Final Project: The Cahn-Hilliard Equation

Jonathan Haydak

December 4, 2017

Contents

Part I
Introduction

1 Purpose

Phase dynamics have fascinated chemists and chemical engineers for centuries. In engineering and science, although correlations and models to exist for describing phase separation and behavior, these are mostly obtained via experiment or semi-empirical models. This is no surprise, as precisely defining the dynamics of these systems is often a daunting task. Consider the situation of oil and water being mixed together.

Because these substances are immiscible, they will eventually separate out into distinct phases. Although this is an everyday phenomena that every one has witnessed countless times, describing what is going on mathematically is quite challenging.

The purpose of this project is to explore the Cahn-Hilliard (C-H) equation. Both the one-dimensional and two-dimensional cases will be considered. The accuracy, run time, and stability of various explicit, semi-implicit, and implicit methods will be considered. In addition to this, there will be a run time comparison of Matlab and a Python script importing Fortran modules to investigate how the language influences affects run time. Stability analysis will be attempted on a linearized form of the C-H equation. Because the (C-H) is nonlinear and contains a fourth order spatial derivative, it is very stiff and extremely short time steps will be used. Calculations will be run on the Chemical and Biomolecular Engineering department's cluster at Georgia Tech.

2 Background

Typically mass transport occurs across a concentration gradient according to Fick's laws. The first law describes steady-state mass transfer

$$J = -D\nabla c \tag{2.1}$$

where D is the diffusion coefficient, c is concentration, and J is the flux. Time dependent mass transfer obeys Fick's second law

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{2.2}$$

However, in the case of phase separation mass transfer occurs *against* the concentration gradient and therefore does *not* follow Fick's laws of diffusion. Rather, species undergoing phase separation instead obey the Cahn-Hilliard equation

$$\frac{\partial c}{\partial t} = D\nabla^2(c^3 - c - \alpha^2\nabla^2 c) \tag{2.3}$$

Here α is related to the length of transition region between separate components. The term $\mu = c^3 - c - \alpha^2\nabla^2 c$ is the chemical potential and therefore (2.3) can also be written as

$$\frac{\partial c}{\partial t} = D\nabla^2 \mu \tag{2.4}$$

The derivation of this equation comes from using the gradient of chemical potential as the mechanism behind mass transfer. Unlike Fick's laws, this equation is nonlinear and contains fourth order spatial derivative, giving rise to very interesting phenomena. To begin with, we will be considering an isolate system with no outside flux of mass or chemical potential contributing to the system. In this case, the most natural boundary conditions become

$$\left.\frac{\partial u}{\partial \mathbf{n}}\right|_\Omega = \left.\frac{\partial \mu}{\partial \mathbf{n}}\right|_\Omega = 0 \tag{2.5}$$

where Ω is the boundary of the region being considered and $\mathbf{n}$ is the unit normal to the surface of the boundary. For the one-dimensional case, the region $x \in [0, 1]$ will be considered. For two dimensions, we will extend the region to $(x, y) \in [0, 1] \times [0, 1]$.

3 1-Dimensional Case

In one dimension, the C-H equation becomes

$$\begin{aligned}\frac{\partial c}{\partial t} &= D\frac{\partial^2 \mu}{\partial x^2} \\ &= D\frac{\partial^2}{\partial x^2}\left(c^3 - c - \alpha^2\frac{\partial^2 c}{\partial x^2}\right)\end{aligned} \tag{3.1}$$

with the boundary conditions

$$\left.\frac{\partial c}{\partial x}\right|_{x=0} = \left.\frac{\partial c}{\partial x}\right|_{x=1} = \left.\frac{\partial \mu}{\partial x}\right|_{x=0} = \left.\frac{\partial \mu}{\partial x}\right|_{x=1} \tag{3.2}$$

3.1 Explicit

The first method we will consider is an explicit method using a forward difference in time and centered differences in the spatial derivatives. Let $\{(x_j, t_n)\}$ denote the set of lattice points on the region where $x_j = j\Delta x$ for $j = 1, 2, \ldots, N$ and $\Delta x = \frac{1}{N-1}$. Similarly, let $c_j^n = c(x = x_j, t = t_n)$ where $t_n = n\Delta t$ for $n = 1, 2, \ldots, M$ and $\Delta t = \frac{T}{M-1}$.The discrete version of (3.1) becomes

$$\begin{aligned}\frac{c_j^{n+1} - c_j^n}{\Delta t} &= D\frac{\mu_{j+1}^n - 2\mu_j^n + \mu_{j-1}^n}{\Delta x^2} \\ &= D\left[\frac{(c_{j+1}^n)^3 - 2(c_j^n)^3 + (c_{j-1}^n)^3}{\Delta x^2} - \frac{c_{j+1}^n - 2c_j^n + c_{j-1}^n}{\Delta x^2} - \alpha^2\frac{c_{j+2}^n - 4c_{j+1}^n + 6c_j^n - 4c_{j-1}^n + c_{j-2}^n}{\Delta x^4}\right]\end{aligned}$$

$$c_j^{n+1} = \Delta t D\left[\frac{(c_{j+1}^n)^3 - 2(c_j^n)^3 + (c_{j-1}^n)^3}{\Delta x^2} - \frac{c_{j+1}^n - 2c_j^n + c_{j-1}^n}{\Delta x^2} - \alpha^2\frac{c_{j+2}^n - 4c_{j+1}^n + 6c_j^n - 4c_{j-1}^n + c_{j-2}^n}{\Delta x^4}\right] + c_j^n \tag{3.3}$$

From the boundary conditions,

$$\frac{c_1^n - c_0^n}{\Delta x} = 0 \implies c_1^n = c_0^n \tag{3.4}$$

$$\frac{c_{N+1}^n - c_N^n}{\Delta x} = 0 \implies c_{N+1}^n = c_N^n \tag{3.5}$$

Here, we are treating the points at $j = 0$ as a ghost point. The boundary conditions relating to chemical potential involve a little more work

$$\begin{gathered}\frac{\mu_1^n - \mu_n^0}{\Delta x} = 0 \\ \frac{(c_1^n)^3 - (c_0^n)^3}{\Delta x} - \frac{c_1^n - c_0^n}{\Delta x} - \frac{(c_2^n - 2c_1^n + c_0^n) - (c_1^n - 2c_0 + c_{-1}^n)}{\Delta x^4} = 0\end{gathered}$$

using the fact that $c_0^n = c_1^n$

$$\begin{gathered}c_2^n - 3c_1^n + 3c_0^n - c_{-1}^n \\ c_2^n = c_{-1}^n\end{gathered} \tag{3.6}$$

By the same argument, the final condition becomes

$$\frac{\mu_N^n - \mu_{N-1}^n}{\Delta x} = 0 \implies c_{N+2}^n = c_{N-1}^n \tag{3.7}$$

3.2 Semi-implicit

The nonlinearity and fourth order spatial derivative make a full implicit scheme of the C-H equation horribly expensive in terms of computing time. This is because a full implicit method would involve solving systems of nonlinear equations with very short Δt. To get around this, we consider a semi-implicit method that can be implemented without having to solve nonlinear equations. The main trick here is to replace $(c_j^n)^3$ with $c_j^{n+1}(c_j^n)^2$. For ease of notation, let the operator D_{xx} be defined as

$$D_{xx}c_j^n = c_{j+1}^n - 2c_j^n + c_j^n$$

Now, slightly modifying (3.3) to contain implicit terms

$$c_j^{n+1} = D\Delta t \left[\frac{D_{xx} c_j^{n+1} (c_j^n)^2}{\Delta x^2} - \frac{D_{xx} c_j^{n+1}}{\Delta x^2} - \alpha^2 \frac{D_{xx}^2 c_j^n}{\Delta x^4} \right] + c_j^n$$

Letting $r = \frac{D\Delta t}{\Delta x^2}$,

$$r D_{xx} c_j^{n+1} - r D_{xx} c_j^{n+1} (c_j^n)^2 + c_j^{n+1} = -\alpha^2 D \Delta t \frac{D_{xx}^2 c_j^n}{\Delta x^4} + c_j^n$$

$$r(c_{j+1}^{n+1} - 2c_j^{n+1} + c_{j-1}^{n+1}) - r\left(c_{j+1}^{n+1}(c_{j+1}^n)^2 - 2c_j^{n+1}(c_j^n)^2 + c_{j-1}^{n+1}(c_{j-1}^n)^2\right) + c_j^{n+1} = -\alpha^2 D \Delta t \frac{D_{xx}^2 c_j^n}{\Delta x^4} + c_j^n$$

$$c_{j+1}^{n+1}\left(r - r(c_{j+1}^n)^2\right) + c_j^{n+1}\left(-2r + 2r(c_j^n)^2 + 1\right) + c_{j-1}^{n+1}\left(r - r(c_{j-1}^n)^2\right) = -\alpha^2 D \Delta t \frac{D_{xx}^2 c_j^n}{\Delta x^4} + c_j^n$$

Finally, expanding the operator on the RHS,

$$c_{j+1}^{n+1}\left(r - r(c_{j+1}^n)^2\right) + c_j^{n+1}\left(-2r + 2r(c_j^n)^2 + 1\right) + c_{j-1}^{n+1}\left(r - r(c_{j-1}^n)^2\right) = -\alpha^2 D \Delta t \frac{c_{j+2}^n - 4c_{j+1}^n + 6c_j^n - 4c_{j-1}^n + c_{j-2}^n}{\Delta x^4} + c_j^n \tag{3.8}$$

Using this relation for $j = 1, 2, \ldots, N$ yields a system of equations:

$$\begin{bmatrix} -r + r(c_1^n)^2 + 1 & r - r(c_2^n)^2 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & r - r(c_1^n)^2 & -2r + 2r(c_2^n)^2 + 1 & r - r(c_3^n)^2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & r - r(c_2^n)^2 & -2r + 2r(c_3^n)^2 + 1 & r - r(c_4^n)^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & \\ 0 & 0 & \cdots & 0 & r - r(c_{N-1}^n)^2 & -r + r(c_N^n)^2 + 1 & & \end{bmatrix} \begin{bmatrix} c_1^{n+1} \\ c_2^{n+1} \\ c_3^{n+1} \\ c_4^{n+1} \\ \vdots \\ c_{N-1}^{n+1} \\ c_N^{n+1} \end{bmatrix} =$$

$$-\frac{\alpha^2 D \Delta t}{\Delta x^4} \begin{bmatrix} c_3^n - 4c_2^n + 6c_1^n - 4c_1^n + c_2^n \\ c_4^n - 4c_4^n + 6c_2^n - 4c_1^n + c_1^n \\ c_5^n - 4c_4^n + 6c_3^n - 4c_2^n + c_1^n \\ \vdots \\ c_{N-1}^n - 4c_N^n + 6c_N^n - 4c_{N-1}^n + c_{N-2}^n \end{bmatrix} + \begin{bmatrix} c_1^n \\ c_2^n \\ c_3^n \\ \vdots \\ c_N^n \end{bmatrix}$$

4 Stability

In this section, we seek to find conditions for the L^2 stability of several schemes. As noted previously, the non-linearity of the C-H equation makes normal Von-Neumann analysis difficult. Instead, we will examine the stability of a slightly modified equation:

$$\frac{\partial c}{\partial t} = D \frac{\partial^2}{\partial x^2} \left(A^2 c - c - \alpha^2 \frac{\partial^2 c}{\partial x^2} \right) \tag{4.1}$$

where A is some constant.

4.1 Explicit Method

Starting from the modified form of (3.3) and taking the discrete Fourier Transform

$$\begin{aligned} \hat{C}^{n+1} &= \left[rA^2 e^{i\xi} - 2rA^2 + rA^2 e^{-i\xi} - re^{i\xi} + 2r - re^{-i\xi} - Re^{2i\xi} + 4Re^{i\xi} - 6R + 4Re^{-i\xi} - Re^{-2i\xi} \right] \hat{C}^n \\ &= \left[2rA^2(\cos(\xi) - 1) + 2r(1 - \cos(\xi)) - 2R\cos(2\xi) + 8R\cos(\xi) - 6R \right] \hat{C}^n \end{aligned}$$

where $r = D\Delta t/\Delta x^2$ and $R = \alpha^2 D\Delta t/\Delta x^4$ Now we consider the Fourier coefficient:

$$\begin{aligned}
\rho &= 2r\left(\cos(\xi) - 1\right)\left(A^2 - 1\right) + 2R\sin^2(\xi) - 2R\cos^2(\xi) + 8R\cos(\xi) - 6R \\
&= 2r\left(1 - \cos(\xi)\right)\left(1 - A^2\right) + 2R - 2R\cos^2(\xi) - 2R\cos^2(\xi) + 8R\cos(\xi) - 6R \\
&= 4r\sin^2(.5\xi)(1 - A^2) - 4R\cos^2(\xi) + 8R\cos(\xi) - 4R \\
&= 4r\sin^2(.5\xi)(1 - A^2) - 4R\left(\cos^2(\xi) - 2\cos(\xi) + 1\right)
\end{aligned}$$

To find conditions for the Fourier coefficient to be less than one in magnitude, we consider that the possible extrema in practice occur will occur at $\xi = \pm\pi$, where this is inferenced by looking at linear combinations of the two above terms.

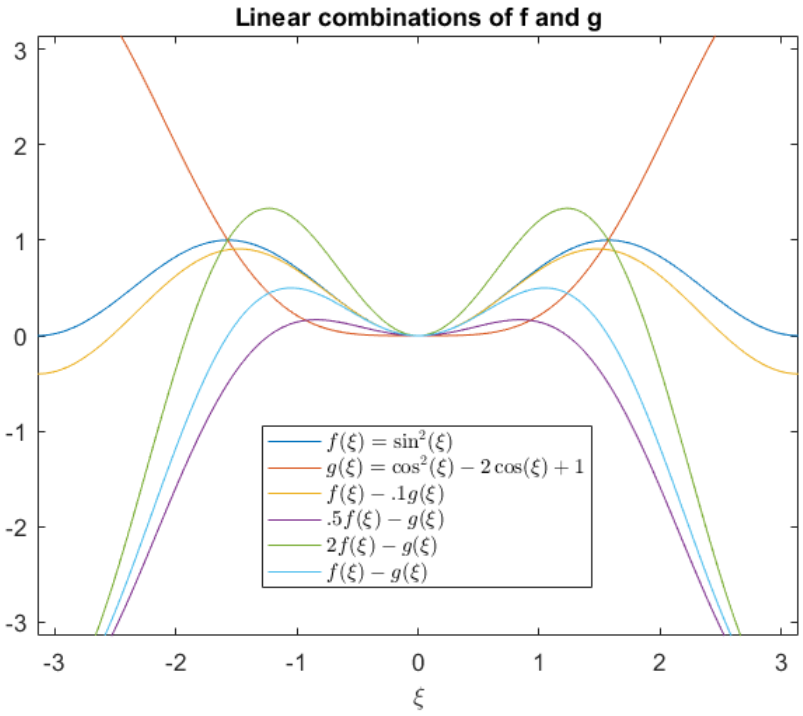


Figure 1: Linear combinations of the terms in the Fourier coefficient.

