundary conditions are only sufficient to specify r up to a constant [1], we require a further constraint on our solution r. We therefore impose that

$$\int_\Omega r\,\mathrm{d}x = 0, \tag{14}$$

by introducing a Lagrange multiplier $c \in \mathbb{R}$ which we will solve for simultaneously [5].[2]

We will refer to this as 'the energy sub-problem', as it needs to be solved for any given time at which we wish to calculate $E(u)$.

3 Variational Form

To convert the system into its variation form, our first step will be to discretise time over the range $[0, T]$ into N intervals, such that $t_n = n\Delta t$ for $n \in [0, N]$ and $\Delta t = T/N$. We use the implicit Euler approximation of the time derivative, such that at t_{n+1} the system in (7) becomes

$$u_{n+1} - u_n - \Delta t\left(\Delta_N w_{n+1} - \sigma(u_{n+1} - m)\right) = 0, \tag{15}$$

$$w_{n+1} + \varepsilon^2\Delta_N u_{n+1} - \mathcal{N}(u_{n+1}) = 0, \tag{16}$$

which is a time independent boundary value problem (BVP) to be solved at each time step.

[2]Furthermore, we note that the existence of a solution r to (13) requires that $\int_\Omega (u-m)\,\mathrm{d}x = 0$ [5]. Fortunately, in our case this is satisfied because $m = \frac{1}{|\Omega|}\int_\Omega u\,\mathrm{d}x$.

Next, we take an $L^2(\Omega)$ inner product of each equation with an arbitrary test function in $H^1(\Omega)$, such that $u_{n+1}, w_{n+1} \in H^1(\Omega)$ satisfy

$$\langle u_{n+1}, q\rangle - \langle u_n, q\rangle - \Delta t\left(\langle \Delta w_{n+1}, q\rangle - \sigma\langle u_{n+1} - m, q\rangle\right) = 0, \qquad \forall q \in H^1(\Omega), \quad (17)$$

$$\langle w_{n+1}, v\rangle + \varepsilon^2\langle \Delta u_{n+1}, v\rangle - \langle u_{n+1}(u_{n+1}^2 - 1), v\rangle = 0, \qquad \forall v \in H^1(\Omega), \quad (18)$$

as well as the Neumann boundary conditions[3].

We note that

$$\begin{aligned}
&\nabla \cdot (q\nabla u) = \nabla q \cdot \nabla u + q\Delta u, \\
\iff &\int_\Omega q\Delta u \, \mathrm{d}x = \int_\Omega \nabla \cdot (q\nabla u) - \nabla q \cdot \nabla u \, \mathrm{d}x, \\
\iff &\int_\Omega q\Delta u \, \mathrm{d}x = \int_{\partial\Omega} q\nabla u \cdot \mathbf{n} \, \mathrm{d}s - \int_\Omega \nabla q \cdot \nabla u \, \mathrm{d}x, \\
\iff &\int_\Omega q\Delta u \, \mathrm{d}x = \int_{\partial\Omega} q\frac{\partial u}{\partial n} \, \mathrm{d}s - \int_\Omega \nabla q \cdot \nabla u \, \mathrm{d}x, \\
\iff &\int_\Omega q\Delta u \, \mathrm{d}x = -\int_\Omega \nabla q \cdot \nabla u \, \mathrm{d}x,
\end{aligned}$$

and so

$$\langle \nabla u, \nabla q\rangle = -\langle \Delta u, q\rangle, \quad (19)$$

where we have used the divergence theorem and the fact that $\frac{\partial u}{\partial n} = 0$ on the boundary. Using these results, and the fact that $\frac{\partial w}{\partial n} = 0$ on the boundary as well, we arrive at the following final variational form of our problem: we seek, at each time step, $u_{n+1}, w_{n+1} \in H^1(\Omega)$ such that

$$\langle u_{n+1}, q\rangle - \langle u_n, q\rangle + \Delta t\left(\langle \nabla w_{n+1}, \nabla q\rangle + \sigma\langle u_{n+1} - m, q\rangle\right) = 0 \qquad \forall q \in H^1(\Omega), \quad (20)$$

$$\langle w_{n+1}, v\rangle - \varepsilon^2\langle \nabla u_{n+1}, \nabla v\rangle - \langle u_{n+1}(u_{n+1}^2 - 1), v\rangle = 0 \qquad \forall v \in H^1(\Omega). \quad (21)$$

The energy sub-problem

At each time step we also need to solve the problem given in (13). As this problem is time-independent, we omit the subscript used in the previous section to specify the time step of the solution u.

[3] In all the steps which follow in this section, we always seek u_{n+1}, w_{n+1} such that both the variational form as well as the boundary conditions are satisfied, though the latter is not explicitly repeated.

Solving (13) is equivalent to finding r which minimises the Dirichlet energy [2]

$$\tilde{E} = \frac{1}{2}\int_\Omega \nabla r \cdot \nabla r \ \mathrm{d}x + \int_\Omega (u - m) \cdot r \ \mathrm{d}x, \tag{22}$$

subject to the constraint

$$\int_\Omega r \ \mathrm{d}x = 0. \tag{23}$$

These are equivalent as taking the Gateaux derivative of the Dirichlet energy returns the weak form of Poisson's equation.

To solve this problem, we define our Lagrangian $L : V \times \mathbb{R} \longrightarrow \mathbb{R}$,

$$L(r, c) = \frac{1}{2}\int_\Omega \nabla r \cdot \nabla r \ \mathrm{d}x + \int_\Omega (u - m) \cdot r \ \mathrm{d}x + c\int_\Omega r \ \mathrm{d}x, \tag{24}$$

where V is the function space in which r lives and c is a Lagrange multiplier. We are looking for stationary points of this Lagrangian, and therefore we seek $r \in V$ and $c \in \mathbb{R}$ such that

$$L_r(c, r, p_1) = 0 \tag{25}$$

$$L_c(c, r, p_2) = 0, \tag{26}$$

for all $(p_1, p_2) \in V \times \mathbb{R}$. Taking the Frechet derivative of L with respect to r in the direction of p_1 gives

$$L_r(c, r, p_1) = \int_\Omega \nabla r \cdot \nabla p_1 \ \mathrm{d}x + \int_\Omega (u - m) \cdot p_1 \ \mathrm{d}x + c\int_\Omega p_1 \ \mathrm{d}x, \quad \forall p_1 \in V, \tag{27}$$

and after differentiating with respect to c we have that

$$L_c(c, r, p_2) = p_2 \int_\Omega r \ \mathrm{d}x, \quad \forall p_2 \in \mathbb{R}. \tag{28}$$

Together these specify the variational form of the energy sub-problem solved at every time step to calculate the free energy $E(u)$.

