

## PHY307, Science and Computers I

**Lab #5, September 17, 2002**  
**(due by end of lab THURSDAY, SEPT. 19)**

# The Curves of Chaos

### Summary of what you will do:

We will now do some graphing to more carefully classify the chaos we can find in the logistic map. You should take a look at the notes online about the logistic map at some point.

You will (1) experiment some with plots and lists, (2) plot the “map history” for a few values of the parameter **a** in the logistic map, (3) see how the histories may or may not diverge, depending on initial conditions, and (4) draw a “bifurcation diagram” for the logistic map.

In your lab report, give a narrative of what you do, include code, and put in snapshots of the plot. To take a snapshot of a window, choose the window to take a picture of, then, while holding down the ALT key, press the PrtSc or PrintScreen key. Then when you paste into a program like Word, you will paste in an image of that window.

### **FIRST, GRAPH PRACTICE**

These first few steps should be very familiar:

1. Log in to your workstation using your CMS/SUnix name & password.
2. Open up some editor or word processor to start your report. Put in a title, your name, and date. As you write the report, make it clear what each section is. At the end of the lab, print your report out, staple it together, and hand it in.
3. Get IDLE for VPython going by going through the Menu: Start ? SU Academic Departments ? Physics.
4. Open a program file editor by “New” under the “File” menu and save your (currently empty) work to your I: or K: drive as “graph1.py” or a similar name.

Now,

5. Based upon what you learned at the beginning of class and using the **visual.graph** module, have VPython plot the function  $\sin(x/10.) * \exp(-x/40.)$ , sampled at integer **x**, for **x** from 0 to 120, using a solid yellow curve. Include the Python code and a snapshot of the plot in your report.
6. [BONUS TIME] If you have extra time, you might want to come back and make the color of the plot change during the plot. When you add a point to a curve, set its color, using **color = ( x/120., 0, 1.-x/120.)** in the **curveA.plot()** function call. Record what you see.

## SECOND, PLOT SEQUENCES FOUND USING THE LOGISTIC MAP

7. Start with a logistic map history generator, set to generate 50 points, at the rate of 5 points a second. This code, 'logistic1.py' will do the trick:

```
from visual import *

a = 2.5                # set logistic map parameter
xA = 0.25             # set initial value ("population")

for j in range(50):   # repeat map for 50 steps
    rate(5)
    xA = a * xA * (1.-xA) # here is the map
    print "xA =", xA, "at step", j # print value of xA
```

Run it. What do you see? Record the value that  $x_A$  converges to.

8. Save your code under a new name 'logisticplot1.py'. Add in some visualization. Previously you visualized the map by giving a cone the dynamics of the logistic map. Here, you will *plot* how the value of  $x_A$  changes in time. You will need to:
- Add **from visual.graph import \*** near the beginning.
  - Set up an empty curve, say named **curveA**. Pick a color for it.
  - Each time through the loop, add the point (**j**, **xA**) to the curve.
  - (You can remove the **print** statement if you like, though it is a useful check.)

Run it – what do you see? Include the final plot in your report, along with a written description.

9. Repeat the above for values **a=3.2**, **a=3.5**, and **a=3.8**. Carefully look at the character of the plots for each value of **a**:
- Is there an initial settling in?
  - Does the system or population settle into a stable or repeating pattern?

### **THIRD, COMPARE TWO SEQUENCES FOUND USING THE LOGISTIC MAP**

You will now repeat Lorenz's experiments with his computer simulations of the weather, but you will instead use the logistic map to investigate the butterfly effect.

10. Start from your program (**logisticplot1.py**) that plots the history of the population from a single starting point. Save your new code as **logisticplot2.py**.
  - a. Add a second variable, named **x<sub>B</sub>**, near where you initialize the variable **x<sub>A</sub>**. Give it an initial value close to that of the initial value for **x<sub>A</sub>**.
  - b. Add a second curve, **curveB**. Make it a color different from **curveA**. Do this outside the loop, as you only want one extra curve.
  - c. Inside the loop, update the value of **x<sub>B</sub>** using the logistic map (of course, with the same **a** used for **x<sub>A</sub>**.)
  - d. For each value of **j** (that is, in the body of the loop), add the point (**j,x<sub>B</sub>**) to **curveB**.

