Spiral Waves in λ–ω Reaction-Diffusion Systems
AMATH 581 Homework 4

Erik Neumann
610 N. 65th St., Seattle, WA 98103
erikn@MyPhysicsLab.com

December 7, 2001

Abstract

The λ–ω reaction–diffusion system is investigated. Spectral transform methods are used to numerically integrate the system. Filtered and unfiltered spectral methods are compared. Various initial conditions are examined including rectangular cells, and one or two-armed spirals. Stable rotating spiral waves are demonstrated. The effect of changing parameters in the reaction–diffusion system is also investigated. In particular, we look for where the solution becomes unstable and chaotic.

Contents

1 Prolegomenon

We consider the λ–ω reaction–diffusion system in two dimensions which is defined by

$$U_t = \lambda(A)U - \omega(A)V + D_1\nabla^2 U \tag{1a}$$
$$V_t = \omega(A)U + \lambda(A)V + D_2\nabla^2 V \tag{1b}$$

where $A^2 = U^2 + V^2$ and $\nabla^2 = \partial_x^2 + \partial_y^2$. U and V represent two different species of chemicals that react with each other and diffuse on their own. The diffusion term in equation (1a) is $D_1\nabla^2 U$. The reaction terms are the remaining terms on the right hand side.

The λ–ω functions we will consider are

$$\lambda(A) = \epsilon - aA^2 \tag{2}$$
$$\omega(A) = c - \beta A^2 \tag{3}$$

where all the parameters ϵ, a, c, β are greater than zero. We will investigate the effects on the system of modifying these four parameters.

The system (1) will be numerically solved using the spectral transform method, and also a filtered spectral transform method. These two methods will be compared as to running time.

2 Theoretical Background

2.1 Spectral Transform

The Fourier transform and its inverse are defined as

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} f(x)\, dx \tag{4}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} \hat{f}(k)\, dk \tag{5}$$

We will denote the Fourier transform in x of a function f by $\hat{f}$. The key relation is among derivatives of functions. The general result for the n-th derivative of a function f is

$$\hat{f^{(n)}} = (-ik)^n \hat{f} \tag{6}$$

where k is the wave number. To numerically integrate equation (1a) we begin by taking the Fourier transform in x.

$$\hat{U}_t = \widehat{\lambda(A)U} - \widehat{\omega(A)V} - D_1 k_x^2 \hat{U} + D_1 \widehat{\partial_y^2 U} \tag{7}$$

We denote the Fourier transform in y of f by $\tilde{f}$. Then taking the Fourier transform in y of equation (7) gives

$$\tilde{\hat{U}}_t = \widetilde{\widehat{\lambda(A)U}} - \widetilde{\widehat{\omega(A)V}} - D_1(k_x^2 + k_y^2)\tilde{\hat{U}} \tag{8}$$

Similarly, taking the Fourier transform in x and y of equation (1b) gives us

$$\tilde{\hat{V}}_t = \widetilde{\widehat{\omega(A)U}} + \widetilde{\widehat{\lambda(A)V}} - D_2(k_x^2 + k_y^2)\tilde{\hat{V}} \tag{9}$$

Equations (8) and (9) can now be solved together using a Runge-Kutta ODE (ordinary differential equation) solver. We advance the equations in time while the variables $\tilde{\hat{U}}$ and $\tilde{\hat{V}}$ remain in the spectral domain.

2.2 Filtered Spectral Transform

The filtered spectral transform uses a trick from the theory of ordinary differential equations. We first write equation (8) as

$$\tilde{\hat{U}}_t + (D_1(k_x^2 + k_y^2) - \epsilon)\tilde{\hat{U}} = -a\widetilde{\widehat{A^2U}} - c\tilde{\hat{V}} + \beta\widetilde{\widehat{A^2V}} \tag{10}$$

Next we multiply by the integrating factor $e^{(D_1(k_x^2+k_y^2)-\epsilon)t}$

$$\tilde{\hat{U}}_t e^{(D_1(k_x^2+k_y^2)-\epsilon)t} + (D_1(k_x^2 + k_y^2) - \epsilon)\tilde{\hat{U}} e^{(D_1(k_x^2+k_y^2)-\epsilon)t} = e^{(D_1(k_x^2+k_y^2)-\epsilon)t}\left(-a\widetilde{\widehat{A^2U}} - c\tilde{\hat{V}} + \beta\widetilde{\widehat{A^2V}}\right) \tag{11}$$

We recognize the left hand side as the derivative of a product

$$(\tilde{\hat{U}} e^{(D_1(k_x^2+k_y^2)-\epsilon)t})_t = e^{(D_1(k_x^2+k_y^2)-\epsilon)t}\,(-a\widetilde{\widehat{A^2U}} - c\tilde{\hat{V}} + \beta\widetilde{\widehat{A^2V}}) \tag{12}$$

Define the variable G by

$$\tilde{\hat{G}} = e^{(D_1(k_x^2+k_y^2)-\epsilon)t}\,\tilde{\hat{U}} \tag{13}$$

Then equation (12) becomes

$$\tilde{\hat{G}}_t = e^{(D_1(k_x^2+k_y^2)-\epsilon)t}\,(-a\widetilde{\widehat{A^2U}} - c\tilde{\hat{V}} + \beta\widetilde{\widehat{A^2V}}) \tag{14}$$

We have solved explicitly for the linear, constant coefficient terms. This removes the numerical stiffness of these terms.

A similar treatment for equation (9) leads to the new variable H defined by

$$\tilde{\hat{H}} = e^{(D_2(k_x^2+k_y^2)-\epsilon)t}\,\tilde{\hat{V}} \tag{15}$$

and the following differential equation

$$\tilde{\hat{H}}_t = e^{(D_2(k_x^2+k_y^2)-\epsilon)t}\,(c\tilde{\hat{U}} - \beta\widetilde{\widehat{A^2U}} - a\widetilde{\widehat{A^2V}}) \tag{16}$$

We can now use a Runge-Kutta ODE solver to advance equations (14) and (16) in time. At each time step, we must update the values of U and V used in the right hand sides of these equations. We can use equations (13) and (15) to find U and V from the current value of G and H.

3 Algorithm Implementation and Development

3.1 Initial Conditions

A dynamic reaction can be sustained for the λ–ω reaction diffusion system when U and V have sinusoidal profiles that are offset in space. Here are the initial conditions that we consider: We consider the following initial conditions for ω

- One armed spiral.
- Two armed spiral.
- Square grids.
- Parallel rows.

Let L be the width and height of the two dimensional space over which U and V are defined. Let x, y be vectors of discretized values from $-L/2$ to $L/2$. We can set up sinusoidal parallel rows by

$$U = \sin(m\pi x/L) \tag{17}$$
$$V = \cos(m\pi x/L) \tag{18}$$

where m controls the number of rows.

A sinusoidal wave on square grid is defined by

$$U = \sin(m\pi x/L)\,\sin(m\pi y/L) \tag{19}$$
$$V = \cos(m\pi x/L)\,\cos(m\pi y/L) \tag{20}$$

where m controls the number of squares.

A spiral can be defined by

$$U = \tanh(\sqrt{x^2+y^2})\,\cos(m\,\mathrm{Arg}(x+iy) - \sqrt{x^2+y^2}) \tag{21}$$
$$V = \tanh(\sqrt{x^2+y^2})\,\sin(m\,\mathrm{Arg}(x+iy) - \sqrt{x^2+y^2}) \tag{22}$$

where m controls the number of spiral arms.

3.2 Spectral Transform Implementation

The spectral transform for the reaction–diffusion system is implemented by the Matlab file `rd2.m`. Various parameters can be set near the top of the file such as

- Which initial condition to use.
- Values of parameters ϵ, a, c, β
- Diffusion factor D_1.
- Number of grid points N.
- Time step Δt.
- How long to run the simulation.
- Time between displayed frames.

