Experimental Soft Matter Physics
Module 3: The Lorenz Equations

Learning Objectives

1. To confirm, using scientific programming techniques, that a *deterministic* system of ordinary differential equations can generate what appear to be unpredictable, non-repeating behaviors.
2. To explore and characterize forms of order and structure within such systems, including bifurcations and attractors in phase space.
3. To appreciate the broader significance of Lorenz's work in the history of nonlinear science.

Introduction

The stubborn unpredictability of the weather, in defiance of the tremendous scientific and technological tools at our disposal, is something we're all familiar with. But how many of us stop to wonder *why* weather should be so hard to understand? Is it so much more complicated than orbital mechanics and atomic behavior, which seem far easier to predict? In the early 1960s, an atmospheric scientist named Ed Lorenz made a startling discovery that changed how we think about such questions: *simple deterministic models, consisting of just few nonlinear differential equations, can have solutions that never repeat.* This, together with earlier work by Poincaré and other mathematicians interested in dynamics, was the beginning of what we now *chaos theory.*

In this lab, you will retrace some of Lorenz's steps using his famous system of equations,

$$\begin{aligned} \dot{X} &= \sigma(Y - X), \\ \dot{Y} &= rX - Y - XZ, \\ \dot{Z} &= XY - bZ. \end{aligned} \tag{1}$$

As we discussed in class, Lorenz derived these ordinary differential equations from a much more complicated set of partial differential equations describing thermal convection in a heated fluid. The variable X is related to the fluid velocities, while Y and Z are related to the temperature field. While this simple model ignores many of the complexities of thermal convection, it does capture one of its most basic features. Recall that the *Rayleigh number*, Ra, is a dimensionless number proportional to the applied temperature gradient. For sufficiently small Ra, we expect the fluid to remain totally motionless. As Ra is increased beyond a *critical value*, Ra_c, however, the fluid suddenly starts moving and convection patterns emerge. In the Lorenz equations, the control parameter r compares Ra to its critical value: $r = \mathrm{Ra}/\mathrm{Ra}_c$. Thus, your first task will be to confirm that the system shows a sharp transition from a motionless state to one with steady convection at a well-defined value of r. Afterwards, you will confirm that the dynamics become much more complicated at larger values of r, offering a groundbreaking example of the phenomenon now known as *deterministic chaos.*

Solving the Lorenz Equations

In this lab, many of the techniques explored in previous lab will be put to good use. To get started, for example, we're going to need code that gives us complete control of the dynamics. You already learned how to do this in the excitable systems lab. Building on these earlier efforts, you simply type

```
>> edit lorenzeqns
```

and then translate the Lorenz equations into language suitable for Matlab's differential equation solvers:

```
function dy = lorenzeqns(t,y,r)
sig = 10;
b = 8/3;
dy = zeros(3,1);
dy(1) = sig*(y(2) - y(1));
dy(2) = ...;
dy(3) = ...;
end
```

Note that X, Y, and Z from Eqn. (1) are here written as $y(1)$, $y(2)$, and $y(3)$. Note also that the above is an incomplete translation of Eqn. (1): only the first of the three lines defining dy has been completed and the other two have been left for you as an exercise. When you're finished, save your work and close the editor window you've been working in. Now type

```
>> edit lorenzlab
```

and enter the following commands:

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

% pick initial values of y(1), y(2), y(3)
init = [0, 1, 2];

% choose a value for r
r = 0.1;

% use the equations you defined
ode = @(t,y) lorenzeqns(t,y,r);

% solve the equations
options = odeset('RelTol',1e-6);
[t,y] = ode45(ode,tspan,init,options);

% plot results
plot(t,y(:,1))
```

When you're ready to go, just save your changes and type

```
>>  lorenzlab
```

r In its current form, your "lorenzlab" script produces a time series plot of $X(t)$ as its last step, but already know how to produce other plots. If you want to see what $Z(t)$ looks like, for example, try

```
>>  plot(t,y(:,3))
```

or, if you want to see the three-dimensional phase trajectory projected onto the X-Z plane, try

```
>>  plot(y(:,1),y(:,3))
```

If you want to change your time span, initial conditions, or r value, all you have to do is modify your "lorenzlab" script, save your changes, and run it again. Play around for a while now. Try out a range of different r values between 0 and 10 or so. Does anything interesting change as you increase r from very low fractions of 1 to progressively larger values? Be sure to take thorough notes and save plots that capture some of your insights and discoveries!

Bifurcation Diagrams

When analyzing a nonlinear system, plotting what happens to the solution as control parameters vary can provide useful information. In your exploration of the Lorenz system, for example, you probably noticed that your variables X, Y, and Z always settle down to particular values that depend on your choice of r. This naturally suggests the possibility of plotting the steady state value of one of these variables as a function of r. Finding this steady state analytically is a nice challenge, which you could take a stab at, but since this is a lab course we're going to do it empirically. All you have to do is let your code run for a long enough time span and then store the final value of, say, $X(t)$, before trying again with a different value of r. You'll want to make a copy of your "lorenzlab" script and give it a new name (or just type "edit [filename]", like you did before). Here's an idea for what your new script could look like:

