Computational Physics
Roy Rinberg
Andy Haas
HW 6

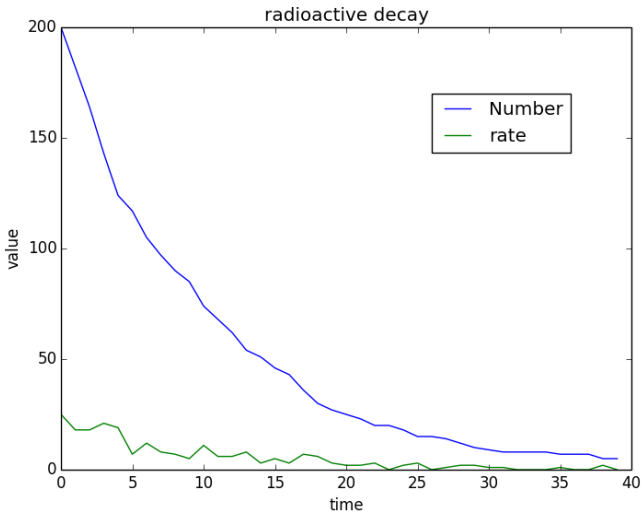
- 7.1/7.2.1 (quantum states in a box) - try bisection, Newton, and scipy.root

Looking at the programs in the book, and using their code, I was able to solve for the root of some functions.
I was unable to find the root using newton raphson, most likely because I did not include backtracking, however, I was able to find a result that agreed to the 8th decimal using bisection and the scipy method (of plugging it in to root.optimize.scipy)

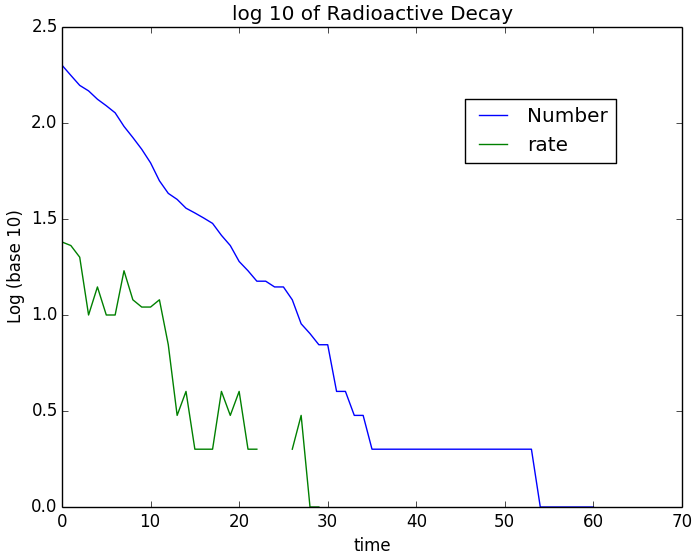
('Root by bisection=', 0.0040192094276339605)
('Root by root.optimize.scipy=', 0.0040192624533292491)

- **7.8 1 (fit ln(decay data) to a line), using code like in Fit.py (listing 7.4)**

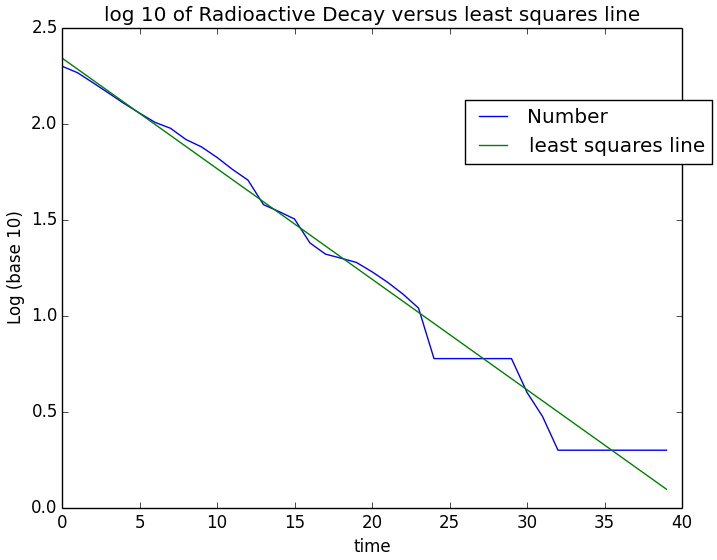
To solve it by taking the log of the data, and fitting it to a line, we can take the log of the data, do least squares or Scipy.optimize.curvefit (as we do in the 3rd problem) and then take it to the exponent again.



Rather than spending time guessing what the plotted figure they had in figure 7.6, I used the exponential decay function that we made in the 3rd week. The specific decay rate is not extremely informative, as we are developing a method, not a physical thing. (The error of this data set is a little bit unclear, but we are not really asked to do anything with the error).



Then, I applied the least squares method used in numpy, to generate a least squares function:



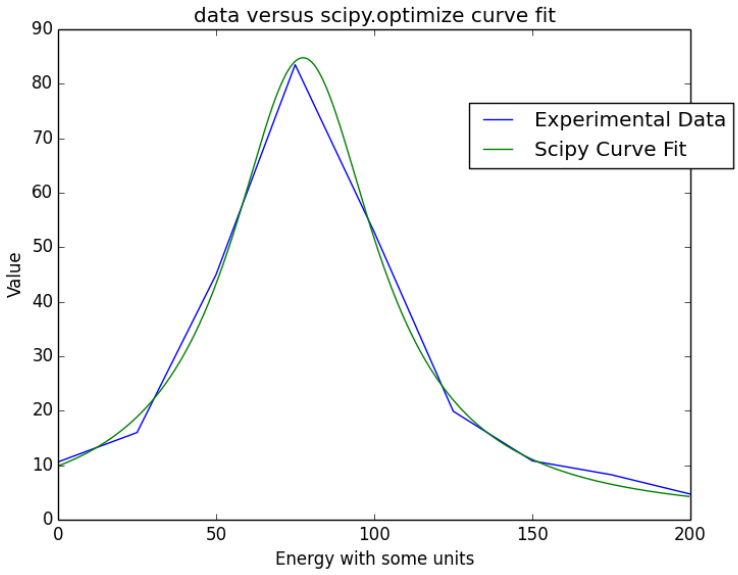
Slope and offset:
(-0.011902817902817922, 7.9684575424575517)
I operated this least squares on the number, rather than the rates, because the rates are sometimes found to be NaN, and so, difficult to apply least squares. The rate and the number should have the same slope and only different offsets.

This fit seems pretty good. But I am unclear how to actually quantify the error of the least squares fit.

The lifetime is like around

- Problem 5 (7.8.2): Fit the Breit-Wigner function to the data, using scipy.optimize.curve_fit. For extra credit, write a program to do this without using curve_fit (it's OK to use scipy.optimize.newton)

Using Scipy.optimize.curvefit- I get this:



for the function

$$f(x) = \frac{A}{(x-B)^2 - C}$$

A = 66886.39690237 B = 77.493579 C = 788.88396468

C = G2/4
G = 56.1738

With a standard deviation (error)s of the 3 variables
[3.43863531e+03 7.26104083e-01 4.97785894e+01]