The Matlab code in `rd2.m` is mostly a straightforward implementation of the theory given in Section 2.1. We begin by creating U and V according to the initial conditions as outlined in Section 3.1. We then apply Fourier transform in x and y to get $\tilde{\tilde{U}}$ and $\tilde{\tilde{V}}$. We form the matrices $\tilde{\tilde{U}}$ and $\tilde{\tilde{V}}$ into column vectors and concatenate them to form a single column vector $\tilde{\hat{Z}}$. Then equations (8) and (9) define the right hand side for the Runge-Kutta `ode23` solver. The right hand side is coded as the Matlab file `rd2_rhs.m`.

Note that we need to inverse Fourier transform within the right hand side to recover U and V from the frequency domain so that we can calculate the non-linear portions of equations (8) and (9). We then re-apply the Fourier transform to the result, since the equations are being solved in the frequency domain.

We make an adjustment to Matlab's `fft` function as follows. As part of `fft`, Matlab multiplies the result by n, the number of data points. To undo this effect we divide by n after each `fft`, otherwise working in the frequency domain the right hand side of equations (8) and (9) become very large and so `ode23` takes extremely small time steps. And we multiply by n before the inverse Fourier transform `ifft`.

To ensure that U and V are real data, we use the function `real` on the results of `ifft`.

3.3 Filtered Spectral Implementation

The spectral transform for the reaction–diffusion system is implemented by the Matlab file `rd3.m`. The code is very similar to that for the non-filtered version, and so most of the comments in Section 3.2 apply. The difference is that the right hand side is defined by equations (14) and (16).

After creating U and V according to the initial conditions as outlined in Section 3.1 we apply the Fourier transform and use equations (13) and (15) to arrive at initial values for $\tilde{\hat{G}}$ and $\tilde{\hat{H}}$. These are concatenated into column vectors to form a single column vector $\hat{Z}$. Then equations (14) and (16) define the right hand side for the Runge-Kutta `ode23` solver. The right hand side is coded as the Matlab file `rd3_rhs.m`.

Within the right hand side function we need to inverse Fourier transform and use equations (13) and (15) to recover U and V from the frequency domain so that we can calculate the non-linear portions of equations (14) and (16). We then re-apply the Fourier transform and inverses of equations (13) and (15) since the equations are being solved in the filtered frequency domain.

4 Computational Results

The website `www.MyPhysicsLab.com` has some of these results available to view as movies. Look for the icon on the first page labeled "Reaction–Diffusion Simulations".

4.1 Running Time Comparison

The two algorithms, spectral transform and filtered spectral, were compared with the same initial conditions. The initial condition was for a one-armed spiral, and the code was run to time $t = 20$. The parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. The time step used was $\Delta t = 0.5$. The running times as measured by `cputime` are shown below.

method	cputime (secs)
spectral transform	215
filtered spectral	194

Figure 1 shows the last frame for each run. It is apparent that there is not a significant difference, so we judge the two methods as having roughly the same accuracy.

Note that we had to set the `ode23` absolute tolerance to infinity for the filtered spectral method, so that only relative tolerance was used. Otherwise the filtered spectral method ran over 4 times slower than the regular spectral method.

Here is why we must set absolute tolerance to infinity: the maximum absolute value of the right-hand side function was observed during the two runs (filtered and regular). While the regular spectral method remained at about 0.3, the filtered spectral method grew by about 5% each invocation, which rapidly becomes a huge number. With the default absolute tolerance setting of 0.001 the `ode23` method must take smaller and smaller steps as the right hand side gets larger.

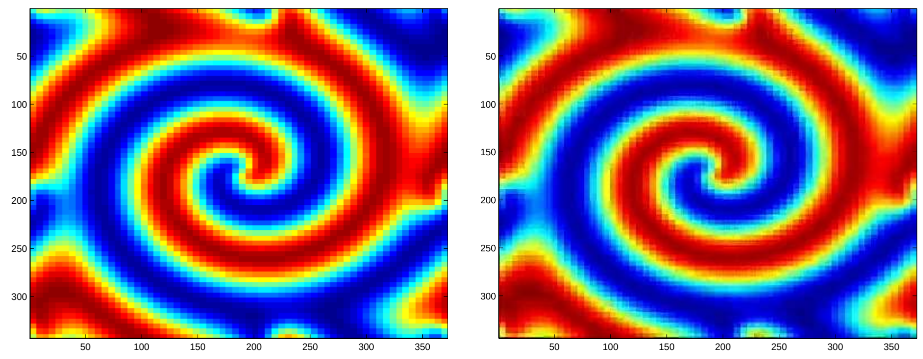
The regular spectral method was used for the remainder of the investigation.

4.2 Varied Initial Conditions

Various initial conditions were described in Section 3.1. The results from these initial conditions are shown in this section. The regular spectral transform was used for all of these computations. The diffusion was set to $D_1 = D_2 = 0.1$ in each case. The other parameter settings were $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. The time step used with `ode23` was $\Delta t = 0.5$ unless noted.

Figure 2 shows a one-armed spiral rotating counter-clockwise. This pattern rotates indefinitely without decaying (out to at least time $t = 500$). The one armed spiral initial condition has some sharp transitions at the boundaries, i.e. it is not smooth at the periodic boundaries. But the diffusion quickly smooths out these sharp edges.

If the one-armed spiral initial condition is scaled so that U varies between -0.2 and 0.2, it rapidly grows to a range of -1 to 1. Similarly, if it is scaled to vary



(a) Regular Spectral (b) Filtered Spectral

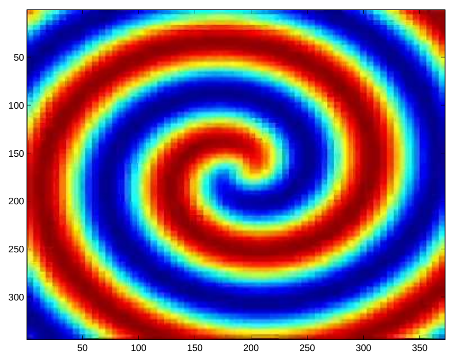
Figure 1: Comparison of last frames ($t = 20$) from regular spectral and filtered spectral methods. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).

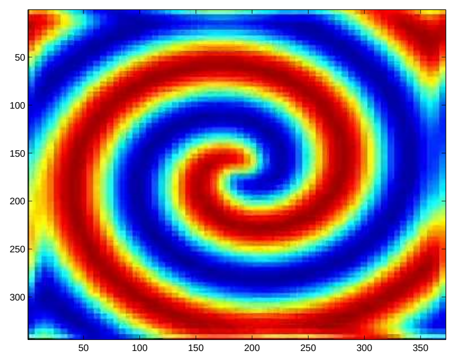
between -2 and 2, then it rapidly shrinks to the range of -1 to 1. So it seems that an amplitude range of -1 to 1 is stable for the given parameter settings.

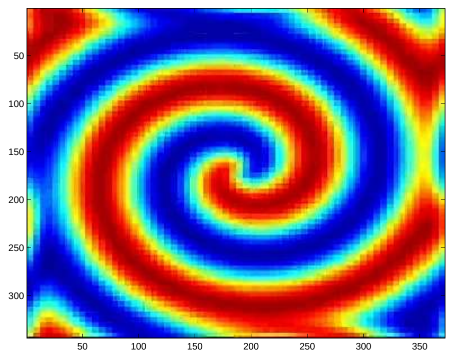
Figure 3 shows a two-armed spiral rotating counter-clockwise. This pattern rotates indefinitely.

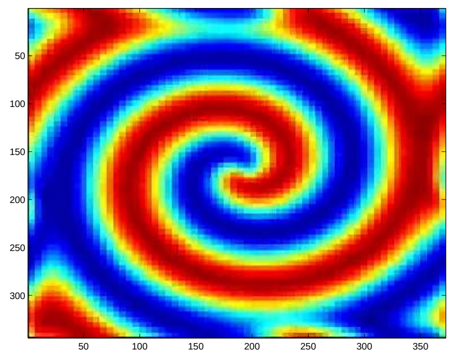
Figure 4 shows a square grid with a smooth sine function. The resulting patterns oscillate indefinitely at a rapid rate without decaying. A time step of $\Delta t = 0.25$ was used due to the very rapid oscillation.

