Swift-Hohenberg Numerics - Testing with Chebfun

Edward McDugald

October 22, 2022

1 Operator Splitting for SH

- 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
$$\begin{aligned} L(w) &= -(2\nabla^2 + \nabla^4)w \\ NL(w) &= (R-1)w - w^3. \end{aligned}$$
- Basic procedure is as follows:
  (i) Let $A = (-\nabla^4 - 2\nabla^2)$. Consider the pair of PDEs
  $$\begin{aligned} w_t &= Aw \\ w_t &= (R-1)w - w^3. \end{aligned}$$
  (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,
  $$w(t+\Delta t) \approx \Delta t\left[(R-1)w(t) - w(t)^3\right] + w(t).$$
  .

  (iii) Apply operator splitting (strang splitting) now

The code is implemented in python as follows:

```
def non_lin_rhs(w,R):
    return (R-1)*w - w**3

def integrateSH(w0,R,dt,nSteps,L,o4=False):
     """
     :param w0: initial temperature surface
     :param R: bifurcation parameter- can be a constant, or of same shape as w0
     :param dt: time step length
     :param nSteps: number of time steps to take
     :param L: Length of square over which w0 is defined
     :return w0: time evolution of w0 at time 0+dt*nSteps
     Ideally, the size of w0 is fft friendly, ie 2^n x 2^n
     """
     print("Starting time integration of Swift Hohenberg")
     ny, nx = np.shape(w0)
     print("Dimensions of w0:", nx, ny)
     kx = (2.*np.pi/L)*sp.fft.fftfreq(nx,1./nx)
     ky = (2.*np.pi/L)*sp.fft.fftfreq(ny,1./ny)
     Kx, Ky = np.meshgrid(kx,ky)
     fourierLaplacian = -(Kx**2+Ky**2)
     A = -(fourierLaplacian*fourierLaplacian)-2*fourierLaplacian
     for i in range(0,nSteps):
         # if i%100 == 0:
             # print("step number:",i)
         w1 = np.real(sp.fft.ifft2(np.exp(A*.5*dt)*sp.fft.fft2(w0)))

         #rk4 version
         if o4:
            k1 = dt*w1
            k2 = dt*non_lin_rhs(w1+.5*k1, R)
            k3 = dt*non_lin_rhs(w1+.5*k2, R)
            k4 = dt*non_lin_rhs(w1+k3, R)
            w2 = (k1+2*k2+2*k3+k4)/6 + w1

         #fwd euler version
         else:
            w2 = dt*((R-1)*w1-w1**3)+w1

         w0 = np.real(sp.fft.ifft2(np.exp(A*.5*dt)*sp.fft.fft2(w2)))
     return w0
```

Note: It seems the Forward Euler scheme is working better than RK4 Scripts

to generate these files can be found here python and here matlab

2 Comparison with Chebfun

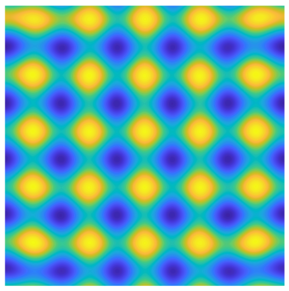
2.1 Chebfun vs Python 1

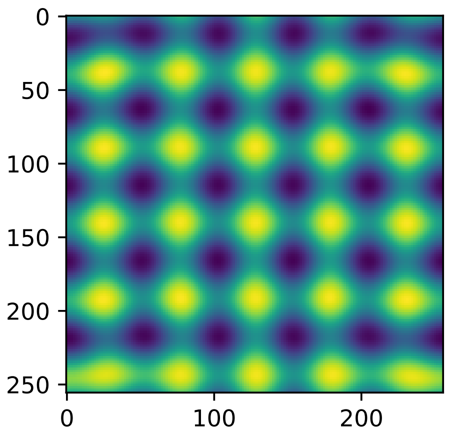
Chebfun 1 code

```
dom = [-16 16 -16 16]
tspan = [0 10];
S = spinop2(dom, tspan)
S.lin = @(u) -2*lap(u)-biharm(u)
R = .5
S.nonlin = @(u) (R-1)*u - u.^3;
u0 = 1/20*chebfun2(@(x,y) cos(x) + sin(2*x) + sin(y) + cos(2*y), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-10*pi).^2 + (y-10*pi).^2)), dom, 'trig');
S.init = u0;
u = spin2(S, 256, .1, 'plot', 'off');
plot(u), view(0,90), axis equal, axis off
saveas(gcf,'/Users/edwardmcdugald/Research/convection_patterns_matlab/figs/sh_tsts_1018/
close all
```

