Probing a Simplified Model of the Belousov-Zhabotinsky Reaction

18.300 Continuum Applied Mathematics

Jade Chongsathapornpong
MIT, Cambridge, MA 02139
(Dated: January 6, 2023)

I. INTRODUCTION

Excitable media are those which, from an initial quiescent state, exhibit traveling waves excited by defects or perturbations which exceed some threshold. After a period of recovery, the system supports further waves [1]. The Belousov-Zhabotinsky (BZ) reaction exhibits oscillatory behavior which can produce such waves, including visually striking spirals. The system of reactions is highly complex; an early attempt at explaining the phenomenon involved numerous reaction steps between 11 species [2]. Fortunately, it was found that the dynamics of excitable media can be captured with simplified models. The so-called Oregonator model [3] is perhaps the most well-known, and is based on studies of the chemical mechanism. In this project, we will explore a less-studied simplified model proposed by science writer Philip Ball, which is claimed to vaguely model the BZ system. We'll approach this analytically and numerically, to characterize the simplified model and attempt to reproduce traveling and spiral waves.

Code for this project can be found here:
`https://github.com/Certaingemstone/reactdiff`

II. REACTION-DIFFUSION MODEL

The underlying mechanism of the BZ reaction is commonly understood as the interplay between two competing processes which dominate under different conditions, in particular dependent on the concentration of bromide ions in solution. Under high bromide concentrations, one process consumes bromide ions monotonically, until descending through a certain threshold concentration where the other process dominates. This second process rapidly oxidizes a transition metal catalyst—for example, an Fe(II) phenanthroline complex becomes Fe(III) or, in the case of the original reaction, Ce(III) becomes Ce(IV). This oxidation corresponds to a visible color change. Diffusion pushes neighboring solution through the threshold, so this color change propagates. Another set of slower reactions are catalyzed which eventually regenerate bromide ion and reduces the catalyst back to its original state and color, at which point the cycle can repeat [2].

A simplified model for this reaction was described in

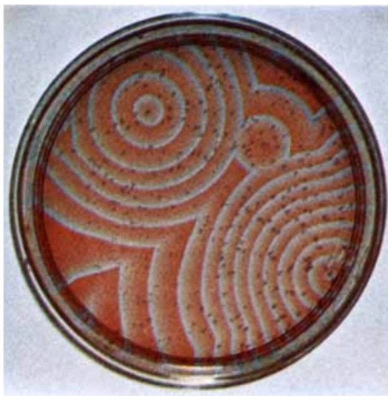

FIG. 1: Target patterns in BZ reaction, from [2]. Periodic "trigger" waves propagate from imperfections in the solution. The imperfections can under some circumstances be removed, leaving the reagents in a steady state.

the work of Field, Körös, and Noyes, who had previously investigated the pertinent chemistry. Their scheme, dubbed the "Oregonator," is as follows:

$$\begin{aligned} \mathrm{A} + \mathrm{Y} &\rightleftharpoons \mathrm{X} \\ \mathrm{X} + \mathrm{Y} &\rightleftharpoons \mathrm{P} \\ 2\,\mathrm{B} + 2\,\mathrm{X} &\rightleftharpoons 4\,\mathrm{X} + 2\,\mathrm{Z} \\ 2\,\mathrm{X} &\rightleftharpoons \mathrm{Q} \\ 2\,\mathrm{Z} &\rightleftharpoons 2\,\mathrm{Y} \end{aligned}$$

where Y corresponds to the bromide ions and Z to the oxidized transition metal species. It turns out that this model leads to a coupled system of nonlinear differential equations of which one is "stiff"—in certain regions of state space, an explicit numerical solution becomes sensitive to small errors around an exact solution [3]. This renders the equations not amenable to simple explicit solvers without prohibitively small time steps. And the problematic equation being nonlinear, an implicit scheme is likely "not a picnic," according to R. Rosales.

For the purposes of this project, we will investigate an even simpler model which no longer corresponds directly to the physical BZ mechanism, but purportedly exhibits the same excitation-recovery behavior. This model is a modified version of one due to P. Ball [4]:

$$\mathrm{A + B \rightleftharpoons 2\,A}$$
$$\mathrm{B + C \rightleftharpoons 2\,B}$$
$$\mathrm{C + A \rightleftharpoons 2\,C}$$

Here, a symmetric set of 3 reactions transform circularly between A, B, and C. Each is autocatalyzed, that is, the interaction between A and B tends to convert B into A, and so on. To write equations describing the species' time evolution, we employ the law of mass action, which proposes that reaction rates will be proportional to the rates of collision between reacting species. Assuming a homogeneous mixture, this would imply that the rates are proportional to the product of reactant concentrations. For example, the first forward reaction proceeds at rate $k_{1+}ab$, and its reverse reaction at $k_{1-}a^2$, where a, b, and c denote the respective concentrations, and k are rate constants. Combining the reactions,

$$\frac{da}{dt} = k_{1+}ab - k_{3+}ca - k_{1-}a^2$$
$$\frac{db}{dt} = k_{2+}bc - k_{1+}ab - k_{2-}b^2$$
$$\frac{dc}{dt} = k_{3+}ca - k_{2+}bc - k_{3-}c^2$$

