An open problem in pattern forming systems

Edward McDugald

November 27, 2022

1 Introduction

The aim of this report is to describe an open problem in pattern forming systems, pertaining specifically to patterns deriving from microscopic gradient flows that are translationally and rotationally invariant, and that admit straight parallel roll solutions on infinite domains. On finite domains however, such rolls may emerge in different locations with different orientations, and they meet in the interior of the domain generating point and line defects. Such pattern forming systems, while arising from various microscopic gradient systems, obey a universal PDE describing the evolution of the phase variable, θ.

A canonical example of such patterns in nature is seen through Rayleigh-Bénard convection. Rayleigh-Bénard convection may be observed by trapping a thin layer of fluid between two plates, and applying heat to the bottom plate. In the microscopic model which we will consider, the stress parameter R is the difference in heat between the top and bottom plate. Initially heat will transfer through the fluid by conduction, however past a critical value, heat will transfer through convection, generating a velocity field that manifests as temperature rolls in the fluid. In the figures we will present in this report, we may consider the surfaces as being the average temperature along the z direction.

The procedure for deriving such a phase equation involves studying the vector field whose components are the instantaneous frequencies of the pattern generated by the microscopic equations. We call this vector field $\boldsymbol{k}$, and note that this is the gradient of the phase, $\boldsymbol{k} = \nabla\theta$. The phase gradient is modulated over distances L, the average distance between defects, which is long with respect to the local wavelength l. We introduce the following

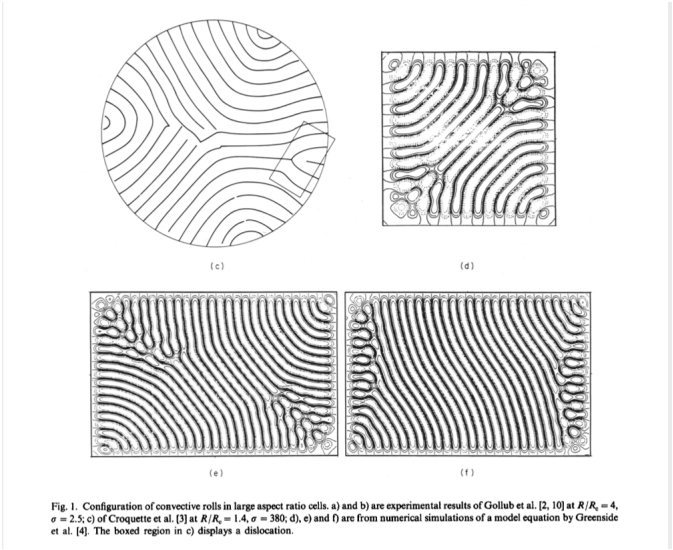

Fig. 1. Configuration of convective rolls in large aspect ratio cells. a) and b) are experimental results of Gollub et al. [2, 10] at $R/R_c = 4$, $\sigma = 2.5$; c) of Croquette et al. [3] at $R/R_c = 1.4$, $\sigma = 380$; d), e) and f) are from numerical simulations of a model equation by Greenside et al. [4]. The boxed region in c) displays a dislocation.

Figure 1.1: Numerical and experimental depictions of convection rolls.

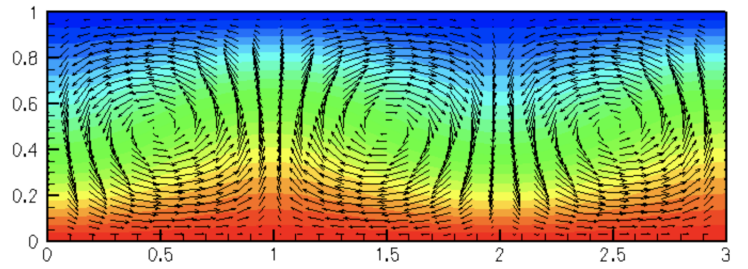

Figure 1.2: Local Periodicity of convection rolls.

variables:

$$\begin{aligned} \boldsymbol{X} &= \epsilon \boldsymbol{x} \\ T &= \epsilon^2 t \\ \Theta(\boldsymbol{X}, T) &= \epsilon \theta(\boldsymbol{x}, t) \\ k &= \|\boldsymbol{k}\| \\ \epsilon &= \frac{l}{L} \quad \text{The inverse aspect ratio.} \end{aligned}$$

Letting w be the variable of the microscopic equations, and E the associated energy functional, we apply the following modulational Ansatz:

$$w = \sum A_n(k^2)\cos(n\theta) \quad \text{(for real fields)} \tag{1.1}$$

$$w = A(k^2, \boldsymbol{X}, T)e^{i\theta(\boldsymbol{x},t)} \quad \text{(for complex fields).} \tag{1.2}$$

We then average the energy over the peridicity of the pattern

$$\overline{E} = \frac{1}{2\pi}\int E d\theta \tag{1.3}$$

and analyze the dependence of the energy on A and Θ in orders of ϵ^n. We find that the only orders of ϵ that contribute are $\mathcal{O}(\epsilon^0)$ and $\mathcal{O}(\epsilon^2)$, and that the energy can be interpreted as a sum of stretching and bending energies. We also find that stationary solutions can be deduced by balancing the stress and strain energies. We pose an open challenge, where a certain pattern appearing in simulations of microscopic gradient flows, and in experiments, is not yet deduced by the Cross-Newell equation. We also discuss some potential uses of machine learning to approach this problem.

2 The Regularized Cross-Newell Phase Diffusion Equation for Complex Fields

2.1 Deriving the Unegularized Equation

We consider pattern forming systems that arise as PDEs that are equivalently gradient flows. These are a very general object, and we take as our preferred example to be the Swift Hohenberg equation. In its complexified form, the swift hohenberg equation has energy density

$$E = \int \left((\nabla^2 + 1)w(\nabla^2 + 1)w^* - Rww^* + \frac{1}{2}w^2 w^{*^2} \right) dx. \tag{2.1}$$

Let us compute δE,

$$\begin{aligned}\delta E &= \lim_{h\to 0} \frac{E[w+h\delta w, w^*+h\delta w^*] - E[w,w^*]}{h} \\ &= \int \big((\nabla^2+1)w(\nabla^2+1)\delta w^* - Rw\delta w^* + w^2 w^*\delta w^* + \\ &\quad (\nabla^2+1)\delta w(\nabla^2+1)w^* - Rw^*\delta w + w{w^*}^2\delta w\Big)\, d\boldsymbol{x}\end{aligned}$$

Using integration by parts, one finds that

$$\begin{aligned}(\nabla^2+1)w(\nabla^2+1)\delta w^* &= (\nabla^2+1)^2 w\delta w^* \\ (\nabla^2+1)\delta w(\nabla^2+1)^2 w^* &= (\nabla^2+1)^2 w^*\delta w.\end{aligned}$$