python 1 code

```
x = np.linspace(-16,16,256)
y = np.linspace(-16,16,256)
X,Y = np.meshgrid(x,y)
w0 = (1./20.)*(np.cos(X)+np.sin(2*X)+np.sin(Y)+np.cos(2*Y))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-10*np.pi)**2 +(Y-10*np.pi)**2)))
dt = .1
R=.5
L=x[len(x)-1]-x[0]
nSteps = 100
```


(a) Chebfun



(b) Python

Figure 1: Chebfun v Python 1

```
W1 = integrateSH(w0,R,dt,nSteps,L)
fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax1.imshow(W1)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH1.pdf")


#Running the Methods on the Ellipse
R=.5*(-1./(1.+np.exp(-(X**2+2*Y**2-256.)))+1.)

W1_Ell = integrateSH(w0,R,dt,nSteps,L)
fig1_Ell, ax1_Ell = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax1_Ell.imshow(W1_Ell)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH1_Ell.pdf")
```

Test 1: Chebfun v Python Results

Python Result on Ellipse

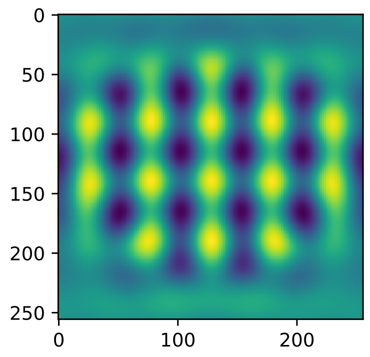


Figure 2: Python Result on Ellipse

2.2 Chebfun vs Python 2

Chebfun 2 code

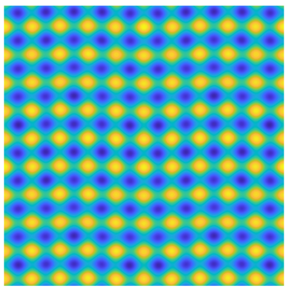
```
dom = [0 20*pi 0 20*pi];
tspan = [0 200];
S = spinop2(dom, tspan);
S.lin = @(u) -2*lap(u) - biharm(u);
r = 1e-2;
S.nonlin = @(u) (-1 + r)*u - u.^3;
u0 = 1/20*chebfun2(@(x,y) cos(x) + sin(2*x) + sin(y) + cos(2*y), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-10*pi).^2 + (y-10*pi).^2)), dom, 'trig');
S.init = u0;
plot(S.init), view(0,90), axis equal, axis off
close all
u = spin2(S, 96, 2e-1, 'plot', 'off');
plot(u), view(0,90), axis equal, axis off
saveas(gcf,'/Users/edwardmcdugald/Research/
convection_patterns_matlab/figs/sh_tsts_1018/cf2.pdf');
close all
```

python 2 code

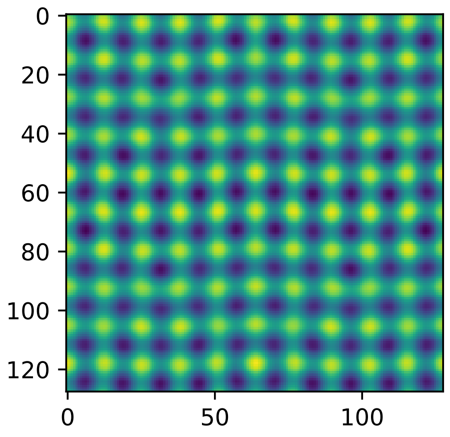
```
x = np.linspace(0,20*np.pi,128)
y = np.linspace(0,20*np.pi,128)
X,Y = np.meshgrid(x,y)
```

```
w0 = (1./20.)*(np.cos(X)+np.sin(2*X)+np.sin(Y)+np.cos(2*Y))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-10*np.pi)**2 +(Y-10*np.pi)**2)))
dt = 2e-1
R= 1e-2
L=x[len(x)-1]-x[0]
nSteps = 1000

W2 = integrateSH(w0,R,dt,nSteps,L)
fig2, ax2 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax2.imshow(W2)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH2.pdf")

#Running the Methods on the Ellipse
R=.5*(-1./(1.+np.exp(-((X-10*np.pi)**2+2*(Y-10*np.pi)**2-256.)))+1.)

W2_Ell = integrateSH(w0,R,dt,nSteps,L)
fig2_Ell, ax2_Ell = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax2_Ell.imshow(W2_Ell)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH2_Ell.pdf")
```