4 Solving with FEniCS

FEniCS is very powerful, in that it allows you to perform otherwise very difficult scientific computing with comparative ease and with comparatively very few lines of code [8][10]. In this section we describe the important parts of the FEniCS code used to solve the equations defined in the previous sections and obtain the results

in Section 5. Wherever the new methods in Sections 7 and 8 require significant modification of this code, or require extra steps to be added, we explain these in the relevant section.

We begin by defining the various model parameters (σ, ε and m), as well as the mesh on which we will solve the equations (in our case, we begin with the '`UnitSquareMesh`').

Next we construct our finite element '`V`', which we choose to be a linear Lagrange element. We use this element to construct two function spaces. The first is a mixed-element function space (which we call `ME`) in which we will look for our solutions u and w. The second (which we call `P2`) is the subspace of $H^1(\Omega)$ spanned by the Lagrange basis. into which we will later project our function u. Moreover, we will interpolate '`Constant(1.0)`' in this function space P2 when tracking the mass to confirm that $\frac{1}{|\Omega|}\int_\Omega u \, \mathrm{d}u = m$.

We also create a second mixed-element function space, this time from a linear Lagrange element and a Real element, which corresponds to the function space $V \times \mathbb{R}$ in which we will be searching for solutions to the energy sub-problem.

We define two functions in our mixed space `ME`, one (`y0`) to keep track of the solution at the previous time step, which will be used to solve for the second function (`y`) at each time step. We split `y0` into `u0` and `w0`, and `y` into `u` and `w` (without making deep copies). These then become references to the components of `y` and `y0`, which allows us write our variational form in terms of `u`, `u0` and `w`. After defining `q` and `v` as test functions in the mixed space `ME`, the weak form of our problems is coded as

```
L1 = inner(u,q)*dx - inner(u0,q)*dx + dt*dot(grad(w), grad(q))*dx +
    dt*sigma*inner(u-m,q)*dx
L2 = inner(w,v)*dx - (epssi**2)*dot(grad(u), grad(v))*dx -
    inner(dphidu,v)*dx
L = L1 + L2
```

Following the demo in [4], we define two new helpful subclasses, though we amend the suggested code to best suit the purposes of these particular simulations.

First we define an '`InitialConditions`' class, as a subclass of '`Expression`'. This enables us to easily initialise our system with a random initial condition, and is thus very useful given the number of times we are required to do this throughout the investigation.

[3] We will project `u-u0` into `P2` to access the '`.array()`' attribute, in order to approximate $||u - u_0||_{L_\infty}$ in Section 8

```
class InitialConditions(Expression):
   def __init__(self, mass, s, **kwargs):
      random.seed(s)
      self.mass = mass
   def eval(self, values, x):
      values[0] = self.mass + 2*(0.5 - random.random())
      values[1] = 0.0
   def value_shape(self):
      return (2,)
```

Our constructor takes in the mass m as a parameter, which is used by the '`eval`' function to ensure a sensible initial condition for each simulation, with (roughly) the appropriate mean value, as well as a parameter which sets the seed for the random number generation. Once we have instantiated our initial condition, we interpolate it in our `ME` function space, and assign this to our function `y0` with '`y0.assign(ICs)`'.

The second class we define is '`OKsystem`' (a subclass of '`NonlinearProblem`'), which represents our Ohta-Kawasaki problem. The '`NonlinearProblem`' class has member functions to calculate the residual and Jacobian matrix [6]. We initialise an instance of '`OKsystem`' by providing our full variational form `L`, as well as the derivative of `L` with respect to `y`.

After specifying our solver as a '`NewtonSolver`, we manually set a number of parameters, some of which we change in later sections. To begin we use LU decomposition to solve each linear system, and set our convergence criterion 'incremental', that is, to check the size of the incremental step (as opposed to the residual) and set a tolerance of 1e-6.

Our final section of the code, which is where all the 'solving' actually happens, is a loop over time, from $t = 0$ up to the specified final time for the simulation.

At each time step, we solve our two stationary BVPs: first, we solve the OKDE system for `y` (after which we update `y0`), and second, we solve the energy sub-problem for r, after which we use '`assemble`' to evaluate the integral and calculate the energy $E(u)$ at that time step.

Finally, we use '`assemble`' once again to calculate $\frac{1}{|\Omega|}\int_\Omega u \, \mathrm{d}x$ at each time step, to confirm that our system obeys the conservation of mass property which is built into the equations.

We output our results in two formats. First we have the option of adding u at each time step to a '`.pvd`' file, to be viewed in Paraview. Second, we save our energy, mass,

and time-stepping data to a '`.mat`' file, with the '`io.savemat`' command. Hence, our plots in the sections which follow are produced either in Paraview or MATLAB.

5 Results on the Unit Square

We begin by reproducing a number of the results presented by Parsons for a unit square domain [12], which were obtained using MATLAB.

Our first set of results compare the effect of varying the mass m, that is, the (conserved) average value of u over the whole domain. We discretise time setting $\Delta t = \varepsilon^2$ as Parsons does, a time step size for which this scheme is stable [12][4]. We discretise our domain with a 41×41 grid (that is, 40 intervals in each direction), such that $h = 0.025$, and we choose the same parameter values as Parsons, setting $\sigma = 10$, $\varepsilon = 0.08$.

We compare simulations with $m = 0$, the symmetric case (corresponding to equal proportions of monomers A and B), to the case when $m = 0.4$. In each case, we begin with a random initial condition, with a maximum amplitude of 1 on either side of a mean (roughly) equal to the mass in each simulation[5].

In both cases, by $t = 3$, the interesting part of the evolution of u and the pattern formation in the domain has been completed, reaching 'metastable' end states with plateaus in the free energy profile before this time.

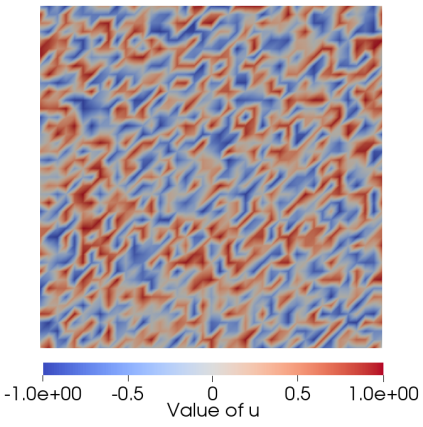
The initial conditions for these simulations, as well as snapshots of the evolution of u at times $t = 0.2$ seconds, and $t = 3$ seconds, are shown in Figure 1.