Thus, one obtains

$$\frac{\delta E}{\delta w^*} = (\nabla^2+1)^2 w - Rw + w^2 w^* \tag{2.2}$$

$$\frac{\delta E}{\delta w} = (\nabla^2+1)^2 w^* - Rw^* + {w^*}^2 w. \tag{2.3}$$

The complexified Swift-Hohenberg equations can be written as

$$w_t = -\frac{\delta E}{\delta w^*}, \quad w_2^* = -\frac{\delta E}{\delta w}. \tag{2.4}$$

Using the Ansatz for complex fields (1.2), one can write

$$\delta E = \frac{\delta E}{\delta w}\delta w + \frac{\delta E}{\delta w^*}\delta w^* = -2A_t\delta A - 2A^2\Theta_t\delta\theta. \tag{2.5}$$

We want now to find expressions for $\frac{\delta E}{\delta A}$ and $\frac{\delta E}{\delta \Theta}$ Using the fact that $\nabla_{\boldsymbol{x}} = \epsilon\nabla_{\boldsymbol{X}}$, and writing $w = Ae^{i\theta} = A(\boldsymbol{x})e^{i\theta(\boldsymbol{x})}$, one obtains

$$\nabla_{\boldsymbol{x}} w = e^{i\theta}(i\boldsymbol{k} + \epsilon\nabla_{\boldsymbol{X}})A \tag{2.6}$$

$$\nabla_{\boldsymbol{x}}^2 w = e^{i\theta}(-k^2 + i\epsilon(2\boldsymbol{x}\cdot\nabla_{\boldsymbol{X}} + \nabla_{\boldsymbol{X}}\cdot\boldsymbol{k}) + \epsilon^2\nabla_{\boldsymbol{X}}^2)A \tag{2.7}$$

$$\nabla_{\boldsymbol{x}} w^* = e^{-i\theta}(-i\boldsymbol{k} + \epsilon\nabla_{\boldsymbol{X}})A \tag{2.8}$$

$$\nabla_{\boldsymbol{x}}^2 w^* = e^{-i\theta}(-k^2 - i\epsilon(2\boldsymbol{x}\cdot\nabla_{\boldsymbol{X}} + \nabla_{\boldsymbol{X}}\cdot\boldsymbol{k}) + \epsilon^2\nabla_{\boldsymbol{X}}^2)A. \tag{2.9}$$

Inserting (1.2) into (2.1) with (2.6)-(2.9), and *ignoring* terms of order $\mathcal{O}(\epsilon^3)$, one obtains

$$\begin{aligned}E = &\int \left((k^2-1)^2A^2 - RA^2 + \frac{1}{4}A^4 + K\right) d\boldsymbol{x} \\ &+ \epsilon^2 \int \left((2\boldsymbol{k}\cdot\nabla A + \nabla\cdot\boldsymbol{k}A)^2 + 2(1-k^2)A\nabla^2 A\right) d\boldsymbol{x}.\end{aligned} \tag{2.10}$$

Now, obtaining macroscopic equations in terms of the phase parameter requires averaging the energy over the periodicity of the pattern. Conveniently, in the complex field case, this averaging occurs automatically, and we have $E = \overline{E}$. Equation (2.5) gives

$$-2A_t = \frac{\delta \overline{E}}{\delta A} = \frac{\delta \overline{E_0}}{\delta A} + \epsilon^2 \frac{\delta \overline{E_2}}{\delta A}. \tag{2.11}$$

By construction, $A_t = \epsilon^2 A_T$, and therefore A_t is at most $\mathcal{O}(\epsilon^2)$. Thus, to leading order, the main equation contributing to the evolution of the amplitude A is

$$\frac{\delta \overline{E_0}}{\delta A} = 0. \tag{2.12}$$

This gives

$$A^2 = R - (K^2 - 1)^2. \tag{2.13}$$

This is known as the eikonal solution, which demands that the amplitude be slaved algebraically to the modulus of the phase gradient. This property holds for stress parameter values far from onset.

Having found a dominant relation for A, we seek an equation for Θ. Using the eikonal solution, we can write the leading order energy as

$$\overline{E_0} = \int -\frac{1}{2} \left(A^2(k^2) \right)^2 + K. \tag{2.14}$$

Considering just the $\mathcal{O}(\epsilon^0)$ contribution, equation (2.5) gives

$$A^2 \theta_t = -\frac{1}{2} \frac{\delta \overline{E_0}}{\delta \theta}. \tag{2.15}$$

We have

$$\begin{aligned} \frac{\delta \overline{E_0}}{\delta \theta} \delta\theta &= \delta \left(-\frac{1}{2} (A^2(k^2))^2 \right) \\ &= -A^2(k^2) \frac{d}{dk^2} A^2(k^2) 2\boldsymbol{k} \cdot \delta \nabla \theta. \end{aligned}$$

Using integration by parts, one finds that

$$-A^2(k^2) \frac{d}{dk^2} A^2(k^2) 2\boldsymbol{k} \cdot \delta \nabla \theta = A^2(k^2) \frac{d}{dk^2} A^2(k^2) 2\nabla \cdot \boldsymbol{k} \delta\theta.$$

Thus, we obtain

$$A^2 \theta_t = -A^2(k^2) \frac{dA^2(k^2)}{dk^2} 2\nabla \cdot \boldsymbol{k}$$

Noting that $\theta_t = \epsilon \Theta_t$, $\nabla_{\boldsymbol{x}} = \epsilon \nabla_{\boldsymbol{X}}$, and setting

$$B(k^2) = A^2(k^2) \frac{dA^2(k^2)}{dk^2}, \tag{2.16}$$

one obtains the *unregularized* Cross-Newell phase diffusion equation

$$A^2 \Theta_T + \nabla_{\boldsymbol{X}} \cdot \boldsymbol{k} B(k^2) = 0. \tag{2.17}$$

2.2 Regularizing the Cross-Newell phase diffusion equation

Recall the form of $\overline{E_0}$, $\overline{E_0} = \int -\frac{1}{2}A^4(k^2) + K$. Let us define k_B to be the preferred wave number, by which we mean the wave number that minimizes E_0. That is, k_B is defined to satisfy the equation

$$\frac{d}{dk^2}A^4(k^2) = 0. \tag{2.18}$$

We define K to be $-\frac{1}{2}A^4(k^2)$ evaluated at k_B, and we can then write

$$\overline{E_0} = \int \left(-\frac{1}{2}A^4(k^2)\right)_{k^2}^{k_B^2} d\boldsymbol{x} \tag{2.19}$$

Note that this choice of K ensures that the leading order energy is 0 whenever $k = k_B$.

