Machine Learning Experiment

Edward McDugald

October 11, 2022

1 Introduction

Convection rolls are known for their pattern forming properties. Experimental results demonstrate a range of patterns, which appear to be well-modeled by the Swift-Hohenberg PDE.

$$w_t = -(1 + \nabla^2)^2 w + Rw - w^3.$$

Work by Cross and Newell has sought to find generic models for pattern forming systems meeting certain conditions. The most prevalent model in the literature appears to be essentially the minimizers of the regularized Cross-Newell energy functional

$$\varepsilon^{(\mu)}(\Theta) = \mu \int_\Omega (\nabla_{\vec{X}} \Theta)^2 d\vec{X} + \frac{1}{\mu} \int_\Omega \left(1 - |\nabla_{\vec{X}} \Theta|^2\right)^2 d\vec{X}.$$

The Cross-Newell framework does not capture all possible patterns and pattern defects seen in experiments, nor seen in SH simulations. The problem at hand is to find a Cross-Newell like equation that predicts a wider class of patterns than the current Cross-Newell equations.
The current document outlines machine learning models to a range of related problems. Typically, machine learning experiments on PDE data either seek reduced order models as output, or seek to generate more efficient simulation algorithms. The range of machine learning experiments that will be explored here is not yet determined. Some of the main papers of interest here are the following:
OG Kutz paper on SINDy- applicable to Dynamical Systems.
Kutz paper on PDE-FIND
Git for PDE-FIND
PySINDy
Describes SINDy like approach for PDE data
Published paper on LaSDI- Bill's work
gLaSDI preprint - Bill's work
Reduced Order model via 2 point correlations (2017) Materials Science
Materials Science CNN
Learning PDEs via data discovery and sparse optimization

2 Original Work on SINDy

2.1 Some Context

- The goal is to extract parsimonious models from data. Can be called the dynamical system discovery problem.
- The framework is made possible by work from Bongard and Lipson (Automated reverse engineering of nonlinear dynamical systems), and Schmidt and Lipson (Distilling free-form natural laws from experimental data). The method uses *symbolic regression* to find nonlienar differential equations, balancing complexity with accuracy.
- Symbolic regression is expensive, does not scale well to large systems, and can be prone to overfitting.
- A good review of methods for the dynamical system discovery problem can be found here: Automated adaptive inference of phenomenological dynamical models.

2.2 Sparse Identification of Nonlinear Dynamics (SINDy)

Note, a useful appendix is found here: Appendix

- Views dynamical system discovery from persepctive of sparse regression and compressed sensing.
- Assumes most physical systems have only a few relevant terms, making governing equations sparse in a high-dimensoonal nonlinear function space.
- Consider dynamical systems of the form
$$\frac{d}{dt}\boldsymbol{x}(t) = f(\boldsymbol{x}(t))$$
- To determine the function $\boldsymbol{f}$ from data, we collect a time history of the state $\boldsymbol{x}(t)$ and either measure the derivative $\dot{\boldsymbol{x}}(t)$ or approximate it numerically from $\boldsymbol{x}(t)$. The data are sampled at $t_1, t_2, \ldots, t_m$ and arranged into two matrices:
$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^T(t_1) \\ \boldsymbol{x}^T(t_2) \\ \vdots \\ \boldsymbol{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \ldots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \ldots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \ldots & x_n(t_m) \end{bmatrix},$$
and
$$\dot{\boldsymbol{X}} = \begin{bmatrix} \dot{\boldsymbol{x}}^T(t_1) \\ \dot{\boldsymbol{x}}^T(t_2) \\ \vdots \\ \dot{\boldsymbol{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \ldots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \ldots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \ldots & \dot{x}_n(t_m) \end{bmatrix}.$$