We can ignore other possible extreme because for the α being considered in this project, r and R on the same order of magnitude, and the only relevant extrema are those are $\pm\pi$. Hence,

$$\rho = 4r(1 - A^2) - 16R$$

where in practice this will be negative because it is the dominating term (see previous figure). this requirement yields,

$$\begin{aligned}
16R - 4r(1 - A^2) \leq 16R \leq 1 \\
16R \leq 1 \\
\Delta t \leq \alpha^2 D\Delta x^4/16
\end{aligned}$$

So we see that for the fully explicit method, we have a time step that must scale with Δx^4 for stability. Moreover, physical diffusion coefficients are often very small values. For this reason, we will assume $D \geq 1$ in most simulations. Otherwise, the stability requirements would be insurmountable.

5 1-D Simulations

Because mass should be conserved, this simple principle gives us a measure by which to assess the quality of each numerical scheme. Ideally, the mass should remain unchanged regardless of the time over which the simulation is run. However, the change in mass with time can provide insight into the reliability of these schemes.

	$\int_0^1 c(x)\,\mathrm{d}x$							
	$\Delta t = 1 \times 10^{-11}$, $\Delta x = .5 \times 10^{-3}$, $D = 100$, $\gamma = .2$							
scheme	t=0	t=1e-10	t=1e-9	t=1e-8	t=1e-7	t=1e-6	t=1e-5	t=1e-4
explicit	-.3300	-.3300	-.3300	-.3300	-.3300	-.3296	-.3365	-.1317
semi-implicit	-.3300	-.3300	-.3300	-.3300	-.3300	-.3301	-.3293	-.3303
	$\Delta t = 1 \times 10^{-10}$, $\Delta x = .5 \times 10^{-3}$, $D = 100$, $\gamma = .2$							
explicit	-.3300	-.3300	NaN	NaN	NaN	NaN	NaN	NaN
semi-implicit	-.3300	-.3300	-.3300	NaN	NaN	NaN	NaN	NaN

As shown in the above table, extremely small time step sizes are required for modest spatial step sizes. The semi-implicit method offers greater stability than the fully explicit scheme as expected.

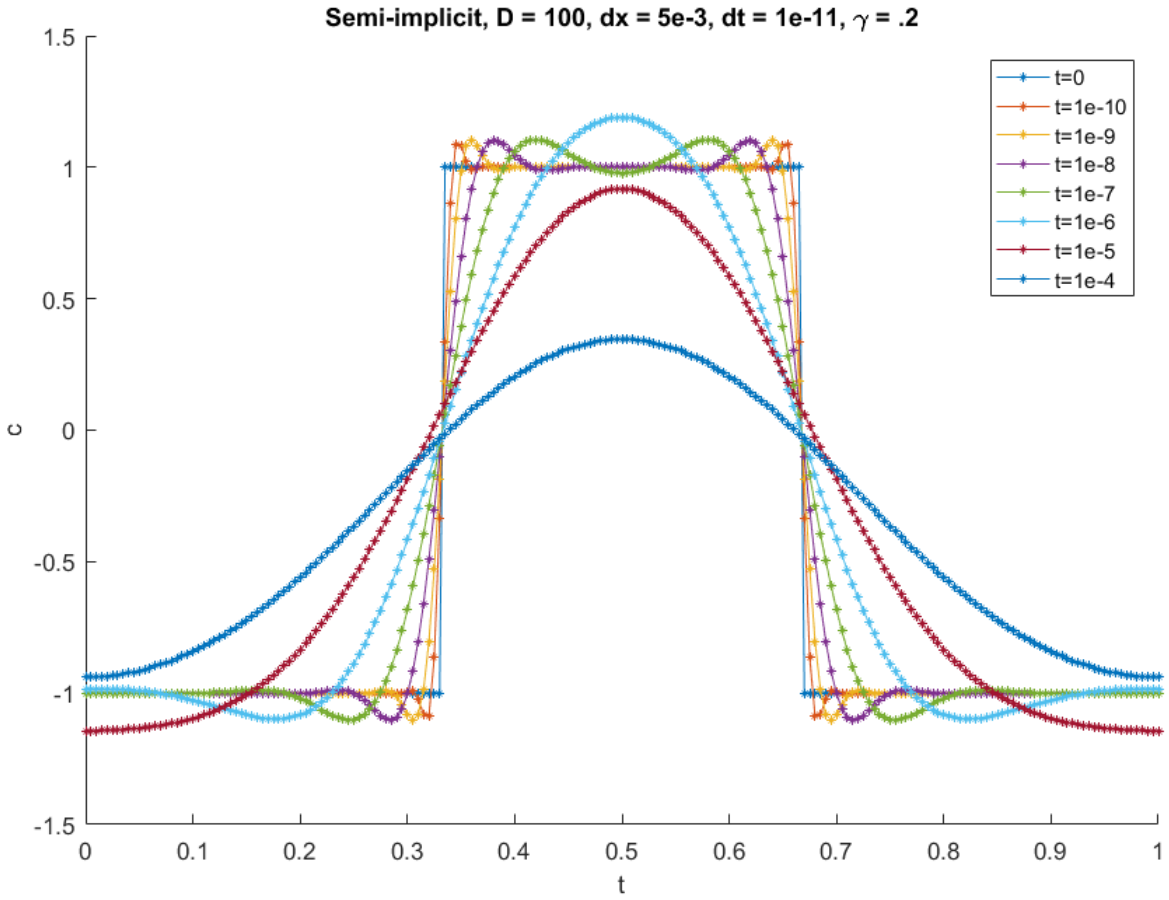


Figure 2: Semi-implicit scheme time evolution of C-H equation

6 C-H 2 Dimensional Case

The 2-D case of the C-H equation is introduces more complications than the 1-D case. Namely, the algebra is more involved now that there are two spatial variables and the computation time is longer now that the number of grid points increases by a factor of N.

6.1 Explicit Method

The derivation for the 2-D case is analogous to the 1D case. Thus, we can come up with a numerical scheme just by extending (3.3) to two dimensions.

$$\begin{aligned} c_{j,k}^{n+1} = \Delta t D \Bigg[& \frac{(c_{j+1,k}^n)^3 - 2(c_{j,k}^n)^3 + (c_{j-1,k}^n)^3}{\Delta x^2} + \frac{(c_{j,k+1}^n)^3 - 2(c_{j,k}^n)^3 + (c_{j,k-1}^n)^3}{\Delta y^2} - \frac{c_{j+1,k}^n - 2c_{j,k}^n + c_{j-1,k}^n}{\Delta x^2} \\ & - \frac{c_{j,k+1}^n - 2c_{j,k}^n + c_{j,k-1}^n}{\Delta y^2} - \alpha^2 \frac{c_{j+2,k}^n - 4c_{j+1,k}^n + 6c_{j,k}^n - 4c_{j-1,k}^n + c_{j-2,k}^n}{\Delta x^4} \\ & - \alpha^2 \frac{c_{j,k+2}^n - 4c_{j,k+1}^n + 6c_{j,k}^n - 4c_{j,k-1}^n + c_{j,-2k}^n}{\Delta y^4} \Bigg] + c_{j,k} \end{aligned} \tag{6.1}$$

The corresponding boundary conditions are

$$\begin{aligned} c_{1,k}^n &= c_{0,k}^n \\ c_{j,1}^n &= c_{j,0}^n \\ c_{N+1,k}^n &= c_{N,k}^n \\ c_{j,N+1}^n &= c_{j,N}^n \\ c_{2,k}^n &= c_{-1,k}^n \\ c_{j,2}^n &= c_{j,-1}^n \\ c_{N+2,k}^n &= c_{N-1,k}^n \\ c_{j,N+2}^n &= c_{j,N-1} \end{aligned}$$

6.2 Alternating Direction Implicit

As before, we define the operators

$$D_{xx} c_{j,k}^n = c_{j+1,k}^n - 2c_{j,k}^n + c_{j-1,k}^n, \qquad D_{yy} c_{j,k}^n = c_{j,k+1}^n - 2c_{j,k}^n + c_{j,k-1}^n$$

Now, consider a semi-implicit scheme split into the following two steps

$$c_{j,k}^{n+.5} = .5 D \Delta t \left[\frac{D_{xx} c_{j,k}^{n+.5} (c_{j,k}^n)^2}{\Delta x^2} + \frac{D_{yy} (c_{j,k}^n)^3}{\Delta y^2} - \frac{D_{xx} c_{j,k}^{n+.5}}{\Delta x^2} - \frac{D_{yy} c_{j,k}^n}{\Delta y^2} - \alpha^2 \frac{D_{xx}^2 c_{j,k}^n}{\Delta x^4} - \alpha^2 \frac{D_{yy}^2 c_{j,k}^n}{\Delta y^4} \right] + c_{j,k}^n$$

$$c_{j,k}^{n+1} = .5 D \Delta t \left[\frac{D_{xx} (c_{j,k}^{n+.5})^3}{\Delta x^2} + \frac{D_{yy} c_{j,k}^{n+1} (c_{j,k}^{n+.5})^2}{\Delta y^2} - \frac{D_{xx} c_{j,k}^{n+.5}}{\Delta x^2} - \frac{D_{yy} c_{j,k}^{n+1}}{\Delta y^2} - \alpha^2 \frac{D_{xx}^2 c_{j,k}^{n+.5}}{\Delta x^4} - \alpha^2 \frac{D_{yy}^2 c_{j,k}^{n+.5}}{\Delta y^4} \right] + c_{j,k}^{n+.5}$$

Expanding out the operators for the first step and letting $r_x = \frac{.5D\Delta t}{\Delta x^2}$, $r_{xx} = \frac{.5D\alpha^2\Delta t}{\Delta x^4}$, $r_y = \frac{.5D\Delta t}{\Delta y^2}$, $r_{yy} = \frac{.5D\alpha^2\Delta t}{\Delta y^4}$

$$c_{j,k}^{n+.5} - r_x D_{xx} c_{j,k}^{n+.5}(c_{j,k}^n)^2 + r_x D_{xx} c_{j,k}^{n+.5} = r_y D_{yy}(c_{j,k}^n)^3 - r_y D_{yy} c_{j,k}^n - r_{xx} D_{xx}^2 c_{j,k}^n - r_{yy} D_{yy}^2 c_{j,k}^n + c_{j,k}^n$$

$$c_{j,k}^{n+.5} - r_x\left(c_{j+1,k}^{n+.5}(c_{j+1,k}^n)^2 - 2c_{j,k}^{n+.5}(c_{j,k}^n)^2 + c_{j-1,k}^{n+.5}(c_{j-1,k}^n)^2\right) + r_x\left(c_{j+1,k}^{n+.5} - 2c_{j,k}^{n+.5} + c_{j-1,k}^{n+.5}\right) =$$
$$r_y\left((c_{j,k+1}^n)^3 - 2(c_{j,k}^n)^3 + (c_{j,k-1}^n)^3\right) - r_y\left(c_{j,k+1}^n - 2c_{j,k}^n + c_{j,k-1}^n\right) - r_{xx}\left(c_{j+2,k}^n - 4c_{j+1,k}^n + 6c_{j,k}^n - 4c_{j-1,k}^n + c_{j-2,k}^n\right)$$
$$- r_{yy}\left(c_{j,k+2}^n - 4c_{j,k+1}^n + 6c_{j,k}^n - 4c_{j,k-1}^n + c_{j,k-2}^n\right) + c_j, k^n$$

$$c_{j+1,k}^{n+.5}\left(r_x - r_x(c_{j+1,k}^n)^2\right) + c_{j,k}^{n+.5}\left(-2r_x + 2r_x(c_{j,k}^n)^2 + 1\right) + c_{j-1,k}^{n+.5}\left(r_x - r_x(c_{j-1,k}^n)^2\right) = r_y\left((c_{j,k+1}^n)^3 - 2(c_{j,k}^n)^3 + (c_{j,k-1}^n)^3\right)$$
$$- r_y\left(c_{j,k+1}^n - 2c_{j,k}^n + c_{j,k-1}^n\right) - r_{xx}\left(c_{j+2,k}^n - 4c_{j+1,k}^n + 6c_{j,k}^n - 4c_{j-1,k}^n + c_{j-2,k}^n\right) + c_{j,k}^n$$
$$- r_{yy}\left(c_{j,k+2}^n - 4c_{j,k+1}^n + 6c_{j,k}^n - 4c_{j,k-1}^n + c_{j,k-2}^n\right) + c_j, k^n$$

This equation can be solved as a system of equations in N^2 unknowns. Alternatively, it can also be solved as N system of equations each with N unknowns. This second way is easier to conceptualize and organize, and so will be method illustrated.

$$\begin{bmatrix} -r_x + r_x(c_{1,k}^n)^2 + 1 & r_x - r_x(c_{2,k}^n)^2 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & r_x - r_x(c_{1,k}^n)^2 & -2r_x + 2r_x(c_{2,k}^n)^2 + 1 & r_x - r_x(c_{3,k}^n)^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & r_x - r_x(c_{2,k}^n)^2 & -2r_x + 2r_x(c_{3,k}^n)^2 + 1 & r - r_x(c_{4,k}^n)^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & \\ & & & & & & & \end{bmatrix} \begin{bmatrix} c_{1,k}^{n+.5} \\ c_{2,k}^{n+.5} \\ c_{3,k}^{n+.5} \\ c_{4,k}^{n+.5} \\ \vdots \\ c_{N-1,k}^{n+.5} \\ c_{N,k}^{n+.5} \end{bmatrix}$$

$$= \begin{bmatrix} r_y D_{yy}(c_{1,k}^n)^3 - r_y D_{yy} c_{1,k}^n - r_{xx} D_{xx}^2 c_{1,k}^n - r_{yy} D_{yy}^2 c_{1,k}^n + c_{1,k}^n \\ r_y D_{yy}(c_{2,k}^n)^3 - r_y D_{yy} c_{2,k}^n - r_{xx} D_{xx}^2 c_{2,k}^n - r_{yy} D_{yy}^2 c_{2,k}^n + c_{2,k}^n \\ r_y D_{yy}(c_{3,k}^n)^3 - r_y D_{yy} c_{3,k}^n - r_{xx} D_{xx}^2 c_{3,k}^n - r_{yy} D_{yy}^2 c_{3,k}^n + c_{3,k}^n \\ \vdots \\ r_y D_{yy}(c_{N,k}^n)^3 - r_y D_{yy} c_{N,k}^n - r_{xx} D_{xx}^2 c_{N,k}^n - r_{yy} D_{yy}^2 c_{N,k}^n + c_{N,k}^n \end{bmatrix}$$

