Due by class Monday 23 February 2015

Exercise 1: Plotting experimental data

In the on-line resources you will find a file called `sunspots.txt`, 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.

   a) Write a program that reads in the data and makes a graph of sunspots as a function of time.

   b) Modify your program to display only the first 1000 data points on the graph.

   c) Modify your program further to calculate and plot the running average of the data, defined by

$$Y_k = \frac{1}{2r} \sum_{m=-r}^{r} y_{k+m} \, ,$$

where $r = 5$ in this case (and the y_k are the sunspot numbers). Have the program plot both the original data and the running average on the same graph, again over the range covered by the first 1000 data points.

Exercise 2: Curve plotting

Although the `plot` function is designed primarily for plotting standard xy graphs, it can be adapted for other kinds of plotting as well.

   a) Make a plot of the so-called *deltoid* curve, which is defined parametrically by the equations

$$x = 2\cos\theta + \cos 2\theta, \qquad y = 2\sin\theta - \sin 2\theta,$$

where $0 \leq \theta < 2\pi$. Take a set of values of θ between zero and 2π and calculate x and y for each from the equations above, then plot y as a function of x.

   b) Taking this approach a step further, one can make a polar plot $r = f(\theta)$ for some function f by calculating r for a range of values of θ and then converting r and θ to Cartesian coordinates using the standard equations $x = r\cos\theta$, $y = r\sin\theta$. Use this method to make a plot of the Galilean spiral $r = \theta^2$ for $0 \leq \theta \leq 10\pi$.

   c) Using the same method, make a polar plot of "Fey's function"

$$r = e^{\cos\theta} - 2\cos 4\theta + \sin^5 \frac{\theta}{12}$$

in the range $0 \leq \theta \leq 24\pi$.

Exercise 3: Using the program from Example 3.2 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.

b) The face-centered cubic (fcc) lattice, which is the most common lattice in naturally occurring 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:
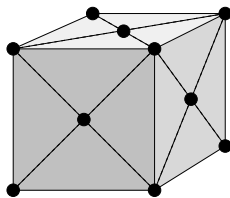
Create a visualization of an fcc lattice with a single species of atom (such as occurs in metallic iron, for instance).

Exercise 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:

1. 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.)

2. 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*.

3. 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.

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.

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?" (Although arguably the first person to suggest the butterfly effect was not a physicist at all, but the science fiction writer Ray Bradbury in his famous 1952 short story *A Sound of Thunder*, in which a time traveler's careless destruction of a butterfly during a tourist trip to the Jurassic era changes the course of history.)

Comment: 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.

Exercise 5: Least-squares fitting and the photoelectric effect

It's a common situation in physics that an experiment produces data that lies roughly on a straight line, like the dots in this figure:
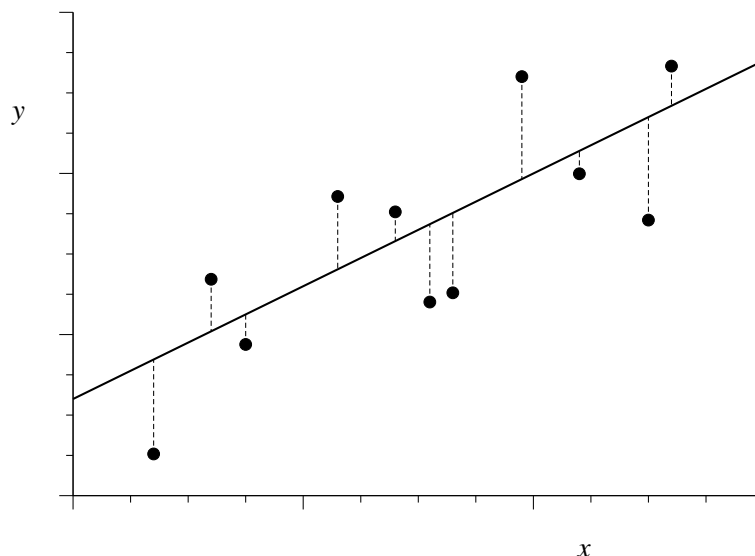
The solid line here represents the underlying straight-line form, which we usually don't know, and the points representing the measured data lie roughly along the line but don't fall exactly on it, typically because of measurement error.

The straight line can be represented in the familiar form $y = mx + c$ and a frequent question is what the appropriate values of the slope m and intercept c are that correspond to the measured data. Since the data don't fall perfectly on a straight line, there is no perfect answer to such a question, but we can find the straight line that gives the best compromise fit to the data. The standard technique for doing this is the *method of least squares*.