The results are as we would expect and corroborate those presented in [12]. When $m = 0$, and thus the total proportion of each monomer type in the domain is equal, the metastable end state toward which the system moves is a striped pattern, with alternating vertical stripes of $u = \pm 1$. When $m = 0.4$, the pattern evolves instead towards multiple circular regions of $u = -1$, surrounded by $u = 1$[6].

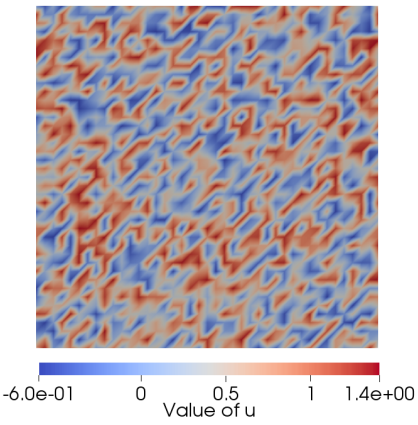
[4] See [12] for the full stability results for this scheme.

[5] Note that although this means in the case with $m = 0.4$ the initial condition is not physical, as $|u| > 1$ in some parts of the domain, we still achieve physical end-states with $|u| < 1$ everywhere.

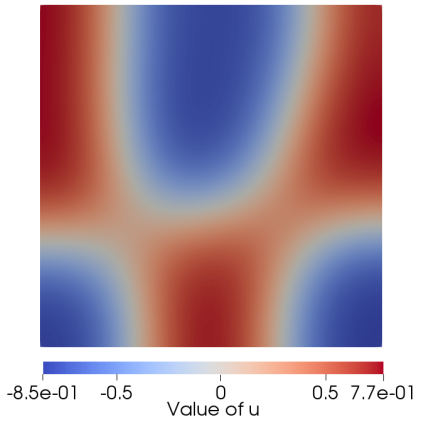
[6] The orientation and order of the stripes when $m = 0$, and the number and location of the circles when $m = 0.4$, depend on the initial condition u_0. For example, the results in Figure 1 were produced by initialising the '`InitialConditions`' with the seed set to 1. Setting the seed equal to 2 produced an $m = 0$ end-state with vertical instead of horizontal stripes.



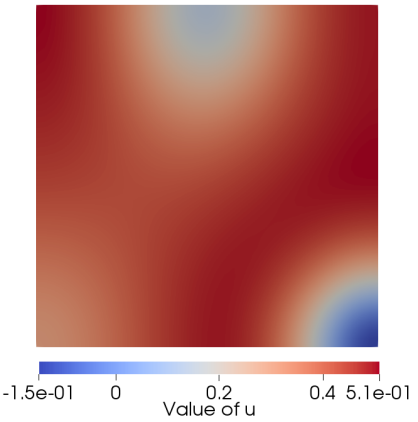
(a) $t = 0$, $m = 0$



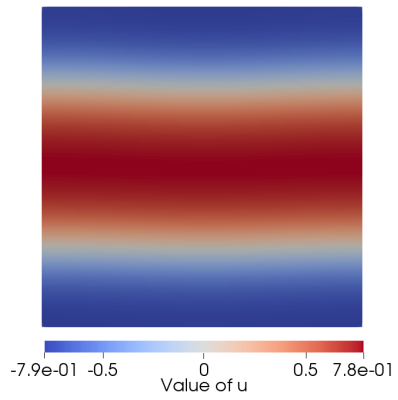
(b) $t = 0$, $m = 0.4$



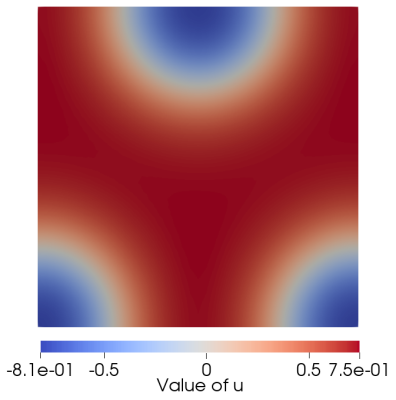
(c) $t = 0.2$, $m = 0$



(d) $t = 0.2$, $m = 0.4$



(e) $t = 3$, $m = 0$



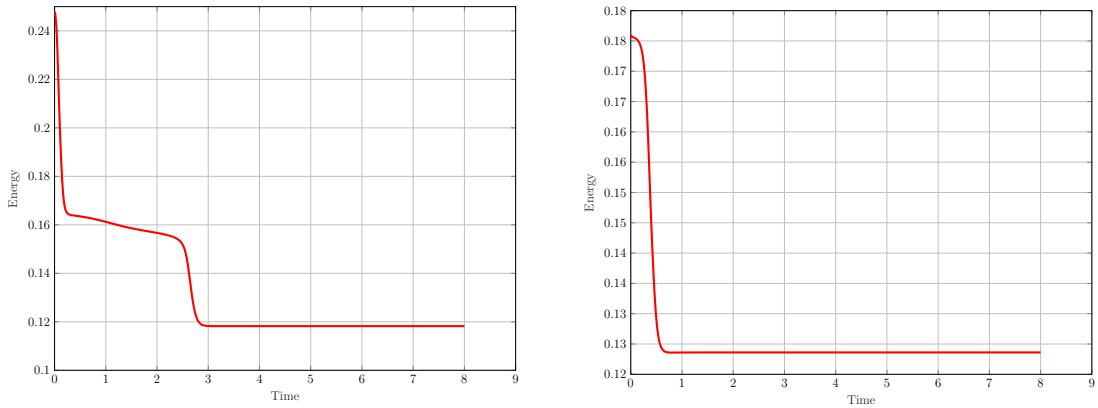
(f) $t = 3$, $m = 0.4$

Figure 1: The evolution of u with $m = 0$ and $m = 0.4$, given $\sigma = 10$, $\varepsilon = 0.08$, $\Delta t = \varepsilon^2$, and $h = 0.025$.

We note that by $t = 3$, the peak absolute values in each region do not yet reach

± 1. We expect that this is as a result of the value of ε, the interfacial thickness, since our simulations below with a decreased $\varepsilon = 0.02$ show minimum and maximum values of u which correspond much more accurately to ± 1.

The evolution of the free energy of these systems is typically characterised by one or more short periods of very rapid change early on, followed by a sharp plateau, after which the arrangement of monomers A and B changes very slowly. To see this clearly, we can examine the evolution of the free energy over time, as shown in Figure 2.