If we neglect reverse reactions as was done in [4], effectively assuming that $k_{i+} \gg k_{i-}$, we get a simpler system. Now we introduce diffusion, allowing species to move through space along concentration gradients. We obtain the partial differential equations:

$$\frac{\partial a}{\partial t} = k_1ab - k_3ca + \nu\nabla^2 a \tag{1}$$
$$\frac{\partial b}{\partial t} = k_2bc - k_1ab + \nu\nabla^2 b \tag{2}$$
$$\frac{\partial c}{\partial t} = k_3ca - k_2bc + \nu\nabla^2 c \tag{3}$$

where the Laplacian ∇^2 tends to flatten down bumps and fill in valleys in the concentrations. We've assumed each of the species diffuse according to ν.

III. ANALYSIS

In this section we analyse the model. First, we restrict the problem to a plane in the 3-dimensional space of reactant concentrations. Then, linear stability is evaluated, revealing a critical point likely to exhibit oscillation. A numerical computation investigates the behavior of the reaction around this equilibrium.

To begin, we could nondimensionalize our equations with

$$a = \frac{\nu}{k_1}u, \quad b = \frac{\nu}{k_1}v, \quad c = \frac{\nu}{k_3}w, \quad \text{and} \quad t = \frac{1}{\nu}\tau$$

However, this breaks a symmetry in the equations, which turns out to be a pain in the neck. Instead, to begin we only scale time to eliminate ν:

$$a = \nu u, \quad b = \nu v, \quad c = \nu w, \quad \text{and} \quad t = \frac{1}{\nu}\tau$$

Note that

$$\frac{\partial}{\partial t} = \frac{\partial \tau}{\partial t}\frac{\partial}{\partial \tau} = \nu\frac{\partial}{\partial \tau}$$

Substituting these in, we get the system

$$\frac{\partial u}{\partial \tau} = u(k_1v - k_3w) + \nabla^2 u \tag{S.1}$$
$$\frac{\partial v}{\partial \tau} = v(k_2w - k_1u) + \nabla^2 v \tag{S.2}$$
$$\frac{\partial w}{\partial \tau} = w(k_3u - k_2v) + \nabla^2 w \tag{S.3}$$

Then we'll say $t \equiv \tau$ for convenience.

Note that the equations are not hyperbolic, and so the method of characteristics which we have studied in class unfortunately will not help us here.

A. Dimension Reduction

Now, we should try to see how this system behaves, and whether it's a reasonable model for an excitable medium. For traveling waves, we need the medium to be able to "reset" after a wide excursion through its state space. If it could get "stuck" after one wave passes through, we have a problem. We might look for a limit cycle, that is, a closed trajectory in its state space which has other trajectories spiraling into it. Or, there might be a center, around which a family of closed trajectories exists.

Let's imagine a spatially homogeneous state, so that diffusion can be neglected in S.1-S.3. Firstly, notice that the problem can be simplified with a change of coordinates. Intuitively from the underlying chemical mechanism, each unit of species A, B, and C can only transform into one unit of a different species; the overall sum should be conserved. Formally, define the total amount of reactants

$$R(t) \equiv u + v + w$$

for which the spatially homogeneous system gives the time derivative

$$\dot{R} = k_1uv - k_3uw + k_2vw - k_1uv + k_3uw - k_2vw = 0$$

implying the total R is conserved, as expected. This means that instead of the full (u, v, w)-space, our system is restricted to a certain affine subset—a plane parameterized by R. Let's define a coordinate system on this space.

The plane for a given R includes the points $(R,0,0)$, $(0,R,0)$, and $(0,0,R)$. Thus a normal vector for it is $(1,1,1)$, and a vector in the plane is $(0,R,0)-(R,0,0) \propto (-1,1,0)$. We can obtain an orthogonal basis with one more vector from the cross product,

$$(1,1,1)\times(-1,1,0) = (-1,-1,2)$$

Now note that $(R/3,R/3,R/3)$ is a point in the plane, which we'll define as the origin. Call $S \equiv R/3$. This lets us introduce new variables, (ξ,χ), to parameterize the plane as follows:

$$(u,v,w) = (S,S,S)+\xi(-1,-1,2)+\chi(-1,1,0) \tag{4}$$

To go in the opposite direction, obtaining χ and ξ for a given (u,v,w), we solve the system

$$\begin{aligned} u &= S-\chi-\xi \\ v &= S+\chi-\xi \\ w &= S+2\xi \end{aligned}$$