Let us look at the unregularized equation (2.17). We can write this as a diffusion equation in standard form

$$A^2\Theta_T = -\nabla_{\boldsymbol{X}}\left[B(k^2)\nabla\theta\right]. \tag{2.20}$$

Expanding the right hand side yields

$$A^2\Theta_T = \theta_{xx}\left(-\frac{dB(k^2)}{dk^2}2\theta_x^2 - B(k^2)\right) + \theta_{xy}\left(-\frac{dB(k^2)}{dk^2}4\theta_x\theta_y\right) + \theta_{yy}\left(-\frac{dB(k^2)}{dk^2}2\theta_y^2 - B(k^2)\right). \tag{2.21}$$

Consider the simple case where $\boldsymbol{k} = \langle\theta_x, 0\rangle$. We then obtain

$$A^2\Theta_T = \theta_{xx}\left(-\frac{dB(k^2)}{dk^2}2\theta_x^2 - B(k^2)\right) + \theta_{yy}\left(-B(k^2)\right).$$

To have a well-posed diffusion equation, we need that

$$-\frac{dB(k^2)}{dk^2}2\theta_x^2 - B(k^2) > 0 \tag{2.22}$$

$$-B(k^2) > 0 \tag{2.23}$$

Noting that

$$\frac{d}{dk}\left(kB(k^2)\right) = B(k^2) + k\frac{d}{dk}B(k^2) = B(k^2) + 2k^2\frac{d}{dk^2}B(k^2), \tag{2.24}$$

We see that we require

$$\frac{d}{dk}(kB(k^2)) < 0 \tag{2.25}$$

$$B(k^2) < 0. \tag{2.26}$$

It turns out that $B, (kB)'$ are the eigenvalues of the second order quasi-linear operator, and for all k the above must hold. Let us consider the plot of k vs kB. For the case of Swift-Hohenberg with $R = .5$, we have $B = -4(k^2-1)(.5-(k^2-1)^2)$. We see that there is a

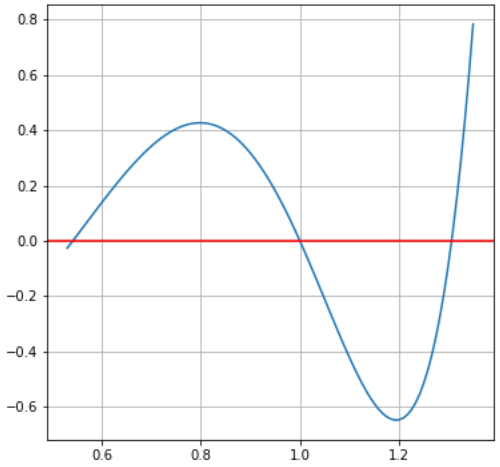

Figure 2.1: Busse Balloon for Swift Hohenberg, $R = .5$

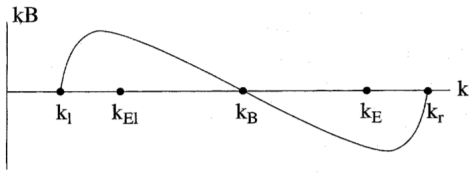

Fig. 2. Graph of kB versus $k.k_l(R), k_r(R)$ are the borders of the neutral stability curve. $\nabla \cdot kB$ is elliptic negative in $k_B < k < k_E$, hyperbolic in $k_{El} < k < k_B$ and $k_E < k < k_r$, elliptic positive in $k_l < R < k_{El}.k_{El}(R)$ and $k_E(R)$ are the left and right Eckhaus instability boundaries. k_B is the zig-zag instability boundary. The region in (R, k) space for which $k_B(R) \leq k < k_E(R)$ is called the Busse balloon. k_{El} and k_E are the wavenumbers when kB is maximum and minimum respectively.

Figure 2.2: Generic Busse Balloon

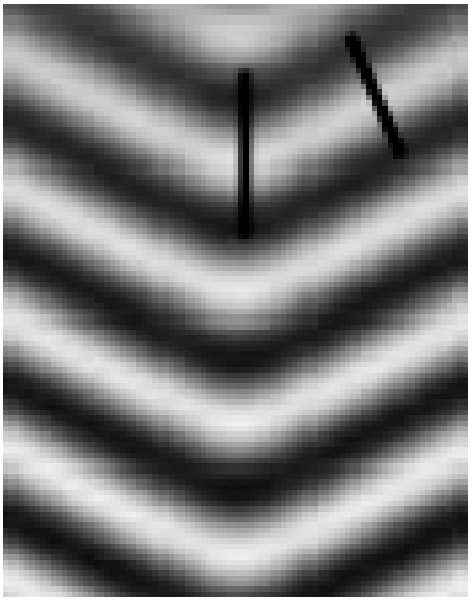

Figure 2.3: Change in wavelength along line defects

limited range of wave numbers where these values are satisfied. This range of wave numbers is knowen as the Busse-Balloon, and is the range $k_B < k < k_E$ as labeled in figure 2.2. It turns out for natural patterns, almost everywhere $k = k_B$, however along line defects we will have $k < k_B$.

Consider, for example, the scenario depicted in figure 2.3. We see that along the line defect, the pattern has a longer wavelength. This means that the local spatial frequences are smaller than they are along most of the pattern, where the wavenumber is close to k_B. Thus, along the line defect $k < K_B$, which lies outside of the Busse Balloon. Hence, the unregularized Cross-Newell phase diffusion equation is ill-posed along line defects, and regularization is required.

To regularize the Cross-Newell phase diffusion equation, we analyze the $\mathcal{O}(\epsilon^2)$ part of the energy, $\overline{E_2}$. We have that

$$\overline{E_2} = \int \left((2\boldsymbol{k} \cdot \nabla A + \nabla \cdot \boldsymbol{k} A)^2 + 2(1 - k^2) A \nabla^2 A \right) d\boldsymbol{x}. \tag{2.27}$$

It turns out that since $\nabla_{\boldsymbol{x}} = \boldsymbol{k} \frac{\partial}{\partial \theta} + \epsilon \nabla_{\boldsymbol{X}}$, the terms either involve differentiating $\boldsymbol{k}$ as it changes direction along the phase contour, or differentiation of k^2 in the direction of $\boldsymbol{k}$. The terms $2\boldsymbol{k} \cdot \nabla A$ and $A \nabla^2 A$ are small, and $\overline{E_2}$ can be well approximated by

$$\int A^2(k_B^2)(\nabla \cdot \boldsymbol{k})^2 d\boldsymbol{x}. \tag{2.28}$$

Computing its variation, we obtain the regularized Cross-Newell phase diffusion equation

$$A^2(k_B^2)\Theta_T + \nabla \cdot \boldsymbol{k} B + \epsilon^2 A^2(k_B^2) \nabla^4 \Theta = 0. \tag{2.29}$$

Moreover, we can approximate the energy $\overline{E}$ by

$$\overline{E} = \int \left(\left((\frac{d}{dk^2} B(k^2)) \right)_{k_B^2} (k^2 - k_B^2) + \epsilon^2 (\nabla \cdot \boldsymbol{k})^2 A^2(k_B^2) \right) d\boldsymbol{x}. \tag{2.30}$$

The $\mathcal{O}(\epsilon^0)$ contribution is the "strain" energy, and the $\mathcal{O}(\epsilon^2)$ contribution is the "bending" energy.