This system is solved for $k = 1, 2, \dots, N$. Similarly, the second step can be solved as

$$c_{j,k+1}^{n+1}\left(r_y - r_y(c_{j,k+1}^{n+.5})^2\right) + c_{j,k}^{n+1}\left(-2r_y + 2r_y(c_{j,k}^{n+.5})^2 + 1\right) + c_{j,k-1}^{n+1}\left(r_y - r_y(c_{j,k-1}^{n+.5})^2\right) = r_x\left((c_{j+1,k}^{n+.5})^3 - 2(c_{j,k}^{n+.5})^3 + (c_{j+1,k}^{n+.5})^3\right.$$
$$- r_x\left(c_{j+1,k}^{n+.5} - 2c_{j,k}^{n+.5} + c_{j-1,k}^{n+.5}\right) - r_{xx}\left(c_{j+2,k}^{n+.5} - 4c_{j+1,k}^{n+.5} + 6c_{j,k}^{n+.5} - 4c_{j-1,k}^{n+.5} + c_{j-2,k}^{n+.5}\right) + c_{j,k}^{n+.5}$$
$$- r_{yy}\left(c_{j,k+2}^{n+.5} - 4c_{j,k+1}^{n+.5} + 6c_{j,k}^{n+.5} - 4c_{j,k-1}^{n+.5} + c_{j,k-2}^{n+.5}\right) + c_j, k^{n+.5}$$

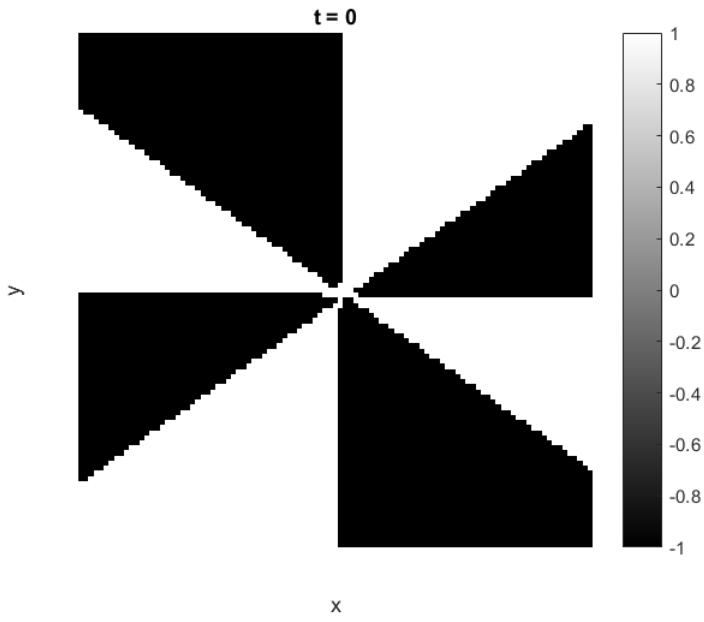
the the N equations to be solved for $j = 1, 2, \ldots, N$ are

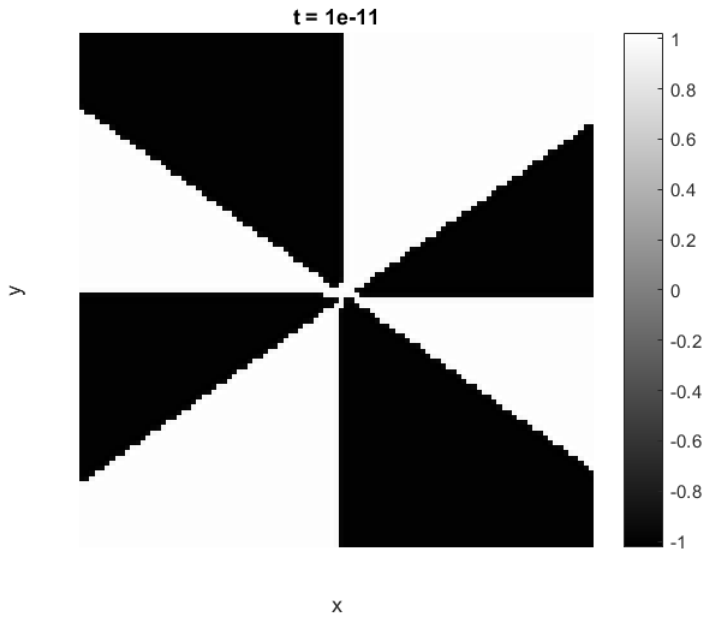
$$\begin{bmatrix} -r_y + r_y(c_{j,1}^{n+.5})^2 + 1 & r_y - r_y(c_{j,2}^{n+.5})^2 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & r_y - r_y(c_{j,1}^{n+.5})^2 & -2r_y + 2r_y(c_{j,2}^{n+.5})^2 + 1 & r_y - r_y(c_{j,3}^{n+.5})^2 & 0 & \ldots & 0 & 0 \\ 0 & 0 & r_y - r_y(c_{j,2}^{n+.5})^2 & -2r_y + 2r_y(c_{j,3}^{n+.5})^2 + 1 & r - r_y(c_{j,4}^{n+.5})^2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & \\ & & & & & & & \end{bmatrix}$$

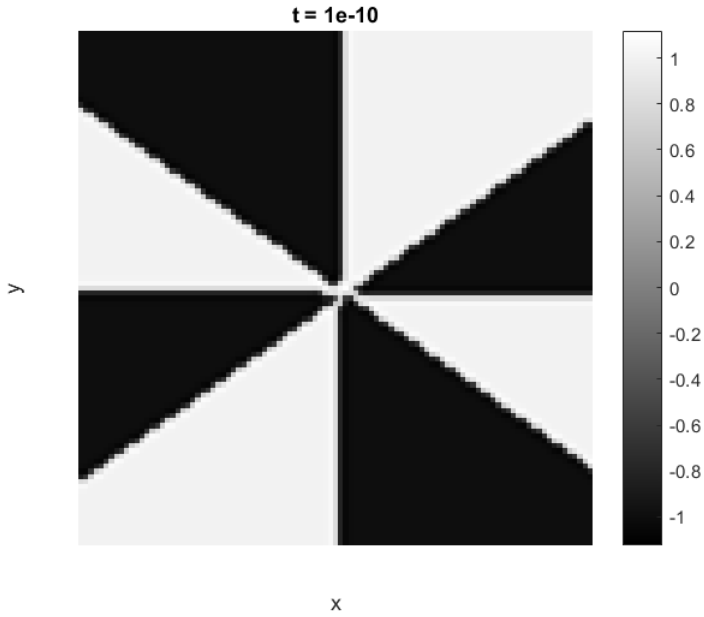
$$\begin{bmatrix} c_{j,1}^{n+1} \\ c_{j,2}^{n+1} \\ c_{j,3}^{n+1} \\ c_{j,4}^{n+1} \\ \vdots \\ c_{j,N-1}^{n+1} \\ c_{j,N}^{n+1} \end{bmatrix} = \begin{bmatrix} r_x D_{xx}(c_{j,1}^{n+.5})^3 - r_x D_{xx} c_{j,1}^{n+.5} - r_{xx} D_{xx}^2 c_{j,1}^{n+.5} - r_{yy} D_{yy}^2 c_{j,1}^{n+.5} + c_{j,1}^{n+.5} \\ r_x D_{xx}(c_{j,2}^{n+.5})^3 - r_x D_{xx} c_{j,2}^{n+.5} - r_{xx} D_{xx}^2 c_{j,2}^{n+.5} - r_{yy} D_{yy}^2 c_{j,2}^{n+.5} + c_{j,2}^{n+.5} \\ r_x D_{xx}(c_{j,3}^{n+.5})^3 - r_x D_{xx} c_{j,3}^{n+.5} - r_{xx} D_{xx}^2 c_{j,3}^{n+.5} - r_{yy} D_{yy}^2 c_{j,3}^{n+.5} + c_{j,3}^{n+.5} \\ \vdots \\ r_x D_{xx}(c_{j,N}^{n+.5})^3 - r_x D_{xx} c_{j,N}^{n+.5} - r_{xx} D_{xx}^2 c_{j,N}^{n+.5} - r_{yy} D_{yy}^2 c_{j,N}^{n+.5} + c_{j,N}^{n+.5} \end{bmatrix}$$

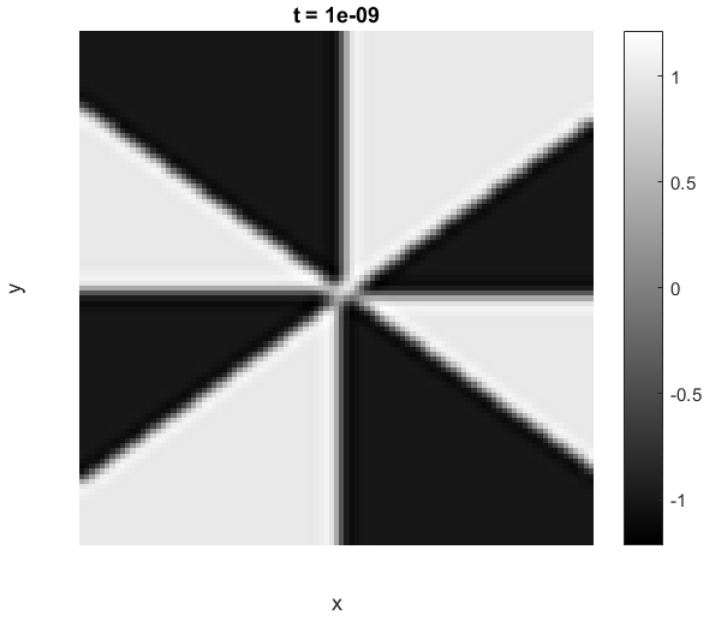
7 2-D Simulations

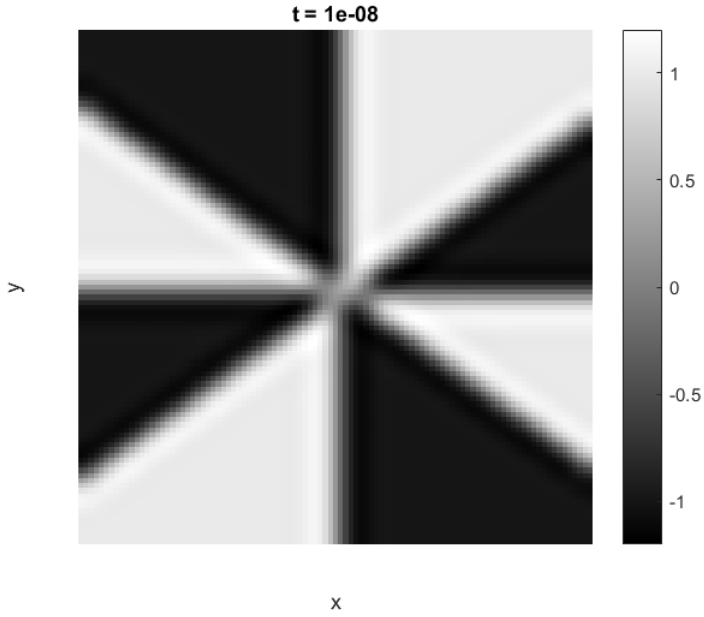
The first interesting cases to consider are initial conditions where the two components are initially separated into regions.