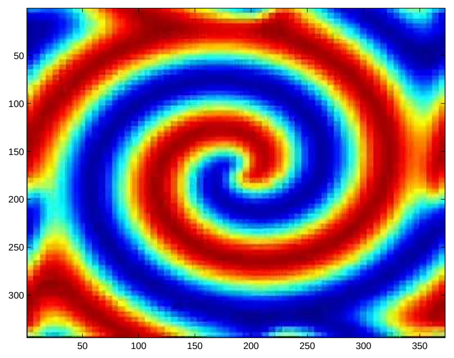
Figure 5 shows an initial condition of parallel sinusoidal rows. These move left at a constant velocity indefinitely.


(a) $t = 0$


(b) $t = 1.5$


(c) $t = 3.0$


(d) $t = 4.5$


(e) $t = 6.0$

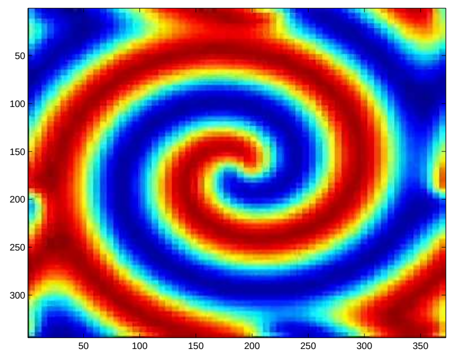
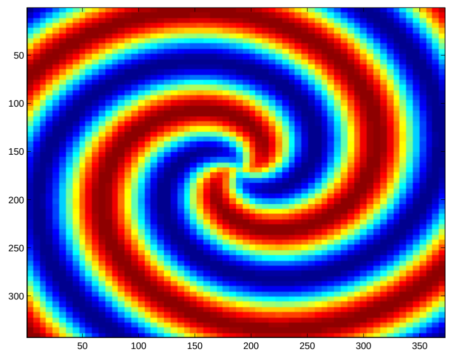
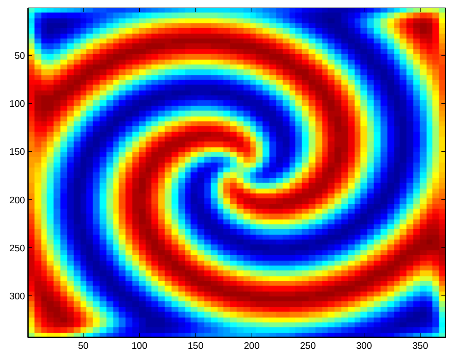

(f) $t = 7.5$

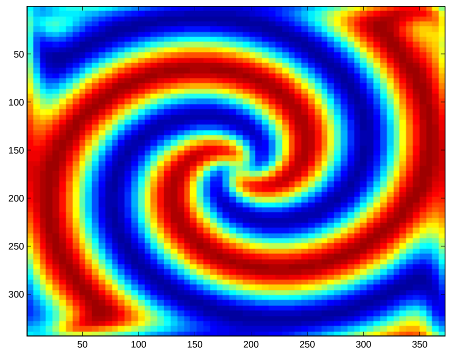
Figure 2: One armed spiral initial condition rotates counter-clockwise. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).



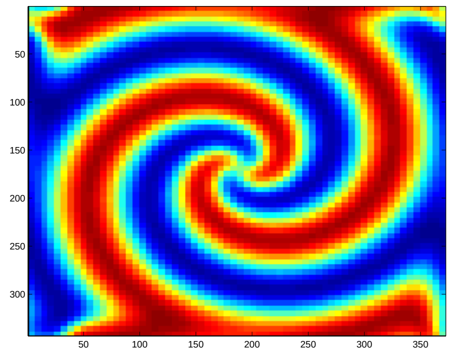
(a) $t = 0$



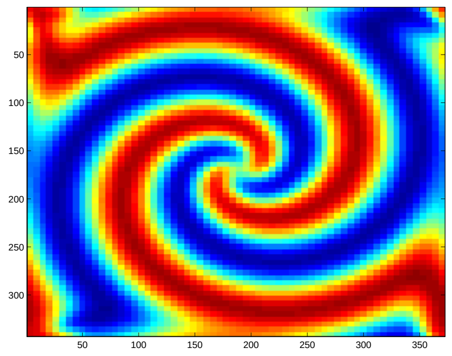
(b) $t = 2$



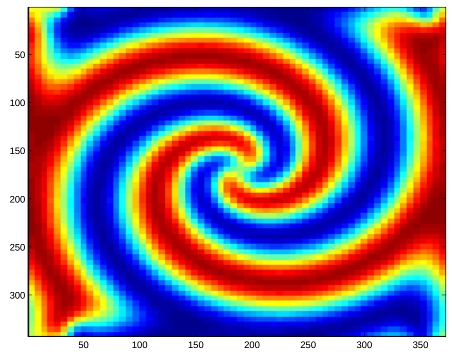
(c) $t = 4$



(d) $t = 6$



(e) $t = 8$



(f) $t = 10$

Figure 3: Two armed spiral initial condition rotates counter-clockwise. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).

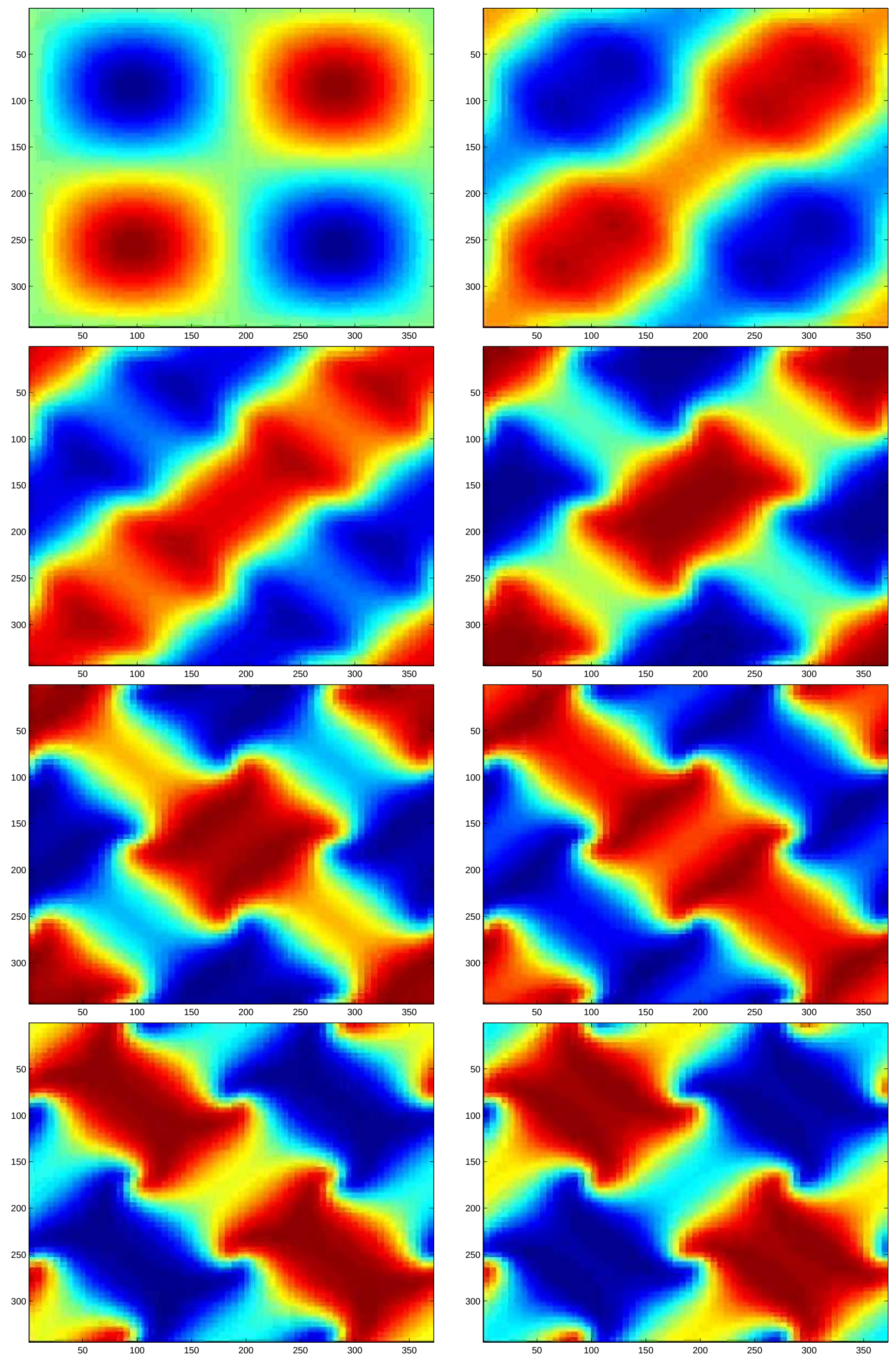
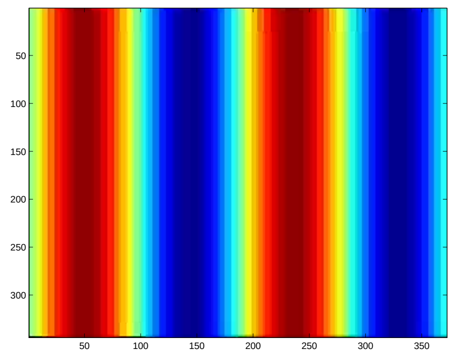
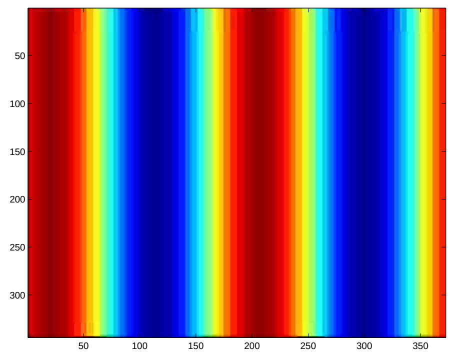