3 The regularized phase diffusion equation for real fields

In the case of real fields, the Swift-Hohenberg equation has microscopic energy functional

$$E = \int \left(((\nabla^2 + 1)w)^2 - \frac{1}{2} R w^2 + \frac{1}{4} w^4 \right) d\boldsymbol{x}. \tag{3.1}$$

In this case, the microscopic equation is

$$w_t = -\frac{\delta E}{\delta w} = -(\nabla^2 + 1)^2 w + Rw - w^3. \tag{3.2}$$

Note that equation (3.2) admits stationary solutions for striped patterns

$$w(\boldsymbol{x}, t) = w(\theta; \{A_n(k^2)\}, R) = \sum A_n(k^2) \cos(n\theta). \tag{3.3}$$

In the real field case, our modulation ansatz is given by (1.1),

$$w(\boldsymbol{x}, t) = w(\theta; \{A_n(k^2)\}, R; \nabla\theta = \boldsymbol{k}(\boldsymbol{X}, T) = \nabla_{\boldsymbol{X}} \Theta(\boldsymbol{X}, t)). \tag{3.4}$$

The procedure to derive the phase diffusion equation is the same as it was for the case of complex fields. However, the energy is not automatically averaged, making the procedure far more delicate. We simply state the key results.

The microscopic equations evolve according to

$$w_t \delta w = -\delta E. \tag{3.5}$$

Upon averaging over θ, one obtains

$$\langle w_\theta^2 \rangle \theta_t \delta\theta = -\delta \overline{E} \tag{3.6}$$

The averaged energy to leading order is given by

$$\overline{E_0} = \left(-\frac{1}{4} \langle w^4 \rangle + K \right) dxdy \tag{3.7}$$

Again, k_B^2 is defined so as to satisfy

$$\frac{d}{dk^2} \left(\frac{1}{4} \langle w^4 \rangle \right) = 0, \tag{3.8}$$

and defining K as we did for the complex case, obtain

$$\overline{E_0} = \int \left(\frac{1}{4} \langle w^4 \rangle \right)_{k^2}^{k_B^2} dxdy. \tag{3.9}$$

Considering just the leading order contribution of the energy, we can obtain the unregularized diffusion equation by taking the variaton of $\overline{E_0}$, and obtain

$$\langle w_\theta^2 \rangle \Theta_T + \nabla_{\boldsymbol{X}} \boldsymbol{k} B(k^2) = 0, \tag{3.10}$$

where

$$B(k^2) = 2 \frac{dE_0}{dk^2}, \tag{3.11}$$

where E_0 here is used to denote the integrand of $\overline{E_0}$. Note that in the complex case, we had $B(k^2) = \frac{dE_0}{dk^2}$.

The fact that the $\mathcal{O}(\epsilon)$ terms vanish in the averaged energy is not as clear in the case of the real field, but it still holds. The argument as to why is omitted for the time being.

Considering the $\mathcal{O}(\epsilon^2)$ terms, we argue in a similar fashion as we did for the case of complex fields. All terms involving $(\boldsymbol{k} \cdot \nabla_{\boldsymbol{X}})$ acting on the amplitudes yield terms proportional to $\frac{dA_n(k^2)}{dk^2}(\boldsymbol{k} \cdot \nabla_{\boldsymbol{X}} k^2)$ which can be written as $\boldsymbol{k} \cdot \nabla_{\boldsymbol{X}}(k^2 - k_B^2)$. Since $k^2 - k_B^2$ is small almost everywhere, there terms are negligible compared to the terms proportional to $\nabla \cdot \boldsymbol{k}$. Via integration by parts, one can obtain

$$\epsilon^2 \int \langle w_\theta^2 \rangle (\nabla \cdot \boldsymbol{k})^2 d\boldsymbol{x} \approx \epsilon^2 \langle w_\theta^2 \rangle_{k_B} \int (\nabla \cdot \boldsymbol{k})^2 d\boldsymbol{x}. \tag{3.12}$$

The regularized phase diffusion equation is then

$$\langle w_\theta^2 \rangle \Theta_T + \nabla \cdot \boldsymbol{k} B + \epsilon^2 \langle w_\theta^2 \rangle \nabla^4 \Theta = 0. \tag{3.13}$$

As we did for the complex field case, we can write the phase diffusion equation as a gradient flow, with the form

$$\eta \nabla^4 \theta + \nabla \cdot \boldsymbol{k} B = -\frac{\delta \overline{E}}{\delta \theta} \tag{3.14}$$

$$\overline{E} = \int \left(\frac{1}{2} \eta |\nabla^2 \theta|^2 + \frac{\alpha}{2} G^2 \right) d\boldsymbol{x} \tag{3.15}$$

$$G^2 = -\frac{1}{\alpha} \int_{k_B^2}^{k^2} B dk^2 \tag{3.16}$$

$$\alpha = -\frac{1}{4 k_B} |B'(k_B)|. \tag{3.17}$$

In this form, the first term measures the "bending" energy, whereas the second term measures the "stretching" energy.

As a quick demonstration of the fact that the wavenumber is almost constant throughout the pattern, and that the solution evolves so as to minimize an energy, figure 3.1 shows a solution of Swift-Hohenberg on the square alongside its energy density, and keep track of the total energy. We see that the energy tends to condense around bends within the pattern. We use random initial conditions for this solution.

Total energy corresponding to figure 3.1

```
total energy at time 15:  -33.888597248321645
total energy at time 20:  -52.26404570245001
total energy at time 25:  -77.2887255255116
total energy at time 30:  -109.74938099881015
total energy at time 199:  -338.9569147336093
total energy at time 200:  -338.9892785820569
```

