[PHYS-GA2000] Problem Set 6

Dylan Lane
Github: dl4729

November 15, 2023

Problem 1

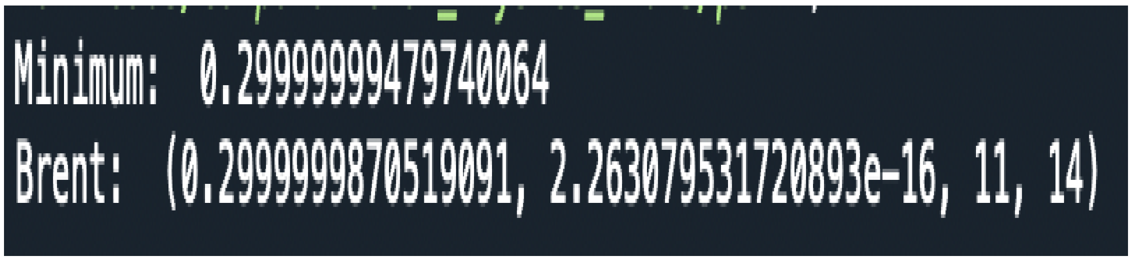
Methods and Results

I employed a bracketing method of specifying the two endpoints a, c and calculating the midpoint of these to use as b, swapping a, b, c such that the $f(b)$ is the smallest of the three. I then used quadratic interpolation between the three points; if the resulting s is within the interval determined by a, b, c, then the smallest subinterval bounded by any pair of a, b, c for which s is on the interior is chosen as the bracket for the next iteration. If s lies outside a, b or c then a golden mean step is chosen; b is set such that the distance between a, b is a golden ratio of the distance from a to c, another point x is calculated to be a golden distance away from the other endpoint, and the subinterval is chosen between among these points according to function evaluations at x and b. This is iterated until the distance between a and b is below some tolerance ($1e-7$ in my implementation).

There are several notes to be made about this result. Firstly, convergence is highly dependent upon the initial choice of bracket. Convergence to the true minimum (trivially 0.3 in the case of $f(x) = (x - 0.3)^2 e^x$ as it is the only point at which $f(x)$ is not strictly positive) occurs in certain "nice" brackets (for example, $(0, 1)$, $(-1, 1)$, $(0, 10)$); for a very low lower bound (very negative), the result will tend towards the lower bound (which is reasonable as a local minimum on a compact interval), and for a very high upper bound, it will generally converge to something that is not a local minimum at all. I am unconvinced my implementation is correct; I do not explicitly compute an acceptable value of b and I specify a and c instead of b as in the hints. I expect the first iteration to correct this inadequecy and find a suitable interior point immediately, but I have not proven that this is the case and I can justify it only by the fact that the results are (sometimes) correct.

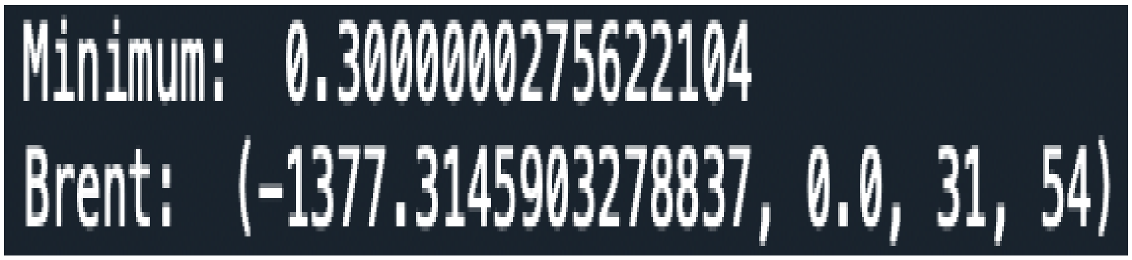
The comparison to Scipy's implementation of the Brent method is mixed. For good initial brackets like $(0, 1)$, Scipy retrieves an answer in 11 iterations while mine takes 30. This makes me concerned my implementation is wrong and is not properly taking parabolic steps, but the values for s are such that parabolic steps should still be being taken. Mine has greater precision despite using the same tolerance in both methods. For worse brackets (say $(-1, 1)$), my method still converges to the true minimum of 0.3 while Scipy results in a large negative number after a comparable number of iterations. Two such results are shown in the figures.

Figures



```
Minimum:  0.2999999479740064
Brent:  (0.2999999870519091, 2.263079531720893e-16, 11, 14)
```

Figure 1: Compared Results: Minimum (mine) vs. Scipy Brent, initial bracketing $(0, 1)$. Mine took approximately 30 iterations.



```
Minimum:  0.3000000275622104
Brent:  (-1377.3145903278837, 0.0, 31, 54)
```

Figure 2: Compared Results: Minimum (mine) vs. Scipy Brent, initial bracketing $(-1, 1)$.