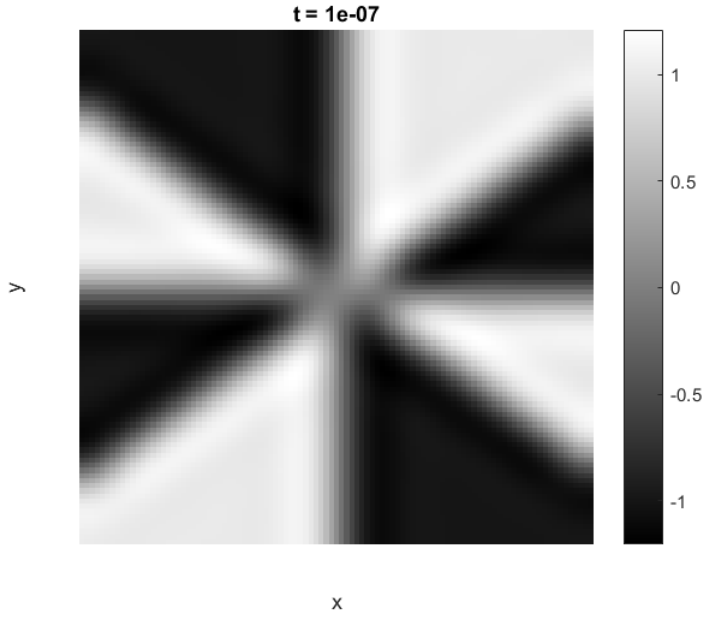

t = 0
y
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1


t = 1e-11
y
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1


t = 1e-10
y
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1


t = 1e-09
y
x
1
0.5
0
-0.5
-1


t = 1e-08
y
x
1
0.5
0
-0.5
-1


t = 1e-07
y
x
1
0.5
0
-0.5
-1

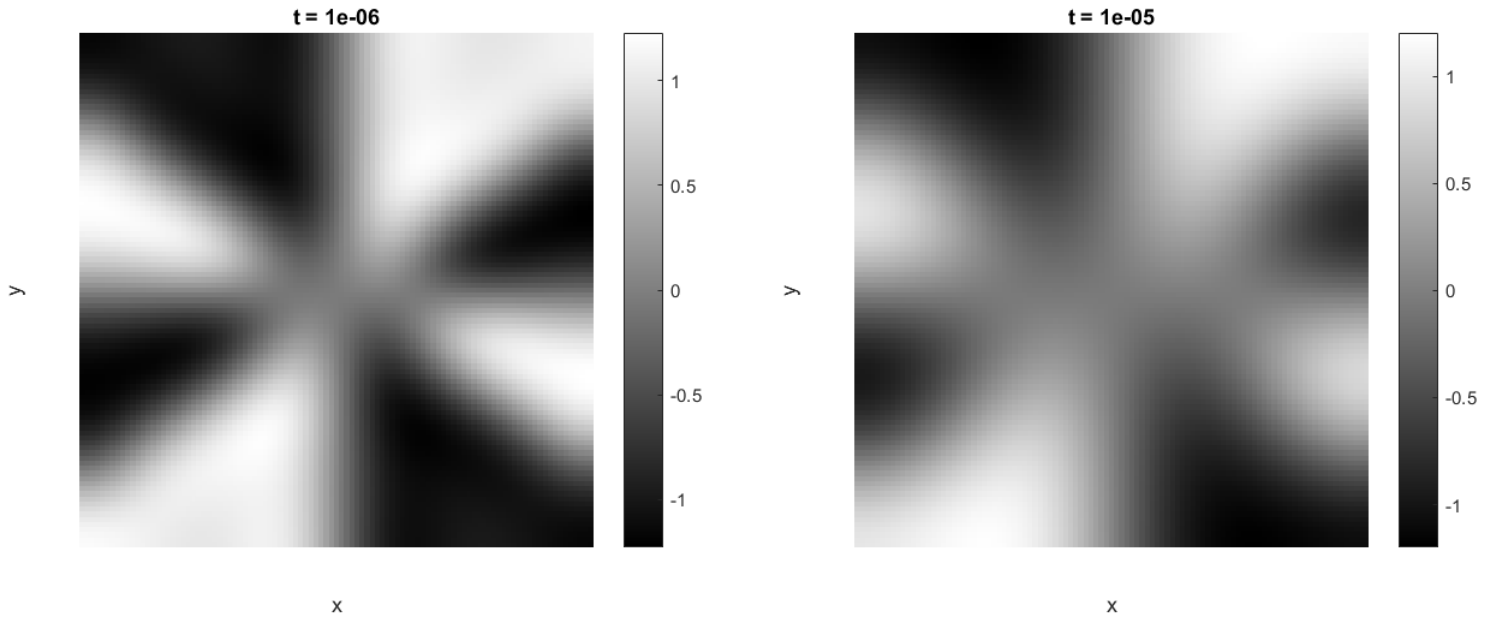
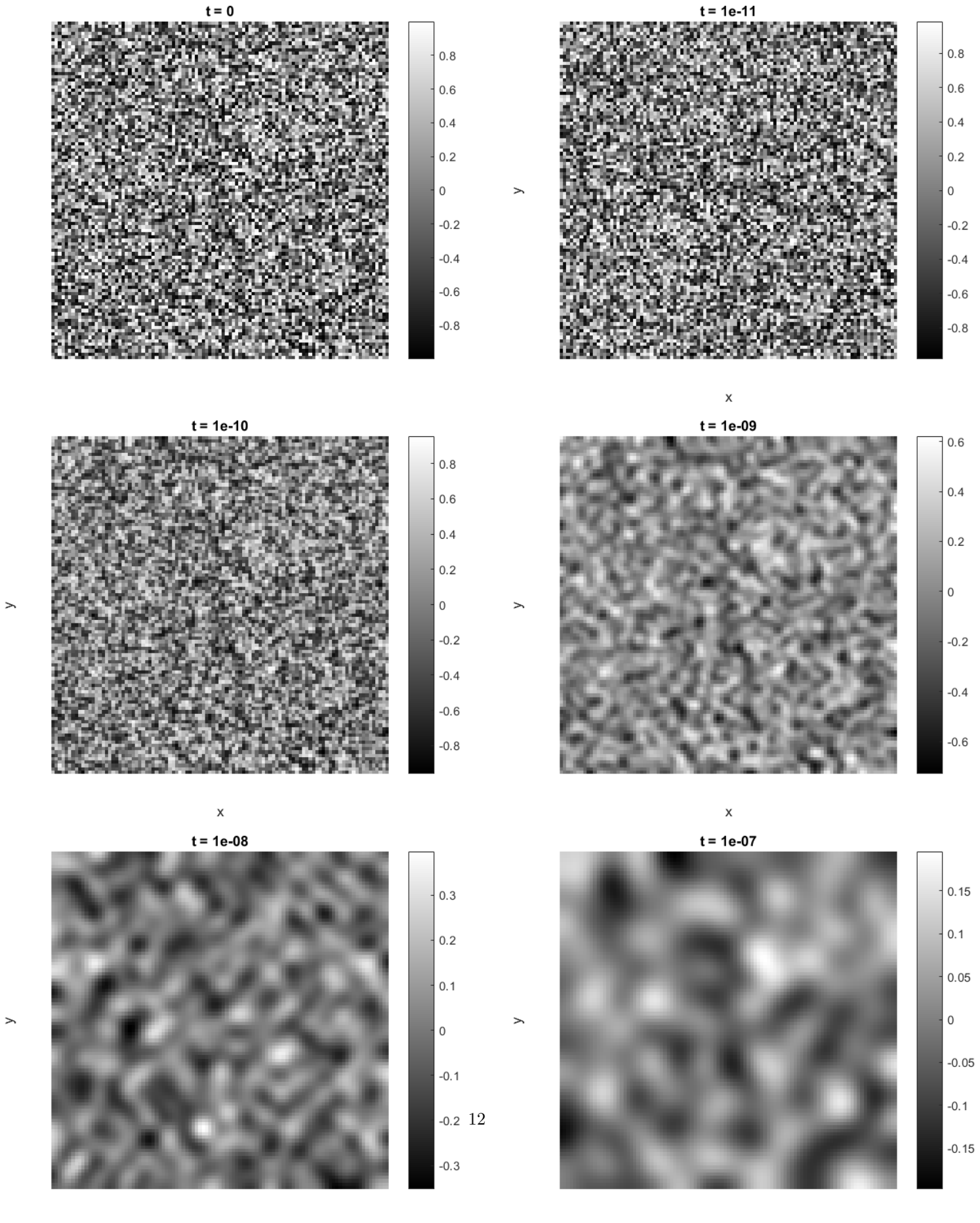


Figure 3: Explicit Scheme, $\Delta t = 1e - 11$, $\Delta x = \Delta y = .01$, $\gamma = .2$, $D = 100$, Separated Initial Conditions

The next case to consider is two components that are initially randomly distributed. In this case, as the system evolves the two components naturally separate.


t = 0
t = 1e-11
t = 1e-10
t = 1e-09
t = 1e-08
t = 1e-07
x
y
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.3
0.1
-0.1
-0.3
0.15
0.05
-0.05
-0.15

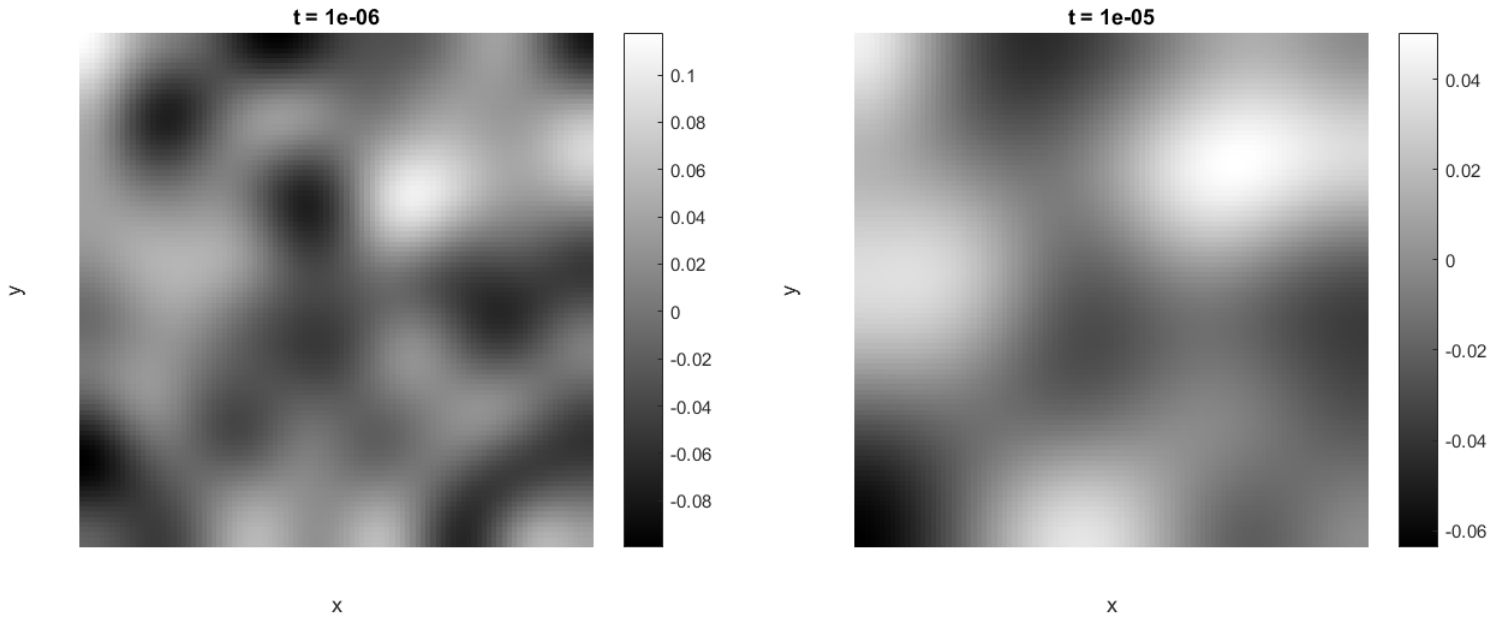
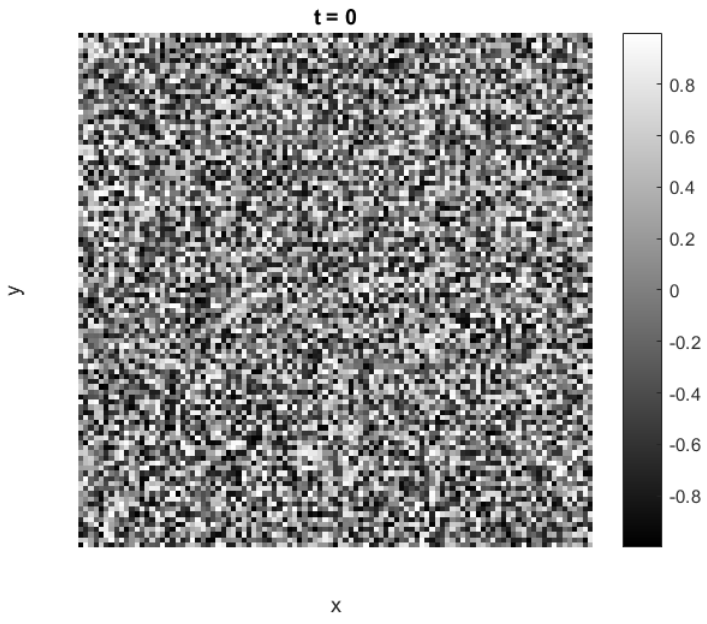
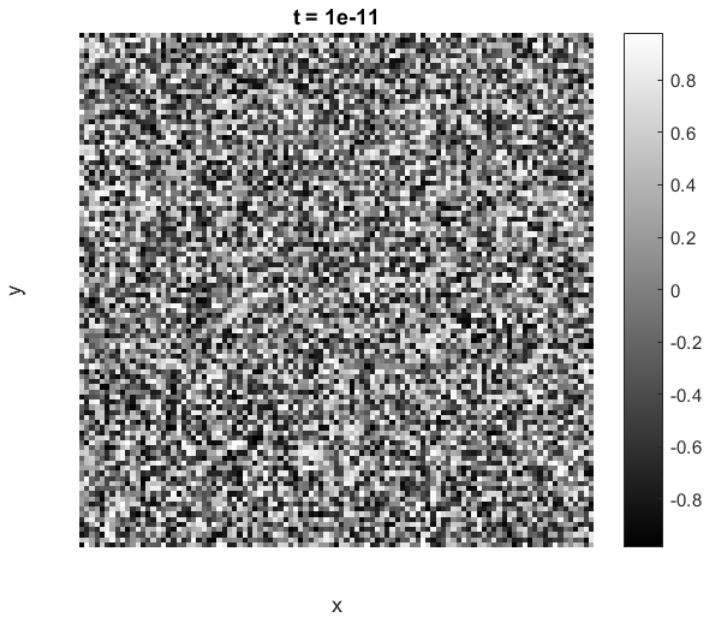
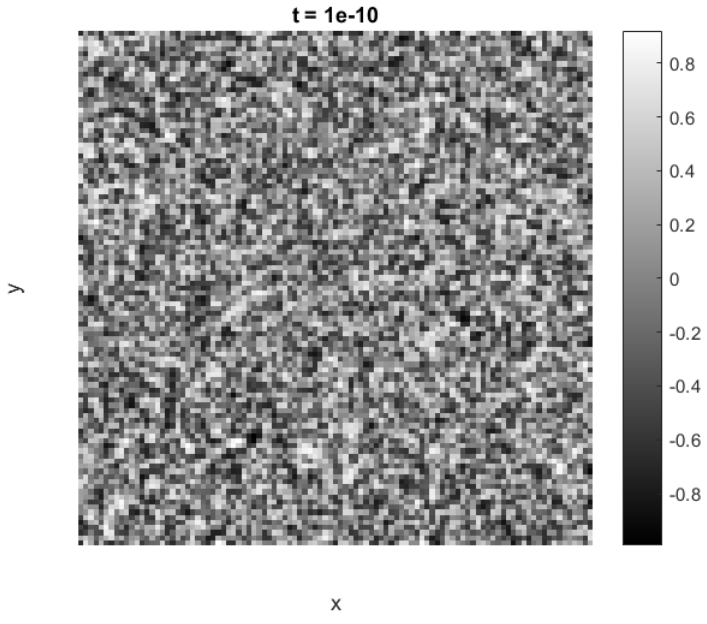
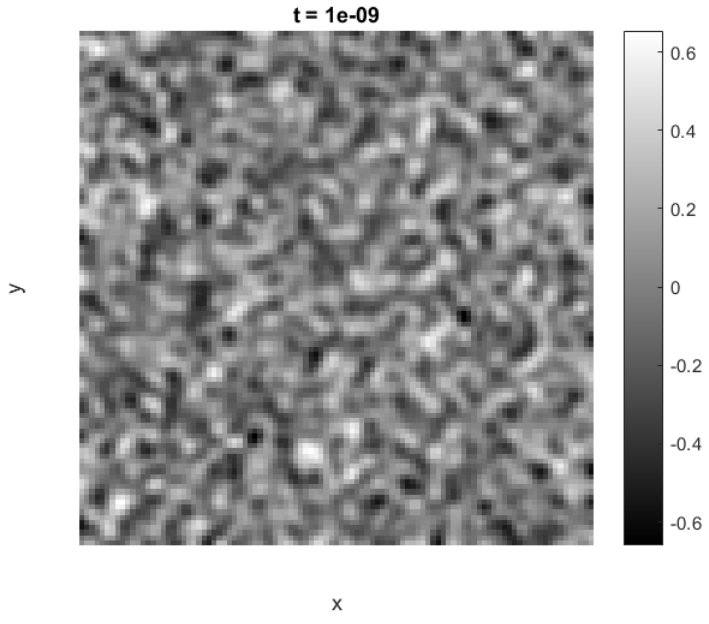


