Exercise 4

We have reached a point where we have explored a variety of mathematical problems and features within `FiPy`. At this stage, it is possible to understand and implement a model that is still an area of active research from both an theoretical and application stand point. Whilst implementing and running the model is relatively straightforward, making progress on the problems which I will highlight is much more complicated.

Learning objectives

1. Understand how the Cahn-Hilliard equation is derived
2. Implement the model in `FiPy`
3. Understand the numerical issues associated with this equation system
4. (Not too important) Output files as `*.vtk` for visualization in Paraview

Problem statement and derivation

The Cahn-Hilliard equation can model the demixing and precipitation of components from a mixture. If you recall Thermodynamics II, we can study the Gibbs energy of mixing ΔG_{mix} which gives us insight on whether the system will exhibit liquid-liquid equilibria (LLE). We first start by considering the conventional diffusive flux expression that you should be familiar with for a species A:

$$\mathbf{j} = -D\nabla a$$
$$\mathbf{j} = -D\frac{da}{dx} \text{ (in one dimension)}$$

where $\mathbf{j}$ is the flux, D is the diffusion coefficient and a is the volume fraction of species A. We can then apply the continuity / mass conservation equation:

$$\frac{\partial a}{\partial t} + \nabla \cdot \mathbf{j} = 0$$

Which gives us the transient diffusion equation was have explored extensively before:

$$\frac{\partial a}{\partial t} = D\nabla^2 a$$

When we model demixing, the mass transport is against the concentration gradient (we are moving in the direction of increasing concentration). This has been called "uphill" diffusion. We can see that the concentration gradient ∇a is no longer suitable as the driving force. We can actually write a more generalizable definition for diffusion which can model uphill diffusion. The driving force is gradients in the chemical potential μ rather than a.

The flux expression for species i , $\mathbf{j}_i$ is written as follows:

$$\mathbf{j}_i = -\sum_j L_{ij}\nabla\mu_j$$

Where L_{ij} is the mobility coefficient and μ_i is the chemical potential of species i. For species A in a binary mixture:

$$\mathbf{j}_A = -(L_{AB}\nabla\mu_A + L_{AB}\nabla\mu_B)$$

The following constraints apply:

$$L_{ij} = L_{ji}$$
$$\sum_i L_{ij} = 0$$
$$\sum_i \mathbf{j}_i = 0$$

We can therefore write:

$$L_{AA} + L_{AB} = 0$$
$$L_{AA} = -L_{AB}$$

When substituting this result back into $\mathbf{j}_A$:

$$\mathbf{j}_A = L_{AB}(\nabla\mu_A - \nabla\mu_B)$$

Since the gradient operator is linear, we can further compactify the above expression:

$$\mathbf{j}_i = L_{AB}\nabla\mu_{AB}$$

where $\mu_{AB} = \mu_A - \mu_B$. Working in differences for chemical potentials i.e. $\mu_{ij} = \mu_i - \mu_j$ is convenient for subsequent analysis. The mobility coefficient needs to be a function of composition, but the diffusion coefficient can be a constant:

$$L_{AB} = D_{AB}a(1-a)$$

where D_{AB} is the diffusion coefficient.

We can then apply the same continuity equation as we did for the vanilla diffusion equation which gives us the following equation:

$$\frac{\partial a}{\partial t} = \nabla \cdot (D_{AB}a(1-a)\nabla\mu_{AB})$$

We now need to figure out an expression for the chemical potential To figure that out, we need to consider the total Gibbs energy of the system. We consider the Landau-Ginzburg free energy functional:

$$G_{\text{system}} = \int_V g(\phi_1, \phi_2, \dots \phi_N) + \sum_i^{N-1} \frac{\kappa_i}{2}(\nabla\phi_i)^2 + \sum_{j>i}^{N-1}\sum_i \kappa_{ij}(\nabla\phi_i)(\nabla\phi_j)\; dV$$