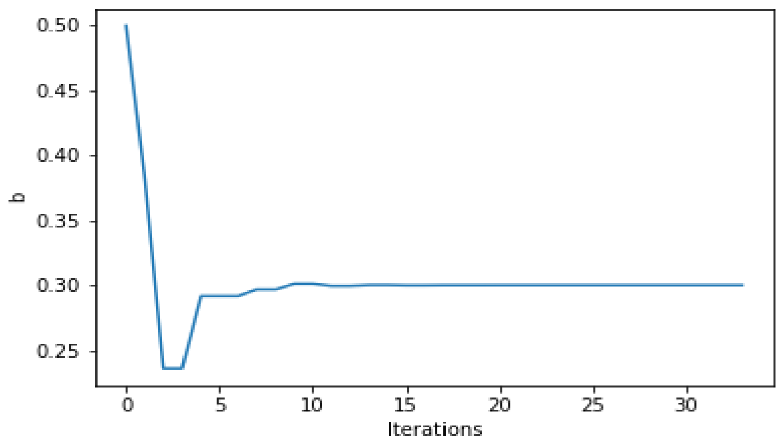


Figure 3: Plot of the interior values of Brent's method on each iteration.

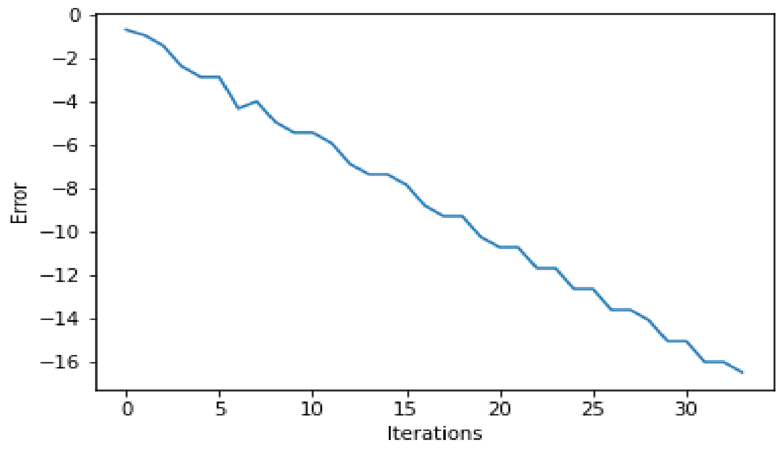


Figure 4: Errors $|a - b|$ on each iteration of Brent's method, log-scaled.

Problem 2

Methods and Results

I begin by loading the data and defining the logistic function to be minimized. The log likelihood is calculated according to

$$L(\beta_0, \beta_1) = \sum_{i=0}^{N} (y_i \log(\frac{p_i}{1 - p_i}) + \log(1 - p_i)) \tag{1}$$

where $p = \frac{1}{1+\exp(-(\beta_0+\beta_1 x))}$. The negative log likelihood is then minimized by scipy.optimize.minimize, outputting the values of β_0, β_1 which minimize L, the inverse hessian matrix, and the variance with some processing. The resulting probability curve is shown in figure (5); it agrees qualitatively well with the data, and the logistic function reaches the vertical midpoint (0.5) after most of the $y = 0$ points have been passed but before most of the $y = 1$ points have been attained.

Note: I was unable to install Jax to do the derivatives by hand. I encountered consistent dependency issues regarding circular installations that I could not resolve.

Figures

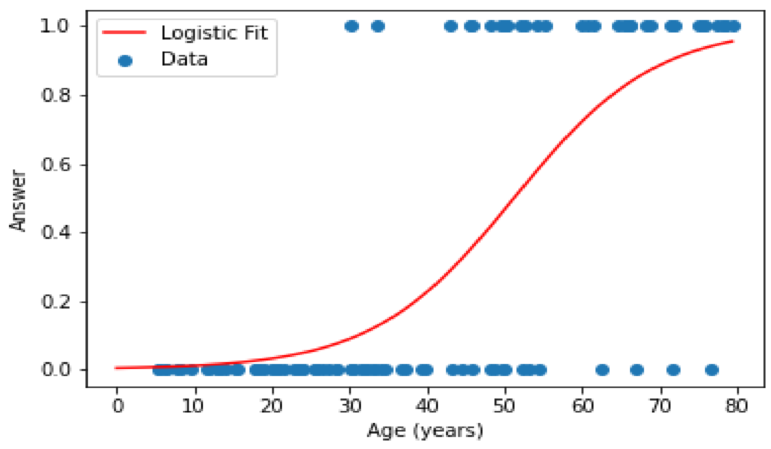


Figure 5: Logistic Fit using log-likelihood minimization.