

Lab 5 - Chaotic Dynamics

Thursday 28 September, 2006 - Due: Thursday 05 October

The aim of this lab is to explore some aspects of chaotic dynamics. First, we have another look at the Pendulum example presented in lecture. Then, we will study the Logistic Map, a simple but quite illustrative example of a chaotic dynamical system.

As always, please include in your writeup any sections of Python code you write.

1 Visualizing the Motion of a Pendulum

First, let us revisit the model of a more realistic pendulum, under the influence of gravitation, of damping, and of a periodic driving force. Although the pendulum swings in a plane, the situation essentially reduces to a “one-dimensional problem” since the pivot point is fixed and the pendulum’s length l is constant. In fancier terms, since there is “one degree of freedom”, we have one “configuration variable”, the “angular position” θ . (If you specify θ , we know how the pendulum is oriented.)

So, as was said in lecture, this is [almost] mathematically identical to the 1D-problem we studied earlier:

$$\begin{aligned}\frac{d\omega}{dt} &= -\frac{g}{l} \sin(\theta) - k \frac{d\theta}{dt} + F \sin(\omega_D t) \\ \frac{d\theta}{dt} &= \omega\end{aligned}$$

The first equation relates the “angular acceleration” (the time-derivative of the “angular velocity” ω) to the gravitational, damping, and driving forces. The second equation expresses the “angular velocity” as the time-derivative of the angular position. I said “almost” above because the 2π -periodicity associated with the “angular position”. We’ll see this treated in the code below.

So, we can reuse the earlier code... either by renaming a bunch of variables, or by keeping the names, but changing the physical interpretation of those variables. To minimize the changes we’d have to make, let’s keep the names... but let’s make comments to remind us of what we did. Refer to [lab5_2pendula.py](#), which is a commented version of the code shown in class. **The key interpretation to keep in mind is that the variable `pos.x` represents the pendulum’s angular position θ .**

Execute [lab5_2pendula.py](#). Recall from lecture that we are plotting as functions of time the *differences* in angular position and angular velocity between two pendula with nearly identical initial conditions. With various choices of the driving force amplitude **F** (say, $F = 0.5$, $F = 1.2$, and $F = 1.35$), we saw plots of the differences.

It might be nice to see a more realistic visualization of the motions of each pendulum. That’s what you are going to do.

()★ Create a visualization of each pendulum. (We’ll help you with this below.)

- We want our VPython display back on. So, replace `scene.visible=0` with `scene=display(visible=1, x=0, y=400, width=400, height=400, autoscale=0, range=3)`
- Since we have just turned on the `display` with its `visible=1` parameter, we need to hide the spheres `ball1` and `ball2` with their own `visible=0`. (Realize that we need them to exist since they represent angular positions for us. However, we don't want to display them.)
- Now, let's visualize in Pendulum 1. Before the `while` loop, insert

```

box(pos=(0,0,0), axis=(0,0,1), length=0.5, width=0.5, height=0.5)

theta1=ball1.x
pendulum1=sphere(pos=vector(sin(theta1), -cos(theta1), -.25),
                    radius=.2, color=color.red)
rod1=curve(pos=[vector(0,0,-.25), pendulum1.pos], color=pendulum1.color)
force1=arrow(axis=vector(0,0,0), color=pendulum1.color)

```

For clarity, we use a new variable `theta1` to remind us that we really are dealing with an angular position. Now, we draw a red small-radius sphere called `pendulum1` that will travel on a circle of radius $l = 1$, sitting in the plane $z = -0.25$. The trig functions were chosen so that " $\theta = 0$ " is downward and θ is positive in the counterclockwise direction from our viewpoint. A radius called `rod1` is drawn from the center of the circle to `pendulum1`. Finally, we set up an arrow called `force1` that will be drawn later.

Now, inside the `while` loop, near the end of the block, we need to update the positions and directions with the values we just computed in the while loop.

```

theta1=ball1.x
pendulum1.pos=vector(sin(theta1), -cos(theta1), -.25)
rod1.pos[1]=pendulum1.pos
force1.pos=pendulum1.pos
force1.axis=(a1_initial.x+a1_final.x)*0.5*
              vector(cos(theta1), sin(theta1), 0)

```

Note that `rod1.pos[1]` [the second element, with index "1", in that curve's `pos` array] is set to the current position of the pendulum1. The first element, with index "0", holds the fixed center of the pendulum's circle.

Note that, in `force1.axis`, the signed-size of the net tangential force on the pendulum is given by `(a1_initial.x+a1_final.x)*0.5`, and the forward direction is given by a unit vector that is tangent to the circle.