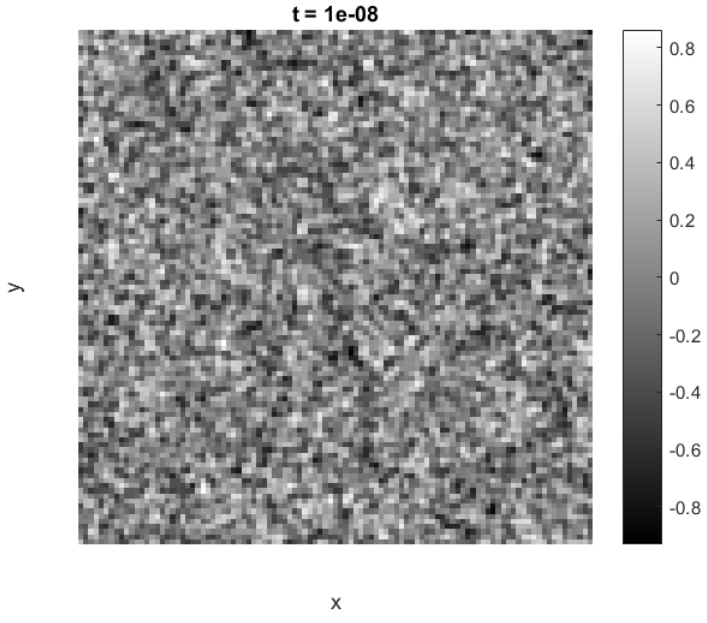
Figure 4: ADI Scheme, $\Delta t = 1e - 11$, $\Delta x = \Delta y = .01$, $\gamma = .2$, $D = 100$, Separated Initial Conditions

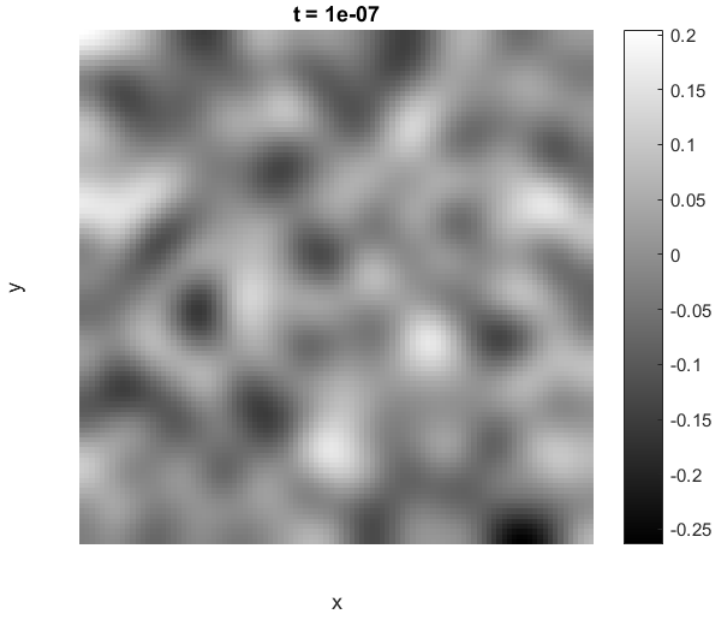

t = 0
y
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


t = 1e-11
y
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


t = 1e-10
y
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


t = 1e-09
y
x
0.6
0.4
0.2
0
-0.2
-0.4
-0.6


t = 1e-08
y
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


t = 1e-07
y
x
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25

	$\int_0^1 \int_0^1 c(x,y)\,\mathrm{d}x\,\mathrm{d}y$							
	$\Delta t = 1 \times 10^{-11}$, $\Delta x = \Delta y = .01$, $D = 100$, $\gamma = .2$							
scheme	t=0	t=1e-11	t=1e-10	t=1e-9	t=1e-8	t=1e-7	t=1e-6	
explicit Random IC	-7.4764e-3	-7.4646e-3	-7.3780e-3	-7.1783e-3	-7.1468e-3	-6.9684e-3	-6.8760e-3	
explicit Separated IC	-1.8800e-2	-1.8798e-2	-1.8783e-2	-1.8732e-2	-1.8705e-2	-1.8712e-2	-1.8698e-2	
ADI Random IC	-7.4764e-3	-7.4647e-3	-7.3788e-3	-7.1785e-3	-7.1468e-3	-6.9684e-3	-6.8760e-3	
ADI Separated IC	-1.8800e-2	-1.8798e-2	-1.8783e-2	-1.8732e-2	-1.8705e-2	-1.8712e-2	-1.8698e-2	
	$\Delta t = 1 \times 10^{-10}$, $\Delta x = \Delta y = .01$, $D = 100$, $\gamma = .2$							
scheme	t=0	t=1e-10	t=1e-9	t=1e-8	t=1e-7	t=1e-6	t=1e-5	
explicit Random IC	-7.4764e-03	-7.3582e-03	-7.1745e-03	-7.1469e-03	-6.9683e-03	-6.8760e-03	-6.7857e-03	
explicit Separated IC	-1.8800e-2	-1.8798e-2	-1.8783e-2	-1.8732e-2	-1.8705e-2	-1.8712e-2	-1.8698e-2	
ADI Random IC	-7.4764e-3	-7.3706e-03	-7.1765e-03	-7.1469e-03	-6.9683e-03	-6.8760e-03	-6.7857e-03	
ADI Separated IC	-1.8800e-2	-1.8782e-2	-1.8732e-2	-1.8705e-2	-1.8712e-2	-1.8698e-2	-1.8697e-2	
	$\Delta t = 1 \times 10^{-9}$, $\Delta x = \Delta y.01$, $D = 100$, $\gamma = .2$							
scheme	t=0	t=1e-9	t=1e-8	t=1e-7	t=1e-6	t=1e-5	t = 1e-4	
explicit Random IC	-7.4764e-03	-6.2944e-03	NaN	NaN	NaN	NaN	NaN	
explicit Separated IC	-1.8800e-2	-1.8600e-02	NaN	NaN	NaN	NaN	NaN	
ADI Random IC	-7.4764e-3	-7.5343e-03	NaN	NaN	NaN	NaN	NaN	
ADI Separated IC	-1.8800e-2	-1.8768e-02	NaN	NaN	NaN	NaN	NaN	

Table 1: Verification of mass conservation for various schemes

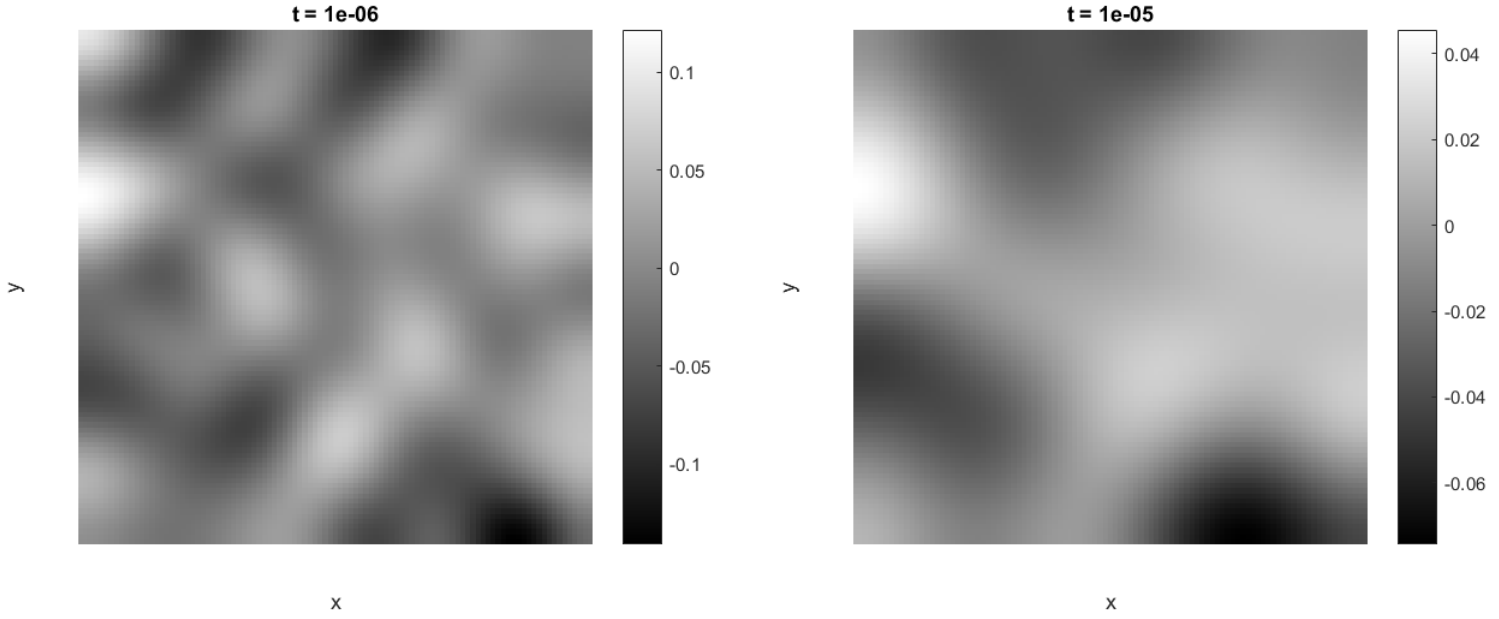


Figure 5: ADI Scheme, $\Delta t = 1e-11$, $\Delta x = \Delta y = .01$, $\gamma = .2$, $D = 100$, Random Initial Conditions

From inspecting the results of the explicit and ADI schemes, we can see that for these parameters and these time steps they both perform reasonably well. As before, another test we can perform to check if these schemes are failing is whether mass is being preserved throughout the system.

8 Run time: Matlab vs Fortran

Although Matlab is a very practical programming language for engineering and scientific applications, one downfall of the language is that it can take an unreasonable amount of time to run complex calculations and simulations. The same problem applies to other high level programming languages such as Python. One way to get around this is to write modules in a lower level language to do the most computationally expensive calculations and then import them. This section seeks to address the question: How much faster can these calculations be run if they are done in a Python script importing Fortran modules versus just Matlab?

Validation of Fortran Code

To validate that the Fortran code works, it is necessary to verify that it reproduces the results of the Matlab code used previously. This was tested for several of the simulations previously reported, and in each case the results matched identically with the only difference being in extremely small digits.

Run time (seconds)									
	$\Delta t = 1 \times 10^{-11}$, $\Delta x = \Delta y = .01$, $D = 100$, $\gamma = .2$								
Scheme	t=0	t=1e-11	t=1e-10	t=1e-9	t=1e-8	t=1e-7	t=1e-6	t=1e-5	
Fortran, Explicit, Random IC	0	1.2660e-3	2.2570e2	1.3308e-1	1.1357	1.1060e1	1.1018e2	1.1011e3	
Matlab, Explicit, Random IC	0	4.6429e-02	2.2316e-01	1.6081	1.5275e+01	1.5121e+02	1.5039e+03	1.5003e+04	
Fortran, Explicit, Separated IC	0	1.2281e-3	1.7526e-2	1.2600e-1	1.1251	1.1019e1	1.0982e2	1.0977e3	
Matlab, Explicit, Separated IC	0	5.4452e-02	1.7891e-01	1.4944	1.5036e+01	1.5143e+02	1.5326e+03	1.5504e+04	
Fortran, ADI, Separated IC	0	4.3671e-3	5.8516e-2	4.3944e-1	4.0513	3.9304e1	3.9736e2	3.9508e3	
Matlab, ADI, Separated IC	0	7.6708e-02	3.0411e-01	2.5384	2.5113e+01	2.5009e+02	2.5166e+03	2.5444e+04	
Fortran, ADI, Random IC	0	4.8969e-3	1.5951e-1	5.9967e-1	4.1643	3.9603e1	1.5715e2	1.5710e3	
Matlab, ADI, Random IC	0	8.3956e-02	3.2848e-01	2.6590	2.5833e+01	2.5658e+02	2.5640e+03	2.5690e+04	
	$\Delta t = 1 \times 10^{-10}$, $\Delta x = \Delta y = .01$, $D = 100$, $\gamma = .2$								
scheme	t=0	t=1e-10	t=1e-9	t=1e-8	t=1e-7	t=1e-6	t=1e-5	t=1e-4	
Fortran, Explicit, Random IC	0	1.2760e-03	2.2863e-02	1.3348e-01	1.1376	1.1046e+01	1.1007e+02	1.0996e+03	
Matlab, Explicit, Random IC	0	5.9090e-02	2.2038e-01	1.6775e+00	1.6073e+01	1.5912e+02	1.5751e+032	1.5768e+04	
Fortran, Explicit, Separated IC	0	1.2069e-03	1.7302e-02	1.2632e-01	1.1207	1.0963e+01	1.0926e+02	1.0922e+03	
Matlab, Explicit, Separated IC	0	4.2587e-02	1.6187e-01	1.4696	1.4567e+01	1.4721e+02	1.4883e+03	1.4961e+04	
Fortran, ADI, Separated IC	0	1.6959e-03	2.3126e-02	1.7563e-01	1.5998	1.5740e+01	1.5698e+02	1.5686e+03	
Matlab, ADI, Separated IC	0	1.2708e-01	3.5641e-01	2.6060	2.5148e+01	2.5184e+02	2.5374e+03	2.5436e+04	
Fortran, ADI, Random IC	0	1.7390e-03	2.7306e-02	1.7965e-01	1.6026	1.5730e+01	1.5760e+02	1.5684e+03	
Matlab, ADI, Random IC	0	8.9244e-02	3.5411e-01	2.6927	2.5947e+01	2.5635e+02	2.5610e+03	2.5659e+04	

Table 2: Comparison of run times using Matlab and a Fortran module called from a Python script.