These are linearly dependent. Solving gives

$$(\xi,\chi) = \left(\frac{w-S}{2}, \frac{2v+w-3S}{2}\right) \tag{5}$$

Of course we can also calculate time derivatives, recalling that $S \propto R$ is constant.

$$\begin{aligned} u_t &= -\chi_t-\xi_t \\ v_t &= \chi_t-\xi_t \\ w_t &= 2\xi_t \end{aligned}$$

If we substitute into the space-independent S.1-S.3,

$$\begin{aligned} -\chi_t-\xi_t =& k_1(S-\chi-\xi)(S+\chi-\xi) \\ &- k_3(S-\chi-\xi)(S+2\xi) \\ \chi_t-\xi_t =& k_2(S+\chi-\xi)(S+2\xi) \\ &- k_1(S-\chi-\xi)(S+\chi-\xi) \\ 2\xi_t =& k_3(S-\chi-\xi)(S+2\xi) \\ &- k_2(S+\chi-\xi)(S+2\xi) \end{aligned}$$

we obtain the derivative ξ_t from the third equation and χ_t by subtracting the first from the second.

$$\begin{aligned} 2\xi_t =& k_3(S-\chi-\xi)(S+2\xi) \\ &- k_2(S+\chi-\xi)(S+2\xi) \end{aligned} \tag{Q.1}$$

$$\begin{aligned} 2\chi_t =& k_2(S+\chi-\xi)(S+2\xi) \\ &- 2k_1(S-\chi-\xi)(S+\chi-\xi) \\ &+ k_3(S-\chi-\xi)(S+2\xi) \end{aligned} \tag{Q.2}$$

B. Linear Stability

A 2-dimensional system in hand, we look for the critical points, where the derivatives ξ_t and χ_t vanish. From above, we see that ξ_t and χ_t are linear combinations of u_t, v_t, and w_t. Therefore we can equivalently look for the stationary points of the 3-dimensional system for the appropriate value of R. Note that (u,v,w) remains in the positive octant; concentrations begin positive, and S.1-S.3 will not allow them to cross zero. In the first octant, by inspection we have solutions for (u,v,w),

$$\tilde{X}_1 = (R,0,0), \quad \tilde{X}_2 = (0,R,0), \quad \tilde{X}_3 = (0,0,R) \tag{6}$$

No equilibria exist where just one component is 0; this would clearly require one of the other two to be zero to be consistent. Cross-substituting, we also find the line

$$\tilde{X}_0^* = \left(r^*, \frac{k_3}{k_2}r^*, \frac{k_1}{k_2}r^*\right)$$

where r^* is such that the components sum to R. Since R is arbitrary, so is r^*, and we're free to scale along the line by k_2 to get a nicer looking point,

$$\tilde{X}_0 = (k_2 r,\ k_3 r,\ k_1 r) \tag{7}$$

where r is defined by $r(k_1+k_2+k_3) = R$. Let's fix R and use 5 to transform to (ξ,χ).

$$X_1 = \left(-\frac{S}{2}, -\frac{3S}{2}\right) \tag{8}$$

We could do the same for $\tilde{X}_2$ and $\tilde{X}_3$, but due to symmetry in the equations, the qualitative behavior of each will be the same as X_1. As for $\tilde{X}_0$, in (ξ,χ) coordinates

$$X_0 = \left(\frac{k_1 r - S}{2}, \frac{2k_3 r + k_1 r - 3S}{2}\right)$$

which for consistency we can write in terms of S or R, as

$$X_0 = \left(\frac{S(2k_1-k_2-k_3)}{2(k_1+k_2+k_3)}, \frac{3S(k_3-k_2)}{2(k_1+k_2+k_3)}\right) \tag{9}$$

Next, we'll look at the linear stability of these points, so we linearize Q.1-Q.2 for small perturbations around the equilibria. That is,

$$\begin{pmatrix}\xi \\ \chi\end{pmatrix}_t = \begin{pmatrix}(\xi_t)_\xi & (\xi_t)_\chi \\ (\chi_t)_\xi & (\chi_t)_\chi\end{pmatrix}\begin{pmatrix}\delta\xi \\ \delta\chi\end{pmatrix} = J\begin{pmatrix}\delta\xi \\ \delta\chi\end{pmatrix} \tag{10}$$

where $\delta\xi$ and $\delta\chi$ are small deviations from an equilibrium, and the Jacobian is evaluated at that equilibrium. We calculate the derivatives:

$$2\xi_{t\xi} = -k_2(S+2\chi-4\xi)+k_3(S-2\chi-4\xi)$$

$$2\xi_{t\chi} = -k_2(S+2\xi) - k_3(S+2\xi)$$

$$2\chi_{t\xi} = 4k_1(S-\xi) + k_2(S+2\chi-4\xi) + k_3(S-2\chi-4\xi)$$

$$2\chi_{t\chi} = 4k_1\chi + k_2(S+2\xi) - k_3(S+2\xi)$$

Let's start with X_1, for which

$$S+2\chi-4\xi = 0, \quad S-2\chi-4\xi = 6S = 2R,$$
$$S+2\xi = 0, \quad S-\xi = \frac{3S}{2} = \frac{R}{2}$$