Test 2: Chebfun v Python Results



(a) Chebfun



(b) Python

Figure 3: Chebfun v Python 2

Python Result on Ellipse

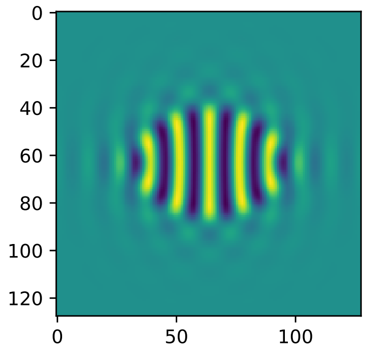


Figure 4: Python Result on Ellipse

2.3 Chebfun vs Python 3

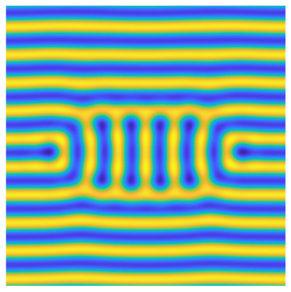
Chebfun 3 code

```
dom = [0 20*pi 0 20*pi];
tspan = [0 200];
```

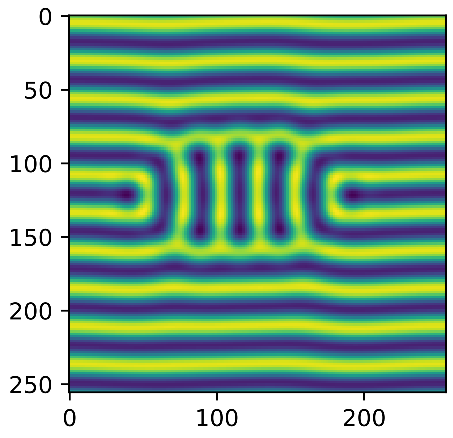
```
S = spinop2(dom, tspan);
S.lin = @(u) -2*lap(u) - biharm(u);
u0 = 1/20*chebfun2(@(x,y) cos(x) + sin(2*x) + sin(y) + cos(2*y), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-5*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-15*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-15*pi).^2 + (y-5*pi).^2)), dom, 'trig');
u0 = u0 + chebfun2(@(x,y) exp(-((x-10*pi).^2 + (y-10*pi).^2)), dom, 'trig');
S.init = u0;
r = 7e-1;
S.nonlin = @(u) (-1 + r)*u - u.^3;
u = spin2(S, 256, 2e-1, 'plot', 'off');
plot(u), view(0,90), axis equal, axis off
saveas(gcf,'/Users/edwardmcdugald/Research/
convection_patterns_matlab/figs/sh_tsts_1018/cf3.pdf');
close all
```

python 3 code

```
x = np.linspace(0,20*np.pi,256)
y = np.linspace(0,20*np.pi,256)
X,Y = np.meshgrid(x,y)
w0 = (1./20.)*(np.cos(X)+np.sin(2*X)+np.sin(Y)+np.cos(2*Y))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-5*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-15*np.pi)**2)))
w0 = w0 + (np.exp(-((X-15*np.pi)**2 +(Y-5*np.pi)**2)))
w0 = w0 + (np.exp(-((X-10*np.pi)**2 +(Y-10*np.pi)**2)))
dt = 2e-1
R= 7e-1
L=x[len(x)-1]-x[0]
nSteps = 1000

W3 = integrateSH(w0,R,dt,nSteps,L)
fig3, ax3 = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
ax3.imshow(W3)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH3.pdf")

#Running the Methods on the Ellipse
R=.5*(-1./(1.+np.exp(-((X-10*np.pi)**2+2*(Y-10*np.pi)**2-256.)))+1.)

W3_Ell = integrateSH(w0,R,dt,nSteps,L)
fig3_Ell, ax3_Ell = plt.subplots(nrows=1, ncols=1, figsize=(3,3))
```