Where ϕ_i is the volume fraction of species i. For two components:

$$G_{\text{system}} = \int_V g(a) + \frac{\kappa}{2}(\nabla a)^2\; dV$$

where G_{system} is the total Gibbs energy of the system, $g(x_1)$ is the homogenous free energy of mixing and κ is the gradient energy parameter. When a mixture demixes, this is a spontaneous process which means that the total Gibbs energy of the system decreases. There are two forces that are considered:

1. The system can reduce G_{system} by firstly demixing which results in species that have unfavorable interactions concentrating in different phases

2. However, demixing results in the formation of an interface between the two phases. The interface itself has an energy associated with it and it is unfavourable. Hence the κ term penalizes demixing.

By definition, we know that the chemical potential μ_i for a conventional system can be written as follows:

$$\mu_i = \left(\frac{\partial G}{\partial \phi_i}\right)_{T,P}$$

However, this definition is inadequate for inhomogeneous systems. We can apply the variational derivative to obtain a generalized expression for the chemical potential:

$$\mu_i = \frac{\delta G_{system}}{\delta \phi_i} = \frac{\partial G}{\partial \phi_i} - \nabla \cdot \frac{\partial G}{\partial \nabla \phi_i}$$

Therefore, μ_i can be written as follows:

$$\mu_A = \frac{\partial g}{\partial a} - \nabla \cdot (\kappa \nabla a)$$

We assume that κ is not dependent on composition, so:

$$\mu_A = \frac{\partial g}{\partial a} - \kappa \nabla^2 a$$

and:

$$\mu_B = 0$$

$$\therefore \mu_{AB} = \frac{\partial g}{\partial a} - \kappa \nabla^2 a$$

We can observe that the system forms a 4th order PDE. However, solving a 4th order PDE imposes severe time-stepping requirements (referencing exercise 2). Hence, the Cahn-Hilliard equation is typically treated as a set of coupled 2nd order PDEs:

$$\frac{\partial a}{\partial t} = \nabla \cdot (D_{AB} a(1-a) \nabla \mu_{AB})$$

$$\mu_{AB} = \frac{\partial g}{\partial a} - \kappa \nabla^2 a$$

We now need an expression for $g(a)$. For polymer blends and polymer-solvent systems, we can use the Flory-Huggins equation:

$$g(a) = \frac{a}{N_A} \ln a + \frac{(1-a)}{N_B} \ln{(1-a)} + \chi_{AB} a(1-a)$$

where N_A, N_B is the number of segments of the polymer chain of species A and B respectively and χ_{AB} is the interaction parameter between species A and B. From literature, the following expression can be used for κ:

$$\kappa = \frac{2}{3} R_G^2 \chi_{AB}$$

Where R_G is the radius of gyration of the polymer which we assume to be identical for both species. To wrap things up, we scale our equations by introducing the following scaling relationships:

$$\tilde{t} = \frac{D_{AB}t}{R_G^2}$$
$$\mathbf{x} = \tilde{\mathbf{x}}R_G$$

We thus obtain the final equation system of interest for us:

$$\tilde{\mu}_{AB} = \frac{\partial g}{\partial a} - \tilde{\kappa}\tilde{\nabla}^2 a$$
$$\frac{\partial a}{\partial \tilde{t}} = \tilde{\nabla} \cdot (a(1-a)\tilde{\nabla}\tilde{\mu}_{AB})$$

Ideal limit

For sanity's sake, this part outlines how the modified Cahn-Hilliard equation reduces to the vanilla diffusion equation when the system is ideal. When the system is ideal, $\kappa = 0$ since there are no enthalpic / residual interactions / contributions. In an ideal system where the individual particles have the same volume, the mole fraction and volume fraction become identical. For clarity, we can replace ϕ_i/a with x_i so that it is more recognisable to you.

