Physics 411: Homework 2

1. **Plotting experimental data:**

   (a) In the on-line resources you will find a file called `sunspots.dat`, which contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns of numbers, the first being the month and the second being the sunspot number. Write a program that reads in the data and makes a graph of sunspots as a function of time.

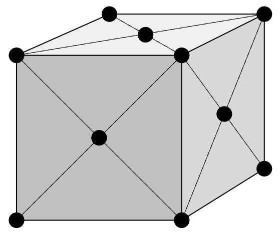
   (b) The file `stm.dat` contains a grid of values from scanning tunneling microscope measurements of the (111) surface of silicon. A scanning tunneling microscope (STM) is a device that measures the surface of a solid at the atomic level by tracking a sharp tip over the surface and measuring quantum tunneling current as a function of position. The end result is a grid of values that represent the height of the surface and the file `stm.dat` contains just such a grid of values. Write a program that reads the data contained in the file and makes a density plot of the values. Use the various options and variants you have learned about to make a picture that shows the structure of the silicon surface clearly.

   ☑ **For full credit** turn in printouts of your two programs and printouts of your plots.

2. **Visualizing lattices:** Example 4.2 in the book gives a program that creates a computer visualization of a simple cubic lattice. Using that program as a starting point, or starting from scratch if you prefer, do the following.

   (a) A sodium chloride crystal has sodium and chlorine atoms arranged on a cubic lattice but the atoms alternate between sodium and chlorine, so that each sodium is surrounded by six chlorines and each chlorine is surrounded by six sodiums. Create a visualization of the sodium chloride lattice using two different colors to represent the two types of atoms. (If you print out the result in black-and-white, make sure to use colors that are clearly distinguishable.)

   (b) The face-centered cubic (fcc) lattice, which is the most common lattice in naturally occurring monatomic crystals, consists of a cubic lattice with atoms positioned not only at the corners of each cube but also at the center of each face, like this:

   Create a visualization of an fcc lattice with a single species of atom (such as occurs in metallic iron, for instance).

   ☑ **For full credit** turn in a printout of your program for part (b) and printouts of the two pictures you created.

Hint: Printing out 3D graphics is a little tricky. The simplest way to do it is to run the program and then take a screenshot of the window containing the graphics. On a PC running Windows hold down ALT and then press the "Print Screen" button to take a screenshot of the current active window. On a Mac hold down Command and Shift and press the number 4, then press the space bar, then click on a window to take a screenshot of that window. Once you have your screenshot you can paste it into a document and print it out.

3. **Visualization of the solar system:** The innermost six planets of our solar system revolve around the Sun in roughly circular orbits that all lie approximately in the same (ecliptic) plane. Here are some basic parameters:

Object	Radius of object (km)	Radius of orbit (millions of km)	Period of orbit (days)
Mercury	2440	57.9	88.0
Venus	6052	108.2	224.7
Earth	6371	149.6	365.3
Mars	3386	227.9	687.0
Jupiter	69173	778.5	4331.6
Saturn	57316	1433.4	10759.2

(a) Write down equations for the coordinates x, y of a planet in the plane of the ecliptic at time t, assuming that it lies on the x-axis at $t = 0$ and travels in a circular orbit with radius R.

(b) Using the facilities provided by the `visual` package, create an animation of the solar system that shows the following:

i. The Sun and planets as spheres in their appropriate positions and with sizes proportional to their actual sizes. Because the radii of the planets are tiny compared to the distances between them, represent the planets by spheres with radii c_1 times larger than their correct proportionate values, so that you can see them clearly. Find a good value for c_1 that makes the planets visible. You'll also need to find a good radius for the Sun. Choose any value that gives a clear visualization. (It doesn't work to scale the real radius of the Sun by the same factor as you use for the planets, because it'll come out looking way too large. So just use whatever works.) For added realism, you may also want to make your spheres different colors. For instance, Earth could be blue and the Sun could be yellow.

ii. The motion of the planets as they move around the Sun (by making the spheres of the planets move). In the interests of alleviating boredom, construct your program so that time in the animation runs a factor of c_2 faster than actual time. Find a value of c_2 that makes the motion of the orbits easily visible but not unreasonably fast. Use the `rate` function to make your animation run smoothly.