(a) $m = 0$ (b) $m = 0.4$

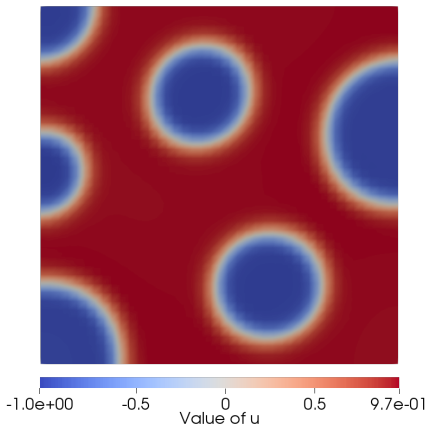
Figure 2: Free energy evolution over time, with $h = 0.025$, $\sigma = 10$, $\varepsilon = 0.08$, and $\Delta t = \varepsilon^2$, for $m = 0$ and $m = 0.4$.

In Figure 2 we plot $E(u)$ starting from $t = \Delta t$ not from $t = 0$. Given the way that our initial conditions for these simulations were created, the values of u_0 at the vertices do not correspond to a differentiable function, and hence $E(u)$, which includes a $|\nabla u|^2$ term, is not properly defined. In other words, the finite difference approximation used to create ∇u is not approximating any derivative. This can potentially cause strange and unphysical results in the first few time steps, as '$u(x + h)$' and '$u(x)$' can be far apart, regardless of how small h is, meaning that each term in the finite difference derivative can blow up, with order $\frac{1}{h}$. For this particular simulation, it is sufficient to omit only $E(u(0, x))$ in order to remove the irregular behaviour.

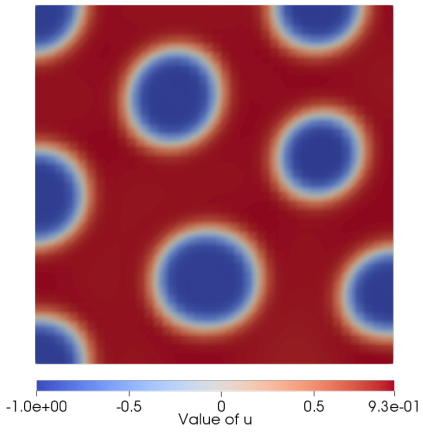
Both when $m = 0$ and $m = 0.4$, the majority of the spatial segregation happens within the first 3 seconds, with short periods for which the negative energy gradient is very steep. Thereafter, the energy gradient flattens out to decrease extremely slowly in comparison to the initial period.

In our next set of simulations we vary the non-local energy coefficient σ. Keeping $h = 0.025$, and again replication Parsons' parameter values [12] by setting $\varepsilon = 0.02$

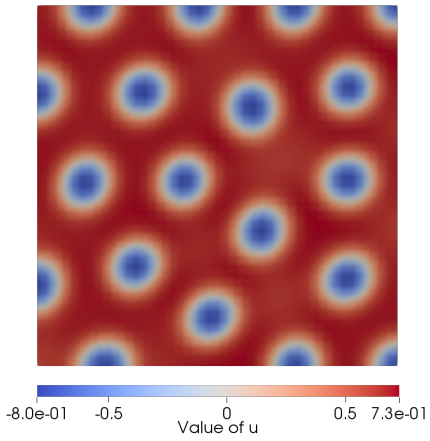
and $m = 0.4$, $\Delta t = \varepsilon^2$, we simulate $\sigma = 2$, 20, 200 and 800. Our results, shown in Figure 3, confirm the results presented by Parsons [12].



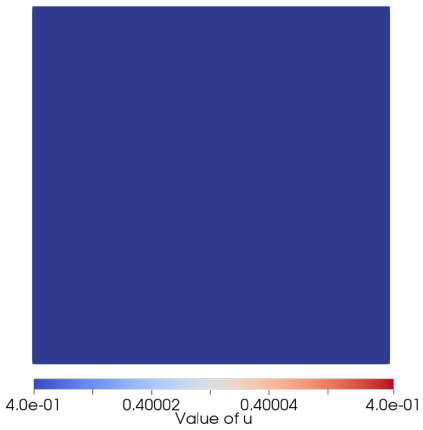
(a) $\sigma = 2$



(b) $\sigma = 20$



(c) $\sigma = 200$



(d) $\sigma = 800$

Figure 3: Varying σ, with $h = 0.025$, $\varepsilon = 0.02$, $\Delta t = \varepsilon^2$, and $m = 0.4$.

In short, the effect of increasing σ is to increase the 'periodicity' with which substructures form in the given domain. That is, as we increase σ, the number of circles which form in our domain increases, and their size decreases. However, when σ is too large, for example when $\sigma = 800$ as shown in Figure 3d, the copolymer melt exhibits no structures or spatial patterns at large times, and the value is almost exactly equal to the average value at every point in the domain. This state is known

in the literature as 'disorder' [12].[7]

6 Method of Manufactured Solutions

In this section we confirm that our FEniCS implementation is producing the correct results. In particular we want to compare the error convergence we get in practice to the predicted convergence derived by Parsons[8].

To this end we apply the method of manufactured solutions (MMS). We construct a closed-form 'solution' $\hat{u}$ (and therefore $\hat{w}$), and substitute this into the left hand side of our equation, to analytically calculate the corresponding right hand side 'forcing' term. We then solve the equation with this added term using our FEniCS code, which, if working correctly, should produce a solution which converges, at the expected rate, to the closed-form solution we 'manufactured'[9].

The full convergence result for u is given by

$$||e||_{L_\infty(0,T,L^2(\Omega))} \leq C(\Delta t + h^2), \tag{29}$$

for some constant C which is independent of Δt and h, so long as $\Delta t \lessapprox \varepsilon^2$ [12].

We will limit our MMS investigation to the 1D case, with our domain as the unit interval $x \in [0, 1]$. This is in part motivated by computational time: for each simulation we need to set $\Delta t = \tilde{C}h^2$, and so given that we need to keep the final time T constant across all simulations, the finest mesh requires an extremely long simulation[10]. For example, to vary h by a factor of 100 requires simulating 10 000 as many time steps.

Fortunately, the 1D case is sufficient for our purposes, as changing the domain over which we are solving our problem involves very little change in the FEniCS implementation itself (indeed it can require nothing more than changing the specified mesh.)

[7] Parsons continues his analysis by examining the evolution of the different constituent 'parts' of the Energy functional over time, examining which dominates in each case and why. While replicating these results is beyond the scope of this paper, we have deliberately written our FEniCS code in a such a way as to calculate each of these 'parts' individually at each time step. It would therefore be a straightforward exercise to store these values at each time step and plot the evolution of each part individually as Parsons does.

[8] To the best of this author's knowledge, this convergence result has yet to be published.

[9] Note that once this right hand side function is added, the equation no longer necessarily models anything physical. The purpose of this exercise is simply to confirm that our code is working as it should.