Run your program for **a = 2.5**. How do **curveA** and **curveB** look, compared to each other?

11. OK, as you might guess, it would be interesting to do Step 10 again, for the values **a=3.2**, **a=3.5**, and **a=3.8**. So do this! How do **curveA** and **curveB** differ?
12. Compare how the divergence of the curves differ, for **a=3.8**, for different initial separations. Compare for initial values of **x<sub>A</sub>** and **x<sub>B</sub>** differing by 0.01 and for the case where they differ by 0.0000001 (1 part in 1e7.) That is, plot the two curves, using **logisticplot2.py** for, say, **x<sub>A</sub>=0.25**, **x<sub>B</sub>=0.26** and compare with what you see for, say, **x<sub>A</sub>=0.25**, **x<sub>B</sub>=0.2500001**.
13. [BONUS TIME] If you have extra time, you might want to plot the sequence (**j,x<sub>A</sub>-x<sub>B</sub>**), without plotting the sequences (**j,x<sub>A</sub>**) or (**j,x<sub>B</sub>**). This allows you to see how the difference between the two changes with time, especially for very small differences between the initial values of **x<sub>A</sub>** and **x<sub>B</sub>**. (Or even, what if **x<sub>A</sub>** and **x<sub>B</sub>** start really close and you plot **log(abs(x<sub>A</sub>-x<sub>B</sub>))** vs. **j**?)
14. [EXTRA CREDIT – TAKES QUITE A BIT MORE THINKING] Can you set up a list of curves, corresponding to a list of initial values, so that you can plot 100 curves of 100 different colors, representing the map histories for 100 different initial values for **x**, at once?

## FINALLY, MAKE A BIFURCATION DIAGRAM

Time for some pretty pictures. This is a little subtle now. What we want to do is plot a figure that will *summarize the behavior of the logistic map for a large number of values of a*. This plot is called a bifurcation diagram (bifurcation referring to the splitting into two that you will see.) We will build up to the bifurcation diagram in a few steps.

15. First, what we want to do is plot the *long term behavior* for each value of **a**. We want to ignore the initial “settling in” or “transient behavior”. This initial behavior depends on the initial conditions and we want to neglect it to look at what type of behavior the system settles into (whether it is stable, oscillating, or chaotic.) So set up a logistic map simulator that makes a plot after iterating the map **Ninitial** times, where the variable **Ninitial** might be, say, 100. You can do this by inserting an “equilibration” loop before the main loop:

```
Ninitial = 100  
for k in range(Ninitial):  
    xA = a * xA * (1.-xA)
```

One sometimes calls this “warming up the system”.

Cut out all of the **B** stuff (**xB** and **curveB**) now – in fact, start with **logisticplot1.py**, saving your new code to **logisticplot3.py**. Replace **gcurve** with **gdots**. We don’t want to connect the dots now, for reasons we will see soon.

How do your plots look now, compared to your original **logisticplot1.py**, for **a=2.5, 3.2** and **3.8**?

16. Next, TAKE OUT ANY **rate()** function calls you might have, as we want to go fast now – you will plot many points.
17. Instead of plotting vs. **j**, plot the **xA** values vs. **a**. So when you call **dotsA.plot()**, use **pos=(a,xA)**. This might look more boring at first, but try it out. What do you see for **a=2.5,3.2** and **3.5**?
18. Now here is the tricky bit: we want to make this plot for many values of **a**, simultaneously. You will want to repeat most of the program (except for the initialization of the **gdots** object and the **import** statements) a block statement that you can repeat. To do this, drag the mouse over the relevant parts of the program to highlight it, then press the Tab key to indent that region. Just before the indented region, put your loop over **a**, starting from **a=2**, ending at **a=4**, in steps of **0.01** which looks like this:

```
for a in arange(2, 4, 0.01):
```

Run your program and discuss in detail the diagram you create, including a snapshot of the diagram in your report, of course. Save your code for use in the homework, say as **bifurcatelogistic.py**.