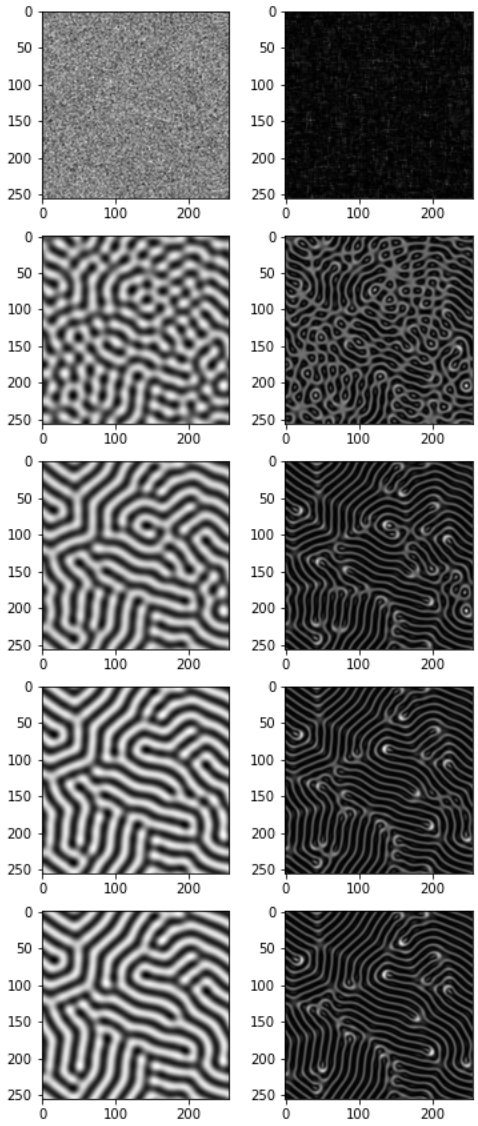

Figure 3.1: Swift-Hohenberg solution on the square, $R = .5$

4 Self-Dual Solutions and an Open Challenge

We are interested in stationary solutions of the RCN, namely solutions of

$$\eta\nabla^4\theta + \nabla\cdot\boldsymbol{k}B = -\frac{\delta\overline{E}}{\delta\theta} = 0. \tag{4.1}$$

Many stationary solutions can be deduced by balancing the strain and bending energies of $\overline{E}$. To name a few, the creation of VX pairs (figure 4.1), and concave and convex disclinations (figure 4.2).

There is a particular scenario that can be observed experimentally and in numerical simulations that is not yet deduced from the RCN. Numerically, this can be depicted by solving Swift-Hohenberg on an ellipse, using the Eikonal Solution as an initial condition. What one observes qualitatively is that the eikonal solution generates pointy edges where the stripes meet along the major axis of the ellipse. These points rapidly transition into a nipple instability, at which point a new state emerges. This state appears like a set of stair steps for stripes close to the major axis. An explanation for how this state emerges, in terms of the self-dual solution, is unclear. Figure 4.3 demonstrates solutions having this behavior, with their associated energy density.

Total energy corresponding to figure 4.3

```
total energy at time 15:  -167.19072805295662
total energy at time 20:  -261.4739829791277
total energy at time 25:  -391.7822674764031
total energy at time 30:  -560.5206723618704
total energy at time 199:  -1518.3264485682134
total energy at time 200:  -1518.3769630101526
```

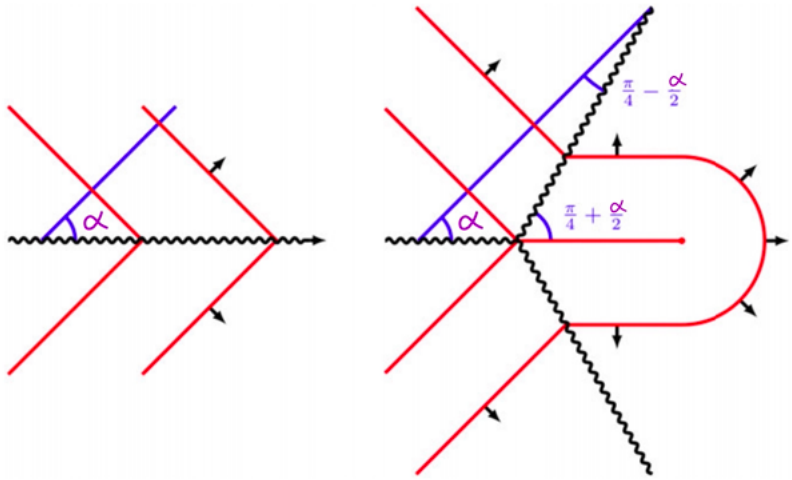

Figure 4.1: The nipple instability and birth of VX pair

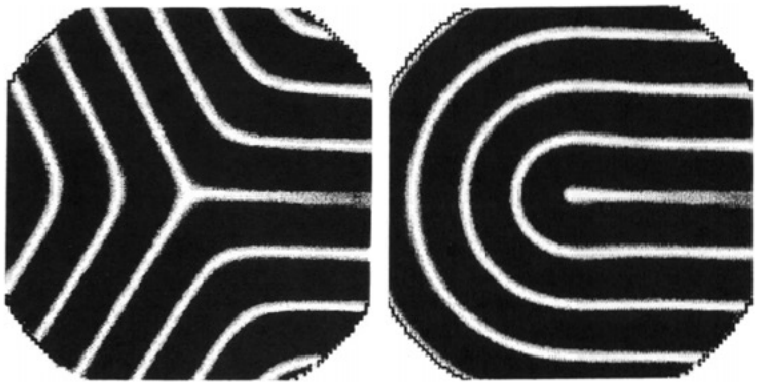

Figure 4.2: A concave and convex disclination

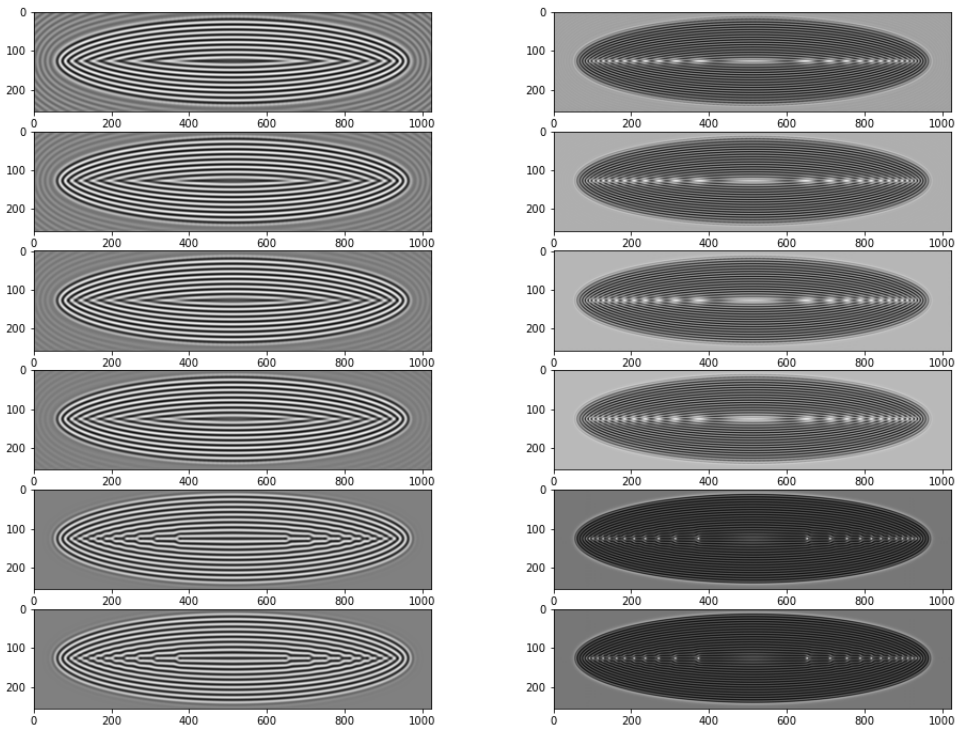

Figure 4.3: Swift Hohenberg solution on an ellipse, with eikonal initial condition

5 Machine Learning Experiments

5.1 SINDy-like methods for PDEs

Note that the problem at hand involves a microscopic model exhibiting behavior that is not yet deduced from the macroscopic model. This raises the possibility that the macroscopic model does not capture all aspects of the microscopic equations. Thus, one may be tempted to find an additional term to the macroscopic model. This idea motivates the study of existing methods for discovering governing equations from data.