[10] The value of $\tilde{C}$ is set to ensure that on the coarsest mesh there is no 'blow-up' or instability.

Our chosen $\hat{u}$ (and $\hat{w} = -\varepsilon^2 \hat{u}_{xx} + \hat{u}^2(\hat{u} - 1)$), must satisfy the zero Neumann boundary conditions. For simplicity, we will perform these calculations with $m = 0$, and hence we require further that $\frac{1}{|\Omega|} \int_\Omega \hat{u} \, \mathrm{d}x = 0$. We choose

$$\hat{u}(t, x) = \sin(t)\cos 2\pi x, \tag{30}$$

and therefore

$$\hat{w}(t, x) = \sin(t)(4\varepsilon^2\pi^2 - 1)\cos(2\pi x) + \sin^3(t)\cos^3(2\pi x), \tag{31}$$

which together satisfy the mean and boundary conditions. We note also that as $\hat{u}$ is differentiable everywhere in the domain, meaning that the energy functional $E(u)$ is always defined, including at $t = 0$.

Substituting this into the left hand side of the OKDE in (5), yields a right hand side f of the form

$$\begin{aligned} f(t, x) = \cos(2\pi x)(\cos(t) + 4\pi^2(4\varepsilon^2\pi^2 - 1 + \sigma)\sin(t) \\ + 12\pi^2 \sin^3(t)(\cos^2(2\pi x) - 2\sin^2(2\pi x))). \end{aligned} \tag{32}$$

Our amended variational form to be solved with our FEniCS implementation at each time step is therefore to find $\hat{u}_{n+1}, \hat{w}_{n+1} \in H^1(\Omega)$ such that

$$\langle \hat{u}_{n+1}, q\rangle - \langle \hat{u}_n, q\rangle + \Delta t \left(\langle \nabla \hat{w}_{n+1}, \nabla q\rangle + \sigma\langle \hat{u}_{n+1}, q\rangle\right) = \Delta t \langle f(t_{n+1}, x), q\rangle \qquad \forall q \in H^1(\Omega), \tag{33}$$

$$\langle \hat{w}_{n+1}, v\rangle - \varepsilon^2 \langle \nabla \hat{u}_{n+1}, \nabla v\rangle - \langle \hat{u}_{n+1}(\hat{u}_{n+1}^2 - 1), v\rangle = 0 \qquad \forall v \in H^1(\Omega). \tag{34}$$

We discretise our domain $[0, 1]$ into 10, 20, 40, 80, 160, and 320 intervals, such that $h = 0.1$, 0.05, 0.025, 0.0125, 0.00625, 0.003125. Given that no blowup happens when $h = 0.1$ and $\Delta t = 0.01$, we can set $\tilde{C} = 1$ such that $\Delta t = h^2$ in each case, and thus $\Delta t \approx 0.01$, 0.0025, 6.25e-4, 1.56e-4, 3.91e-5, and 9.76e-6, for each simulation respectively. We set our final time $T = 3$ for all simulations.

At each time step we must update the value of t in our function f, which is represented in our FEniCS implementation as an '`Expression`'. We do the same for the '`Expression`' representing the manufactured $\hat{u}$ in (30) (written as '`u_ex`' in our code). We implement this by simply adding the following line into the time-stepping loop:

```
f.t = t; u_ex.t = t;
```

Given the potential length of the simulations involved, we make a number of small changes to our code in an attempt to minimise the computational time required to implement MMS.

The first is to switch from using an LU factorisation solver at each step, to using a GMRES solver, which as expected proved to decrease the time taken for each matrix system solve.[11]The second is to turn off the default 'printing to console', using the '`set_log_active(False)`' command, once confident that the code is running smoothly.

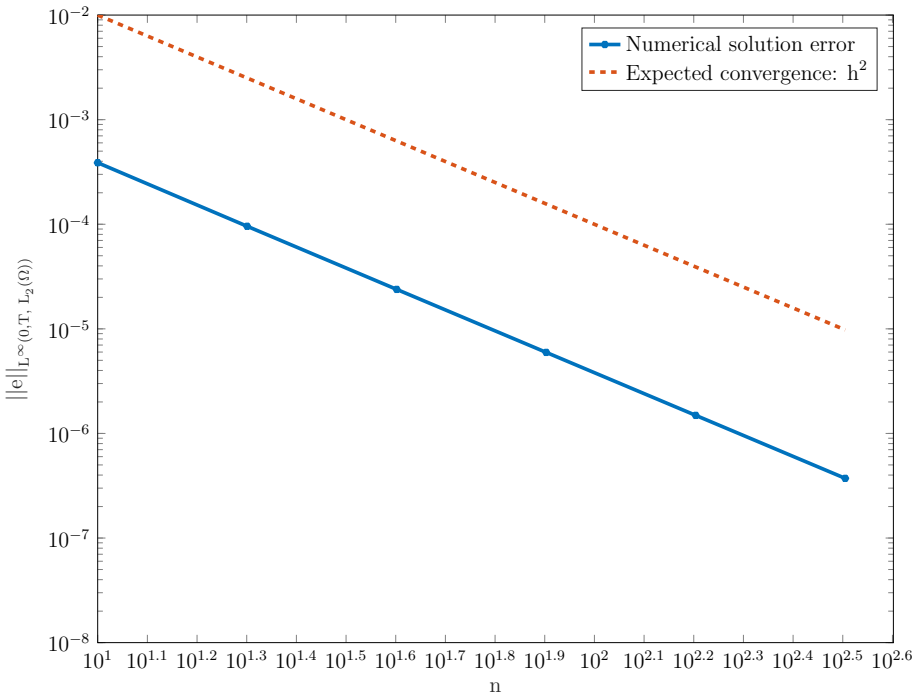


Figure 4: Error convergence in the MMS simulation described in Section 6, compared with the expected rate. The error converges exactly as expected in the specified norm.

Thirdly, we take advantage of the simple parrellelisation functionality offered by FEniCS, and split our code to run in parallel on 10 cores. To run the simulation in parallel, all it takes is to run our code with the command

`mpirun -n 10 python filename.py`.

The results, shown in Figure 4, confirm that the error of our numerical solution converges at exactly the expected rate, and thus give us confidence in the accuracy of the results presented in Section 5.

[11]The code was also run with an LU solver for a slightly smaller range in h values, and achieved the same convergence rate.

7 Alternative Splitting Schemes

In this section we investigate an alternative to the fully implicit scheme which we have been implementing so far.

