Swift-Hohenberg Numerics

Edward McDugald

September 20, 2022

1 Literature

The following resource has a good discussion of the dynamics in 1d case: 1D Swift Hohenberg
This has an interesting, fancy sounding numerical method Reproducing Kernel Hilbert Space Method
1D Method Study of Solution of Swift Hohenberg
Pattern Selection of solutions of SH
SH
One More
Good Exposition for 1d
Another promising one
More- lol

2 Getting Started

The Swift-Hohenberg equation reads:

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

We can expand this out, to read:

$$\begin{aligned} w_t &= -(1 + \nabla^2)^2 w + Rw - w^3 \\ &= -(1 + 2\nabla^2 + \nabla^4)w + Rw - w^3 \\ &= -w - 2\nabla^2 w - \nabla^4 w + Rw - w^3 \\ &= (R-1)w - 2\nabla^2 w - \nabla^4 w - w^3 \\ &= -2\nabla^2 w - \nabla^4 w + (R-1)w - w^3. \end{aligned}$$

Here is some matlab code I found for a 2d (?) solution

```
function Phi=SwiHoEuler(Phi, nSteps)
epsi=0.25;
dt=0.1;

[nR nC]=size(Phi);
if mod(nR, 2)==0
    kR=[0:nR/2-1 -nR/2:-1]*2*pi/nR;
else
    kR=[0:nR/2 -floor(nR/2):-1]*2*pi/nR;
end
Ky=repmat(kR.', 1, nC);

if mod(nC, 2)==0
    kC=[0:nC/2-1 -nC/2:-1]*2*pi/nC;
else
    kC=[0:nC/2 -floor(nC/2):-1]*2*pi/nC;
end
Kx=repmat(kC, nR, 1); % frequencies
K2=Kx.^2+Ky.^2; % used for Laplacian in Fourier space
D0=1.0./(1.0-dt*(epsi-1.0+2.0*K2-K2.*K2)); % linear factors combined

PhiF=fft2(Phi);

for n=0:nSteps
    NPhiF=fft2(Phi.^3); % nonlinear term, evaluated in real space
    if mod(n, 100)==0
        fprintf('n = %i\n', n);
    end
    PhiF=(PhiF - dt*NPhiF).*D0; % update

    Phi=ifft2(PhiF); % inverse transform
end
return
```

I am going to compare results of this code with chebfun simulations. Note that these simulations won't exatly match the specs listed in the following Global description of patterns far from onset- a case study. In particular, I am not going to use the same initial condition, ie, the cosine of the solution of the eikonal equation

$$|\nabla v|^2 - 1 = 0.$$

I am also not concerned about multiplying by a constant to enforce $k = 1$. I will digest that material soon. For now, I am just trying to compare results of a numerical solver of my own implementation in python, with the results obtained by using the open source solver, "chebfun".

2.1 Chebfun Simulation

I am going to run a simulation with $R = .5$, on a square of width 20π, using 256^2 grid points. I will take as an initial condition the surface $cos(x) + sin(y)$. We use 200 time steps, with $dt = .1$. With $R = .5$, SH reads

$$w_t = -2\nabla^2 w - \nabla^4 w + -.5w - w^3.$$

We consider the linear term to be $-2\nabla^2 w - \nabla^4 w$, and the nonlinear term to be $-.5w - w^3$. Using chebfun, the code generate the simulation is

```
dom = [0 20*pi 0 20*pi];
tspan = [0 200];
S = spinop2(dom,tspan);
S.lin = @(w) -2*lap(w)-biharm(w);
S.nonlin = @(w) -.5*w-w.^3;
S.init = .1*chebfun2(@(x,y) cos(x)+sin(y),dom,'trig');
w = spin2(S,256,1e-1,'plot','off');
plot(w), view(0,90), axis equal, axis off
saveas(gcf,'/Users/edwardmcdugald/Research/convection_patterns_matlab/figs/cf1.png');
```