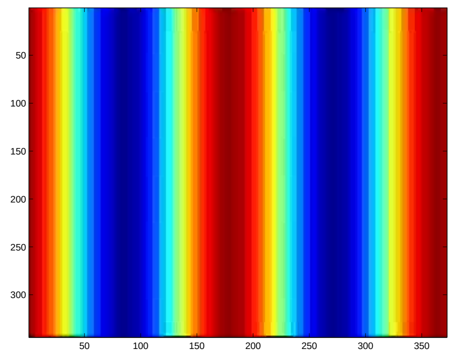
Figure 4: Square grid at times $t = 0$, 0.5, 1, 1.5, 2, 2.5, 3, 3.5. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).



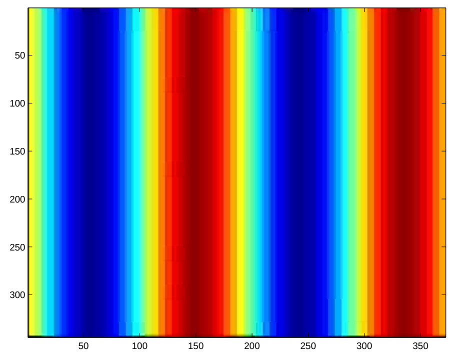
(a) $t = 0$



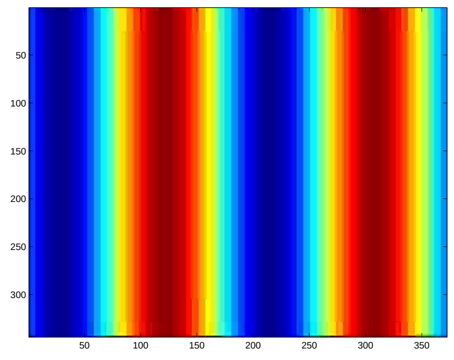
(b) $t = 1$



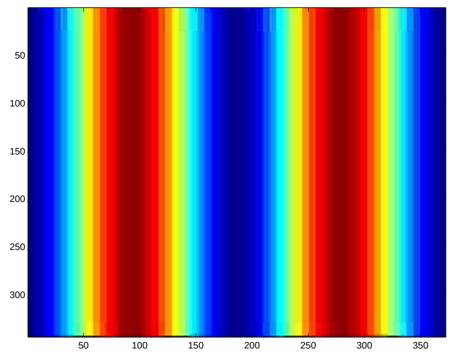
(c) $t = 2$



(d) $t = 3$



(e) $t = 4$



(f) $t = 5$

Figure 5: Parallel rows initial condition. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).

4.3 Effect of Parameters

We investigate the effect of modifying the parameters D, ϵ, a, c, β in equations (1), (2) and (3). The single-arm spiral is the initial condition in each case. The method used is to regard the settings $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$ as the default, and modify a single parameter to see the effects. The time step used with `ode23` was $\Delta t = 0.5$ unless noted.

Figure 6 shows the effect of varying the diffusion parameter D (we have taken $D_1 = D_2$ throughout and so just refer to D). The values used were $D = 0.01, 0.04, 0.1, 0.2, 0.4, 0.8$. The figure shows the state of the calculation at time $t = 20$. Increasing diffusion causes the arms of the spiral to become fatter. The direction or speed of the motion was not affected. At the highest diffusion level, $D = 0.8$, the spiral pattern breaks up and an oscillating pattern similar to that for the grid boxes (see Figure 4) takes over.

Figure 7 shows the effect of varying ϵ. The values of epsilon shown are $\epsilon = 0.1, 0.25, 0.5, 1.5, 2.5, 4.0$. The figure shows the state of the calculation at time $t = 20$. There are three effects from modifying ϵ.

1. The range of the solution rapidly changes, with small ϵ giving small ranges, and high ϵ leading to larger ranges.
2. The speed of the rotation is greatly affected with low ϵ leading to low rotation speeds, and high ϵ giving high rotation speeds.
3. The arms of the spiral become longer, thinner and wrap around more with increasing ϵ. With decreasing ϵ the spiral arms become fatter and shorter.

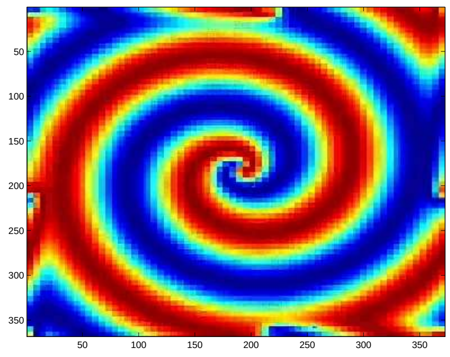
Additionally, we can see a new spiral developing on the side for $\epsilon = 4.0$, which could lead to instability.

Figure 8 shows how chaos results from setting parameter $a = 0.25$. The effects of modifying a are similar to those for modifying ϵ. Smaller a (less than 1) leads to larger range, faster rotation, longer thinner spiral arms, and eventually chaos. Larger a (greater than 1) leads to smaller range, slower rotation, fatter shorter spiral arms.

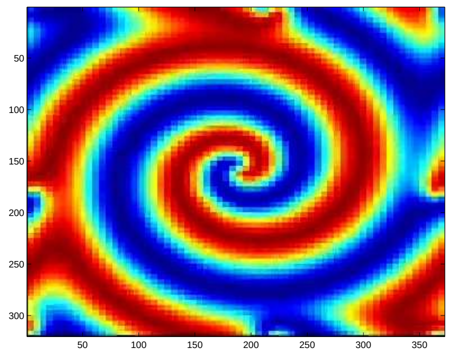
Figure 9 shows that setting parameter $c = 1.5$ causes clockwise rotation instead of the usual counter-clockwise rotation. The only effect of parameter c is to change the rotation rate and direction. The size and shape of the spiral arms is unaffected by changing c. As we increase c from 0 to 0.9, the rotation is counter-clockwise and slows down. At $c = 0.9$ the rotation comes to a complete halt and there is no motion. As c increases above 0.9 the rotation becomes clockwise and increases in speed.