For an ideal system, we know that:

$$g(x_A) = x_A \ln x_A + (1-x_A)\ln(1-x_A)$$

Therefore:

$$\mu_{AB} = \ln x_A - \ln(1-x_A) = \ln\frac{x_A}{1-x_A}$$
$$\nabla\mu_{AB} = \frac{\partial \mu_{AB}}{\partial x_A}\nabla x_A = \frac{1}{x_A(1-x_A)}\nabla x_A$$

Considering the first equation:

$$\frac{\partial x_A}{\partial t} = \nabla \cdot (D_{AB}x_A(1-x_A)\nabla\mu_{AB}) = \nabla \cdot \left(D_{AB}x_A(1-x_A)\frac{1}{x_A(1-x_A)}\nabla x_A\right) = D_{AB}\nabla^2 x_A$$

We recover our original equation for simple diffusion.

Numerical Implementation

The problem is defined on a 2D grid with periodic boundary conditions. For the initial conditions, we need to perturb the system with some noise (i.e. random spatial variation in the concentration field). We replace $1, 2$ with $_{a}, _{b}$ and x_1 with a for clarity.

The simulation parameters are defined as follows:

```
# Simulation parameters
# Mesh resolution and domain size
nx = ny = 50
dx = dy = 1.0
# Initial composition
a_0 = 0.77
# Noise magnitude
noise_mag = 0.03
# Flory-Huggins interaction parameter
chi_AB = 0.006
# Length of polymer chains
n_a = 1000
n_b = 1000
```

Defining the mesh:

```
mesh = PeriodicGrid2D(nx=nx, ny=ny, dx=dx, dy=dy)
```

Defining the variables. We need to perform sweeps at each timestep due to the non-linear mobility coefficient:

```
# Defining the variables
a = CellVariable (name=r"$a$", mesh=mesh, hasOld=1)
mu_AB = CellVariable(name=r"$\mu_{AB}", mesh=mesh, hasOld=1)
```

Adding the initial noise:

```
# Setting the initial composition of the system with noise
noise = UniformNoiseVariable(mesh=mesh, minimum=(a_0-noise_mag), maximum=
(a_0+noise_mag))
a[:] = noise
```

Setting up the equations. We need to manually differentiate $g(x)$. The `FiPy` documentation introduces a trick which is also presented in the code, but is commented out currently, that you can explore.

```
# differentiate g(a)
dgda = ((1.0/n_a) - (1.0/n_b)) + (1.0/n_a)*numerix.log(a) -
(1.0/n_b)*numerix.log(1.0 - a) + chi_AB*(1.0 - 2*a)
d2gda2 = (1.0/(n_a*a)) + (1.0/(n_b*(1.0 - a))) - 2*chi_AB

# Evaluate kappa
kappa = (2.0/3.0)*chi_AB

# Defining the equations
eq1 = (TransientTerm(var=a)) == DiffusionTerm(coeff=a*(1.0-a), var=mu_AB)
eq2 = (ImplicitSourceTerm(coeff=1.0, var=mu_AB)) == dgda -
DiffusionTerm(coeff=kappa, var=a)
# eq2 = (ImplicitSourceTerm(coeff=1.0, var=mu_AB)) ==
ImplicitSourceTerm(coeff=d2gda2, var=a) - d2gda2*a + dgda -
DiffusionTerm(coeff=kappa, var=a)

# Coupling the equations
eq = eq1 & eq2
```

We now introduce the idea of different solvers and solver settings. We can define this in `FiPy` quite simply. But do not worry about this.