(a) Chebfun



(b) Python

Figure 5: Chebfun v Python 3

```
ax3_Ell.imshow(W3_Ell)
plt.savefig("/Users/edwardmcdugald/Research/
convection_patterns/code/figs/sh_num_tsts_1018/mySH3_Ell.pdf")
```

Test 3: Chebfun v Python Results

Python Result on Ellipse

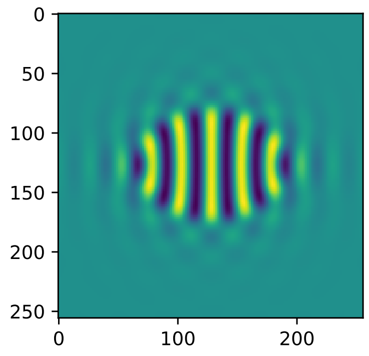


Figure 6: Python Result on Ellipse

3 Shankars Code

Implementation based on the following ideas

- Global description of patterns far from onset: a case study
- fourth order time stepping for stiff pdes

The original code

```
%% Swift-Hohenberg equation

% We use a pseudospectral discretization along with Exponential
% time-differencing/RK4 to solve the Swift-Hohenberg equation on
% "arbitrary" domains.

% The equation we are solving is
%   u_t = -(\Delta + 1)^2 u + R u - u^3

% To use a spectral discretization, we fit the domain inside a rectangle
% [0,L_x] X [0, L_y] and extend periodically. Since the patterns only form
% for R > 0, we choose R = -0.5 + R \chi_{\Omega}, where \chi is a smoothed
% characteristic function of \Omega.

% To prevent aliasing due to the nonlinearity u^3, we need to keep twice as
% many sample points in each direction as the number of useful (i.e.
% low-frequency) modes. We can keep R "sharp" on the scale of the grid
% spacing, although we will typically use a low pass filtered version of R
% since it shouldn't play a big role in determining the pattern, and a
% "softer" boundary seems more robust.

% This idea borrows from "Global description of patterns far from onset: a
% case study" by Ercolani, Indik, Newell and Passot for the setup of
% Swift-Hohenberg, and from "Fourth-order time-stepping for stiff PDEs" by
% Kassam and Trefethen for the implementation of the ETD.

%%  Spatial   grid   and   initial   condition:

Lx = 120*pi;    % Size of enclosing periodic rectangle
Ly = 60*pi;
% we want N ~ L_x because max wavenumber is like k_x ~ pi*N/Lx which we
% want to be about 3. Likewise for Ly.
```

```
Nx = 512;
Ny = 256;
% Number of grid points in x and y. We will keep them as powers of 2.
% Useful number of modes = Nx Ny/4

beta = 0.45; % Less than 0.5, sets relative size of the ellipse in the box
amplitude = 0.1; % for initial pattern. want this "small" compared to sqrt(R)

xx = Lx*(-Nx/2+1:Nx/2)'/Nx;
yy = Ly*(-Ny/2+1:Ny/2)'/Ny;
[X,Y] = ndgrid(xx,yy);

R = 0.5*tanh(sqrt(Lx^2+Ly^2)*(beta-sqrt((X/Lx).^2+(Y/Ly).^2))/2);

if (init_flag == 1)
    u0 = randn(N);
    u0 = amplitude*u0/norm(u0, inf);
elseif (init_flag == 2)
    u0 = amplitude*sin(Y);
elseif (init_flag == 3)
    u0 = ellipse_init(X,Y,beta*Lx,beta*Ly,amplitude);
end

figure(21)
surf(X,Y,u0.*(R+0.5),'mesh','none');
axis equal;
view(2);

%%   Precompute   various   ETDRK4   scalar   quantities:

h = 3;   %   time   step

kx = 2*pi*[0:Nx/2 -Nx/2+1:-1]'/Lx;   %   wave   numbers
ky = 2*pi*[0:Ny/2 -Ny/2+1:-1]'/Ly;

[xi,eta]=ndgrid(kx,ky);

L = -(1-xi.^2-eta.^2).^2;   %   Fourier   multipliers