Figure 10 shows that chaos results from setting parameter $\beta = 2.5$. Decreasing β from 1 to 0.1 slows the rotation, but has no effect on the shape of the spiral.

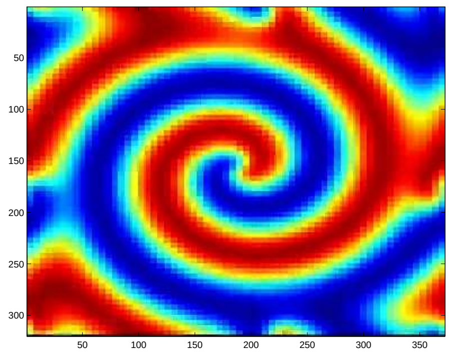
Increasing β above 1 increases the rate of rotation and causes the spiral arms to thin and wrap around the center. As β is increased further, unstable areas appear and grow until there are only chaotic oscillations.



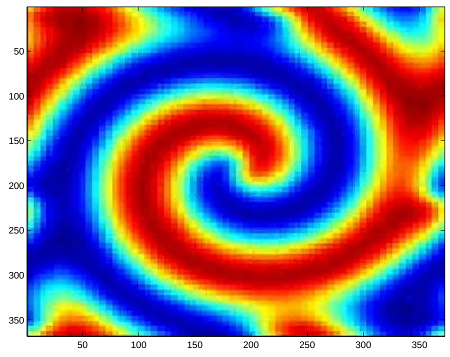
(a) $D = 0.01$



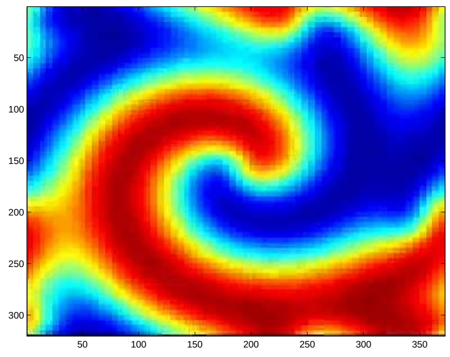
(b) $D = 0.04$



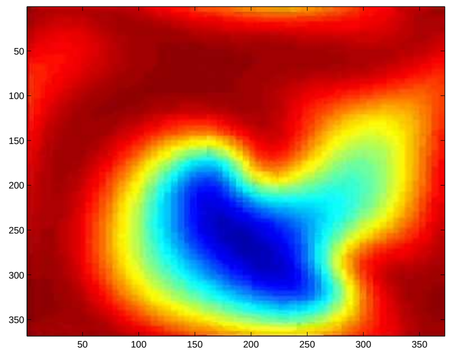
(c) $D = 0.1$



(d) $D = 0.2$

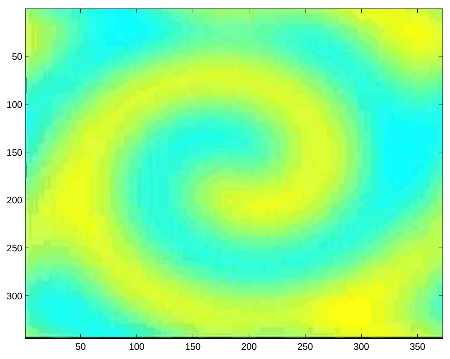


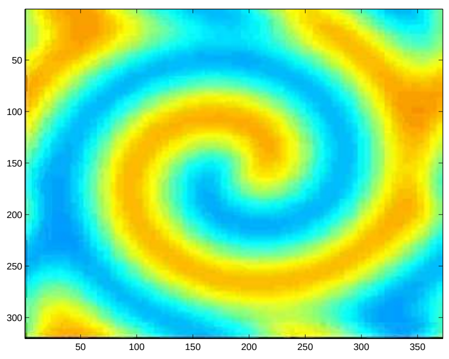
(e) $D = 0.4$

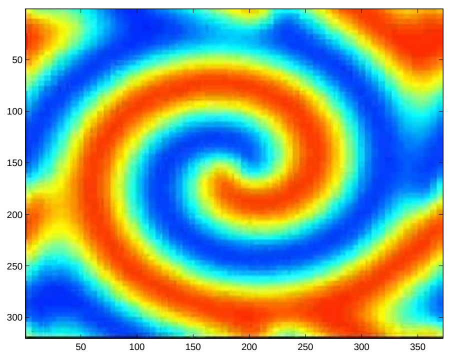


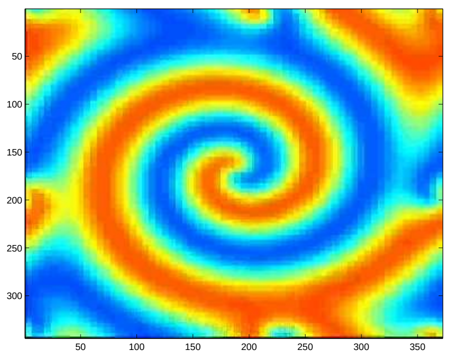
(f) $D = 0.8$

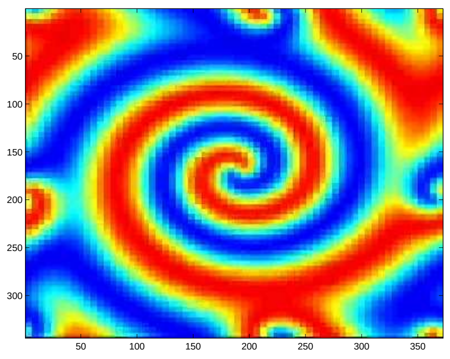
Figure 6: Effect of varying diffusion parameter D. Frames shown are at time $t = 20$. Amplitude ranges from -1 (blue) to 1 (red).


(a) $\epsilon = 0.1$ range $= \pm 0.2$


(b) $\epsilon = 0.25$ range $= \pm 0.3$


(c) $\epsilon = 0.5$ range $= \pm 0.5$


(d) $\epsilon = 1.5$ range $= \pm 1.2$


(e) $\epsilon = 2.5$ range $= \pm 1.5$

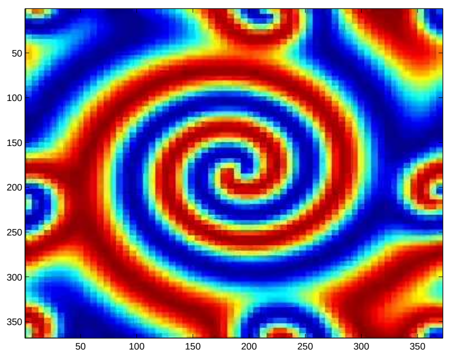
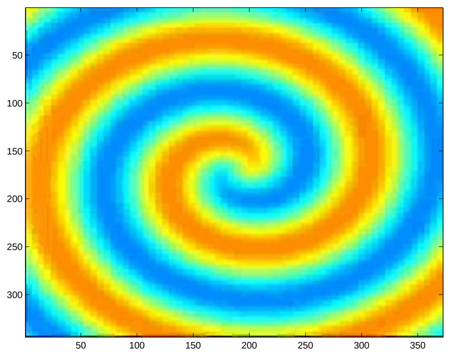
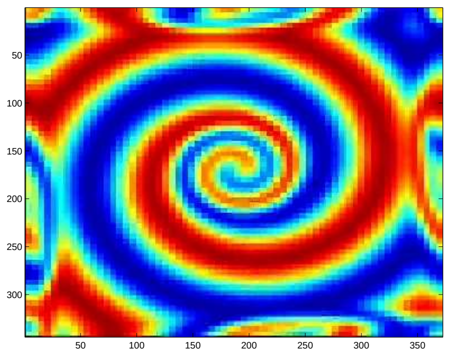
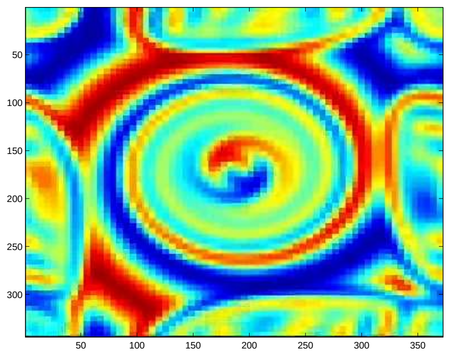
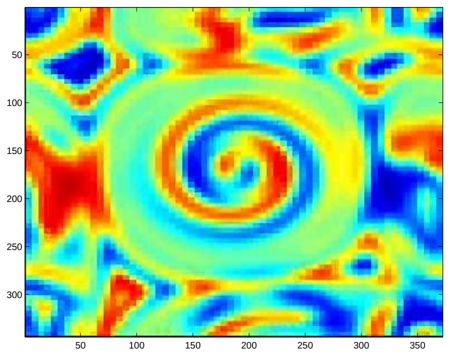