9 Conclusion

Based on the results seen here, we see that for the parameters shown, the stability of explicit and implicit schemes breaks down when the step size is less than $\Delta x = \Delta y = 1 \times 10^{-10}$. This approximately agrees with the stability condition previously described. For very fine meshes, the explicit and semi-implicit/ADI schemes seem to perform reasonably well, but the implicit methods do seem to have better stability. If the step sizes were more closely investigated, it is very likely we would find a point where the semi-implicit schemes perform significantly better.

In terms of run time, we see that the Fortran code performs *significantly* than the Matlab code, on the order of an entire order of magnitude better. Whereas it takes the Fortran code anywhere from 10-30 minutes to complete 1 million integration steps, it takes the Matlab code roughly 7 hours. Moreover, the Fortran was not compiled to be optimized for the architecture being used. Based on this, it is reasonable to say that an engineer solving computationally intensive problem can very easily save orders of magnitude of time by just simply writing Fortran modules with little worry about optimizing them for performance.

10 Appendix

1-D explicit scheme

```
function CHproject1D_explicit
close all
global D
global gamma
D = 100;
gamma = .2;
dx = 5e-3;
dt = 1e-10;
save_fh = 'explicit_dt1e-10.mat';
[x1,~,c1] = my_CD(1e-10,1,dx,dt);
[x2,~,c2] = my_CD(1e-9,1,dx,dt);
[x3,~,c3] = my_CD(1e-8,1,dx,dt);
save(save_fh)
[x4,~,c4] = my_CD(1e-7,1,dx,dt);
save(save_fh)
[x5,~,c5] = my_CD(1e-6,1,dx,dt);
save(save_fh)
[x6,~,c6] = my_CD(1e-5,1,dx,dt);
save(save_fh)
[x7,~,c7] = my_CD(1e-4,1,dx,dt);
save(save_fh)
% hold all
% plot(x1,c1(:,end),'*-','MarkerSize',3)
% plot(x2,c2(:,end),'*-','MarkerSize',3)
% plot(x3,c3(:,end),'*-','MarkerSize',3)
%  plot(x4,c4(:,end),'*-','MarkerSize',3)
%   plot(x5,c5(:,end),'*-','MarkerSize',3)
% xlabel('t'); ylabel('c')
% title('Semi-implicit, D = 100, dx = 5e-3, dt = 1e-11, \gamma = .2')
end

function u = create_initial(u)
numx = length(u);
midpt = round(numx/3);
for i = 1:midpt
   %u(i) = sin(2*pi*(i/numx));
```

```
    u(i) = -1;
end
for i = midpt+1:midpt*2
  %u(i) = sin(2*pi*(i/numx));
  u(i) = 1;
end

for i = midpt*2+1:numx
    u(i) = -1;
end

end
function [xout,tout,uout] = my_CD(t_f,x_f,dx,dt)
global D
global gamma
t = 0:dt:t_f;
x = 0:dx:x_f;
numx = length(x); numt = length(t);
uold  = zeros(numx,1);
uold = create_initial(uold);
unew = uold;
%plot initial conditions
% figure
plot(x,uold,'-*','MarkerSize',3)
xlabel('x'); ylabel('c');
title('Initial Conditions')

%mu = @(j,n) u(j,n)^3 - u(j,n) - gamma^2* (u(j+1,n)-2*u(j,n)+u(j-1,n))/dx^2;
for n = 1:numt-1
    display(['Iteration ' num2str(n) ' of '  num2str(numt)])
    for j = 3:numx-2
        %RHS = D* (mu(j+1,n) - 2*mu(j,n) - mu(j-1,n))/dx^2;
        %u(j,n+1) = RHS*dt + u(j,n);
        term1 = uold(j+1)^3 - 2*uold(j)^3 + uold(j-1)^3;
        term2 = uold(j+1) - 2*uold(j) + uold(j-1);
        term3= uold(j-2) - 4* uold(j-1) + 6*uold(j) - 4*uold(j+1) + uold(j+2);
        RHS = term1/dx^2 - term2/dx^2 - gamma^2*term3/dx^4;
        unew(j) = D*RHS*dt + uold(j);
    end
    j = 2;
    term1 = uold(j+1)^3 - 2*uold(j)^3 + uold(j-1)^3;
    term2 = uold(j+1) - 2*uold(j) + uold(j-1);
    term3= uold(1) - 4* uold(j-1) + 6*uold(j) - 4*uold(j+1) + uold(j+2);
    RHS = term1/dx^2 - term2/dx^2 - gamma^2*term3/dx^4;
    unew(j) = D*RHS*dt + uold(j);
    j = 1;
    unew(j) = unew(j+1);
    j = numx-1;
    term1 = uold(j+1)^3 - 2*uold(j)^3 + uold(j-1)^3;
    term2 = uold(j+1) - 2*uold(j) + uold(j-1);
    term3= uold(j-2) - 4* uold(j-1) + 6*uold(j) - 4*uold(j+1) + uold(numx);
    RHS = term1/dx^2 - term2/dx^2 - gamma^2*term3/dx^4;
    unew(j) = D*RHS*dt + uold(j);
    j = numx;
    unew(j) = unew(j-1);

    uold = unew;

end

xout = x;
tout = t;
uout = unew;
end
```

1-D semi-implicit scheme

```
function CHproject1D_semi
close all
global D
global gamma
D = 100;
gamma = .2;
dx = 5e-3;
dt = 1e-10;
save_fh = 'semi_dt1e-10';
[x1,~,c1] = my_CD(1e-10,1,dx,dt);
[x2,~,c2] = my_CD(1e-9,1,dx,dt);
[x3,~,c3] = my_CD(1e-8,1,dx,dt);
save(save_fh)
[x4,~,c4] = my_CD(1e-7,1,dx,dt);
save(save_fh)
[x5,~,c5] = my_CD(1e-6,1,dx,dt);
save(save_fh)
[x6,~,c6] = my_CD(1e-5,1,dx,dt);
save(save_fh)
[x7,~,c7] = my_CD(1e-4,1,dx,dt);
save(save_fh)
hold all
plot(x1,c1(:,end),'*-','MarkerSize',3)
plot(x2,c2(:,end),'*-','MarkerSize',3)
plot(x3,c3(:,end),'*-','MarkerSize',3)
plot(x4,c4(:,end),'*-','MarkerSize',3)
plot(x5,c5(:,end),'*-','MarkerSize',3)
plot(x6,c6(:,end),'*-','MarkerSize',3)
plot(x7,c7(:,end),'*-','MarkerSize',3)
xlabel('t'); ylabel('c')
legend('t=0','t=1e-10','t=1e-9','t=1e-8','t=1e-7','t=1e-6','t=1e-5','t=1e-4')
title('Semi-implicit, D = 100, dx = 5e-3, dt = 1e-11, \gamma = .2')
end
function u = create_initial(u)
numx = length(u);
midpt = round(numx/3);
for i = 1:midpt
   %u(i) = sin(2*pi*(i/numx));
   u(i) = -1;
end
for i = midpt+1:midpt*2
  %u(i) = sin(2*pi*(i/numx));
  u(i) = 1;
end

for i = midpt*2+1:numx
   u(i) = -1;
end

end

function [xout,tout,u] = my_CD(t_f,x_f,dx,dt)
global D
global gamma
t = 0:dt:t_f;
x = 0:dx:x_f;
numx = length(x); numt = length(t);
uold  = zeros(numx,1);
uold = create_initial(uold);
%plot initial conditions
figure
plot(x,uold,'-*','MarkerSize',3)
```

```
xlabel('x'); ylabel('c');
title('Initial Conditions')



%apply BC's
x = [x(1)-2*dx, x(1)-dx, x, x(end)+dx, x(end)+2*dx];
uold = [uold(2); uold(1); uold; uold(end); uold(end-1);];
new_numx = length(uold);
r = D*dt/dx^2;
for n = 1:numt-1
    display(['Iteration ' num2str(n) ' of '  num2str(numt)])
     A = zeros(new_numx,new_numx);
     b = zeros(new_numx,1);
     A(1,1) = 1;
     A(1,4) = -1;
     A(2,2) = 1;
     A(2,3) = -1;
     A(end,end) = 1; A(end,end-3) = -1;
     A(end-1,end-1) = 1; A(end-1,end-2) = -1;
    for j = 3:numx+2
        A(j,j-1) = r - r*(uold(j-1)^2);
        A(j,j) = -2*r + 2*r*(uold(j)^2)+1;
        A(j,j+1) = r - r*(uold(j+1)^2) ;
        b(j) = -gamma^2*D*dt/dx^4 * (uold(j+2)-4*uold(j+1) + 6*uold(j) -4*uold(j-1) ...
            +uold(j-2));
        b(j) = b(j) + uold(j);
    end
    unew = A \ b;
    uold = unew;


end

xout = x(3:numx+2);
tout = t;
u = unew(3:numx+2);
end
```

2-D explicit scheme

```
function CHproject_2D_explicit2
close all
D = 100;
gamma = .2;
dr = .01;
dt = 1e-11;
t0 = 0;
tf1 = 1e-11;

save_str = '2d_explicit_not_rand_2';
numx = length(0:dr:1);
c0 = create_initial_circ(numx,numx);
%imshow(c0)
tic
for i = 1:7
    istr = num2str(i);
    thisstr = ['[x' istr ',y' istr ',t' istr ',c' istr' ']'];
    if i == 1
        thisstr = [thisstr '=my_CD(0,tf1,1,1,dr,dr,dt,gamma,D,c0);'];
    else
        thisstr = [thisstr '=my_CD(t' num2str(i-1), ',t' num2str(i-1)];
        thisstr = [thisstr '*10,1,1,dr,dr,dt,gamma,D,c' num2str(i-1) ');'];
    end
```

```
    eval(thisstr)
    str = ['this_time' istr '=toc'];
    eval(str)
    save(save_str);
end

high = max(c(:));
low = min(c(:));
close all
imshow(c)
caxis([low high])
end
function A = create_initial_rand(numx,numy)
A = zeros(numx,numy);
for i = 1:numx
   for j = 1:numy
      A(i,j) = -1 + 2*rand;
   end
end
end
function A = create_initial_circ(numx,numy)
mdpntx = round(numx/2); mdpnty = round(numy/2);
A = zeros(numx,numy) -1;
for i = 1:numx
   for j = 1:numy
      dx = mdpntx - i;
      dy = mdpnty - j;
      if 2*dx^2 + dy^2 + 5*dx^2*dy - 20*dy^2*dx + 20*dx^3*dy - 10*dy^3*dx < 15
         A(i,j) = 1;
      end

   end
end
end

function [xout,yout,tout,new] = my_CD(t0,t_f,x_f,y_f,dx,dy,dt,gamma,D,u0)
t = t0:dt:t_f;
x = 0:dx:x_f;
y = 0:dy:y_f;
numx = length(x); numy = length(y); numt = length(t);
old = u0;
%apply BC's
new_old = zeros(length(x)+4,length(y)+4);
new_old(3:end-2,3:end-2) = old;
new_old(1,3:end-2) = old(2,:); new_old(3:end-2,1) = old(:,2);
new_old(2,3:end-2) = old(1,:); new_old(3:end-2,2) = old(:,1);
new_old(end,3:end-2) = old(end-1,:); new_old(3:end-2,end) = old(:,end-1);
new_old(end-1,3:end-2) = old(end,:); new_old(3:end-2,end-1) = old(:,end);

old = new_old;

new = zeros(size(old));

for n = 1:numt-1

    for j = 3:numx+2
        for k = 3:numy+2
            term1 = old(k,j+1)^3 - 2 * old(k,j)^3 + old(k,j-1)^3;
            term2 = old(k+1,j)^3 - 2* old(k,j)^3 + old(k-1,j)^3;
            term3 = old(k,j+1) - 2*old(k,j) + old(k,j-1);
            term4 = old(k+1,j) - 2*old(k,j) + old(k-1,j);
            term5 = old(k,j-2) -4*old(k,j-1) + 6* old(k,j) -4*old(k,j+1) + old(k,j+2);
            term6 = old(k-2,j) -4*old(k-1,j) + 6*old(k,j) -4*old(k+1,j) + old(k+2,j);
            RHS = term1/dx^2 + term2/dy^2 - term3/dx^2 -term4/dy^2 - gamma^2*term5/dx^4 - ...
                gamma^2*term6/dy^4;
```

```
            new(k,j) = D*RHS*dt + old(k,j);
        end
    end
    %apply the BC's
    new_old = new;
    %new_old(3:end-2,3:end-2) = old;
    new_old(1,3:end-2) = new(4,3:end-2); new_old(3:end-2,1) = new(3:end-2,4);
    new_old(2,3:end-2) = new(3,3:end-2); new_old(3:end-2,2) = new(3:end-2,3);
    new_old(end,3:end-2) = new(end-3,3:end-2); new_old(3:end-2,end) = new(3:end-2,end-3);
    new_old(end-1,3:end-2) = new(end-2,3:end-2); new_old(3:end-2,end-1) = new(3:end-2,end-2);
    new = new_old;
    old = new;
end
xout = x;
yout = y;
tout = t(end);
new = new(3:numx+2,3:numy+2);
end
```