yielding

$$J_1 = R\begin{pmatrix} k_3 & 0 \\ (k_1+k_3) & -k_1 \end{pmatrix}$$

with eigenvalues, one of which is positive,

$$\lambda_1 \propto -k_1 \quad \text{and} \quad k_3$$

Then for X_0, I used the Sage computer algebra system to evaluate the derivatives.

$$J_0 = \frac{R}{2(k_1+k_2+k_3)}\begin{pmatrix} k_1k_2 - k_1k_3 & -k_1k_2 - k_1k_3 \\ k_1k_2 + k_1k_3 + 4k_2k_3 & k_1k_3 - k_1k_2 \end{pmatrix}$$

with purely imaginary eigenvalues

$$\lambda_0 \propto \pm 2i\sqrt{k_1^2k_2k_3 + k_1k_2^2k_3 + k_1k_2k_3^2}$$

These linearizations suggest that the equilibria X_1-X_3 at the corners of the plane are unstable; a certain direction of perturbation has a positive eigenvalue, and therefore grows. This is good, as to produce waves, the system cannot get "stuck" anywhere. Meanwhile, with purely imaginary eigenvalues, the equilibrium X_0 in the middle exhibits promising oscillatory behavior; no small perturbation direction tends to grow or shrink.

C. Behavior near X_0

The linear analysis suggests we should investigate the behavior of solutions near X_0. *A priori*, due to the Poincaré-Bendixson theorem [6], we can infer that these solutions should have periodic cycles and no chaotic behavior. However, the system is nonlinear, and we cannot expect the linearization to capture its dynamics farther from the critical point; whether limiting periodic behavior—a limit cycle—exists, is unclear to me. To figure this out, we'll construct a phase portrait numerically.

We integrate the equations starting from points near the equilibrium using the common Runge-Kutta 4th order scheme, with my previous implementation in C++ (`/misc/rk4.cpp`). A subset of the points from each trajectory is plotted (`/misc/phase.py`) in Figure 2.

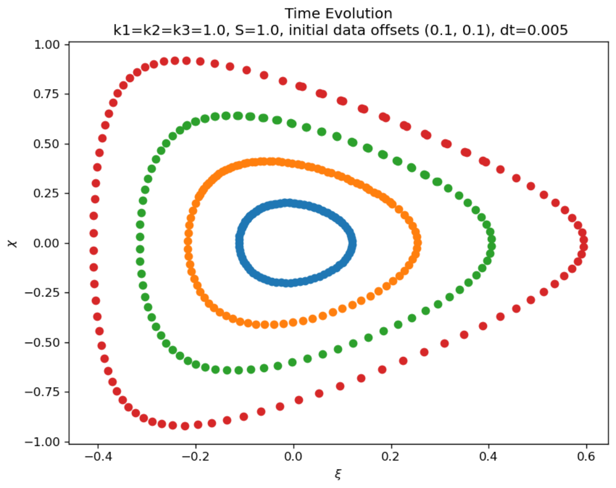

FIG. 2: Phase portrait of system around X_0 in $\xi-\chi$ plane for all rate constants equal. Trajectories start at the equilibrium point plus multiples of $(0.1, 0.1)$. The equilibrium point behaves as a center, with a continuous set of closed orbits.

We can see that there exist closed orbits around the equilibrium. No trajectories seem to eventually spiral into a specific orbit, that is, there is no limiting periodic behavior. The period of these oscillations depends on the initial conditions, as shown in Figure 3—different orbits take different amounts of time to close, and specifically larger deviations from equilibrium take longer.

This reveals a distinction between the model in [4] and the physically-based Oregonator model. According to experimental characterization in [5], BZ reactions exhibit limit cycle behavior; the reaction should eventually approach oscillations with amplitude and period determined only by the rates of the reaction mechanism, independent of initial concentrations [3]. Qualitatively, this is very different from what we see in our simplified model, where perturbations of the system will shift the orbit permanently, and the orbits depend not only on the rate constants, but also reactant species concentrations.

D. Wavefront Propagation

That traveling waves should arise when diffusion is reintroduced to the system is not necessarily clear. Unfortunately, with still two second-order equations, I'm not sure how to study this analytically. Our discovery of closed orbits implies the existence of yet another conserved quantity in the system. This would permit us to reduce the dimension once more, and from there it may be possible to look for traveling wave solutions analytically. Unfortunately, it is not obvious what this quantity is, so we will move on to purely numerical means.