(f) $\epsilon = 4.0$ range $= \pm 2.0$

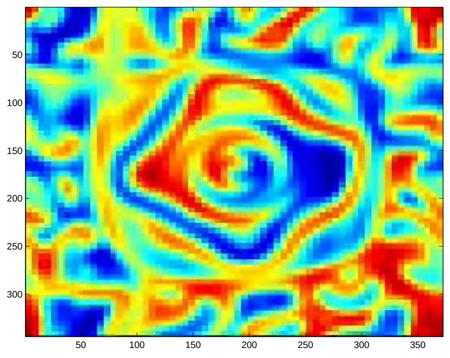
Figure 7: Effect of varying parameter ϵ. Frames shown are at time $t = 20$. Color range for figures (a-c) is ± 1. Color range for figures (e-f) is ± 2.


(a) $t = 0$


(b) $t = 4$


(c) $t = 8$


(d) $t = 12$


(e) $t = 16$

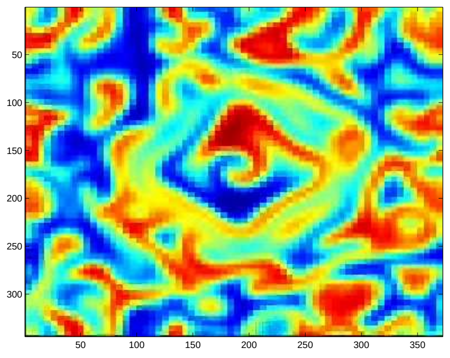
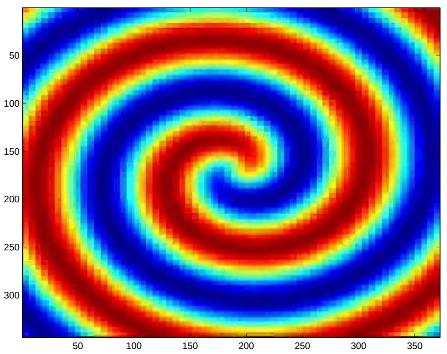
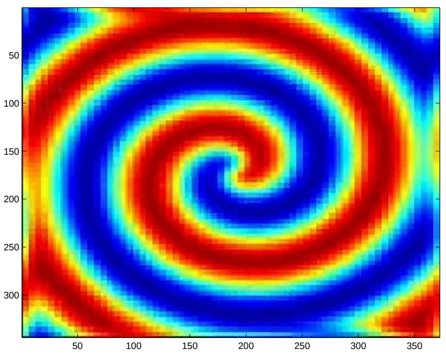

(f) $t = 20$

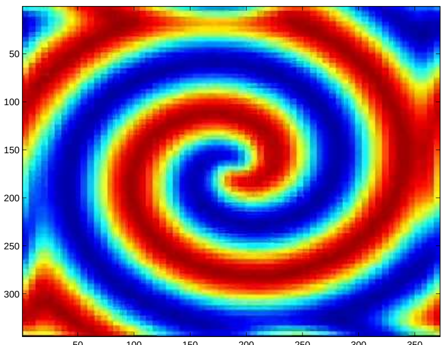
Figure 8: Chaos results from setting parameter $a = 0.25$. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 0.25$, $c = 0$, $\beta = 1$. Amplitude ranges from -2 (blue) to 2 (red).