Perhaps the most well known method is the Sparse Identification of Nonlinear Dynamics, or SINDy, introduced by Kutz et al for discovery of equations governing dynamical systems. Since the original publication, SINDy-like methods have been proposed to discover PDEs. As a preliminary first step, we combined ideas from Kutz and Schaefer to discover the Swift Hohenberg PDE from data consisting of solutions on the square.

Using a square grid of dimension 256×256 on the interval $[-25\pi, 25\pi]^2$, we integrate Swift-Hohenberg from $t = 0$ to $t = 100$ using time step $h = .5$. Thus, we obtain a dataset of dimension $256 \times 256 \times 201$. We used backward finite differences to compute 200 time derivatives $u_t(x_i, y_j, t_k)$. We then form various features out of the data $u(x_i, y_j, t_k)$. These features can be any term we suspect might be in the PDE, and primarily consists of polunomial functions, trigonometric functions, and spatial derivatives of u. We then form the matrix equation $u_t = \Theta(u)\xi$, where Θ is a matrix whose columns are the features constructed from u. The task then is to solve $u_t = \Theta(u)\xi$ where ξ is a sparse vector of coefficients. Figure 5.1 shows an example of the matrix equation one might form where u depends on only one spatial variable.

Once the time derivatives are obtained and feature matrix is constructed, the task is to approximate a solution of

$$\xi = \text{argmin}_{\hat{\xi}} \|\Theta(u)\hat{\xi} - u_t\|_2^2 + \lambda\|\hat{\xi}\|_0. \tag{5.1}$$

$$\mathbf{U}_t = \boldsymbol{\Theta}(\mathbf{U}, \mathbf{Q})\xi$$

$$\begin{bmatrix} u_t(x_0,t_0) \\ u_t(x_1,t_0) \\ u_t(x_2,t_0) \\ \vdots \\ u_t(x_{n-1},t_m) \\ u_t(x_n,t_m) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & u(x_0,t_0) & u_x(x_0,t_0) & \dots & u^5u_{xxx}(x_0,t_0) \\ 1 & u(x_1,t_0) & u_x(x_1,t_0) & \dots & u^5u_{xxx}(x_1,t_0) \\ 1 & u(x_2,t_0) & u_x(x_2,t_0) & \dots & u^5u_{xxx}(x_2,t_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u(x_{n-1},t_m) & u_x(x_{n-1},t_m) & \dots & u^5u_{xxx}(x_{n-1},t_m) \\ 1 & u(x_n,t_m) & u_x(x_n,t_m) & \dots & u^5u_{xxx}(x_n,t_m) \end{bmatrix}}_{} \begin{bmatrix} \\ \\ \\ \\ \\ \xi \end{bmatrix}$$

Figure 5.1: SINDy equation for PDEs with one spatial variable

Algorithm 1: STRidge($\Theta, \mathbf{U}_t, \lambda, tol, \text{iters}$)

$\hat{\xi} = arg\,min_{\xi} \|\Theta\xi - \mathbf{U}_t\|_2^2 + \lambda\|\xi\|_2^2$ # ridge regression

bigcoeffs = $\{j : |\hat{\xi}_j| \geq tol\}$ # select large coefficients

$\hat{\xi}[\sim \text{bigcoeffs}] = 0$ # apply hard threshold

$\hat{\xi}[\text{bigcoeffs}] = \text{STRidge}(\Theta[:, \text{bigcoeffs}], \mathbf{U}_t, tol, \text{iters} - 1)$ # recursive call with fewer coefficients

return $\hat{\xi}$

Figure 5.2: Algorithm 1: STRidge

In practice, what we actually solve is

$$\xi = \text{argmin}_{\hat{\xi}} \|\Theta(u)\hat{\xi} - u_t\|_2^2 + \lambda\|\hat{\xi}\|_1. \tag{5.2}$$

Following the approach of Kutz, we use Sequential Threshold Ridge Regression with a test/-train split, outlined in algorithm 1 and algorithm 2.

Following the approach of Schaefer, we use coarsened grids to reduce the data being passed to the optimization procedure, and use spectral methods to form the spatial derivatives being passed to our library. Note that all polynomials, spatial derivatives and time derivatives are reshaped into column vectors and stacked vertically. For this preliminary proof of concept, our library is constructed with the terms $1, u, u^2, u^3, u^4, u_x, u_y, u_x^2, u_y^2, u_x u_y, u_{xy}, u_{xx}, u_{yy}, \nabla^2 u$, and $\nabla^4 u$. Figure 5.4 displays the code used for the coarsening procedure and the formation of $\Theta(u)$.

Having formed u_t and $\Theta(u)$, we call the TrainSTRidge algorithm, and recover the Swift-Hohenberg PDE, as seen in figure 5.5.

5.2 Combining SINDy with an Autoencoder

Having "discovered" the form of the microscopic model from numerical simulations, a natural next step is to discover the macroscopic model. The use of autoencoder neural networks have been demonstrated to recover dynamics in terms of an alternative coordinate system. Consider for example, a video of a swinging pendulum. Kutz et al were able to combine an autoencoder with the SINDy framework to obtain the equation $\ddot{\theta} = \sin(\theta)$. However, we are not aware of examples where the recovered model is a PDE. The one example explored on PDE data was with a coupled reaction-diffusion system, from which the SINDy + Autoencoder method recovered a dynamical system. Thus, we are unaware of any examples of using an autoencoder with SINDy to recover a PDE.