- Define the corresponding elements `theta2`, `pendulum2`, `rod2`, and `force2` for the second pendulum, a green pendulum traveling on a circle on the plane $z = +0.25$. Within the while loop, you must include the corresponding update statements.
- (Q1)★ • Include your completed Python program and a screen capture when the two pendula have diverged, which indicates "a sensitivity to initial conditions".

2 The Logistic Map

We are going to consider the following dynamical model

$$x_{n+1} = 4r x_n(1 - x_n) \quad \text{with } 0 \leq x_i \leq 1 \text{ and } 0 < r \leq 1.$$

This could be thought of like the integrator methods (Euler, etc.) that we have used, with $dt = 1$. According to Wikipedia, this model was introduced by the mathematician P.F. Verhulst (1838) in the context of “population dynamics” and popularized by the biologist R. May (1976). The constant “4” is chosen out of convenience. [When x_0 starts within $[0, 1]$, it turns out that the subsequent x_n ’s stay within $[0, 1]$ as long as $4r$ was chosen within $(0, 4]$, or more conveniently, r was chosen within $(0, 1]$.]

Let’s study the behavior of this model.

- Open [lab5_logistic_a.py](#) and run it.
 - This program features an example of “formatted printing”. `info` is a string that will have two values from a “tuple” [in this case, an ordered pair] of values substituted into a format string. `%8.6f` means “in 8 character spaces, display the value as floating-point value with 6 decimal places”

Keeping `r` fixed at 0.20000, observe the behavior of the system for at least FOUR (4) choices of the starting value of `x` in the open range $(0, 1)$. Make use of the graph and of the numbers printed in the shell window.

(Q2)★ Briefly describe what you observe. Repeat for $r = 0.10000$.

- With `r` to 0.30000, observe the behavior of the system for at least FOUR (4) choices of `x` in the open range $(0, 1)$.

(Q3)★ Briefly describe what you observe.

What is the x -value (call it x_∞) that each sequence appears to converge to?

- Open [lab5_logistic_varying_x.py](#) .

Here, we’ve used a `for`-loop over a Python-list of starting `x`-values in order to take out the tedium of changing the starting values of `x` . Note how we indented the entire block that is to be executed for each `x` .

(Q4)★ Use this program to complete this table:

r	x_∞
0.3	
0.4	
0.5	
0.6	
0.7	
0.8	

When the table is complete, summarize what you observed.

(At home)★ Can you determine a formula for x_∞ in terms of r using the values for $0.3 \leq r \leq 0.7$?
Hint: for these r ’s, try to approximate x_∞ as a ratio of two integers.

- In order to better understand what is happening for $r = 0.8$, return to `lab5_logistic_a.py` and make the following changes:

- increase `nmax` to 2000 (or greater, if needed)
- change `gcurve` to `gdots` (`gdots` is clearer and can handle more points)
- remove or comment-out the `rate` statement (that is, full-speed ahead!)

With (say) $x = 0.3$, run this modified program for $r = 0.7$ and then for $r = 0.8$.

(Q5)★ Include this program in your writeup.

(Q6)★ To (say) FIVE (5) decimal places in `r`, determine “ r_2 ”, the smallest r that displays this “period doubling” phenomenon as seen in $r = 0.8$.

If there are some ambiguities in categorizing what you observe, you may have to increase `nmax` to 5000 or greater. This is likely to happen as you approach r_2 . In addition, consult the values displayed in the shell window.

If the shell window is too cluttered, close the window. A clean window will appear upon execution.

- Now, starting near $r = 0.875$ search downward for “ r_3 ”, the smallest r that displays a further “period doubling” phenomenon.

- *Slowly* crawl upward from $r = 0.875$ until you see the next doubling. Then from there, search downward to find r_4 .

(Q7)★ With r_2 , r_3 and r_4 , you can make a first estimate δ_3 of the Feigenbaum number δ discussed in lecture:

$$\delta_k = \frac{r_k - r_{k-1}}{r_{k+1} - r_k}$$

where

$$\delta = \lim_{k \rightarrow \infty} \delta_k = \lim_{k \rightarrow \infty} \left(\frac{r_k - r_{k-1}}{r_{k+1} - r_k} \right)$$

(Possibly continued at home Q8)★

- Find as many r_k values as you can. Then, use those to get better estimates of δ .

(Before you leave Q9)★

- Take a look at `lab5_logistic_bifurcation.py`, which plots, for each value of r , a sample of its x -values after many iterations. This is called a “bifurcation diagram” for this Logistic Map. By clicking on a data point, its r -value is revealed and the view zooms in on that point. Note the chaotic region as r approaches 1.0.

After selecting some interesting points, take a screenshot and include it in your writeup.