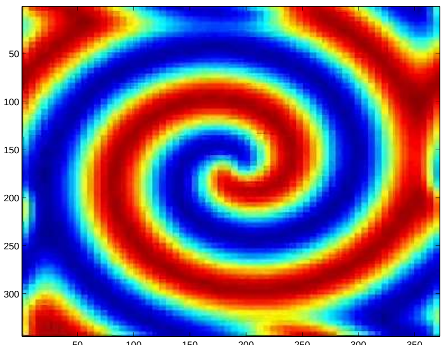
(a) $t = 0$



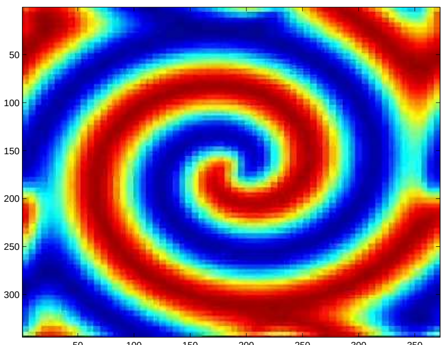
(b) $t = 1.5$



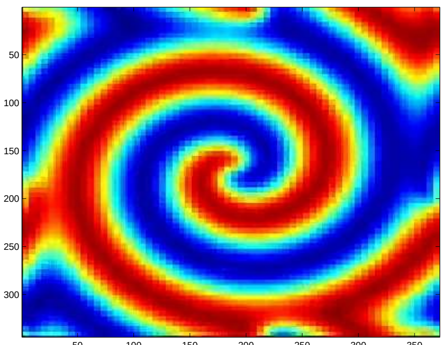
(c) $t = 3.0$



(d) $t = 4.5$

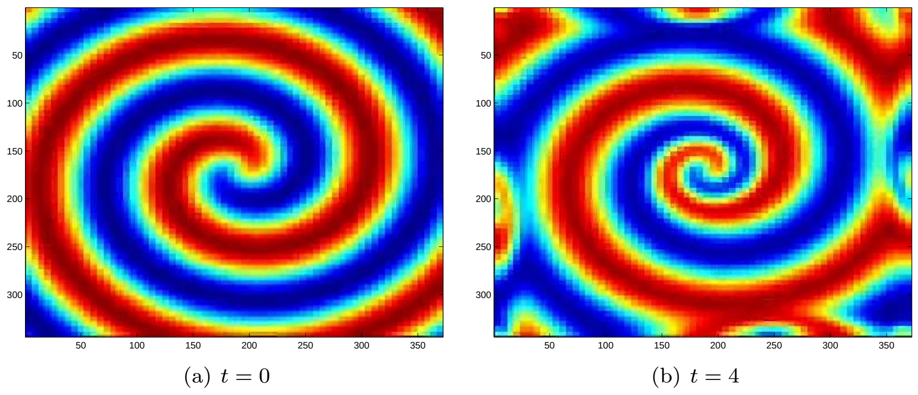


(e) $t = 6.0$

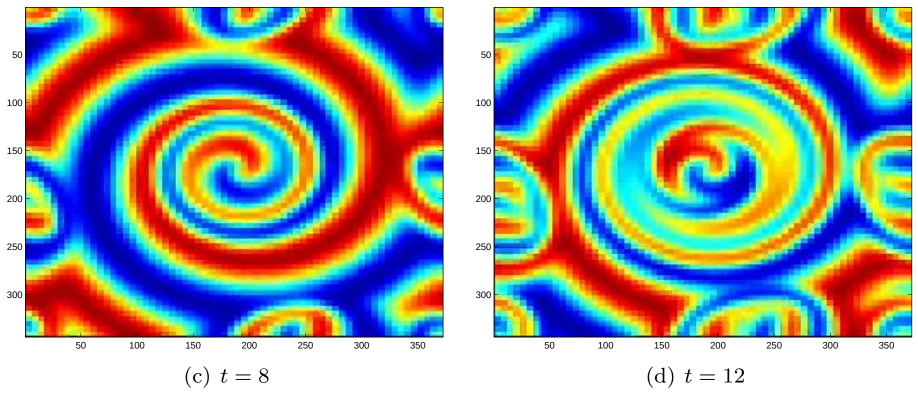


(f) $t = 7.5$

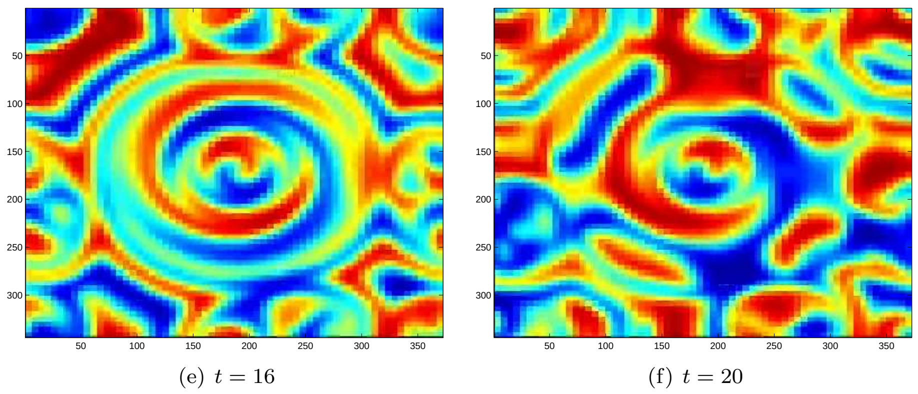
Figure 9: Clockwise rotation results from setting parameter $c = 1.5$. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 1.5$, $\beta = 1$. Amplitude ranges from -1 (blue) to 1 (red).



(a) $t = 0$ (b) $t = 4$



(c) $t = 8$ (d) $t = 12$



(e) $t = 16$ (f) $t = 20$

Figure 10: Chaos results from setting parameter $\beta = 2.5$. Parameters were $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 2.5$. Amplitude ranges from -1 (blue) to 1 (red).

5 Summary and Conclusions

The λ–ω reaction–diffusion system was solved numerically with a variety of initial conditions. Rotating one and two-armed spiral waves were observed. Rectangular grids of cells led to a complex but regular oscillating behavior. Simple sinusoidal rows produced constant linear motion. With the parameter settings of $D = 0.1$, $\epsilon = 1$, $a = 1$, $c = 0$, $\beta = 1$ all of these motions went on indefinitely without decaying.

The regular and filtered spectral methods were compared. They had similar accuracy and speed, though the filtered method was slightly faster. We would expect the filtered method to be significantly faster if higher derivatives were involved.

The effect of modifying the various parameters was examined with the one-armed spiral initial condition. The effects included

- Changing the rate of rotation.
- Changing the direction of rotation.
- Making the spiral arms shorter and fatter, or longer and thinner.
- Changing the range of solution values.

Some parameters had a single effect, while others had a combination of several. Unstable chaotic results were seen in two cases: a low value for a or a high value for β.

A MATLAB functions used

fr.m A tool used to save frames of a movie to a .eps file.

rd2.m Calculates the time evolution of the reaction-diffusion system using regular spectral transform. Produces an .avi movie file.

rd2rhs.m The right hand side used by `rd2.m` in time stepping with the `ode23` solver.

rd3.m Calculates the time evolution of the reaction-diffusion system using filtered spectral transform. Produces an .avi movie file.

rd3rhs.m The right hand side used by `rd3.m` in time stepping with the `ode23` solver.

B MATLAB code

B.1 fr.m

```
function fr(mov,fn,fnam)
%mov=name of .avi file,  fn=vector of frames to display
%fnam=file name to print frames to as color .eps files
%The frame number is appended to the file name.
%fr('m1',1+5*(0:5))  shows every 5th frame
F=aviread(mov);
for j=fn
    figure(1)
    [X,Map] = frame2im(F(j));
    image(X)
    drawnow

    if nargin>=3
        framest = num2str(j);
        while(size(framest,2))<3,
          framest = ['0',framest];
          end;
        fname1 = [fnam framest '.eps'];
        print('-depsc',fname1);
        fprintf('print %s \n',fname1);
    else
        pause(0.5)
    end
end
```

B.2 rd2.m

```
%reaction-diffusion system
clear all; close all;
fixed_scale=1;  %********
nx=2^6; %******
movey=1;  %****** whether to make a movie
% note: turn off screen saver & power saving
%****** parameters
%      e     a   c  beta  d1
p=   [6.0  1.0  0.0  1.0  0.1];
%p=[p; 1.0  1.0  4.0  1.0  0.1];
init='s';  %**** initial condition
num=55;   %***** movie number
end_time=20;  %*******
```

```
scale_multiply=1;  %********
delta_t=0.5;  %****** time between frames

for k=1:size(p,1)
    start_time = cputime;
    clear F u v ut vt zt u2
    e=p(k,1); a=p(k,2); c=p(k,3); beta=p(k,4);d1=p(k,5);
    d2=d1;
    ny=nx;
    N=nx*ny;
    Lx=20; Ly=20;
    time=0;
    frame=1;
    global iter stp;
    x2=linspace(-Lx/2,Lx/2,nx+1); x=x2(1:nx); %periodic BCs
    y2=linspace(-Ly/2,Ly/2,ny+1); y=y2(1:ny);
    % wave frequencies
    kx=(2*pi/Lx)*[0:(nx/2 -1)  (-nx/2):-1]';
    kx(1)=10^(-6);
    ky=(2*pi/Ly)*[0:(ny/2 -1)  (-ny/2):-1]';
    ky(1)=10^(-6);

    % initial conditions

    us=1;
    vs=us;
    fnam=[init int2str(num) '.avi'];
    fprintf('movie %s, scale-multiply=%f, delta_t=%f\n',...
        fnam,scale_multiply,delta_t);
    fprintf('e=%f, a=%f, c=%f, beta=%f\n',e,a,c,beta);
    fprintf('n=%i d1=%f d2=%f end time=%f\n',nx,d1,d2,end_time);

    % spirals
    if init=='s'
        [X,Y]=meshgrid(x,y);
        m=1; % number of spirals
        mag=1;  % with mag=.2, it just grows to mag=1 quickly.
        u=cos(m*angle(X+i*Y)-sqrt(X.^2+Y.^2));
        u=mag*tanh(sqrt(X.^2+Y.^2)).*u;
        v=sin(m*angle(X+i*Y)-sqrt(X.^2+Y.^2));
        v=mag*tanh(sqrt(X.^2+Y.^2)).*v;
    end

    % rows
    if init=='r'
        m=2;  % number of rows
        [X,Y]=meshgrid(x*(m*pi/Lx),y*(m*pi/Lx));
        u=sin(m*X);
        v=cos(m*X);
    end

    % boxes
    if init=='b'
        m=2;  % number of boxes... even number for periodic BC
        [X,Y]=meshgrid(x*(m*pi/Lx),y*(m*pi/Lx));
        m=1; % number of rows
        % try 6 secs for m=1
        u=sin(m*X).*sin(m*Y);
        v=cos(m*X).*cos(m*Y);
    end

    z_range = scale_multiply*[min(min(u)) max(max(u))];
    figure('doublebuffer','on','backingstore','on');
    h=figure(1);
    %get(h,'Position')
    % bottom of screen is zero [left,bottom, width, height]
    set(h,'Position',[5 40   560   420]);

    global Kx Ky
    [Kx,Ky]=meshgrid(kx,ky);

    % form u & v into vector to pass to ode23
    ut=reshape(fft2(u)/N,N,1);
    vt=reshape(fft2(v)/N,N,1);
    zt=[ut;vt];
    while 1
        fprintf('%4.1f ',time);
        if (mod(floor(time/delta_t),12)==0) & (time>0)
            fprintf('\n');
        end
        % plot new solution
        ut=reshape(zt(1:N),nx,ny);
        u=real(ifft2(ut*N));
        % pcolor doesn't show last row or column, so add dummies.
        u2 = [real(u); zeros(1,nx)];
        u2 = [u2, zeros(ny+1,1)];
        pcolor(u2);
        if fixed_scale
            caxis(z_range)
        end
        colorbar;shading flat;
        drawnow;
```

```
        if movey
            F(frame) = getframe;
        end
        frame=frame+1;
        last_frame_time = time;
        if time>=end_time
            fprintf('\nend time reached\n');
            break
        end

        [t,zsol]=ode23('rd2_rhs',[time,time+delta_t],zt,[],...
            d1,d2,nx,ny,e,c,a,beta);
        time=t(end);
        % note: ' gives complex conjugate, so use .' instead
        zt=zsol(end,:).';  %last row corresponds to current time

    end
    fprintf('cputime = %f\n',cputime - start_time);
    if movey & frame>2
        fprintf('movie file %s\n',fnam);
        movie2avi(F,fnam)
        num=num+1;
        close all
    end
end
```