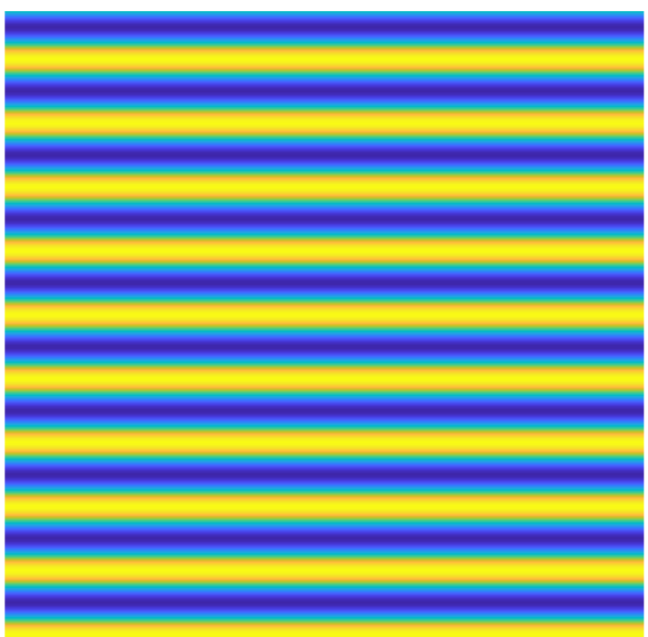

Figure 1: Chebfun Simulation: $[0, 20\pi]^2$, $t \in [0, 20]$, $w_0 = \cos(x) + \sin(y)$, $R = .5$

2.2 Modifying Matlab Code to match Chebfun

In the MATLAB code as found online, the linear and nonlinear terms are grouped as follows. Linear term $(R-1)w - 2\nabla^2 w - \nabla^4 w$, and nonlinear term $-w^3$. Consider the equations

$$\begin{aligned} w_t &= \left((R-1) - 2\nabla^2 - \nabla^4\right) w \\ w_y &= -w^3 \end{aligned}$$

We can solve these separately? Solving the first one, we have something like

$$\begin{aligned} \frac{w(t+\Delta t) - w(t)}{\Delta t} &= \left((R-1) - 2\nabla^2 - \nabla^4\right) w(t) \\ \implies w(t+\Delta t) &= \Delta t \left((R-1) - 2\nabla^2 - \nabla^4\right) w(t) + w(t) \end{aligned}$$

2.3 Ok, the above isnt working cleanly, will try something new

Potentially useful info: Large Time-Stepping Method for SH Ok, so I'm going to split the equation into linear and nonlinear parts

$$\begin{aligned} w_t &= \left(-2\nabla^2 - \nabla^4\right) w \\ w_t &= (R-1)w - w^3. \end{aligned}$$

Solving the first can be done with a fourier transform. Consider the approximation

$$\begin{aligned} \frac{w(t+\Delta t) - w(t)}{\Delta t} &= \left(-2\nabla^2 - \nabla^4\right) \\ \implies w(t+\Delta t) &= \Delta t \left(-2\nabla^2 - \nabla^4\right) w(t) + w(t) \end{aligned}$$

At each time step t, we have the value $w(t)$. We wish to add to it the quantity

$$\Delta t \left(-2\nabla^2 - \nabla^4\right) w(t)$$

To do so, we need to compute $\left(-2\nabla^2 - \nabla^4\right) w(t)$. We have that

$$\left(-2\nabla^2 - \nabla^4\right) w(t) = \text{ifft}((-2k_2 - k_2^2)\text{fft}(w)),$$

where k_2 is a laplacian operator in Fourier space. Thus, we obtain

$$w(t+\Delta t) = \Delta t \text{ifft}((-2k_2 - k_2^2)\text{fft}(w)) + w(t)$$

2.4 Naive Approach

Ok, I have the PDE

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

Discretizing this, I obtain

$$w^{k+1} = \Delta t \left[-(2\nabla^2 + \nabla^4)w^k + (R-1)w^k - (w^k)^3 \right] + w^k.$$