E = exp(h*L);
E2= exp(h*L/2);

M = 16;   %   no.   of   points   for   complex   means
r = exp(1i*pi*((1:M)-.5)/M);   %   roots   of   unity.
% This circle should suffice since max eigenvalue of Lh is 0, at k=0, Lh =
% -1/8
```

```
L2 = L(:); % convert to a single column

LR = h*L2(:,ones(M,1)) + r(ones(Nx*Ny,1),:); % adding r(j) to the jth column
Q = h*real(mean((exp(LR/2)-1)./LR,2));  % means in the 2 direction.
f1 = h*real(mean((-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3,2));
f2 = h*real(mean((2+LR+exp(LR).*(-2+LR))./LR.^3,2));
f3 = h*real(mean((-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3,2));

f1=reshape(f1,Nx,Ny);  % convert from columns to N by N matrices
f2=reshape(f2,Nx,Ny);
f3=reshape(f3,Nx,Ny);
Q=reshape(Q,Nx,Ny);

%clear LR;

%% Dealiasing

Fx = false(Nx,1); % Fx = 1 for high frequencies which will be set to zero
Fy = false(Ny,1);

Fx((Nx/2+1-round(Nx/4):1+ Nx/2+round(Nx/4)))= true; % For dealiasing with u^3
Fy((Ny/2+1-round(Ny/4):1+ Ny/2+round(Ny/4)))= true;

[alxi,aleta] = ndgrid(Fx,Fy);
ind = alxi | aleta; %de-aliasing index.

%filter R and u0

Rhat = fftn(R);
Rhat(ind) = 0;
R = real(ifftn(Rhat));

vv = fftn(u0);
vv(ind) = 0;
u0 = real(ifftn(vv));

Q(ind) = 0; % Q is the only term that multiplies the nonlinear factors

%%   Main   time-stepping   loop:

tmax   =   1000;

nmax   =   round(tmax/h);
nplt   =   floor((tmax/25)/h);
```

```
tt = zeros(1,round(nmax/nplt)+1);
uu   =   zeros(Nx,Ny,round(nmax/nplt)+1);
% vv = fftn(u0);

ii = 1;

tic

for n   =   1:nmax
    t   =   n*h;

    Nv   =  fftn(R.*u0 - u0.^3);

    a   =   E2.*vv   +   Q.*Nv;
%    a(ind) = 0;
    ua  =   real(ifftn(a));
    Na   =  fftn(R.*ua - ua.^3);

    b   =   E2.*vv   +   Q.*Na;
%    b(ind) = 0;
    ub  =   real(ifftn(b));
    Nb   =  fftn(R.*ub - ub.^3);

    c   =   E2.*a   +   Q.*(2*Nb-Nv);
%    c(ind) = 0;
    uc  =   real(ifftn(c));
    Nc   =  fftn(R.*uc - uc.^3);

    vv   =   E.*vv   +   Nv.*f1   +   2*(Na+Nb).*f2   +   Nc.*f3;
%     vv(ind) = 0;
    u0  =   real(ifftn(vv));

    if   mod(n,nplt)==0

        uu(:,:,ii)   =   u0;
        tt(ii)   =  t;
        ii = ii+1;
    end
end

toc

figure(1)
clf;
```

```
surf(X,Y,u0,'mesh','none');
axis equal;
view(2);

%% Processing to find the locations of the convex-concave pairs

xrng = xx(round((0.5-beta)*Nx):round((0.5+beta)*Nx));
onaxis = u0(round((0.5-beta)*Nx):round((0.5+beta)*Nx),Ny/2);
figure(2)
plot(xrng,onaxis)



zci = find(onaxis(2:end).*onaxis(1:end-1) <= 0);     % Returns Zero-Crossing Indices Of
upval = onaxis(zci+1);
downval = onaxis(zci);
zx = (upval.*xrng(zci)-downval.*xrng(zci+1))./(upval-downval);

figure(2)
clf
plot(xrng,onaxis,'-r')
hold on
plot(zx,zeros(size(zci)),'bp')
hold off
grid
legend('Signal', 'Approximate Zero-Crossings');


semi_mjr = beta*Lx;
semi_mnr = beta*Ly;

xc = semi_mjr*min(1,max(-1,semi_mjr*zx/(semi_mjr^2-semi_mnr^2)));