```
% pick a time interval
tspan = [0, 1000];

% pick initial values of y(1), y(2), y(3)
init = [0, 1, 2];

% create an array in which to store X steady state values
n = ...;
x = zeros(n,1);
r = x;

% loop over n values of r
rmax = 4;
for j = 1:n

   % choose a value for r
```

```
    r(j) = rmax*j/n;

    % define "ode" using the actual lorenz eqns for your chosen r
    ode = @(t,y) lorenzeqns(t,y,r(j));

    % solve the lorenz eqns
    options = odeset('RelTol',1e-6);
    [t,y] = ode45(ode,tspan,init,options);

    % store a result
    x(j) = y(end,1);

end

% plot results
plot(r,x)
```

Again, you are here using techniques that you've seen before. Note, for example, the structure and syntax of the "for loop", which allows you scan through a sequence of different r value (one per iteration of the loop). There are many ways of doing this, but the code given above is a good starting place. You'll want to pick some large value for n, something in the hundreds maybe or even bigger, and you may or may not want $r = 4$ to be your largest value. You may even want to change the time interval. Pick whatever gives the nicest plot, whatever best captures the trends you were noticing earlier.

Do you understand the significance of the *bifurcation diagram* produced by this script? Does it tell you anything about the onset of convection? You may need to revisit our discussion of the significance of r and do some reading on your own about Rayleigh-Bénard convection. Be sure to save your favorite plot as an ".eps" file and be sure to get everything into your notebook.

Deterministic Chaos

Now for the real surprise, the feature of these equations that made them famous. It only happens when r is increased well beyond the range you've been looking at so far. Try $r = 28$, the value Lorenz devoted a considerable portion of his paper to. His initial condition was [0, 0.001, 0], if you'd like to try that as well, but feel free to try other initial conditions. Play around with different time spans and different ways of plotting. Do the variables settle down to steady-state values? *Have fun* and don't forget to take thorough notes on your observations and save your favorite plots! Here are a few ideas you might explore in more detail (try one or both!):

1. *Sensitive dependence on initial conditions* (the butterfly effect): Compare the trajectories produced from very similar (but slightly different!) initial conditions. To do this, you'll have to rename your y array (to something like y0) before changing initial conditions and rerunning your code. You'll also have to remember out how to plot two curves simultaneously (the documentation for "plot" or "hold" will help). What do you learn about predictability in this system? Can small perturbations have significant consequences?

2. *Strange attractors*: The phase trajectories associated with the Lorenz system form beautiful and amazingly intricate patterns. When r is low, the eventually fate is collapse onto a dot in the system's X-Y-Z phase space; this dot is the *attractor* for the system. What happens when r is large enough to produce chaos, however, is not obvious. Plotting the phase trajectories in various ways, for a range of time spans, will reveal an attractor with strange, nested loops. The existence of this structure shows an unexpected form of order lurking within the chaos.

When you feel like you've gotten a good sense for the remarkable complexity this system is capable of, you can move on to another of its unexpected features. Take a closer look at the time series for $Z(t)$. You'll see wiggles at two different levels in the curve, one group at higher Z values and another group at lower values, with seemingly random transitions between these. Lorenz sensed a pattern here, which he made explicit by extracting the maximum value of every peak in this curve and see how these peak values are related. In particular, given a certain peak Z_k, he found a remarkably simple way of predicting where the next peak Z_{k+1} will fall. Lorenz found these peak values can be related through a simple empirical curve of the form,

$$Z_{k+1} = f(Z_k), \tag{2}$$

which works for every single peak in the data. This curve $f(z)$ is plotted in his famous 1963 paper. Your last task here is to reproduce this discovery. To do this, start by running your Lorenz equation solver for a very long time span, *e.g.*, [0,10000]. Use Lorenz's initial condition of [0, 0.001, 0].

Now we just need a way of picking out all of the peaks. We could write code for this ourselves but I found code online that works well enough (there's actually a large community sharing Matlab resources online). When your group is ready, ask and I'll help you set it up with a function called "peakdet". Then just type the following:

```
>> [a,b] = peakdet(y(:,3),0.1,t);
>> plot(a(:,1),a(:,2),'o',t,y(:,3))
```

You should see in this new plot a little circle at every local maximum (you may have to zoom in to be sure about this). If everything looks okay, you're ready for the last step. The following commands should produce something very striking:

```
>> num = length(a(:,2));
>> plot(a(1:num-1,2),a(2:num,2),'.')
```

If you looks carefully at the syntax, you see that this is a plot of the $(k+1)$th peak value of Z as a function of the k-th peak value of Z. If the simple structure suggested by Eqn. (2) exists in this data, all of the points plotted will collapse onto single, $f(Z)$. You may want to compare your plot to the one obtained by Lorenz.