Python lab 2

February 12, 2018

In this lab, we'll explore root finding a bit further. We'll play with the functions in the `scipy.optimize` package and we'll explore a challenging situation where things can go awry. In particular, let's play with $f(x) = x^5 - x - 1$. We can use the `newton` command in `scipy.optimize` starting the iteration from $x = 0$ as follows:

```
In [ ]: from scipy.optimize import newton
        def f(x): return x**5 - x - 1
        newton(f,0)
```

Question 1: What do you think of this answer??

As it turns out, the `newton` command doesn't actually use Newton's method when called this way - it uses the secant method. This makes some sense when you consider the fact that SciPy is *purely numeric* and doesn't really have quick, reliable access to f'. You can coerce `newton` into using, well Newton, buy specifying the derivative in the optional third argument. Thus,

```
In [ ]: def fp(x): return 5*x**4 - 1
        newton(f,0,fp)
```

Damn! At this point, you should grab some code from our Finding roots of real functions so we can try to figure out what the hell is going on!

Question 2: Modify our Newton's method code to print the intermediate values as it goes along. Then, apply that code to this problem to determine why Newton's method failed here.

Actually, the result returned from `newton(f,0)` was much worse; we just get an incorrect answer with no indication that the answer is wrong. One thing that's cool about SciPy, is that we can examine the code to try to get a grip on what's going wrong. You can examine the code for `newton` here: https://goo.gl/9V5sys. The portion that defines the Secant Method that is used in this case starts at line 192. There are two relevant points, if we want to try to diagnose our problem.

1. If we call `newton(f,x0)`, the second point used for the Secant Method is just a small perturbation of `x0`. If `x0=0`, this works out to be 0.0001.
2. The stopping criterion is $|x_i - x_{i+1}| < 1.48 \times 10^{-8}$.

Question 3: Modify our secant method code to print the intermediate values as it goes along. Then, apply that code to this problem to determine why it failed here.

If you look in `scipy.optimize`, you'll notice some other techniques - some of which work when started at $x_0 = 0$. In spite of this, **the correct approach is to start the iteration at a reasonable point**.

Question 4: Take a quick look at a graph to get a grip on how many real roots this function has and choose a good starting point to get one of them. with Newton's method.

Finally, NumPy has a `roots` command that finds all the complex roots of a polynomial. Note that the polynomial must be specified as a list of coefficients. Thus, we can find all the complex roots of this polynomial like so:

```
In [ ]: from numpy import roots
        roots([1,0,0,0,-1,-1])
```