The method trains a neural network to generate its own input, with a latent space (typically of reduced dimension) in the middle. Thus, we can break the autoencoder into two compo-

Algorithm 2: TrainSTRidge($\Theta, \mathbf{U}_t, \lambda, d_{tol}$, tol_iters, STR_iters)

First split the data into training and testing sets

$$\left.\begin{array}{l} \Theta \to [\Theta^{train}, \Theta^{test}] \\ \mathbf{U}_t \to [\mathbf{U}_t^{train}, \mathbf{U}_t^{test}] \end{array}\right\} \text{ 80/20 split}$$

Set an appropriate ℓ^0-penalty. The following worked well empirically

$\eta = 10^{-3}\kappa(\Theta)$

Get a baseline predictor

$\xi_{best} = \left(\Theta^{train}\right)^{-1} \mathbf{U}_t^{train}$

$\text{error}_{best} = \|\Theta^{test}\xi_{best} - \mathbf{U}_t^{test}\|_2^2 + \eta\|\xi_{best}\|_0$

Now search through values of tolerance to find the best predictor

$tol = d_{tol}$

for $iter = 1, \ldots,$ tol_iters:

    # Train and evaluate performance

    ξ =STRidge($\Theta^{train}, \mathbf{U}_t^{train}, \lambda, tol$, STR_iters)

    error= $\|\Theta^{test}\xi - \mathbf{U}_t^{test}\|_2^2 + \eta\|\xi\|_0$

    # Is the error still dropping?

    if error$\leq$ error$_{best}$:

        error$_{best}$ =error

        $\xi_{best} = \xi$

        $tol = tol + d_{tol}$

    # Or is tolerance too high?

    else:

        $tol = \max([0, tol - 2d_{tol}])$

        $d_{tol} = \frac{2d_{tol}}{\text{tol\_iters} - iter}$

        $tol = tol + d_{tol}$

return ξ_{best}

Figure 5.3: Algorithm 2: TrainSTRidge

```
num_t = 20
x_subsample = 8
y_subsample = 8
t_vals = np.arange(1,len(t),math.floor(len(t)/num_t))
x_vals = np.arange(0,nx,x_subsample)
y_vals = np.arange(0,ny,y_subsample)

num_points = num_t*len(x_vals)*len(y_vals)
print("feature vec length: ",num_points)
print(dt)

u = np.zeros((num_points,1))
u_t = np.zeros((num_points,1))
u_x = np.zeros((num_points,1))
u_y = np.zeros((num_points,1))
u_xx = np.zeros((num_points,1))
u_yy = np.zeros((num_points,1))
u_xy = np.zeros((num_points,1))
lapu = np.zeros((num_points,1))
biharmu = np.zeros((num_points,1))


# setting parameters for spectral derivatives
Lx = 2*x[len(x)-1] # Size of enclosing periodic rectangle
Ly = 2*y[len(y)-1]

i=0
for t_idx in t_vals:
    print(t_idx)
    uu = U[:,:,t_idx]
    uu_t = BackwardDiff(U[:,:,t_idx],U[:,:,t_idx-1],dt)
    uu_x = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'x')
    uu_y = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'y')
    uu_xx = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'xx')
    uu_yy = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'yy')
    uu_xy = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'xy')
    lapuu = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'laplacian')
    biharmuu = SpectralDerivs(U[:,:,t_idx],Lx,Ly,'biharmonic')
    for x_idx in x_vals:
        for y_idx in y_vals:
            u[i] = uu[x_idx,y_idx]
            u_t[i] = uu_t[x_idx,y_idx]
            u_x[i] = uu_x[x_idx,y_idx]
            u_y[i] = uu_y[x_idx,y_idx]
            u_xx[i] = uu_xx[x_idx,y_idx]
            u_yy[i] = uu_yy[x_idx,y_idx]
            u_xy[i] = uu_xy[x_idx,y_idx]
            lapu[i] = lapuu[x_idx,y_idx]
            biharmu[i] = biharmuu[x_idx,y_idx]
            i+=1

X = np.hstack([np.ones((num_points,1)), u, u**2, u**3,u**4,
                u_x, u_y, u_x**2, u_y**2, u_x*u_y,u_xy,
               u_xx, u_yy, lapu, biharmu])
description = ['','u','u^2','u^3','u^4',
               'u_{x}','u_{y}','u_{x}^2','u_{y}^2','u_{x}u_{y}','u_{xy}'
    ,'u_{xx}', 'u_{yy}','lapu','biharmu']
['1'] + description[1:]
```

Figure 5.4: Code for coarsening and SINDy library

```
c = TrainSTRidge(X,u_t,10**-5,1)
print_pde(c, description)
```

```
u_t = (-0.507735 +0.000000i)u
    + (-1.016235 +0.000000i)u^3
    + (97.489916 +0.000000i)u_{xx}
    + (97.490143 +0.000000i)u_{yy}
    + (-99.522287 +0.000000i)lapu
    + (-1.016119 +0.000000i)biharmu
```

Figure 5.5: Result of optimization procedure

```
latent_dim = 1024

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(256, activation='linear'),
      layers.Dense(512, activation='sigmoid'),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(2048, activation='linear'),
      layers.Dense(4096, activation='sigmoid'),
      layers.Dense(8192, activation='relu'),
      layers.Dense(16384, activation='linear'),
      layers.Reshape((128, 128))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
autoencoder.fit(u_train, u_train,
                epochs=10,
                shuffle=True,
                validation_data=(u_test, u_test))
```

Figure 5.6: Dense Autoencoder Code

nents, the encoder and the decoder. If x is our input data, z the latent space representation of x, and $\hat{x}$ the output, the autoencoder can be represented as

$$\hat{x} = \psi(z) = \psi(\varphi(x)). \tag{5.3}$$

In this case φ is the encoder and ψ is the decoder. The method consists of minimizing the autoencoder error, coupled with minimizing the SINDy error applied to the latent space representation. The loss function for a single example is then

$$L(x) = \|x - \psi(z)\|_2^2 + \lambda_1 \|(\nabla_x z)\dot{x} - \Theta(z)\xi\|_2^2 + \lambda_2 \|\xi\|_1. \tag{5.4}$$

Despite a lack of working examples, we set out to explore this concept, in the hopes of recovering the RCN from Swift-Hohenberg data. While one could hypothetically generate phase surfaces, and apply the standard SINDy procedure to those surfaces, it is a more interesting challenge to discover the model without asserting knowledge of the expected coordinate transformation. The first step is to obtain a working autoencoder, which we achieve using both a standard feed forward network and a convolutional network. We use data of dimension $1001 \times 128 \times 128$, and select our latent space to be of dimension 32. Figure 6.1 displays code for the network of Dense layers, while figure 6.2 displays code for the convolutional neural network. Figures 6.3 and 6.4 display the corresponding model summaries. We run the autoencoders on 10 random samples from data that was not touched during training, and visualize the latent space, along with the input and its autoencoder approximation. Those comparisons are seen in figure 6.5 for the dense encoder, and 6.6 for the convolutional encoder.