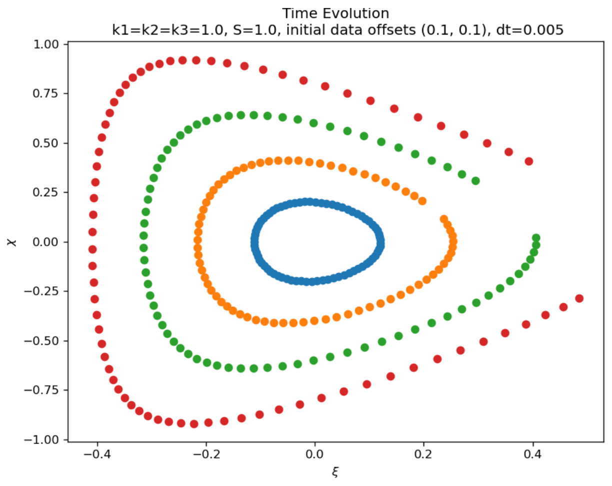

FIG. 3: First 400 time steps (plotted every 5th) of phase portrait. Notice the period of oscillations differs between initial states.

IV. SIMULATION

Here, we attempt to implement the reaction-diffusion model *in silico* with the goal of reproducing traveling and spiral waves from the model. First, we build a crude implementation on a coarsely discretized system, in the form of 2D cellular automata. Second, we construct an implicit-explicit scheme for evolving the system.

A. Cellular Automata

It was suggested by R. Rosales that the wave phenomena we're looking for can appear even in a system where diffusion is replaced by an approximation—averaging neighbors—on a square discrete grid. Indeed, cellular automata have been used to simulate the BZ reaction, and the model we've considered. We build the automata described in [4] with some modifications.

The idea is simple: for each square on a grid, we have a value of u, v, and w. For the next time, we calculate an explicit step using S.1-S.3. Then, to emulate diffusion, use the average of neighboring cells. In our version, we change by a fraction of the difference between a cell and the average of its neighbors, to mimic an exponential relaxation. It is noted in [4] that one should constrain concentrations to a fixed range, a constraint "included so that the values stay in range for the color display." In fact, we found this clamping was important for stability; their method implicitly uses a large timestep, which can explode. In ours, a timestep is explicitly specified.

The algorithm was implemented in Rust (`/src/bin/automaton.rs`). Figures 4 and 5, show that the diffusion works as expected, and that when

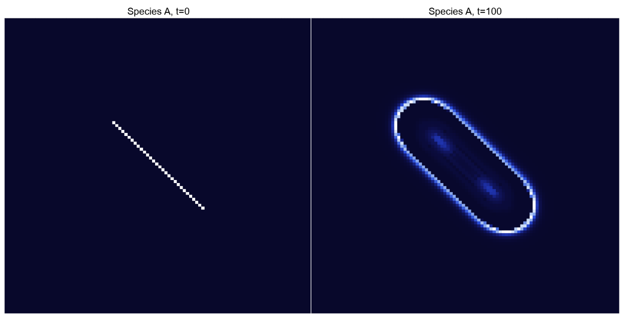

FIG. 4: Time evolution of a line of initial concentrations $(4, 6, 8)$ in cellular automaton, with weak diffusion and $k_1 = k_2 = k_3$. The concentration of species A is shown initially (left) and after 100 iterations (right). A wave propagates outwards and diffuses over time, and oscillations occur in the enclosed region. When diffusion is stronger, the front quickly loses its sharpness.

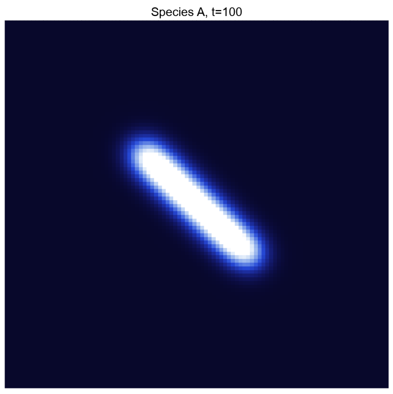

FIG. 5: The effects of diffusion only, under the same conditions as Figure 4.

the reaction mechanism is present, a wavefront forms. In Figure 6 and 7, the cells are initialized with random concentrations, and evolve 100 time steps. Traveling waves form into target and spiral patterns as claimed.

The spirals in Figure 6 don't look like those in the actual BZ reaction; they are much denser and less organized. It may be that the difference in periodic behavior between this model and the Oregonator contributes. On the other hand, it's believed that small perturbations originate waves [2], so it is reasonable that our random initial conditions produce many more sources than would appear in reality. Then because colliding wavefronts annihilate in this medium, the waves cannot propagate long distances, forming this dense structure.

As an aside, in future work it may be interesting to explore what kinds of initial conditions start a spiral or target pattern, by initializing smaller patches of the domain randomly, seeing how they evolve, and characterizing the random patches that yield a given long term behavior.