- Then we construct a library $\Theta(\boldsymbol{X})$ consisting of candidate nonlinear functions of the columns of $\boldsymbol{X}$. For example, $\Theta(\boldsymbol{X})$ may consist of constant, polynomical, and trigonometric terms:
$$\Theta(\boldsymbol{X}) = \left[\mathbf{1}\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{P_2}}\boldsymbol{X}^{\boldsymbol{P_3}}\dots\sin(\boldsymbol{X})\cos(\boldsymbol{X})\dots\right].$$
- Each column of $\Theta(\boldsymbol{X})$ represents a candidate functions for the right-hand side.
- There is tremendous freedom in choosing the entries in this matrix of nonlinearities. Since we assume only a few of them are active in each row of f, we set up a sparse regression problem to determine the sparse vectors of coefficients
$$\boldsymbol{\Xi} = [\boldsymbol{\xi}_1\boldsymbol{\xi}_2\dots\boldsymbol{\xi}_n]$$
that determine which nonlinearities are active:
$$\dot{\boldsymbol{X}} = \Theta(\boldsymbol{X})\boldsymbol{\Xi}.$$

3 SINDy + Coordinate Discovery - Includes PDE Example

The main paper is Data driven discovery of coordinates and governing equations
The code for this paper is here: github Also, there is a "supporting work" pdf available as well.
This work breaks down the "data discovery process" in a few steps:

- Measure data
- Approximate derivatives from data. If data is noisy, there is a method to handle that
- Construct library of functions.
- Jointly solve a sparse regression problem for model discovery, and use an autoeconder neural network for coordiante discover.
- I don't what this means yet, but will figure it out...

3.1 10/4 - replicating results for Reaction-Diffusion equation

- I am going to try replicating result for the PDE presented.
- Step 1 is making or obtaining numerical solver for the PDE.

3.1.1 Reaction Diffusion Data Acquisition/Simulation

We wish to get data from the PDE

$$u_t = \left(1 - (u^2 + v^2)\right) u + \beta(u^2 + v^2)v + d_1(u_{xx} + u_{yy})$$
$$v_t = -\beta(u^2 + v^2)u + \left(1 - (u^2 + v^2)\right) v + d_2(v_{xx} + v_{yy}),$$

with $d_1, d_2 = .1$ and $\beta = 1$. We will test the method on snapshots of the surface $u(x, y, t)$. We will use a single initial condition,

$$u(y_1, y_2, 0) = \tanh\left(\sqrt{y_1^2 + y_2^2}\cos\left(\arg(y_1 + iy_2) - \sqrt{y_1^2 + y_2^2}\right)\right)$$
$$v(y_1, y_2, 0) = \tanh\left(\sqrt{y_1^2 + y_2^2}\sin\left(\arg(y_1 + iy_2) - \sqrt{y_1^2 + y_2^2}\right)\right).$$

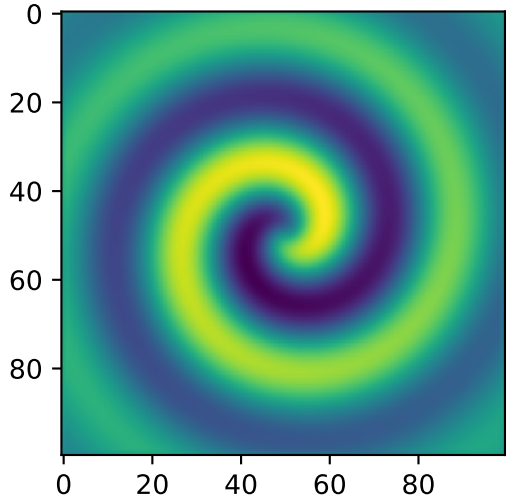
We use $t = 0$ to $t = 500$ with a spacing of $\Delta t = .05$. We use basically turn the PDE into a system of ODE's by taking fourier transform of RHS. We integrate the RHS, and then take the inverse transform.