There is still much work remaining to integrate the autoencoder with the SINDy optimization procedure. The first concern is how to best integrate the SINDy procedure into the training.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 16384)             0

 dense_26 (Dense)            (None, 256)               4194560

 dense_27 (Dense)            (None, 512)               131584

 dense_28 (Dense)            (None, 1024)              525312

=================================================================
Total params: 4,851,456
Trainable params: 4,851,456
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_29 (Dense)            (None, 2048)              2099200

 dense_30 (Dense)            (None, 4096)              8392704

 dense_31 (Dense)            (None, 8192)              33562624

 dense_32 (Dense)            (None, 16384)             134234112

 reshape_4 (Reshape)         (None, 128, 128)          0

=================================================================
Total params: 178,288,640
Trainable params: 178,288,640
Non-trainable params: 0
_________________________________________________________________
Model: "autoencoder_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_8 (Sequential)   (None, 1024)              4851456

 sequential_9 (Sequential)   (None, 128, 128)          178288640

=================================================================
Total params: 183,140,096
Trainable params: 183,140,096
Non-trainable params: 0
```

Figure 5.7: Dense Autoencoder Summary

```
class ConvAutoencoder(Model):
  def __init__(self):
    super(ConvAutoencoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Input(shape=(128, 128,1)),
      layers.Conv2D(16, (3, 3), activation='linear', padding='same', strides=2),
      layers.Conv2D(8, (3, 3), activation='sigmoid', padding='same', strides=2),
      layers.Conv2D(1, (3, 3), activation='relu', padding='same')])

    self.decoder = tf.keras.Sequential([
      layers.Conv2DTranspose(1, (3, 3), activation='linear', padding='same'),
      layers.Conv2DTranspose(8, kernel_size=3, strides=2, activation='sigmoid', padding='same'),
      layers.Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Dense(100,activation='relu'),
      layers.Conv2D(1, kernel_size=(3, 3), activation='linear', padding='same')])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

conv_autoencoder = ConvAutoencoder()
```

```
conv_autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
conv_autoencoder.fit(conv_u_train, conv_u_train,
                epochs=10,
                shuffle=True,
                validation_data=(conv_u_test, conv_u_test))
```

Figure 5.8: Convolutional Autoencoder Code

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)           (None, 64, 64, 16)        160

 conv2d_12 (Conv2D)           (None, 32, 32, 8)         1160

 conv2d_13 (Conv2D)           (None, 32, 32, 1)         73

=================================================================
Total params: 1,393
Trainable params: 1,393
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_transpose_7 (Conv2DT  (None, 32, 32, 1)         10
 ranspose)

 conv2d_transpose_8 (Conv2DT  (None, 64, 64, 8)         80
 ranspose)

 conv2d_transpose_9 (Conv2DT  (None, 128, 128, 16)      1168
 ranspose)

 dense_33 (Dense)             (None, 128, 128, 100)     1700

 conv2d_14 (Conv2D)           (None, 128, 128, 1)       901

=================================================================
Total params: 3,859
Trainable params: 3,859
Non-trainable params: 0
_________________________________________________________________
Model: "conv_autoencoder_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 sequential_15 (Sequential)   (None, 32, 32, 1)         1393

 sequential_16 (Sequential)   (None, 128, 128, 1)       3859

=================================================================
Total params: 5,252
Trainable params: 5,252
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.9: Convolutional Autoencoder Summary

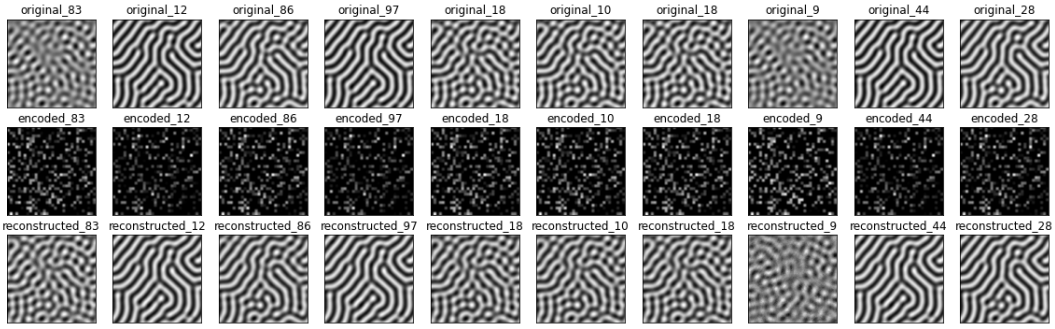

Figure 5.10: Dense Autoencoder Latent Space

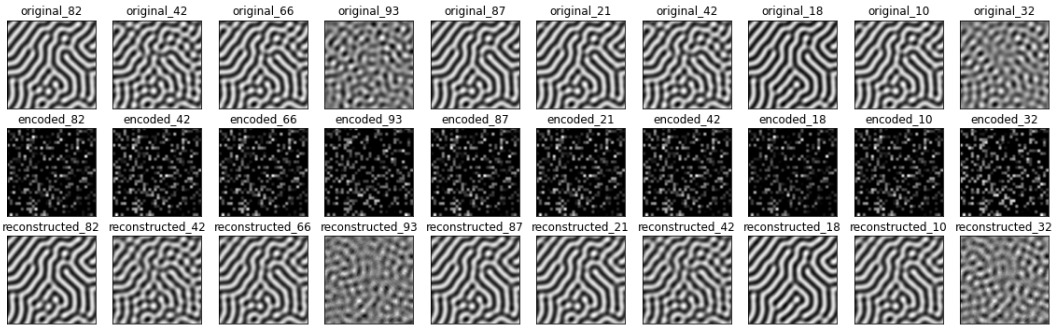

Figure 5.11: Convolutional Autoencoder Latent Space

We will take as our first approach a training procedure that uses batches of 10. After the reconstruction loss is computed, we will perform the SINDy optimization on the latent space for the 10 samples, and combine that error with the autoencoder error. A key uncertainty here is how to best estimate time derivatives of the latent space variable, as well as how to compute its spatial derivatives.

6 Conclusion

We conclude this report by listing the key tasks moving forward.

1. We need to perform a careful analysis of the derivation of the RCN for real fields. Much of the reasoning used for real fields is more delicate than for complex fields.

2. We need to study the self-dual stationary solutions, and the role the Jacobian of the map from (x, y) to $\boldsymbol{k} = (\theta_x, \theta_y)$ plays in the analysis.

3. The autoencoder framework is completely uncoupled from the SINDy optimization procedure, and therefore we must merge them together.

4. Discovering a model through SINDy like methods may not be the way to resolve this problem. However, I believe there are interesting machine learning experiments that can be performed to gain insight. For instance, autoencoders may be used to perform anomaly detection, which may be relevant if training an autoencoder on data prior to the formation of the stair-step pattern. We could also compare the performance of a pair of autoencoders on the elliptical solutions, where one autoencoder uses one latent variable, and the other uses two latent variables. Such a comparison may be used to suggest the emergence (or lack thereof) of another order parameter. Finally, reconstructing high resolution solutions of the microscopic model from coarse samples of the macroscopic model can be useful in determining the relevant order parameters.