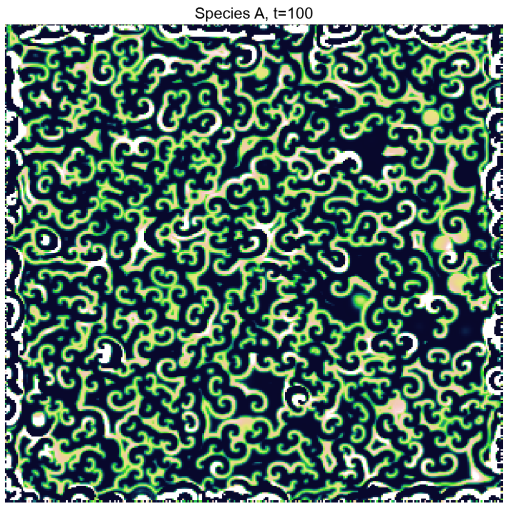

FIG. 6: Results from randomly initialized cells, with weak diffusion (0.3) and $k_1 = k_2 = k_3 = 1$. Dense spiraling patterns form, with many sources.

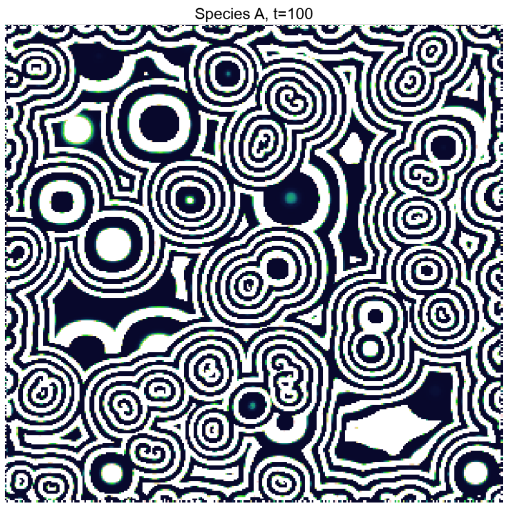

FIG. 7: Results from randomly initialized cells, with weak diffusion (0.3) and $k_1 = 1$, $k_2 = k_3 = 0.7$. Note the similar wavelengths of traveling target pattern waves.

Interestingly, as is most apparent in Figure 7, most of the centers appear to emit waves with a very similar period and wavelength, despite the continuum of periods available in the reaction system. This suggests other factors at play, perhaps some balance between the diffusion rate and the reaction rates which prefers a certain orbit as in Figure 2, thereby determining a wavelength and speed. A study to perform would be to vary the diffusion rate alone, and see how this affects the waves. In brief testing, a faster diffusion rate (0.5) produced thicker waves, and a slower one (0.1) led to more and thinner spiral structures appearing. More systematic characterization would be necessary for insight into the mechanism.

B. Implicit-Explicit

Finally, we attempt a solution to the partial differential equations numerically via an implicit-explicit (ImEx) scheme. The general idea rests on fractional step methods which, as described in class, integrate differential equations with multiple terms: For an equation of form

$$\frac{dy}{dt} = f(y) + g(y)$$

one can evolve the solution $y \to y^*$ with a step corresponding to $f(y)$, and then from the new y^* take another step corresponding to $g(y^*)$. This allows the mixing of different schemes to compute different terms.

This is advantageous because the linear diffusion terms in our equations are known to be "stiff," so we would like to use an implicit method, which would have the key feature of being stable. However, the nonlinear reaction terms are expensive to solve implicitly, so we will use a simpler explicit step.

Like in the cellular automata, we discretize space and time. Only now, we introduce diffusion by considering derivatives, instead of averaging neighbors. To do this, approximate derivatives are taken at a given point using finite differences, e.g. between neighbors in space-time.

The ImEx method is to be implemented in Rust. Explicit steps are calculated with Runge-Kutta 2nd order, and then trapezoidal rule is used for the implicit steps.

As of 4/29/2022, I have not had the time to figure out the implicit scheme. I may try to do so before the last day of classes, but for now, I'm submitting this section as is.

V. SUMMARY

Although we've discovered that Philip Ball's simplified model of the Belousov-Zhabotinsky reaction as described by [4] is rather different from the BZ reaction and its physically-based "Oregonator" model, the system is regardless rich and interesting. A cursory examination found that the system conserves total reactant, reducing its dimension and restricting possible behaviors. We calculated all reachable equilibria to be linearly unstable, except one with oscillations which we deduced to yield closed periodic orbits that contribute to wave phenomena. Computation confirmed this and did not find a limit cycle, leading us to conclude the system is qualitatively different from the BZ reaction. However, simulation in cellular automata showed it still manages to produce traveling and spiral wave phenomena in a manner reminiscent of the real thing, and also suggests some more interesting things to investigate!