☑ **For full credit** turn in a copy of your program and a snapshot showing the animation it produces.

4. **Deterministic chaos and the Feigenbaum plot:** One of the most famous examples of the phenomenon of chaos is the *logistic map*, defined by the equation

$$x' = rx(1-x). \tag{1}$$

For a given value of the constant r you take a value of x—say $x = \frac{1}{2}$—and you feed it into the right-hand side of this equation, which gives you a value of x'. Then you take that value and feed it back in on the right-hand side again, which gives you another value, and so forth. This is a *iterative map*. You keep doing the same operation over and over on your value of x, and one of three things happens:

   (a) The value settles down to a fixed number and stays there. This is called a *fixed point*. For instance, $x = 0$ is always a fixed point of the logistic map. (You put $x = 0$ on the right-hand side and you get $x' = 0$ on the left.)

   (b) It doesn't settle down to a single value, but it settles down into a periodic pattern, rotating around a set of values, such as say four values, repeating them in sequence over and over. This is called a *limit cycle*.

   (c) It goes crazy. It generates a seemingly random sequence of numbers that appear to have no rhyme or reason to them at all. This is *deterministic chaos*. "Chaos" because it really does look chaotic, and "deterministic" because even though the values look random, they're not. They're clearly entirely predictable, because they are given to you by one simple equation. The behavior is *determined*, although it may not look like it.

   Write a program that calculates and displays the behavior of the logistic map. Here's what you need to do.

   For a given value of r, start with $x = \frac{1}{2}$, and iterate the logistic map equation a thousand times. That will give it a chance to settle down to a fixed point or limit cycle if it's going to. Then run for another thousand iterations and plot the points (r, x) on a graph where the horizontal axis is r and the vertical axis is x. You can either use the `plot` function with the options `"ko"` or `"k."` to draw a graph with dots, one for each point, of you can use the `scatter` function to draw a scatter plot (which always uses dots). Repeat the whole calculation for values of r from 1 to 4 in steps of 0.01, plotting the dots for all values of r on the same figure and then finally using the function `show` once to display the complete figure.[1]

   Your program should generate a distinctive plot that looks like a tree bent over onto its side. This famous picture is called the *Feigenbaum plot*, after its discoverer Mitchell Feigenbaum, or sometimes the *figtree plot*, a play on the fact that it looks like a tree and Feigenbaum means "figtree" in German.

[1] There is another approach for computing the Feigenbaum plot, which is neater and faster, making use of Python's ability to perform arithmetic with entire arrays. You could create an array `r` with one element containing each distinct value of r you want to investigate: `[1.0, 1.01, 1.02, ... ]`. Then create another array `x` of the same size to hold the corresponding values of x, which should all be initially set to 0.5. Then an iteration of the logistic map can be performed for all values of r at once with a statement of the form `x = r*x*(1-x)`. Because of the speed with which Python can perform calculations on arrays, this method should be significantly faster than the more basic method above.

Give answers to the following questions:

(a) For a given value of r what would a fixed point look like on the Feigenbaum plot? How about a limit cycle? And what would chaos look like?

(b) Based on your plot, at what value of r does the system move from orderly behavior (fixed points or limit cycles) to chaotic behavior? This point is sometimes called the "edge of chaos."

The logistic map is a very simple mathematical system, but deterministic chaos is seen in many more complex physical systems also, including especially fluid dynamics and the weather. Because of its apparently random nature, the behavior of chaotic systems is difficult to predict and strongly affected by small perturbations in outside conditions. You've probably heard of the classic exemplar of chaos in weather systems, the *butterfly effect*, which was popularized by physicist Edward Lorenz in 1972 when he gave a lecture to the American Association for the Advancement of Science entitled, "Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?"

☑ **For full credit** turn in a copy of your program and a printout of the figure it produces, along with your answers to questions (a) and (b) above.