2-D ADI scheme

```
function CHproject_2D_implicit
close all
D = 100;
gamma = .2;
dr = .01;
dt = 1e-11;
t0 = 0;
tf1 = 1e-11;

save_str = '2d_implicit_rand_test2';
numx = length(0:dr:1);
c0 = create_initial_rand(numx,numx);
%imshow(c0)
tic
for i = 1:7
    istr = num2str(i);
    thisstr = ['[x' istr ',y' istr ',t' istr ',c' istr' ']'];
    if i == 1
        thisstr = [thisstr '=my_CD(0,tf1,1,1,dr,dr,dt,gamma,D,c0);'];
    else
        thisstr = [thisstr '=my_CD(t' num2str(i-1), ',t' num2str(i-1)];
        thisstr = [thisstr '*10,1,1,dr,dr,dt,gamma,D,c' num2str(i-1) ');'];
    end
    eval(thisstr)
    str = ['this_time' istr '=toc'];
    eval(str)
    save(save_str);
end

high = max(c(:));
low = min(c(:));
close all
imshow(c)
caxis([low high])
end
function A = create_initial_rand(numx,numy)
A = zeros(numx,numy);
for i = 1:numx
    for j = 1:numy
        A(i,j) = -1 + 2*rand;
    end
end
end
```

```
function A = create_initial_circ(numx,numy)
mdpntx = round(numx/2); mdpnty = round(numy/2);
A = zeros(numx,numy) -1;
for i = 1:numx
    for j = 1:numy
        dx = mdpntx - i;
        dy = mdpnty - j;
        if 2*dx^2 + dy^2 + 5*dx^2*dy - 20*dy^2*dx + 20*dx^3*dy - 10*dy^3*dx < 15
            A(i,j) = 1;
        end

    end
end
end

function [xout,yout,tout,new] = my_CD(t0,t_f,x_f,y_f,dx,dy,dt,gamma,D,u0)
t = t0:dt:t_f;
x = 0:dx:x_f;
y = 0:dy:y_f;
numx = length(x); numy = length(y); numt = length(t);
uold = u0;
unew = uold;
A = zeros(size(uold));
[nx,ny] = size(A);
b = zeros(nx,1);
rx = .5*D*dt/dx^2;
ry = .5*D*dt/dy^2;
rxx = .5*D*gamma^2*dt/dx^4;
ryy = .5*D*gamma^2*dt/dy^4;
for n = 1:numt-1
    %first do x direction
    for k = 1:numx %assume numx = numy
        for j = 1:numx
            if j == 2
                uoldjm2 = uold(k,1);
                uoldjm1 = uold(k,j-1);
            elseif j == 1
                uoldjm2 = uold(k,2);
                uoldjm1 = uold(k,1);
            else
                uoldjm1 = uold(k,j-1);
                uoldjm2 = uold(k,j-2);
            end
            if k == 2
                uoldkm2 = uold(1,j);
                uoldkm1 = uold(k-1,j);
            elseif k == 1
                uoldkm2 = uold(2,j);
                uoldkm1 = uold(1,j);
            else
                uoldkm1 = uold(k-1,j);
                uoldkm2 = uold(k-2,j);
            end
            if j == nx
                uoldjp1 = uold(k,nx);
                uoldjp2 = uold(k,nx-1);
            elseif j == nx-1
                uoldjp1 = uold(k,j+1);
                uoldjp2 = uold(k,nx);
            else
                uoldjp2 = uold(k,j+2);
                uoldjp1 = uold(k,j+1);
            end
            if k == nx
                uoldkp1 = uold(nx,j);
```

```
109                 uoldkp2 = uold(nx-1,j);
110             elseif k == nx-1
111                 uoldkp1 = uold(k+1,j);
112                 uoldkp2 = uold(nx,j);
113             else
114                 uoldkp1 = uold(k+1,j);
115                 uoldkp2 = uold(k+2,j);
116             end
117             if j > 1 && j < nx
118                 A(j,j-1) = rx - rx*(uoldjm1^2);
119                 A(j,j) = -2*rx + 2*rx*(uold(k,j)^2)+1;
120                 A(j,j+1) = rx - rx*(uoldjp1^2);
121             elseif j==1
122                 A(j,j) = -rx + rx*uold(k,j)^2 + 1;
123                 A(j,j+1) = rx - rx*uoldjp1^2;
124             elseif j == nx
125                 A(j,j) = -rx + rx*uold(k,j)^2 + 1;
126                 A(j,j-1) = rx - rx*uoldjm1^2;
127             end
128             b(j) = ry*(uoldkp1^3 - 2*uold(k,j)^3 + uoldkm1^3);
129             b(j) = b(j) - ry*(uoldkp1 - 2*uold(k,j) + uoldkm1);
130             b(j) = b(j) - rxx*(uoldjp2-4*uoldjp1+6*uold(k,j)-4*uoldjm1+uoldjm2);
131             b(j) = b(j) - ryy*(uoldkp2-4*uoldkp1+6*uold(k,j)-4*uoldkm1+uoldkm2);
132             b(j) = b(j) + uold(k,j);
133         end
134         this_sol = A \ b;
135         unew(k,:) = this_sol;
136     end
137     uold = unew;
138     A(:) = 0; b(:) = 0;%reset
139     %now do y direction
140     for j = 1:numx
141         for k = 1:numx
142             if j == 2
143                 uoldjm2 = uold(k,1);
144                 uoldjm1 = uold(k,j-1);
145             elseif j == 1
146                 uoldjm2 = uold(k,2);
147                 uoldjm1 = uold(k,1);
148             else
149                 uoldjm1 = uold(k,j-1);
150                 uoldjm2 = uold(k,j-2);
151             end
152             if k == 2
153                 uoldkm2 = uold(1,j);
154                 uoldkm1 = uold(k-1,j);
155             elseif k == 1
156                 uoldkm2 = uold(2,j);
157                 uoldkm1 = uold(1,j);
158             else
159                 uoldkm1 = uold(k-1,j);
160                 uoldkm2 = uold(k-2,j);
161             end
162             if j == nx
163                 uoldjp1 = uold(k,nx);
164                 uoldjp2 = uold(k,nx-1);
165             elseif j == nx-1
166                 uoldjp1 = uold(k,j+1);
167                 uoldjp2 = uold(k,nx);
168             else
169                 uoldjp2 = uold(k,j+2);
170                 uoldjp1 = uold(k,j+1);
171             end
172             if k == nx
173                 uoldkp1 = uold(nx,j);
```

```
                uoldkp2 = uold(nx-1,j);
            elseif k == nx-1
                uoldkp1 = uold(k+1,j);
                uoldkp2 = uold(nx,j);
            else
                uoldkp1 = uold(k+1,j);
                uoldkp2 = uold(k+2,j);
            end
            if k > 1 && k < nx
               A(k,k-1) = ry - ry*uoldkm1^2;
               A(k,k) = -2*ry + 2*ry*uold(k,j)^2 + 1;
               A(k,k+1) = ry - ry*uoldkp1^2;
            elseif k == 1
                A(k,k) = -ry + ry*uold(k,j)^2 + 1;
                A(k,k+1) = ry - ry*uoldkp1^2;
            elseif k ==nx
                A(k,k) = -ry + ry*uold(k,j)^2 + 1;
                A(k,k-1) = ry - ry*uoldkm1^2;
            end
            b(k) = ry*(uoldjp1^3 - 2*uold(k,j)^3 + uoldjm1^3);
            b(k) = b(k) - rx*(uoldjp1 - 2*uold(k,j) + uoldjm1);
            b(k) = b(k) - rxx*( uoldjp2 -4*uoldjp1 + 6*uold(k,j) -4*uoldjm1 + uoldjm2 );
            b(k) = b(k) - ryy*(uoldkp2-4*uoldkp1 + 6*uold(k,j) -4*uoldkm1+ uoldkm2 );
            b(k) = b(k) + uold(k,j);
        end
        unew(:,j) = A \ b;
    end
    uold = unew;
end
xout = x;
yout = y;
tout = t(end);
new = unew;
end
```

2-D plotting script

```
load('C:\Users\jhaydak3\Google Drive\School\Sholl Group\Jon ...
    Temp\pde_stuff\2d_implicit_fail_NR.mat')
c0 = double(c0)
close all
figure
imshow(c0,'InitialMagnification','fit');
high = max(c0(:));
low = min(c0(:));
caxis([low high])
title('t = 0')
for i = 1:7
        figure
        str = ['imshow(c' num2str(i) ','''InitialMagnification''','''fit''')'];
        eval(str)
        str = ['high = max(c' num2str(i) '(:));'];
        eval(str)
        str = ['low = min(c' num2str(i) '(:));'];
        eval(str)
        caxis([low high])
        str = ['t_str = num2str(t' num2str(i) ');'];
        eval(str)
        title(['t = ' t_str])
        xlabel('x'); ylabel('y')

end
 t_vec = [this_time1; this_time2; this_time3; this_time4; this_time5; this_time6; this_time7]
```

Python 2-D base code (explicit and ADI)

```
#!/usr/bin/env python
import numpy as np
import time
import twod_explicit
import twod_ADI


def my_2d_explicit(t0,t_f,x_f,y_f,dx,dy,dt,gamma,D,u0):
    t = np.arange(t0,t_f+dt,step = dt)
    x = np.arange(0,x_f+dx,step = dx)
    y = np.arange(0,y_f+dy,step = dy)
    numx = len(x)
    numy = len(y)
    numt = len(t)
    u0 = np.array(u0,order='F')
    twod_explicit.twod_explicit(u0,numt,dx,dy,dt,gamma,D)
    return x,y,t_f,u0

def my_2d_ADI(t0,t_f,x_f,y_f,dx,dy,dt,gamma,D,u0):
    t = np.arange(t0,t_f+dt,step = dt)
    x = np.arange(0,x_f+dx,step = dx)
    y = np.arange(0,y_f+dy,step = dy)
    numx = len(x)
    numy = len(y)
    numt = len(t)
    u0 = np.array(u0,order='F')
    twod_ADI.twod_adi(u0,numt,dx,dy,dt,gamma,D)
    return x,y,t_f,u0

def create_initial_rand(numx,numy):
    np.random.seed(seed=34)
    A = -1 + 2*np.random.rand(numx,numy)
    return A
def create_initial_circ(numx,numy):
    A = np.zeros([numx,numy]) - 1
    mdpntx = round(.5*numx)
    mdpnty = round(.5*numy)
    for i in range(0,numx):
        for j in range(0,numy):
            dx = mdpntx - i
            dy = mdpnty - j
            if 2*dx**2 + dy**2 + 5*dx**2*dy - 20*dy**2*dx + 20*dx**3*dy - 10*dy**3*dx < 15:
                A[i,j] = 1
    return A
def main():
    save_str = 'adi_NR_1'
    D = 100
    gamma = .2
    dx = .01; dy = .01
    dt = 1e-11;
    t0 = 0; tf1 = 1e-11
    numSims = 7
    tvec = [0, tf1, tf1*10, tf1*1e2, tf1*1e3, tf1*1e4, tf1*1e5, tf1*1e6]
    time2run = np.zeros(7)
    tout = np.zeros(7)
    x = np.arange(0,1+dx,step = dx)
    numx = len(x)
    #u0 = create_initial_rand(numx,numx)
    u0 = create_initial_circ(numx,numx)
    fh_string = save_str + '_start.txt'
    this_str = 'D = '+ str(D) +', gamma =  ' + str(gamma) + ', dx = ' + str(dx) + ', dy = ...
        ' + str(dy)
```

```
    this_str = this_str + ', dt = ' + str(dt) + ', t =  ' + str(0) + '\n'
    np.savetxt(fh_string,u0,header = this_str)
    start_time = time.time()
    for i in range(0,numSims):
        #xout,yout,this_t,u0 = ...
            my_2d_explicit(tvec[int(i)],tvec[int(i)+1],1,1,dx,dy,dt,gamma,D,u0)
        xout,yout,this_t,u0 = my_2d_ADI(tvec[int(i)],tvec[int(i)+1],1,1,dx,dy,dt,gamma,D,u0)
        time_now = time.time()
        time2run[i] = time_now - start_time
        tvec[int(i)] = this_t
        #write data with file
        fh_string = save_str + '_' + str(i) + '.txt'
        this_str = 'D = '+ str(D) +', gamma =  ' + str(gamma) + ', dx = ' + str(dx) + ', ...
            dy = ' + str(dy)
        this_str = this_str + ', dt = ' + str(dt) + ', t =  ' + str(this_t) + ', t_elap = ...
            ' + str(time2run[i]) + '\n'
        np.savetxt(fh_string,u0,header = this_str)
        print i

if __name__ == "__main__":
    main()
```

Fortran subroutine, 2-D explicit scheme

```
subroutine twod_explicit(u0,nx,ny,numt,dx,dy,dt,gamma,D)
    implicit none
    real(8), intent(inout), dimension(0:nx-1,0:ny-1) :: u0
!f2p intent(in,out) :: u0
    integer, intent(in) :: nx, ny, numt
!f2p intent(in) :: nx, ny, numt
    real(8), intent(in) :: dx,dy,dt,gamma,D
!f2p intent(in) :: dx,dy,gamma,dt,D
    real(8) :: unew(0:nx+3,0:nx+3) , uold(0:nx+3,0:nx+3)
    real(8) :: term1, term2, term3, term4, term5, term6, RHS
    integer :: n,j,k
!apply the boundary conditions
    unew = 0
    !print *, SHAPE(unew)
    !print *, SHAPE(u0)
    unew(2:nx+1,2:nx+1) = u0
    unew(0,2:nx+1) = u0(1,:)
    unew(2:nx+1,0) = u0(:,1)
    unew(1,2:nx+1) = u0(0,:)
    unew(2:nx+1,1) = u0(:,0)
    unew(nx+3,2:nx+1) = u0(nx-2,:)
    unew(2:nx+1,nx+3) = u0(:,nx-2)
    unew(nx+2,2:nx+1) = u0(nx-1,:)
    unew(2:nx+1,nx+2) = u0(:,nx-1)
    uold = unew
    do n = 1,numt-1
        do j = 2,nx+1
            do k = 2,nx+1
                term1 = uold(k,j+1)**3 - 2 * uold(k,j)**3 + uold(k,j-1)**3
                term2 = uold(k+1,j)**3 - 2 * uold(k,j)**3 + uold(k-1,j)**3
                term3 = uold(k,j+1) - 2*uold(k,j) + uold(k,j-1)
                term4 = uold(k+1,j) - 2*uold(k,j) + uold(k-1,j)
                term5 = uold(k,j-2) -4*uold(k,j-1) + 6*uold(k,j) -4*uold(k,j+1) + uold(k,j+2)
                term6 = uold(k-2,j) - 4*uold(k-1,j) + 6*uold(k,j) - 4*uold(k+1,j) + ...
                    uold(k+2,j)
                RHS = term1/(dx**2) + term2/(dy**2) - term3/(dx**2) - term4/(dy**2)
                RHS = RHS - gamma**2 * term5 / (dx**4) - gamma**2 * term6 / (dy**4)
                unew(k,j) = D*RHS*dt + uold(k,j)
            end do
        end do
```

```
        uold = unew;
        !apply boundary conditions again    unew(2:nx+1,2:nx+1) = u0
        uold(0,2:nx+1) = unew(3,2:nx+1)
        uold(2:nx+1,0) = unew(2:nx+1,3)
        uold(1,2:nx+1) = unew(2,2:nx+1)
        uold(2:nx+1,1) = unew(2:nx+1,2)
        uold(nx+3,2:nx+1) = unew(nx,2:nx+1)
        uold(2:nx+1,nx+3) = unew(2:nx+1,nx)
        uold(nx+2,2:nx+1) = unew(nx+1,2:nx+1)
        uold(2:nx+1,nx+2) = unew(2:nx+1,nx+1)
        unew = uold
    end do
    u0 = unew(2:nx+1,2:nx+1)
end subroutine
```