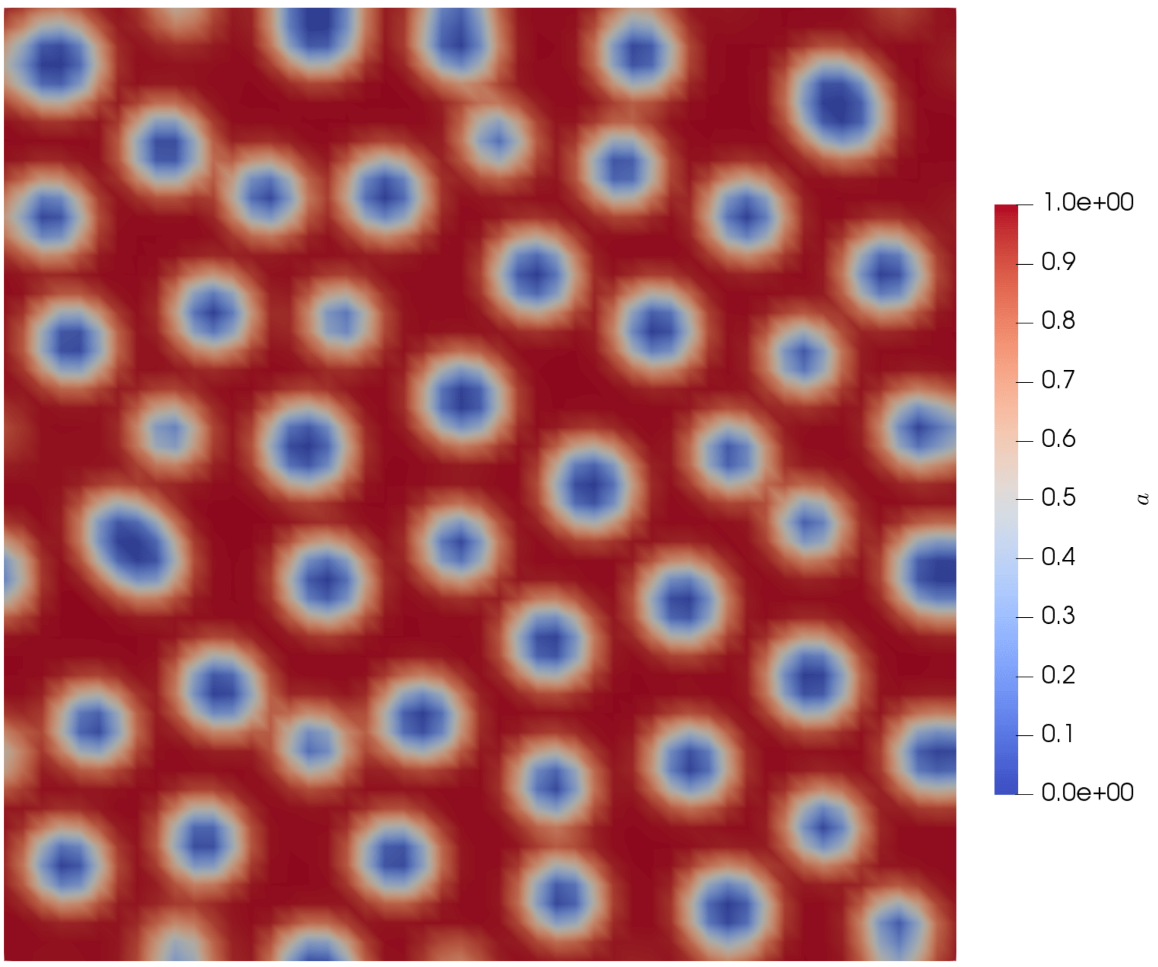
```
# Setting up the solver
solver = LinearLUSolver(tolerance=1e-9, iterations=50, precon="ilu")
```

We now introduce the code needed to output `*.VTK` files that can be processed in software such as Paraview. That will be covered in a separate document.