[1] Zykov, V. S., 2018. Spiral wave initiation in excitable media. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2135), p.20170379.

[2] Winfree, A. T., 1974. Rotating Chemical Reactions. *Scientific American*, 230(6), pp.82-95.

[3] Field, R. & Noyes, R., 1974. Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *The Journal of Chemical Physics*, 60(5), pp.1877-1884.

[4] Turner, A., 2009. A Simple Model of the Belousov-Zhabotinsky Reaction from First Principles. Implementation note.

[5] Ganapathisubramanian, N. et al., 1980. Limit cycle behaviour in the Belousov-Zhabotinskii oscillatory reaction. *Proc. Indian Acad. Sci.*, 89(3), pp.235-239.

[6] Given a differential equation in the plane, for $t \to \infty$, any given solution $x(t)$ which remains in a bounded region must converge to either a fixed point or a periodic cycle. Our solutions are constrained to a plane, and bounded by its intersections with the boundaries of the first octant. Additionally, our linearization suggests they will not descend to the fixed point.

Appendix A: Abridged rk4

```
namespace rk4
{
        rk4Solve::rk4Solve(std::vector<double> state, double time, double timestep,
            std::vector<double (*)(std::vector<double>, double)> inFunctions)
                : t(time), dt(timestep), n(state.size()), stateCurrent(state), derivatives(inFunctions)

        std::vector<double> rk4Solve::iterate()
        {
                std::vector<double> stateNext (n), stateIntermediate (n);
                std::vector<double> k1 (n), k2 (n), k3 (n), k4 (n);

                //Obtain k1 (derivative vector at current state)
                for (int i = 0; i < n; i++) {
                        k1[i] = derivatives[i](stateCurrent, t);
                }

                //Obtain k2 (derivative vector at half timestep, linear from stateCurrent along k1)
                for (int i = 0; i < n; i++) {
                        stateIntermediate[i] = stateCurrent[i] + k1[i] * dt / 2.0;
                }
                for (int i = 0; i < n; i++) {
                        k2[i] = derivatives[i](stateIntermediate, t + dt/2.0);
                }

                //Obtain k3 (derivative vector at half timestep, linear from stateCurrent along k2)
                for (int i = 0; i < n; i++) {
                        stateIntermediate[i] = stateCurrent[i] + k2[i] * dt / 2.0;
                }
                for (int i = 0; i < n; i++) {
                        k3[i] = derivatives[i](stateIntermediate, t + dt/2.0);
                }

                //Obtain k4 (derivative vector at full timestep, linear from stateCurrent along k3)
                for (int i = 0; i < n; i++) {
                        stateIntermediate[i] = stateCurrent[i] + k3[i] * dt;
                }
                for (int i = 0; i < n; i++) {
                        k4[i] = derivatives[i](stateIntermediate, t + dt);
                }

                //Update stateNext using k
                for (int i = 0; i < n; i++) {
                        stateNext[i] = stateCurrent[i]
                        + ((k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6.0) * dt;
                }
                //Update stateCurrent and time
                stateCurrent = stateNext;
                t = t + dt;
                return stateNext;
        }
}
```

```
// client code

//Derivative functions
//x and y are functions of the three reactant concentrations and the sum of concentration
double xprime(std::vector<double> x, double time) {
    double xp = (k3 * (S - x[1] - x[0]) * (S + 2 * x[0])) -
        (k2 * (S + x[1] - x[0]) * (S + 2 * x[0]));
    return xp;
}

double yprime(std::vector<double> x, double time) {
    double yp = k2 * (S + x[1] - x[0]) * (S + 2 * x[0]) -
        2 * k1 * (S - x[1] - x[0]) * (S + x[1] - x[0])
        + k3 * (S - x[1] - x[0]) * (S + 2 * x[0]);
    return yp;
}

int main()
{
    int niter;
    int npoints = 1; // number of starting points
    double dx = 0.1; // size of step away from equilibrium point
    double dt;
    // Initial state
    double x0, y0;

    // equilibrium position
    x0 = S * (2 * k1 - k2 - k3) / (2 * (k1 + k2 + k3));
    y0 = 3 * S * (k3 - k2) / (2 * (k1 + k2 + k3));

    // differential equations
    std::vector<double (*)(std::vector<double>, double)> diffeqs;
    diffeqs.push_back(xprime);
    diffeqs.push_back(yprime);

    // solve and output
    for (int k = 1; k < npoints + 1; k++) {
        std::string fname = std::to_string(k);
        // Initialize solver
        std::vector<double> init{ x0 + k*dx, y0 + k*dx };
        std::cout << "Starting at " << init[0] << " " << init[1] << std::endl;
        rk4::rk4Solve Solver(init, 0.0, dt, diffeqs);

        for (int j = 0; j < niter; j++) {
            Solver.iterate();
            for (double x : Solver.stateCurrent) {
                std::cout << x << " ";
                file << x << ",";
            }
            file << "\n";
            std::cout << std::endl;
        }} return 0; }
```