Fortran subroutine, 2-D ADI scheme

```
subroutine twod_ADI(u0,nx,ny,numt,dx,dy,dt,gamma,D)
    implicit none
    real*8, intent(inout), dimension(nx,ny) :: u0
!f2p intent(in,out) :: u0
    integer, intent(in) :: nx, ny, numt
!f2p intent(in) :: nx, ny, numt
    real*8, intent(in) :: dx,dy,dt,gamma,D
!f2p intent(in) :: dx,dy,gamma,dt,D
    real*8 :: unew(nx,nx) , uold(nx,nx), A(nx,nx)
    real*8:: b(nx), Diag(nx)
    real*8 :: rx, ry, rxx, ryy, uoldkp1, uoldkp2, uoldkm1, uoldkm2, uoldjp1, uoldjp2, ...
        uoldjm2, uoldjm1
    real*8 :: DL(nx-1), DU(nx-1)
    integer :: n,j,k, info, lower, upper, ii, jj, ku, kl
    integer :: IPIV(nx+4)
    CHARACTER(LEN=30) :: rowfmt
!apply the boundary conditions
    unew = 0
    uold = u0
    !first do x directions
    A = 0
    b = 0
    rx = DBLE(.5)*D*dt/(dx**2)
    ry = DBLE(.5)*D*dt/(dy**2)
    rxx = DBLE(.5)*D*(gamma**DBLE(2))*dt/(dx**4.00)
    ryy = DBLE(.5)*D*(gamma**DBLE(2))*dt/(dy**4.00)
    !print *, rx, ry, rxx, ryy
    !stop
    do n = 1,numt-1
        do k = 1,nx
            do j = 1,nx
                IF (j == 2) THEN
                    uoldjm2 = uold(k,1)
                    uoldjm1 = uold(k,j-1)
                ELSE IF (j == 1) THEN
                    uoldjm2 = uold(k,2)
                    uoldjm1 = uold(k,1)
                ELSE
                    uoldjm1 = uold(k,j-1)
                    uoldjm2 = uold(k,j-2)
                END IF
                IF (k == 2) THEN
                    uoldkm2 = uold(1,j)
                    uoldkm1 = uold(k-1,j)
                ELSE IF (k == 1) THEN
                    uoldkm2 = uold(2,j)
                    uoldkm1 = uold(1,j)
```

```
47                  ELSE
48                      uoldkm1 = uold(k-1,j)
49                      uoldkm2 = uold(k-2,j)
50                  END IF
51                  IF (j == nx) THEN
52                      uoldjp1 = uold(k,nx)
53                      uoldjp2 = uold(k,nx-1)
54                  ELSE IF (j == nx-1) THEN
55                      uoldjp1 = uold(k,j+1)
56                      uoldjp2 = uold(k,nx)
57                  ELSE
58                      uoldjp1 = uold(k,j+1)
59                      uoldjp2 = uold(k,j+2)
60                  END IF
61                  IF (k == nx) THEN
62                      uoldkp1 = uold(nx,j)
63                      uoldkp2 = uold(nx-1,j)
64                  ELSE IF (k == nx-1) THEN
65                      uoldkp1 = uold(k+1,j)
66                      uoldkp2 = uold(nx,j)
67                  ELSE
68                      uoldkp1 = uold(k+1,j)
69                      uoldkp2 = uold(k+2,j)
70                  END IF
71                  IF (j > 1 .AND. j < nx ) THEN
72                      A(j,j-1) = rx - rx*(uoldjm1**2)
73                      A(j,j) = DBLE(-2)*rx + DBLE(2)*rx*(uold(k,j)**2) + DBLE(1)
74                      A(j,j+1) = rx - rx*(uoldjp1**2)
75                  ELSE IF (j == 1) THEN
76                      A(j,j) = -rx + rx*(uold(k,j)**2) + DBLE(1)
77                      A(j,j+1) = rx - rx*(uoldjp1**2)
78                  ELSE IF (j == nx) THEN
79                      A(j,j) = -rx + rx*(uold(k,j)**2) + DBLE(1)
80                      A(j,j-1) = rx - rx*(uoldjm1**2)
81                  END IF
82                  b(j) = ry*(uoldkp1**3 - DBLE(2)*uold(k,j)**3 + uoldkm1**3)
83                  b(j) = b(j)  - ry*(uoldkp1 - DBLE(2)*uold(k,j) + uoldkm1)
84                  b(j) = b(j) -rxx*(uoldjp2-DBLE(4)*uoldjp1 + DBLE(6)*uold(k,j) ...
                        -DBLE(4)*uoldjm1+uoldjm2 )
85                  b(j) = b(j) - ryy*(uoldkp2-DBLE(4)*uoldkp1 + DBLE(6)*uold(k,j) ...
                        -DBLE(4)*uoldkm1+uoldkm2 )
86                  b(j) = b(j) + uold(k,j)
87              end do
88              !sub ,super, digonal
89              DO ii = 1,nx-1
90                  DL(ii) = A(ii+1,ii)
91                  Diag(ii) = A(ii,ii)
92                  DU(ii) = A(ii,ii+1)
93              END DO
94              Diag(nx) = A(nx,nx)
95              ! open (unit = 2, file = 'DL.txt')
96              ! do ii = 1,nx-1
97                  ! write(2,'(ES)') DL(ii)
98              ! end do
99              ! close(2)
100             !WRITE(rowfmt,'(A,I4,A)') '(',nx,'(ES))'
101             ! open (unit = 2, file = 'DU.txt')
102             ! do ii = 1,nx-1
103                 ! write(2,'(ES)') DU(ii)
104             ! end do
105             ! close(2)
106             ! open (unit = 2, file = 'c0.txt')
107             ! do ii = 1,nx
108                 ! write(2,FMT=rowfmt) (u0(ii,jj), jj = 1,nx)
109             ! end do
```

```
            ! close(2)
            ! open (unit = 2, file = 'D.txt')
            ! do ii = 1,nx
                ! write(2,'(ES)') Diag(ii)
            ! end do
            ! close(2)
            ! open (unit = 2, file = 'b_b4.txt')
            ! do ii = 1,nx
                ! write(2,'(ES)') b(ii)
            ! end do
            ! close(2)
            CALL DGTSV(nx,1,DL,Diag,DU,b,nx,info)
            ! open (unit = 2, file = 'b_after.txt')
            ! do ii = 1,nx
                ! write(2,'(ES)') b(ii)
            ! end do
            ! close(2)
            ! open (unit = 2, file = 'A.txt')
            ! do ii = 1,nx
                ! write(2,FMT=rowfmt) (A(ii,jj), jj = 1,nx)
            ! end do
            ! close(2)
            ! stop
            !print *, info, j, k
            !stop
            DO ii = 1,nx
                unew(k,ii) = b(ii)
            END DO
            !print *, unew
        end do
        ! open (unit = 2, file = 'unew.txt')
        ! do ii = 1,nx
            ! write(2,FMT=rowfmt) (unew(ii,jj), jj = 1,nx)
        ! end do
        ! close(2)
        ! stop
        !now do y directions
        uold = unew
        A = 0
        b = 0
        do j = 1,nx
            do k = 1,nx
                IF (j == 2) THEN
                    uoldjm2 = uold(k,1)
                    uoldjm1 = uold(k,j-1)
                ELSE IF (j == 1) THEN
                    uoldjm2 = uold(k,2)
                    uoldjm1 = uold(k,1)
                ELSE
                    uoldjm1 = uold(k,j-1)
                    uoldjm2 = uold(k,j-2)
                END IF
                IF (k == 2) THEN
                    uoldkm2 = uold(1,j)
                    uoldkm1 = uold(k-1,j)
                ELSE IF (k == 1) THEN
                    uoldkm2 = uold(2,j)
                    uoldkm1 = uold(1,j)
                ELSE
                    uoldkm1 = uold(k-1,j)
                    uoldkm2 = uold(k-2,j)
                END IF
                IF (j == nx) THEN
                    uoldjp1 = uold(k,nx)
                    uoldjp2 = uold(k,nx-1)
```

```
                ELSE IF (j == nx-1) THEN
                    uoldjp1 = uold(k,j+1)
                    uoldjp2 = uold(k,nx)
                ELSE
                    uoldjp1 = uold(k,j+1)
                    uoldjp2 = uold(k,j+2)
                END IF
                IF (k == nx) THEN
                    uoldkp1 = uold(nx,j)
                    uoldkp2 = uold(nx-1,j)
                ELSE IF (k == nx-1) THEN
                    uoldkp1 = uold(k+1,j)
                    uoldkp2 = uold(nx,j)
                ELSE
                    uoldkp1 = uold(k+1,j)
                    uoldkp2 = uold(k+2,j)
                END IF
                IF (k > 1 .AND. k < nx ) THEN
                    A(k,k-1) = ry - ry*(uoldkm1**2)
                    A(k,k) = DBLE(-2)*ry + DBLE(2)*ry*(uold(k,j)**2) + DBLE(1)
                    A(k,k+1) = ry - ry*(uoldkp1**2)
                ELSE IF (k == 1) THEN
                    A(k,k) = -ry + ry*(uold(k,j)**2) + DBLE(1)
                    A(k,k+1) = ry - ry*(uoldkp1**2)
                ELSE IF (k == nx) THEN
                    A(k,k) = -ry + ry*(uold(k,j)**2) + DBLE(1)
                    A(k,k-1) = ry - ry*(uoldkm1**2)
                END IF

                b(k) = ry*(uoldjp1**3 - DBLE(2)*uold(k,j)**3 + uoldjm1**3)
                b(k) = b(k) - rx*(uoldjp1 - DBLE(2)*uold(k,j) + uoldjm1)
                b(k) = b(k) - rxx*(uoldjp2 -DBLE(4)*uoldjp1 + DBLE(6)*uold(k,j) ...
                    -DBLE(4)*uoldjm1 + uoldjm2 )
                b(k) = b(k) - ryy*(uoldkp2 - DBLE(4)*uoldkp1+ DBLE(6)*uold(k,j) ...
                    -DBLE(4)*uoldkm1 + uoldkm2 )
                b(k) = b(k) + uold(k,j)
            end do
            !sub ,super, digonal
            DO ii = 1,nx-1
                DL(ii) = A(ii+1,ii)
                Diag(ii) = A(ii,ii)
                DU(ii) = A(ii,ii+1)
            END DO
            Diag(nx) = A(nx,nx)
            ! open (unit = 2, file = 'b_b42.txt')
            ! do ii = 1,nx
                ! write(2,'(ES)') b(ii)
            ! end do
            ! close(2)
            ! WRITE(rowfmt,'(A,I4,A)') '(',nx,'(ES))'
            ! open (unit = 2, file = 'A2.txt')
            ! do ii = 1,nx
                ! write(2,FMT=rowfmt) (A(ii,jj), jj = 1,nx)
            ! end do
            ! close(2)
            CALL DGTSV(nx,1,DL,Diag,DU,b,nx,info)
            !print *, info, j, k
            DO ii = 1,nx
                unew(ii,j) = b(ii)
            END DO
            ! open (unit = 2, file = 'b_after2.txt')
            ! do ii = 1,nx
                ! write(2,'(ES)') b(ii)
            ! end do
            ! close(2)
```

```
238           end do
239           uold = unew;
240
241       end do
242       !stop
243       u0 = unew
244   end subroutine
```

Python Plotting script (results were plotted using Matlab)

```
1   base_fh = 'C:\Users\MrBes\Google Drive\School\Sholl Group\Jon Temp\lon_SF6\tc_files\adi_NR_4';
2   close all
3   %read data
4   this_fh = [base_fh , '_start.txt'];
5   this_fh = fopen(this_fh,'r')
6   this_line = fgets(this_fh);
7   this_line = fgets(this_fh); %two lines are junk
8   this_line = fgets(this_fh);
9   c = [];
10  mass_int = [];
11  t_elap= [];
12  while this_line ~= -1
13      c = [c; str2num(this_line)];
14      this_line = fgets(this_fh);
15  end
16   mass_int = [mass_int; trapz(y1,trapz(x1,c,2))];
17  fclose(this_fh);
18  figure
19  imshow(c,'InitialMagnification','fit');
20  high = max(c(:));
21  low = min(c(:));
22  caxis([low high])
23  title('t = 0')
24  xlabel('x')
25  ylabel('y')
26  for i = 1:7
27      this_fh = [base_fh ,'_', num2str(i-1), '.txt'];
28      this_fh = fopen(this_fh,'r');
29      this_line = fgets(this_fh);
30      substr = strfind(this_line,' t = ');
31      substr = this_line(substr:end);
32      comma_ndx = strfind(substr,',')
33      s_ndx = strfind(substr,'=');
34      tsubstr = substr(s_ndx(1)+1:comma_ndx-1);
35      this_t = str2num(tsubstr);
36      t_elap_substr = substr(s_ndx(2)+1:end);
37      t_elap = [t_elap; str2num(t_elap_substr)];
38
39      this_line = fgets(this_fh); %second line is junk
40      this_line = fgets(this_fh);
41      c = [];
42      while this_line ~= -1
43          c = [c; str2num(this_line)];
44          this_line = fgets(this_fh);
45      end
46      mass_int = [mass_int; trapz(y1,trapz(x1,c,2))];
47      fclose(this_fh)
48      figure
49  %     imshow(c,'InitialMagnification','fit');
50  %     high = max(c(:));
51  %     low = min(c(:));
52  %     caxis([low high])
53  %     title(['t = ', num2str(this_t)])
54  %     xlabel('x')
```

```
55 %     ylabel('y')
56 end
57 mass_int
58 t_elap
```