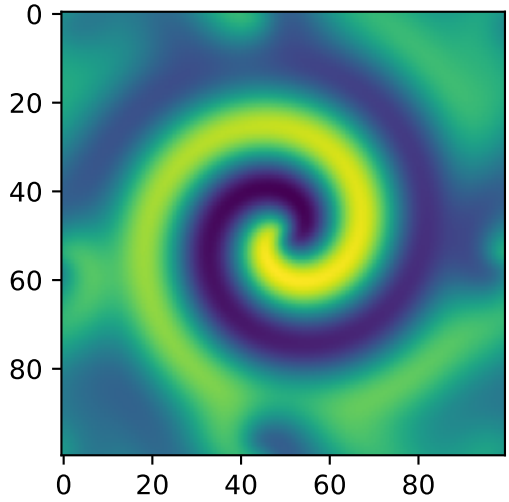
```
import numpy as np
import scipy as sp
from scipy.integrate import solve_ivp
from scipy.io import savemat

t = np.arange(0,500.05,.05)
d1=0.1
d2=0.1
beta=1.0
L=20
n=100
N=n**2
x2 = np.linspace(-L/2,L/2,n+1)
x = x2[0:len(x2)-1]
y=x
kx = (2.*np.pi/L)*sp.fft.fftfreq(n,1./n)
ky=kx
X, Y = np.meshgrid(x,y)
KX, KY = np.meshgrid(kx,ky)
K2 = KX**2+KY**2
K22 = np.reshape(K2,(N,1))
m=1.0
f=np.exp(-.01*(X**2+Y**2))


def reaction_diffusion_rhs(t_dummy,uvt):
```

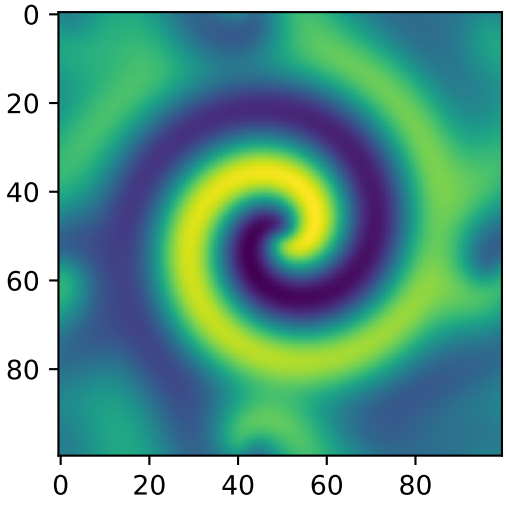
```
    uvt = np.reshape(uvt,(2*N,1))
    ut = np.reshape(uvt[0:N],(n,n))
    vt = np.reshape(uvt[N:2*N],(n,n))
    u = np.real(sp.fft.ifft2(ut.T))
    v = np.real(sp.fft.ifft2(vt.T))
    u3 = u**3
    v3 = v**3
    u2v = (u**2)*v
    uv2 = u*(v**2)
    utrhs = np.reshape(sp.fft.fft2((u-u3-uv2+beta*u2v+beta*v3).T),(N,1))
    vtrhs = np.reshape(sp.fft.fft2((v-u2v-v3-beta*u3-beta*uv2).T),(N,1))
    c1 = -d1*K22*uvt[0:N]+utrhs
    c2 = -d2*K22*uvt[N:2*N]+vtrhs
    return np.array([[c1,c2]]).reshape((2*N,1))


u = np.zeros(shape=(len(x),len(y),len(t)))
v = np.zeros(shape=(len(x),len(y),len(t)))
uf = np.zeros(shape=(len(x),len(y),len(t)))
vf = np.zeros(shape=(len(x),len(y),len(t)))

du = np.zeros(shape=(len(x),len(y),len(t)))
dv = np.zeros(shape=(len(x),len(y),len(t)))
duf = np.zeros(shape=(len(x),len(y),len(t)))
dvf = np.zeros(shape=(len(x),len(y),len(t)))

u[:,:,0]=np.tanh(np.sqrt(X**2+Y**2))*np.cos(m*np.angle(X+Y*1j)-np.sqrt(X**2+Y**2))
v[:,:,0]=np.tanh(np.sqrt(X**2+Y**2))*np.sin(m*np.angle(X+Y*1j)-np.sqrt(X**2+Y**2))
uf[:,:,0]=f*u[:,:,0]
vf[:,:,0]=f*v[:,:,0]

uvt = np.hstack((np.reshape(sp.fft.fft2(u[:,:,0].T),(1,N)),np.reshape(sp.fft.fft2(v[:,:,
uvt_rhs = reaction_diffusion_rhs(t[0],uvt)
du[:,:,0]=np.real(sp.fft.ifft2(np.reshape(uvt_rhs[0:N].T,(n,n))))
dv[:,:,0]=np.real(sp.fft.ifft2(np.reshape(uvt_rhs[N:2*N].T,(n,n))))

uvsol = np.zeros(shape=(N+1,2*N),dtype='complex')
uvsol[0,:]=uvt.flatten()
for i in range(1,len(t)):
    if i%1000==0:
        print(i)
    tspan = [t[i-1],t[i]]
    y0 = uvt.flatten()
    sol = solve_ivp(reaction_diffusion_rhs, tspan, y0,vectorized=True)
    uvsol[i, :] = sol.y[:,len(sol.t)-1]
```

```
    uvt = uvsol[i,:]

for j in range(len(t)-1):
    ut = np.reshape(uvsol[j,0:N].T,(n,n))
    vt = np.reshape(uvsol[j,N:2*N].T,(n,n))
    u[:,:,j+1]=np.real(sp.fft.ifft2(ut))
    v[:,:,j+1]=np.real(sp.fft.ifft2(vt))

    uvt_rhs = reaction_diffusion_rhs(t[j+1],uvsol[j,0:2*N].T)
    du[:,:,j+1] = np.real(sp.fft.ifft2(np.reshape(uvt_rhs[0:N].T,(n,n))))
    dv[:,:,j+1] = np.real(sp.fft.ifft2(np.reshape(uvt_rhs[N:2*N].T,(n,n))))

    uf[:,:,j+1] = f*u[:,:,j+1]
    vf[:,:,j+1] = f*v[:,:,j+1]
    duf[:,:,j+1] = f*du[:,:,j+1]
    dvf[:,:,j+1] = f*dv[:,:,j+1]

t = t[2:N+1]
uf = uf[:,:,2:N+1]
vf = vf[:,:,2:N+1]
duf = duf[:,:,2:N+1]
dvf = dvf[:,:,2:N+1]

mdict={"t":t,"x":x,"y":y,"uf":uf,"vf":vf,"duf":duf,"dvf":dvf}
savemat("/Users/edwardmcdugald/Research/
convection_patterns/code/data/rd1.mat",mdict)
```