Appendix B: Abridged automaton and reactdiff

```
// client code: automaton.rs
let mut grid = reactdiff::ReacDiffGrid::random(300, 3);
for t in 0..time_steps {
    grid.diffuse(relaxation);
    if !diff_only {grid.ball_model([k1, k2, k3, DT]);}
    println!("Time: {}", t);
}

// library: reactdiff
use rand::Rng;
use ndarray::{array, s, Array2, Array3};

// Store the concentrations of A,B,C on a 2D arrays
pub struct ReacDiffGrid {
    pub width: usize,
    pub species: usize,
    pub a: Array3<f32>, // first two dimensions are position, third is species
}

// Initialize
impl ReacDiffGrid {
    // create with zeroes
    pub fn new(width: usize, species: usize) -> Self {
        Self{a: Array3::<f32>::zeros([width, width, species]), width: width, species: species}
    }

    // create with random values between 0 and 1
    pub fn random(width: usize, species: usize) -> Self {
        let mut rng = rand::thread_rng();
        let mut grid = ReacDiffGrid::new(width, species);
        for elem in grid.a.iter_mut() {
            *elem += rng.gen::<f32>();
        }
        grid
    }

    // fill a block
    pub fn write(&mut self, xrange: std::ops::Range<usize>, yrange: std::ops::Range<usize>, spec_idx: us
        for x in xrange {
            let yr = yrange.clone();
            for y in yr {
                self.a[(x,y, spec_idx)] = value;
            }
        }
    }

    // fill a block randomly
    pub fn write_random(&mut self, xrange: std::ops::Range<usize>, yrange: std::ops::Range<usize>, spec_
        let mut rng = rand::thread_rng();
        for x in xrange {
            let yr = yrange.clone();
            for y in yr {
                self.a[(x,y, spec_idx)] = rng.gen::<f32>() * scale;
            }
        }
    }
}

// Evolution
impl ReacDiffGrid {

    // Apply ball reaction model to each cell
    // params: k1, k2, k3, dt
    pub fn ball_model(&mut self, params: [f32; 4]) {
        // du/dt = u(k1*v - k3*w)
        // dv/dt = v(k2*w - k1*u)
        // dw/dt = w(k3*u - k2*v)
        assert_eq!(self.a.shape()[2], 3);
        // for each cell, calculate evolution and add it
        for mut x in self.a.rows_mut() {
            // TODO: vectorize
            let du = x[0] * (params[0]*x[1] - params[2]*x[2]) * params[3];
            let dv = x[1] * (params[1]*x[2] - params[2]*x[0]) * params[3];
            let dw = x[2] * (params[2]*x[0] - params[2]*x[1]) * params[3];
            x += &ndarray::ArrayView::from(&array![du, dv, dw]);
            // clamp values
            for elem in &mut x {
                if *elem < 0.0 {
                    *elem = 0.001;
                }
                if *elem > 10.0 {
                    *elem = 10.0;
                }
            }
        }
    }

    // Apply diffusion, with relaxation towards local average defined by relax
    // e.g. relax 0.5 -> if currently 2 and average is 1, next value is 1.5
    pub fn diffuse(&mut self, relax: f32) {
        assert!(relax > 0. && relax < 1.);
        // calculate averages with hard boundary condition
        let mut d: Array2<f32> = Array2::<f32>::zeros((self.width, self.width));
        for i in 0..self.species {
            // TODO: Get a convolution to work with periodic BC and get rid of the naive implementation
            let avg = self.a.sum() / (self.width * self.width) as f32;
            let species_i = self.a.slice(s![.., .., i]);
            // Calculate averages over 3x3 windows except on edges
            for j in 0..self.width-2 {
                for k in 0..self.width-2 {
                    let window = species_i.slice(s![j..j+3, k..k+3]);
                    d[(j+1, k+1)] = ((window.sum() / 9.) - species_i[(j+1, k+1)]) * relax
                }
            }
            // Pad averages on the edges with zero
            d.slice_mut(s![0, ..]).fill(0.);
            d.slice_mut(s![-1, ..]).fill(0.);
            d.slice_mut(s![.., 0]).fill(0.);
            d.slice_mut(s![.., -1]).fill(0.);
            // Add averages to cells
            let mut slice = self.a.slice_mut(s![.., .., i]);
            slice += &ndarray::ArrayView::from(&d);
            d = Array2::<f32>::zeros((self.width, self.width)); // reset
        }
    }
}

// Misc
impl ReacDiffGrid {
    // Compute the sum of all species
    pub fn sums(&self) -> f32 {
        let mut sum: f32 = 0.;
        for elem in &self.a {
            sum += elem;
        }
        sum
    }
}
```