rho = sqrt((zx-xc).^2 + semi_mnr^2*(1-xc.^2/semi_mjr^2));

figure(3)
if (init_flag < 4)
    clf
end
plot(rho/pi,'-o');
hold on;

figure(4)
if (init_flag < 4)
    clf
end
```

```
plot(0.5+(1:length(zx)-1),diff(rho)/pi,'-*');
hold on

%% Compute energy density

eloc = (1-xi.^2-eta.^2).*fftn(u0);
eloc(ind) = 0;
eloc = ifftn(eloc).^2;

u0sq = fftn(u0.^2);
u0sq(ind) = 0;
u0sq = ifftn(u0sq);

u04th = fftn(u0sq.^2);
u04th(ind) = 0;
u04th = ifftn(u04th);

edens = 0.5*(eloc-R.*u0sq + 0.5*u04th);

figure(5)
clf;
surf(X,Y,edens,'mesh','none');
axis equal;
view(2);

figure(6)
clf;
surf(X,Y,u0sq,'mesh','none');
axis equal;
view([0 0 30]);

%% Processing the phase

%phase = unwrap(angle(hilbert(0.1+u0(round((0.5-beta)*Nx):round((0.5+beta)*Nx),1:Ny/2)')
phase = zeros(Nx/2-round((0.5-beta)*Nx)+1,Ny);

for ii = round((0.5-beta)*Nx):Nx/2

    ind = find(R(ii,:) > 0);
    mindx = ind(1);
    mxndx = ind(end);

    u1 = (0.5+R(ii,mindx:mxndx)).*u0(ii,mindx:mxndx);

    uext = [u1 0 -u1 0];
    phase_loc = unwrap(angle(hilbert(uext)))/(2*pi);
    phase_loc = phase_loc(1:mxndx-mindx+1);
    reverse = phase_loc(end:-1:1);
    phase_loc = min(phase_loc,reverse);

    phase(ii+1-round((0.5-beta)*Nx),:) = ...
        [phase_loc(1)*ones(1,mindx-1) phase_loc phase_loc(end)*ones(1,Ny-mxndx)];
end

nmx = 256;
q = 2*pi*(1:nmx)/nmx;
bdry = beta*[Lx*cos(q);Ly*sin(q)];

imx = length(xrng);
phase0 = zeros(imx,1);

for ii = 1:imx
    phase0(ii) = sqrt(min((xrng(ii)-bdry(1,:)).^2+(bdry(2,:)).^2))/(2*pi);
end

figure(7);
clf
plot(xrng(1:imx),phase0);
hold on;
plot(X(round((0.5-beta)*Nx):Nx/2,Ny/2),phase(:,Ny/2));

% filter to compute amplitude
rho = 2*real(ifftn(exp(-(xi.^2+eta.^2)).*fftn(u0.^2)));

figure(8)
surf(X,Y,rho,'mesh','none');
%axis equal;
view(2);

% figure(9)
% surf(X(round((0.5-beta)*Nx):round((0.5+beta)*Nx),1:Ny/2),...
%     Y(round((0.5-beta)*Nx):round((0.5+beta)*Nx),1:Ny/2),...
%     phase,'mesh','none');
% %axis equal;
% view([0 0 30]);

phase =  [phase; phase(end-1:-1:1,:)];

figure(9)
mesh(X(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    Y(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    2*pi*phase,-phase)
colormap colorcube
%,'mesh','none','colormap','lines');
%axis equal;
view(2);

figure(1)
clf;
surf(X(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    Y(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    u0(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),'mesh','none');
axis equal;
view([0 0 30]);

figure(10)
surf(X(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    Y(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:),...
    (0.5+R(round((0.5-beta)*Nx):round((0.5+beta)*Nx),:))...
    .*cos(2*pi*phase),'mesh','none');
axis equal;
view([0 0 30]);

%% Postscript

% I'm going to split the initialization code from the time stepping. This
% makes it easier to continue an existing solution for a longer time, or
% restarting as appropriate.
```