Experimental Soft Matter Physics
Module 2: BZ Oscillations Lab

Learning Objectives

1. To explore the spontaneous emergence of nonlinear oscillations in chemical and biological systems, using the famous Belousov-Zhabotinsky (BZ) reaction as a laboratory model.
2. To appreciate the broader significance of the BZ reaction in the history of nonlinear science.
3. To gain additional experience with scientific programming, combining quantitative image analysis techniques with computational skills developed in previous labs.

Introduction

In the traditional physics curriculum, the simple harmonic oscillator casts a very long shadow. The classical version offers a generic model for all mechanical oscillations with sufficiently small amplitude. The quantum version is one of a select few problems for which wave functions can be derived analytically and contributes to the foundations of quantum field theory. There are even electrical circuits that act like simple harmonic oscillators. In all of these different applications, we find *linear* equations describing a characteristic transfer between potential and kinetic forms of energy. Elsewhere in the natural world, especially in chemical and biological systems, we find oscillations that require a rather different description. These systems do not violate the law of conservation of energy, which physicists believe to be fundamental, and transfers between potential and kinetic forms of energy can still be identified. However, matter and energy are also flowing into and out of these systems and this, together with *nonlinear* interactions, creates new possibilities.

This lab explores a historically significant example of chemical oscillations. In the classic version of this reaction, after certain reactants are mixed together in water, the color of the entire solution oscillates between colorless and yellow for roughly an hour. Boris Belousov first discovered these oscillations around 1950 while searching for an inorganic model for the citric acid cycle, an important series of biochemical reactions that many cells use to extract energy from carbohydrates, fats, and proteins. He was, however, unable to convince the editors of peer-reviewed journals to publish his results. In the 1960s, during an explosion of interest in biological and biochemical oscillators, a graduate student named Anatol Zhabotinsky rediscovered Belousov's recipe. Within a decade, researchers all over the world were studying this reaction, which is now known as the *Belousov-Zhabotinsky reaction*. This BZ reaction went on to play a central role in many areas of pattern formation and nonlinear science more generally. In this lab, we will use Matlab to analyze photographic evidence of BZ oscillations and explore a famous nonlinear model of these oscillations.

Chemical Oscillations in the Lab

In living cells, the citric acid cycle is catalyzed by a number of organic molecules, many of which involve bound metal ions. Early versions of the reaction discovered by Belousov used the metal cerium as a catalyst. Zhabotinsky later replaced the citric acid in Belousov's recipe with malonic acid. There are now a number of different variations on these original recipes. The one we will be working with requires the following solutions:

Solution A: Dissolve 3.8 g of potassium bromate in 100 mL of ultra- pure water. (This creates 100 mL of a 0.23M $KBrO_3$ solution.)

Solution B: Dissolve 3.2 g of malonic acid with 0.7 g of potassium bromide in 100 mL of ultra-pure water. (This creates a 100 mL solution of 0.31M malonic acid and 0.059M KBr.)

Solution C: Add 14.5 mL of sulfuric acid to 85.5 mL of ultra-pure water and add 1.223 g ceric ammonium sulfate. (This creates a 100 mL solution of 0.019M ceric ammonium sulfate and 2.7M sulfuric acid.)

To initiate the reaction, first mix solutions A and B in a flask with a magnetic stir bar. The mixture will turn brown initially and gradually clear up afterwards. (Note that this brown color is associated with the formation of poisonous bromine gas, so this reaction needs to be conducted inside a chemical fume hood.) Next, add solution C and 5 mL of ferroin indicator solution. The solution will turn a deep red color and, some time later, begin to oscillate between red and blue.

This semester, due to the impossibility of maintaining adequate social distancing in departmental laboratory spaces, we will not be directly observing these oscillations together. We will instead be working with photographs taken by previous generations of Colby physics majors. These photographs were taken at a fixed rate of 4 images every second in a darkened laboratory with the oscillating solution backlit with diffuse blue light. The use of blue light causes the red and blue phases of the BZ reaction to show up as dark and bright regions, respectively, making the oscillation easy to capture in black-and-white photographs. Each laboratory group will be provided with thousands of these photographs to analyze.