The nonlinearity in our equation appears in the term $\mathcal{N}(u) = u^3 - u$. If, in our numerical scheme, we use u from the previous time step in the u^3 term, then we remove this nonlinearity and instead solve a linear system at each time step, with the variational form given by

$$\langle u_{n+1}, q\rangle - \langle u_n, q\rangle + \Delta t\left(\langle \nabla w_{n+1}, \nabla q\rangle + \sigma\langle u_{n+1}, q\rangle\right) = 0 \qquad \forall q \in H^1(\Omega), \tag{35}$$

$$\langle w_{n+1}, v\rangle - \varepsilon^2\langle \nabla u_{n+1}, \nabla v\rangle - \langle u_n^3, v\rangle + \langle u_{n+1}, v\rangle = 0 \qquad \forall v \in H^1(\Omega). \tag{36}$$

Having transformed this into a linear system, it is solved in one step (compared with what is often between 2 and 5 - or more - Newton iterations to converge when the problem is nonlinear). However, as we might expect, given that our method is now more explicit than it was previously, we are forced to take smaller time steps than we are able to take with our fully implicit scheme. Exactly how much smaller these time steps need to be in order to achieve similar accuracy will determine whether or not there is a net gain achieved by using the proposed splitting scheme.

First we compare the time step sizes permitted by the implicit-explicit and fully implicit schemes. We set $h = 0.025$, $\sigma = 10$, $\varepsilon = 0.08$, $m = 0$, and solve our equation on a unit square mesh with each scheme from a random initial condition up until $t = 500\Delta t$.

Under these conditions, both schemes permit a time step size of ε^2, the standard step size used in [3] and [12]. As we investigate the maximum permitted step size before the solver fails to converge, we find that the fully implicit scheme can tolerate a step size of up to approximately $19\varepsilon^2$, while with the implicit-explicit the Newton solver fails to converge for much lower values, when Δt grows larger than $2.1\varepsilon^2$.

Next, we investigate the error of our new proposed scheme. We use MMS again, with the same closed form u and function f on the right hand side as used in Section 6. For this simulation we set $\varepsilon = 0.08$, $m = 0$, $\sigma = 20$, $\Delta t = 0.5h^2$ and refine the mesh from $h = 0.1$ to $h = 0.00625$.

Figure 5 shows our results. Firstly, we confirm that our implicit-explicit scheme error does indeed converge at the same rate as the fully implicit scheme, with

$$||e||_{L_\infty(0,T,L^2(\Omega))} \sim C(\Delta t + h^2). \tag{37}$$

Secondly, we notice that for the same mesh discretisation and time step size,

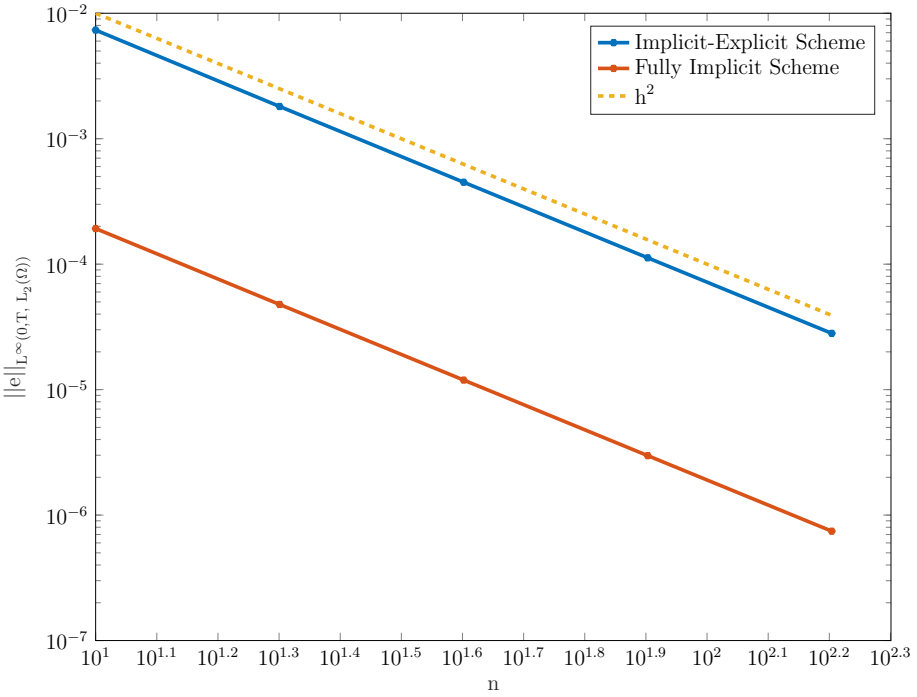


Figure 5: Error Convergence of Implicit-Explicit scheme vs. Fully Implicit scheme

though both schemes avoid blow-up and instability, the error of the fully implicit scheme is orders of magnitude lower than the implicit-explicit scheme error.

These results in Figure 5 show us that in order to achieve an error with the implicit-explicit scheme which is comparable to the fully implicit scheme, one would need to increase the number of grid points by approximately a factor of 10, and adjust the time step accordingly.

This cements a significant advantage of the fully implicit scheme over the implicit-explicit scheme, which appears to outweigh the cost of having a Newton-solver take a few more iterations to converge at each time step. In this case, the benefits of 'implicitness' seem to outweigh the cost of non-linearity.

8 Adaptive Time-Stepping

As shown in the results in Section 5, the evolution of u takes places over varying time scales. In order to capture the dynamics of the short time scale, while u is rapidly changing, and to ensure numerical stability during these periods, it is necessary to take very small time steps. However, using such a small step size uniformly for the whole simulation is unnecessarily inefficient and makes running simulations for longer total time periods vastly more computationally expensive than they need to be. In

this section we introduce a more efficient, adaptive time-stepping algorithm which takes advantage of the varying time-scales, using larger steps for longer time-scales and vice versa.

The scheme used here was previously proposed for the Cahn-Hillard equation by Li et al. [9], as an alternative to a number of other adaptive time-stepping methods previously suggested for the same equation[7][13].

We start by setting $\Delta t_{\min}$ and $\Delta t_{\max}$, and then update Δt at each time step according to

$$\Delta t_n = \min\left[\max\left(\frac{tol}{||u_n - u_{n-1}||_\infty}, \Delta t_{\min}\right), \Delta t_{\max}\right]. \tag{38}$$

The parameters $\Delta t_{\min}$, $\Delta t_{\max}$ and tol are set by trial and error, which is unfortunate, though a very common approach for parameters used in adaptive time-stepping methods [9]. Indeed, our results show how sensitive the accuracy of the method is to the choice of parameters, and that this choice is itself dependent on the other factors.

For the sake of easily visualising our results, we test our method on the 1D case, with our domain as the unit interval $[0, 1]$, setting $\sigma = 40$, $m = 0.4$, and $\varepsilon = 0.02$.