Since we provide an initial condition, this should be a feasible approach to evolve the system (since RHS is in terms of $(k+1)$ exclusively). The terms $(R-1)w^k - (w^k)^3$ and w^k are straightforward to compute. To compute $(2\nabla^2 + \nabla^4)w$, we create a matrix that acts as the laplacian operator in Fourier space, call it K_Δ. Then we have

$$(2\nabla^2 + \nabla^4)w = \text{ifft}((2K_\Delta + K_\Delta^2)\text{fft}(w))$$

The matlab code for this approach is as follows:

```
function w=mySH(w, R, dt, nSteps)

[nR nC]=size(w);
if mod(nR, 2)==0
    kR=[0:nR/2-1 -nR/2:-1]*2*pi/nR;
else
    kR=[0:nR/2 -floor(nR/2):-1]*2*pi/nR;
end
Ky=repmat(kR.', 1, nC);

if mod(nC, 2)==0
    kC=[0:nC/2-1 -nC/2:-1]*2*pi/nC;
else
    kC=[0:nC/2 -floor(nC/2):-1]*2*pi/nC;
end
Kx=repmat(kC, nR, 1); % frequencies
K_Delta=Kx.^2+Ky.^2; % Fourier Laplacian
FourOp = 2*K_Delta+K_Delta.*K_Delta; % Laplacian + Biharmonic

for n=0:nSteps
    linTerm = -ifft2(FourOp.*fft2(w));
    nonLinTerm = (R-1).*w - w.^3;
    w = dt*(linTerm+nonLinTerm)+w;
end
return
```

3 Incorporating Ideas from Meeting with Shankar

Potentially relevant paperolving Linear PDE by Exponential Splitting
Another One Fourth Order time stepping for stiff PDEs

- I believe the following approach falls under the name of "Operator Splitting".
- We have the PDE

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

- We can break this into a linear part and a non-linear part,

$$W_t = L(w) + NL(w),$$

with

$$\begin{aligned} L(w) &= (-(1 + \nabla^2)^2 + R)w \\ NL(w) &= -w^3. \end{aligned}$$

Rearranging the linear part, we have

$$\begin{aligned} L(w) &= (-\nabla^4 - 2\nabla^2 + R - 1)w \\ NL(w) &= -w^3. \end{aligned}$$

- Basic procedure is as follows:
  (i) Let $A = (-\nabla^4 - 2\nabla^2 + R - 1)$. Consider the pair of PDEs

  $$\begin{aligned} w_t &= Aw \\ w_t &= -w^3. \end{aligned}$$

  (ii) Handling the linear and nonlinear PDEs separately, we get the relations

  $$\begin{aligned} & w(t + \Delta t) = \Delta t A w(t) + w(t) \\ \implies & w(t + \Delta t) \approx e^{A\Delta t} w(t). \end{aligned}$$

  And for the nonlinear part,

  $$w(t + \Delta t) \approx -\Delta t w(t)^3 + w(t).$$

  .

  (iii) Let's say we have the array $w(t)$ at time t. We wish to evolve to $w(t + \Delta t)$.
  - First, we evolve the linear part, for a time step of $\frac{\Delta t}{2}$.

  $$w(t + \Delta t)_1 = e^{A\frac{\Delta t}{2}} w(t).$$

  - Then we evolve the result according to the nonlinear part, for a time step of Δt.

  $$w(t + \Delta t)_2 = w(t + \Delta t)_1 - \Delta t w(t + \Delta t)_1^3.$$

  - Then, we evolve the result according to the linear part, for another time step of $\frac{\Delta t}{2}$,

  $$w(t + \Delta t) = e^{A\frac{\Delta t}{2}} w(t + \Delta t)_2.$$

– Collapsing the notation, we have

$$w(t+\Delta t) = e^{A\frac{\Delta t}{2}}\left[e^{A\frac{\Delta t}{2}}w(t) - \Delta t\left(e^{A\frac{\Delta t}{2}}w(t)\right)^3\right].$$

– The best way to handle the linear evolution, $e^{A\frac{\Delta t}{2}}w(t)$, is to do the following:

$$w(t+\Delta t) = \text{ifft}\left(e^{A\frac{\Delta t}{2}}\text{fft}(w(t))\right),$$

where $A = (-\nabla^4 - 2\nabla^2 + R - 1)$.