Quantitative Image Analysis

A surprising number of soft matter experiments use digital photography as a primary source of quantitative data. This requires, in addition to a good camera, lenses, and lighting equipment, a strategy for integrating images into a programming environment that lets you analyze them at the level of individual pixels. Matlab has an entire toolbox of image processing functions and, not surprisingly, this toolbox includes a function that loads images. If you tell this function the name of a particular image file and where this file is located, it will load it as a matrix into your Matlab workspace. Your analysis, however, will almost certainly require looping over a number of files with slightly different names, so you need some way of dealing with these names automatically. Here's a strategy I have found useful for dealing with numbered sequences of images...

The first step is to write a Matlab function that knows where your images are and, given a particular number in the sequence, picks out the associated file correctly. The images you've been given have filenames that look like "BZ0001.tif", "BZ0002.tif", and so on. Let's assume these are sitting on your desktop, on a PC, in a folder named "Data". In this case, your function should look something like this:

```
function image = image_load(number)

prefix = 'C:\Users\jhmccoy\Desktop\Data\BZ';
name = [prefix,sprintf('%04u',number),'.tif'];
image = imread(name,'tif');

end
```

Here, the string "prefix" is everything in the filename, including the full path describing where the file is located, that comes before the four digit number. If you're on a Mac, this line would look slightly different, but the overall idea would be the same:

```
prefix = '/Users/jhmccoy/Desktop/Data/BZ';
```

The next line of code combines this prefix with the four digit number, including zeros, plus the final ".tif" that tells the code what type of file we're dealing with. The last line of code actually imports the chosen file, using a function from the image processing toolbox. So if you type

```
>> M = image_load(4);
```

the image saved as "BZ0004.tif" will appear in your Matlab workspace as a matrix named M. Let's take a look at this image, using another function from the image processing toolbox:

```
>> imshow(M);
```

You should see a mostly dark image, apart from a few bright streaks due to reflected light. In the middle, you should see the faint outlines of a beaker containing 300 mL of dark fluid, which looks dark in this image because of how its red color absorbs blue light.

Next, load and take a look at the 140th image in the sequence. In this image, you should see a bright glow inside the beaker, suggesting that the reaction has moved into its blue phase. Your task will be to extract a quantitative signal that captures these differences. Since the matrices created by your image loading function have the exact dimensions of the original image, with one entry per pixel, you can easily pick out specific pixels or groups of pixels and do anything you like with them. For example, if you want to know the intensity recorded in the pixel at the upper left corner of the image, just type

```
>> M(1,1)
```

If you want to loop over the first 100 images and watch how this particular pixel intensity varies with time, you simply write a new function that contains the following code:

```
% create a vector in which to save the data you're interested in
d = zeros(100,1);

% loop over images
for j = 1:100

   % load the j-th image
   M = image_load(j);

   % pick out a particular pixel and save its intensity value
   d(j) = M(1,1);

end
```

After running this code, you can then plot "d" or use it to perform additional analysis as necessary. You can do all sorts of other things with the data as well, of course, by changing what you do each time you load "M" during for loop or by adding new lines that manipulate "d" after this loop is completed. Instead of looking at a particular pixel, for example, you could look at averages obtained from a group of useful pixels. The exact procedure depends, of course, on what exactly you have in mind but hopefully this example is general enough to get you started. Again, the goal is to extract quantitative information from the raw images, such as periods of oscillation or any other measures that come to mind.