To begin we let $h = 0.02$, $\Delta t_{\min} = 0.5h^2$, and $\Delta t_{\max} = 50h^2$. We choose the same initial condition used by Li et al.[9], with an added constant to take account of our non-zero mass. We therefore define

$$u(0, x) = 0.1\cos(20\pi x) + 0.1\cos(30\pi x^2) + 0.1\cos(40\pi x^3) + 0.4. \tag{39}$$

Figure 6 shows snapshots of the solution u produced with the adaptive time stepping algorithm with varied values for tol, along with the solutions using a uniform time step of different sizes, after 1 second ($5000\Delta t_{\min}$), .

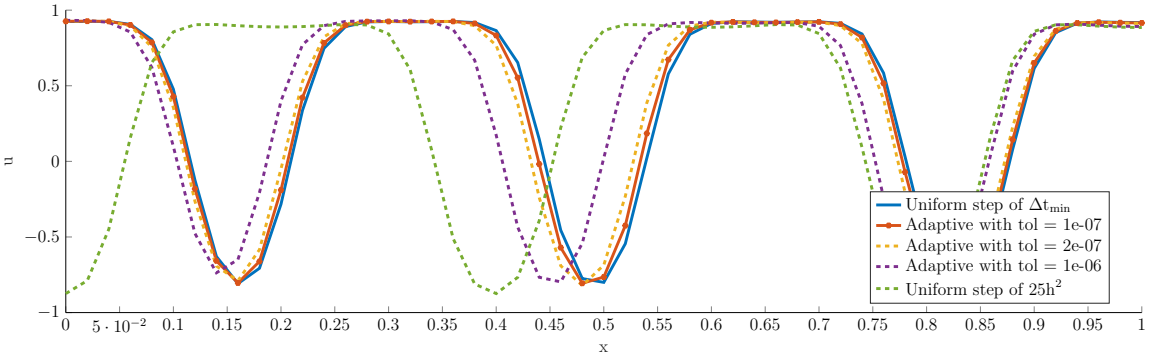
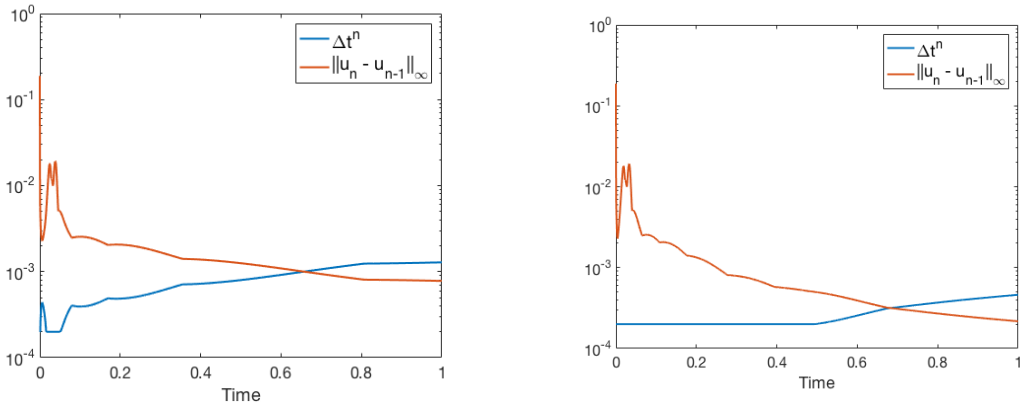


Figure 6: Solutions at time $t = 1$ produced with different time-stepping algorithms.

The most accurate of the solutions plotted in Figure 6 is the solution produced with the uniform time step $\Delta t = \Delta t_{\min}$, and thus we use this as our point of comparison

for the performance of our adaptive time-stepping method. We see that for tol = 1e-07, the adaptive time step solution has remained very close to the $\Delta t_{\min}$ solution for the first 5000 small time steps, which are those during which the most rapid evolution of u occurs. As tol is increased, the adaptive time step solution gets worse, as expected. However, even with tol = 1e-06, the adaptive time step method which allows a potential time step of up to $\Delta t_{\max} = 50h^2$, far outperforms a uniform time step of $25h^2$.

Figure 7 shows the adaptive step size over the course of the simulation, along side $||u_n - u_{n-1}||_{L_\infty}$, which serves as a measure for how fast u is changing at a given point in time. Figure 7a plots this for tol = 1e-06, and Figure 7b plots this for tol = 1e-07.



(a) tol = 1e-06

(b) tol = 1e-07

Figure 7: The evolution of $||u_n - u_{n-1}||_{L_\infty}$ and Δt_n, from $t = 0$ to $t = 1$, with $\sigma = 40$, $\varepsilon = 0.02$ and $m = 0.4$.

The results nicely exhibit the basis of the algorithm, and the impact of the tol parameter. The lower the tolerance, the higher the threshold value for $||u_n - u_{n-1}||_{L_\infty}$ below which the step-size will begin to adapt to changes in $||u_n - u_{n-1}||_{L_\infty}$.

Once the value of $||u_n - u_{n-1}||_{L_\infty}$ drops below this threshold, Figure 7a shows how the time step size responds directly and predictably to changes in $||u_n - u_{n-1}||_{L_\infty}$, bounded of course by $\Delta t_{\min}$ and $\Delta t_{\max}$

A lower tolerance therefore results in better accuracy by requiring smaller values of $||u_n - u_{n-1}||_{L_\infty}$ before the step size will increase, creating a trade-off between the efficiency gained and the accuracy achieved. Yet even with tol = 1e-07, which reproduces accurate results, the efficiency gains over long times will be noticeable.

However, the choice of tolerance is dependent not only on the equation parameters chosen, but also on the resolution of the mesh. Figure 8 compares the solution at $t = 1$

produced using the same choice of tolerance for the adaptive time stepping algorithm of 1e-07, comparing the cases when $n = 50$ ($h = 0.02$) and $n = 100$ ($h = 0.01$). From simply 'eye-balling' the figure, we can see that the accuracy of the solution has decreased significantly on the finer mesh. Indeed, the norm of the difference between the Δt_{min} solution and the adaptive time step solution has increased from approximately 0.02 to approximately 0.4.