Some samples of the output: (u for time step 0, 3000, 9000).


0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80


0
20
40
60
80
0
20
40
60
80

```
import scipy.io as sio
import numpy as np
data = sio.loadmat("code/data/rd1.mat")
import matplotlib.pyplot as plt
fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax1.imshow(data['uf'][:,:,0])
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/RD_10_10_01.pdf")

fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax1.imshow(data['uf'][:,:,3000])
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/RD_10_10_02.pdf")

fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax1.imshow(data['uf'][:,:,9000])
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/RD_10_10_03.pdf")
```

3.1.2 Reaction Diffusion- SINDy + AutoEncoder Results

Having generated test data, we follow along the example from Champion 2019. Data-driven discovery of coordinates and governing equation. The github for this work is here: Champion SINDy + AutoEncoder Git. This may be harder than thought. The original code uses a deprecated version of tensorflow. Have tried running a script to update the code for version 2... will see if i can modify it and make it work.

3.1.3 Reaction Diffusion- PDE-Find Results

We test the results now on PDE-FIND. Original Kutz paper here: Data-driven discovery of partial differential equations. The github for this work is here: PDE-FIND. This seems to only contain the sparse regression idea. May need to use SINDyPy afterall...

3.1.4 Reaction Diffusion- SINDyPy Results (Optional)

There is an open source package that is well maintained for these types of experiments. SINDyPy See this! PySINDy examples