Suppose we make some guess about the parameters m and c for the straight line. We then calculate the vertical distances between the data points and that line, as represented by the short vertical lines in the figure, then we calculate the sum of the squares of those distances, which we denote χ^2. If we have N data points with coordinates (x_i, y_i), then χ^2 is given by

$$\chi^2 = \sum_{i=1}^{N}(mx_i + c - y_i)^2.$$

The least-squares fit of the straight line to the data is the straight line that minimizes this total squared distance from data to line. We find the minimum by differentiating with respect to both m and c and setting the derivatives to zero, which gives

$$m\sum_{i=1}^{N}x_i^2 + c\sum_{i=1}^{N}x_i - \sum_{i=1}^{N}x_iy_i = 0,$$

$$m\sum_{i=1}^{N}x_i + cN - \sum_{i=1}^{N}y_i = 0.$$

For convenience, let us define the following quantities:

$$E_x = \frac{1}{N}\sum_{i=1}^{N}x_i, \qquad E_y = \frac{1}{N}\sum_{i=1}^{N}y_i, \qquad E_{xx} = \frac{1}{N}\sum_{i=1}^{N}x_i^2, \qquad E_{xy} = \frac{1}{N}\sum_{i=1}^{N}x_iy_i,$$

in terms of which our equations can be written

$$mE_{xx} + cE_x = E_{xy},$$
$$mE_x + c = E_y.$$

Solving these equations simultaneously for m and c now gives

$$m = \frac{E_{xy} - E_x E_y}{E_{xx} - E_x^2}, \qquad c = \frac{E_{xx}E_y - E_x E_{xy}}{E_{xx} - E_x^2}.$$

These are the equations for the least-squares fit of a straight line to N data points. They tell you the values of m and c for the line that best fits the given data.

a) In the on-line resources you will find a file called `millikan.txt`. The file contains two columns of numbers, giving the x and y coordinates of a set of data points. Write a program to read these data points and make a graph with one dot or circle for each point.

b) Add code to your program, before the part that makes the graph, to calculate the quantities E_x, E_y, E_{xx}, and E_{xy} defined above, and from them calculate and print out the slope m and intercept c of the best-fit line.

c) Now write code that goes through each of the data points in turn and evaluates the quantity $mx_i + c$ using the values of m and c that you calculated. Store these values in a new array or list, and then graph this new array, as a solid line, on the same plot as the original data. You should end up with a plot of the data points plus a straight line that runs through them.

d) The data in the file `millikan.txt` are taken from a historic experiment by Robert Millikan that measured the *photoelectric effect*. When light of an appropriate wavelength is shone on the surface of a metal, the photons in the light can strike conduction electrons in the metal and, sometimes, eject them from the surface into the free space above. The energy of an ejected electron is equal to the energy of the photon that struck it minus a small amount ϕ called the *work function* of the surface, which represents the energy needed to remove an electron from the surface. The energy of a photon is $h\nu$, where h is Planck's constant and ν is the frequency of the light, and we can measure the energy of an ejected electron by measuring the voltage V that is just sufficient to stop the electron moving. Then the voltage, frequency, and work function are related by the equation

$$V = \frac{h}{e}\nu - \phi,$$

where e is the charge on the electron. This equation was first given by Albert Einstein in 1905.

The data in the file `millikan.txt` represent frequencies ν in hertz (first column) and voltages V in volts (second column) from photoelectric measurements of this kind. Using the equation above and the program you wrote, and given that the charge on the electron is 1.602×10^{-19} C, calculate from Millikan's experimental data a value for Planck's constant. Compare your value with the accepted value of the constant, which you can find in books or on-line. You should get a result within a couple of percent of the accepted value.

This calculation is essentially the same as the one that Millikan himself used to determine of the value of Planck's constant, although, lacking a computer, he fitted his straight line to the data by eye. In part for this work, Millikan was awarded the Nobel prize in physics in 1923.

Exercise 6: Consider the integral

$$I = \int_0^1 \sin^2 \sqrt{100x}\, dx$$

a) Write a program that uses the adaptive trapezoidal rule method of Section 5.3 and Eq. (5.34) to calculate the value of this integral to an approximate accuracy of $\epsilon = 10^{-6}$ (i.e., correct to six digits after the decimal point). Start with one single integration slice and work up from there to two, four, eight, and so forth. Have your program print out the number of slices, its estimate of the integral, and its estimate of the error on the integral, for each value of the number of slices N, until the target accuracy is reached. (Hint: You should find the result is around $I = 0.45$.)

b) Now modify your program to evaluate the same integral using the Romberg integration technique described in this section. Have your program print out a triangular table of values, as on page 161, of all the Romberg estimates of the integral. Calculate the error on your estimates using Eq. (5.49) and again continue the calculation until you reach an accuracy of $\epsilon = 10^{-6}$. You should find that the Romberg method reaches the required accuracy considerably faster than the trapezoidal rule alone.

Exercise 7: Heat capacity of a solid

Debye's theory of solids gives the heat capacity of a solid at temperature T to be

$$C_V = 9V\rho k_B \left(\frac{T}{\theta_D}\right)^3 \int_0^{\theta_D/T} \frac{x^4 e^x}{(e^x - 1)^2} \, dx,$$

where V is the volume of the solid, ρ is the number density of atoms, k_B is Boltzmann's constant, and θ_D is the so-called *Debye temperature*, a property of solids that depends on their density and speed of sound.

a) Write a Python function `cv(T)` that calculates C_V for a given value of the temperature, for a sample consisting of 1000 cubic centimeters of solid aluminum, which has a number density of $\rho = 6.022 \times 10^{28} \, \mathrm{m}^{-3}$ and a Debye temperature of $\theta_D = 428 \, \mathrm{K}$. Use Gaussian quadrature to evaluate the integral, with $N = 50$ sample points.

b) Use your function to make a graph of the heat capacity as a function of temperature from $T = 5 \, \mathrm{K}$ to $T = 500 \, \mathrm{K}$.