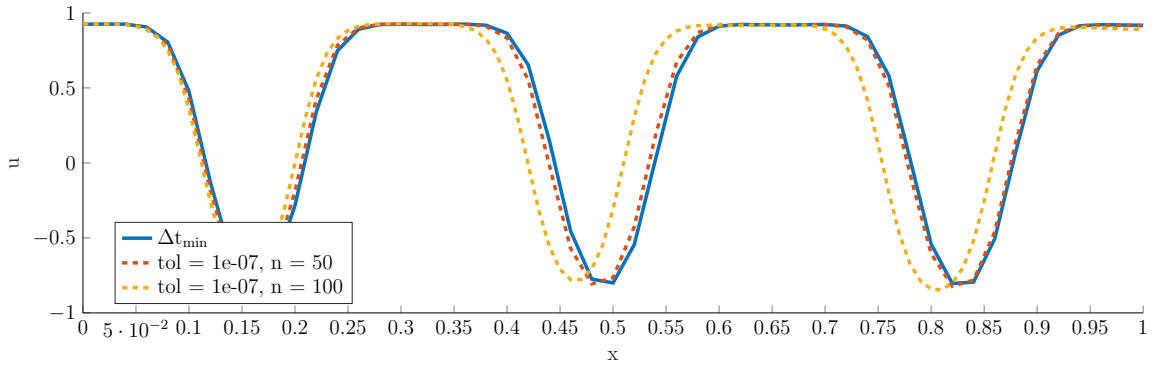


Figure 8: Comparing the performance of the proposed adaptive time-stepping algorithm with the same parameters but differing mesh resolutions.

Given the sensitivity of the method to the choice of parameters, it is unclear whether the cost in terms of time and effort to find the best tolerance tol, Δt_{min} and Δt_{max} for a given simulation, is offset by the gain in efficiency from a well functioning adaptive time step method. For smaller, once off experiments, this is unlikely to be the case, while for much longer simulations, where a decrease by a factor or 10 to 20 in computational time might mean the difference between a simulation being feasible or infeasible, the adaptive time step algorithm may be preferable. This will need to be evaluated on a case by case basis.

9 Conclusion

In this report we have presented and solved the Ohta-Kawasaki equations with FEniCS. We have verified our numerical simulations with the method of manufactured solutions, and explored two potential means of decreasing the computational time necessary to solve these equations. We first investigated an alternative, implicit-explicit splitting which removed the non-linearity from the time independent BVP solved at each time step, concluding that the inaccuracy introduced by making the equation more explicit in this way was more costly than it was beneficial. We then proposed

an adaptive time-stepping algorithm to take advantage of the different time scales on which the dynamics of the solution evolve. The benefit of the proposed algorithm appears to be case-specific, and there is much room for further research into better adaptive time-stepping algorithms.

References

[1] Susanne Brenner and Ridgway Scott. *The mathematical theory of finite element methods*, volume 15. Springer Science & Business Media, 2007.

[2] Patrick E. Farrell. Finite element methods for PDEs. *Finite Element Methods Course Notes*, 2017.

[3] Patrick E Farrell and John W Pearson. A preconditioner for the ohta–kawasaki equation. *SIAM Journal on Matrix Analysis and Applications*, 38(1):217–225, 2017.

[4] FEniCS Project. Cahn-hilliard equation. https://fenics.readthedocs.io/projects/dolfin/en/latest/demos/cahn-hilliard/python/demo_cahn-hilliard.py.html. As seen on 2017/06/30.

[5] FEniCS Project. Poisson equation with pure Neumann boundary conditions. https://fenicsproject.org/olddocs/dolfin/1.4.0/python/demo/documented/neumann-poisson/python/documentation.html. As seen on 2017/06/30.

[6] FEniCS Project. Programmers reference for DOLFIN (Python). https://fenicsproject.org/olddocs/dolfin/1.6.0/python/programmers-reference/cpp/la/NonlinearProblem.html#dolfin.cpp.la.NonlinearProblem. As seen on 2017/06/10.

[7] Héctor Gómez, Victor M Calo, Yuri Bazilevs, and Thomas JR Hughes. Isogeometric analysis of the Cahn–Hilliard phase-field model. *Computer methods in applied mechanics and engineering*, 197(49):4333–4352, 2008.

[8] Hans Petter Langtangen and Anders Logg. Solving PDEs in minutes–the FEniCS tutorial volume i, 2017.

[9] Yibao Li, Yongho Choi, and Junseok Kim. Computationally efficient adaptive time step method for the Cahn–Hilliard equation. *Computers & Mathematics with Applications*, 73(8):1855–1864, 2017.

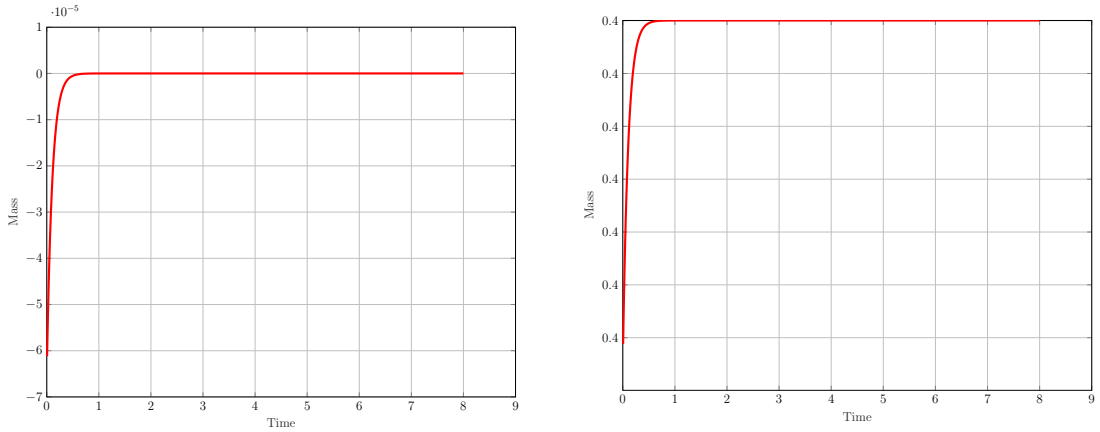
[10] Anders Logg, Garth N. Wells, and Johan Hake. *DOLFIN: a C++/Python Finite Element Library*, chapter 10. Springer, 2012.

[11] Takao Ohta and Kyozi Kawasaki. Equilibrium morphology of block copolymer melts. *Macromolecules*, 19(10):2621–2632, 1986.

[12] Quentin Parsons. Numerical approximation of the Ohta-Kawasaki Functional. *master's thesis*, 2012.

[13] Zhengru Zhang and Zhonghua Qiao. An adaptive time-stepping strategy for the Cahn-Hilliard equation. *Communications in Computational Physics*, 11(4):1261–1278, 2012.

Appendix A: Conservation of Mass



(a) Simulation where $m = 0$

(b) Simulation where $m = 0.4$

Figure 9: The value of m over time corresponding to the simulations plotted in Figure 2. As the initial condition does not have precisely the correct mean, the very slight deviation from the correct mass value near $t = 0$ is to be expected. These results show that mass is indeed conserved in our simulations.