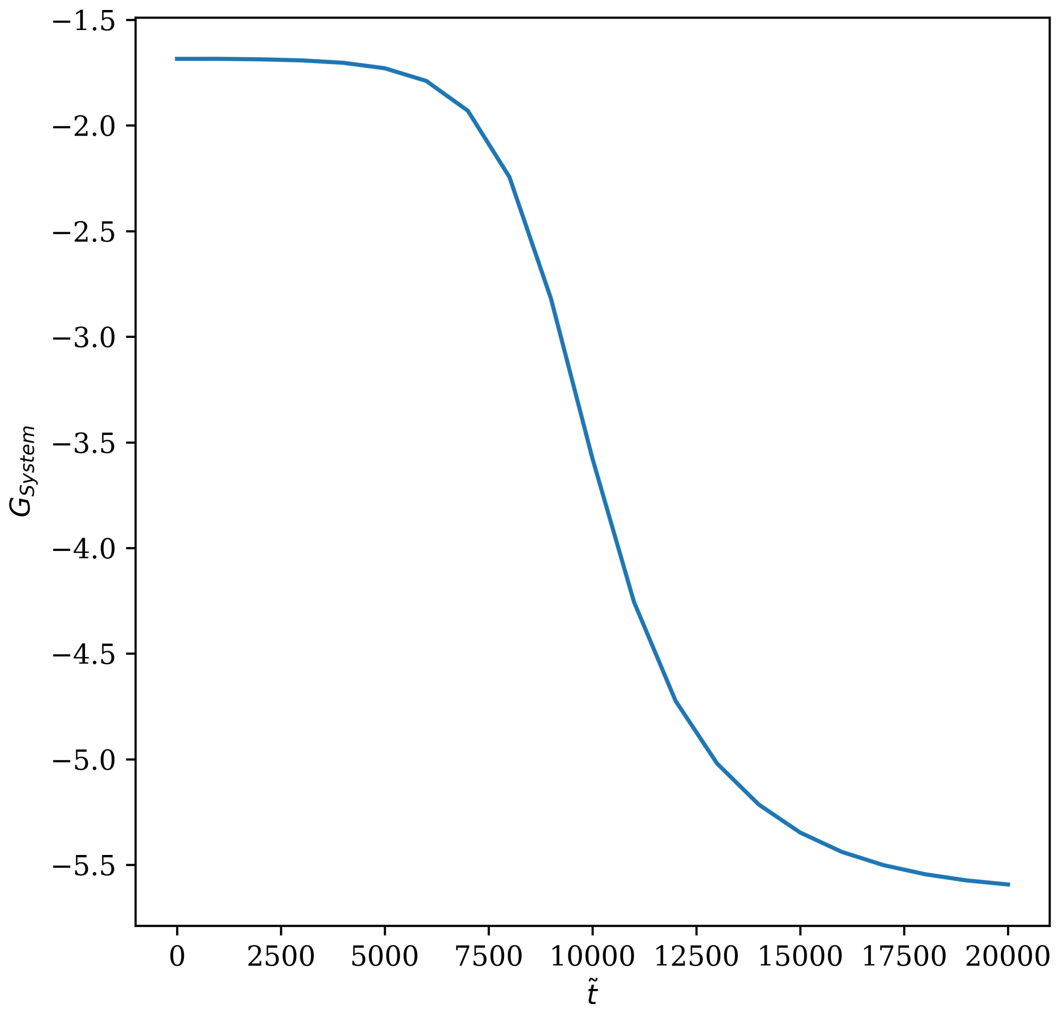
```
# Code for VTK ouput
# while elapsed < duration:
#     if (timestep == 0):
#         vw = VTKCellViewer(vars=(a, mu_AB))
#         vw.plot(filename="0_output.vtk")
#     elapsed += dt
#     timestep += 1
#     a.updateOld()
#     mu_AB.updateOld()
#     res = 1e+10
#     while res > 1e-10:
#         res = eq.sweep(dt=dt, solver=solver)
#         print ("sweep!")
#     print (elapsed)
#     end = time.time()
#     print(end-start)
#     if (timestep % time_stride ==0):
#         vw = VTKCellViewer(vars=(a, mu_AB))
#         vw.plot(filename="%s_output.vtk" %(elapsed))
```

Results and area for study

If you follow the post-processing steps in ParaView, you will be able to obtain the following figures / plots:



And G_{System} looks like this:



If you want to further explore the system, you could consider changing the values of the parameters such as N_A, N_B or χ_{AB}. You will notice that for higher values of χ_{AB}, the solution diverges. This divergence is quite a difficult issue that has yet to be resolved.