A Simple Model of BZ Oscillations

The Belousov-Zhabotinsky reaction mechanism involves dozens of steps, depending on how you count. A research team based at the University of Oregon in the early 1970s, however, discovered that a quantitative model could be constructed by focusing on a small subset of these reaction steps. This simplified model, which is now known as the *Oregonator*, provided a realistic mechanism linking limit cycle dynamics to a real-world chemical oscillator. Assuming certain reactants are available in unlimited quantities, the model requires only two coupled differential equations,

$$\epsilon \dot{U} = U(1-U) - fV(U-q)/(U+q), \tag{1}$$

$$\dot{V} = U - V, \tag{2}$$

where U and V represent the concentrations of $HBrO_2$ and Ce^{4+}, respectively. As we discussed in class there are simpler models that produce oscillators, such as the Brusselator and the Lotka-Volterra model, but these do not describe real-world chemical oscillators that can be examined in the lab. In this next section of our lab, you will explore the behavior of the Oregonator model using Matlab and confirm the presence of a limit cycle. To do this, you will need a modified version of the code you used to numerically investigate the FitzHugh-Nagumo model earlier this semester:

```
% parameter values
epsilon = 9.9e-3;
q = 7.62e-5;
f = 1;

% find equilibrium values of reactants
ueq = 0.5*(1-q-f) + 0.5*sqrt((1-q-f)^2 + 4*(1+f)*q);
veq = ueq;

% pick initial values of y(1), y(2), y(3)
y0 = [ueq + 1e-5, veq + 1e-5];

% pick a time interval
tspan = [0, 50];

% use the equations you defined
ode = @(t,y) BZ_model(t,y,epsilon,q,f);
J = @(t,y) BZ_Jacobian(t,y,epsilon,q,f);
```

```
% solve the equations
options = odeset('RelTol',1e-6,'Jacobian',J);
[t,y] = ode15s(ode,tspan,y0,options);

% plot results
plot(t,y(:,2))
```

Examining these lines, you will find recognize and be able to interpret many of the steps. The equilibrium concentration values provided, for example, are those at which $\dot{U} = 0$ and $\dot{V} = 0$ and the initial conditions are chosen near these values, though you're free to change that. You'll need to create a new function that tells Matlab what the model looks like. To do this, type

```
>> edit BZ_model
```

and enter the following lines:

```
function dy = BZ_model(t,y,epsilon,q,f)

   % model equations
   dy = zeros(2,1);
   dy(1) = (y(1)*(1 - y(1)) - f*y(2)*(y(1) - q)/(y(1) + q))/epsilon;
   dy(2) = y(1) - y(2);

end
```

If you look closely, you'll also notice that a different differential equation solver is being used here. If you try using your earlier code exactly, you'll find that the solver finds the Oregonator model too hard to work with. (I can tell more about this, if you're interested!) The solution, implemented above, is to use "ode15s" instead of "ode45" and to provide the solver with information about the Jacobian of the model equations. To make this work, just type

```
>> edit BZ_Jacobian
```

and enter the following lines:

```
function J = BZ_Jacobian(t,y,epsilon,q,f)

   % Jacobian equations
   J = zeros(2,2);
   J(1,1) = (1 - 2*y(1) - 2*f*y(2)*q/(y(1) + q)^2)/epsilon;
   J(1,2) = (-f*(y(1) - q)/(y(1) + q))/epsilon;
   J(2,1) = 1;
   J(2,2) = -1;

end
```

Running this code, you should be able to confirm that nonlinear oscillations are produced at these parameter values and that these oscillators can be interpreted as a limit cycle. (Remember to plot solutions as functions of time but also as phase trajectories and take notes on what you see!) These

oscillations will not look exactly like those you found using the experimental data, partly because the model parameters that best match our BZ recipe are not known but mostly because we don't have quantitative measurements of the variables used in the model. Nevertheless, you may find it interesting to have examined this historically significant problem from both experimental and theoretical sides. If you want to challenge yourself, you can try mapping the phase diagram by varying the values of ϵ and f to find the Hopf bifurcations at which the limit cycle appears or disappears. I can provide hints on how to do this, if you're interested.