– It is easy to verify that (depending on Fourier Transform definition), we have

$$\hat{\nabla^2 w} = w_{xx} \hat{+} w_{yy} = (ik_x)^2\hat{w} + (ik_y)^2\hat{w} = -(k_x^2+k_y^2)\hat{w}.$$

And

$$\begin{aligned}\hat{\nabla^2 w} &= w_{xxxx} + w_{yyxx} \hat{+} w_{xxyy} + w_{yyyy}\\ &= \left((ik_x)^4 + (ik_y)^2(ik_x)^2 + (ik_x)^2(ik_y)^2 + (ik_y)^4\right)\hat{w} = (k_x^4 + 2k_x^2k_y^2 + k_y^4)\hat{w}.\end{aligned}$$

– Note that $(k_x^2+k_y^2)^2 = (k_x^4 + 2k_x^2k_y^2 + k_y^4)$

– Let us define the Fourier Laplacian Matrix, M_{∇^2}

$$M_{\nabla^2} = k_x^2 + k_y^2.$$

Then, the Fourier Biharmonic Matrix, M_{∇^4} is given by

$$M_{\nabla^4} = M_{\nabla^2}\odot M_{\nabla^2},$$

where $\odot$ denotes element wise multiplication. Thus, we can write our matrix A as

$$A = -(M_{\nabla^2}\odot M_{\nabla^2}) - 2M_{\nabla^2} + (R-1)$$

The resulting code is as follows:

```
function w=mySH2(w, R, dt, nSteps, L)

[ny, nx] = size(w); %recall: number of columns in grid is number of x-coordinates!

%frequency matrix in x direction
if mod(nx, 2)==0
    kx=[0:nx/2-1 -nx/2:-1]*2*pi/L;
else
    kx=[0:nx/2 -floor(nx/2):-1]*2*pi/L;
end
%in python, kx = (2.*np.pi/(x[len(x)-1]-x[0]))*sp.fft.fftfreq(len(x),1./len(x))
```

```
Kx=repmat(kx, ny, 1);

%frequency matrix in y direction
if mod(ny, 2)==0
    ky=[0:ny/2-1 -ny/2:-1]*2*pi/L;
else
    ky=[0:ny/2 -floor(ny/2):-1]*2*pi/L;
end
%in python, ky = (2.*np.pi/(y[len(y)-1]-y[0]))*sp.fft.fftfreq(len(y),1./len(y))
Ky=repmat(ky.', 1, nx);

%in pythin, Kx, Ky = np.meshgrid(kx,ky)


MDelta = Kx.^2+Ky.^2; % Fourier Laplacian Operator
A = -(MDelta.*MDelta)-2*MDelta+R-1; % Linear Operator (Biharmonc, Laplacian, Constant te

for n=0:nSteps
    w1 = real(ifft2(expm(A*(dt/2.0))*fft2(w)));
    w2 = w1 - dt*w1.^3;
    w = real(ifft2(expm(A*(dt/2.0))*fft2(w2)));
    %based on previous work in python, the below code might be whats needed
    %w1 = fftshift(real(ifft2(expm(A*(dt/2.0))*fft2(fftshift(w)))));
    %w2 = w1 - dt*w1.^3;
    %w = fftshift(real(ifft2(expm(A*(dt/2.0))*fft2(fftshift(w2)))));
end
return
```

Testing the Method

- **Without fftshift** Did not work... need to think about this some...
-
-