B.3 rd2rhs.m

```
function rhs=rd2_rhs(t,zt,dummy,d1,d2,nx,ny,e,c,a,beta)
global Kx Ky
N=nx*ny;

ut=reshape(zt(1:N),nx,ny);
vt=reshape(zt(N+1:2*N),nx,ny);

% calc del^2 stuff (diffusion)
ud=-d1*(Kx.^2+Ky.^2).*ut;
vd=-d2*(Kx.^2+Ky.^2).*vt;
rhs=[reshape(ud,N,1); reshape(vd,N,1)];

% non-diffusion terms
u=reshape(real(ifft2(ut*N)),nx,ny);
v=reshape(real(ifft2(vt*N)),nx,ny);
A2=u.*u +v.*v;
lambda=e-a*A2;
```

```
omega=c-beta*A2;
up=lambda.*u - omega.*v;
dn=omega.*u + lambda.*v;
up=reshape(fft2(up)/N,N,1);
dn=reshape(fft2(dn)/N,N,1);
rhs=rhs+[up;dn];
%fprintf('max(rhs)=%f\n',max(abs(rhs)));
```

B.4 rd3.m

```
%reaction-diffusion system
% Filtered pseudo-spectral technique.
clear all; close all;
fixed_scale=1;  %********
d1=0.1;  %********
nx=2^6; %******
movey=1;  %****** whether to make a movie
%****** parameters
% note: turn off screen saver & power saving
%      e      a    c  beta
p=    [1.0  1.0  0.0  1.0];
%p=[p; 1.0  1.0  0.0  1.0];
num=2;   %***** movie number
end_time=20;  %*******
scale_multiply=1;  %********
delta_t=0.5;  %****** time between frames

for k=1:size(p,1)
    start_time = cputime;
    clear F u v ut vt zt u2
    e=p(k,1); a=p(k,2); c=p(k,3); beta=p(k,4);
    ny=nx;
    N=nx*ny;
    Lx=20; Ly=20;
    time=0;
    frame=1;
    global iter stp;
    x2=linspace(-Lx/2,Lx/2,nx+1); x=x2(1:nx); %periodic BCs
    y2=linspace(-Ly/2,Ly/2,ny+1); y=y2(1:ny);
    % wave frequencies
    kx=(2*pi/Lx)*[0:(nx/2 -1)  (-nx/2):-1]';
    kx(1)=10^(-6);
    ky=(2*pi/Ly)*[0:(ny/2 -1)  (-ny/2):-1]';
    ky(1)=10^(-6);
```

```
% initial conditions
fprintf('movie %i, scale-multiply=%f, delta_t=%f\n',...
    num,scale_multiply,delta_t);
fprintf('e=%f, a=%f, c=%f, beta=%f\n',e,a,c,beta);
fprintf('n=%i d1=%f end time=%f\n',nx,d1,end_time);

% spirals
if 1
    [X,Y]=meshgrid(x,y);
    m=1; % number of spirals
    mag=1;  % with mag=.2, it just grows to mag=1 quickly.
    u=cos(m*angle(X+i*Y)-sqrt(X.^2+Y.^2));
    u=mag*tanh(sqrt(X.^2+Y.^2)).*u;
    v=sin(m*angle(X+i*Y)-sqrt(X.^2+Y.^2));
    v=mag*tanh(sqrt(X.^2+Y.^2)).*v;
end

% rows
if 0
    m=2;  % number of rows
    [X,Y]=meshgrid(x*(m*pi/Lx),y*(m*pi/Lx));
    u=sin(m*X);
    v=cos(m*X);
end

% boxes
if 0
    m=2;  % number of boxes... even number for periodic BC
    [X,Y]=meshgrid(x*(m*pi/Lx),y*(m*pi/Lx));
    m=1; % number of rows
    % try 6 secs for m=1
    u=sin(m*X).*sin(m*Y);
    v=cos(m*X).*cos(m*Y);
end

z_range = scale_multiply*[min(min(u)) max(max(u))];
figure('doublebuffer','on','backingstore','on');

h=figure(1);
%get(h,'Position')
% bottom of screen is zero  [left,bottom, width, height]
set(h,'Position',[5 40   560   420]);

% form u & v into vector to pass to ode23
ut=reshape(fft2(u)/N,N,1); %/N ?
vt=reshape(fft2(v)/N,N,1);
```

```
zt=[ut;vt];
global kg
[Kx,Ky]=meshgrid(kx,ky);
kg=d1*(Kx.^2+Ky.^2)-e;
while 1
    fprintf('%4.1f ',time);
    if (mod(floor(time/delta_t),12)==0) & (time>0)
        fprintf('\n');
    end
    % plot new solution
    gt=reshape(zt(1:N),nx,ny);
    % calc linear (exponential) terms
    ud=exp(time*kg);
    ut=gt./ud;
    u=real(ifft2(ut*N));  %ut*N?
    % pcolor doesn't show last row or column, so add dummies.
    u2 = [real(u); zeros(1,nx)];
    u2 = [u2, zeros(ny+1,1)];
    pcolor(u2);
    if fixed_scale
        caxis(z_range)
    end
    colorbar;shading flat;
    drawnow;
    if movey
        F(frame) = getframe;
    end
    frame=frame+1;
    last_frame_time = time;
    if time>=end_time
        fprintf('\nend time reached\n');
        break
    end

    options=odeset('abstol',inf);  %otherwise very slow!
    [t,zsol]=ode23('rd3_rhs',[time,time+delta_t],zt,...
        options,d1,nx,ny,e,c,a,beta);
    time=t(end);
    % note: ' gives complex conjugate, so use .' instead
    zt=zsol(end,:).';  %last row corresponds to current time

end
fprintf('cputime = %f\n',cputime - start_time);
if movey & frame>2
    fnam=['f' int2str(num) '.avi'];
    fprintf('movie file %s\n',fnam);
```

```
        movie2avi(F,fnam)
        num=num+1;
        close all
    end
end
```

B.5 rd3rhs.m

```
function rhs=rd3_rhs(t,zt,dummy,d1,nx,ny,e,c,a,beta)
global  stp kg
N=nx*ny;

% calc linear (exponential) terms
ud=exp(t*kg);
rhs=[reshape(ud,N,1); reshape(ud,N,1)];

% convert gt,ht to ut,vt
ut=reshape(zt(1:N),nx,ny)./ud;
vt=reshape(zt(N+1:2*N),nx,ny)./ud;

% non-linear terms
u=reshape(real(ifft2(ut*N)),nx,ny);  %ut*N?
v=reshape(real(ifft2(vt*N)),nx,ny);  %vt*N?
A2=u.*u +v.*v;
up=-a*A2.*u + beta*A2.*v - c*v;
dn=c*u - beta*A2.*u - a*A2.*v;
up=reshape(fft2(up)/N,N,1);  %fft(up)/N ?
dn=reshape(fft2(dn)/N,N,1);
rhs=rhs.*[up;dn];
%fprintf('max(rhs)=%f\n',max(abs(rhs)));
```