
Problem 1. *Implementing Brent's method for minimization*

Solution.

This problem set tasks us with manually implementing Brent's one-dimensional minimization function. Brent's method combines golden section search and successive parabolic interpolation. Successive parabolic interpolation converges with an order of $\alpha = 1.325$ however, it is not always guaranteed to converge. Although the golden section search converges with an order of $\alpha = 1$, it is a good method to fall back on since it is guaranteed to converge. So Brent's minimization method gives us a good mix of robustness as well as efficiency.

For our implementation, we will use parabolic approximations while keeping track of the bracketing interval and the steps taken up to two iterations before the current one. The function takes user-defined limits for the bracket and sets them to **a** and **c**. It then finds a value in between these using the golden ratio and sets it to **b** as:

$$b = a + \frac{3 - \sqrt{5}}{2} * (c - a)$$

The main loop runs until the tolerance is reached or the maximum number of iterations is reached. At the beginning of each loop, it calculates the minimum of the parabola fitted using the three bracket points (**a**, **b**, and **c**). It then checks two conditions to decide which minimization method should be used in the current iteration:

- New minimum is within the brackets $a \leq \text{pot_min} \leq c$
- Parabolic step is less than the step before last

If those two conditions are met, the function uses the minimum from the parabolic approximation as the new best minimum. If it fails to meet those conditions, the function reverts to the more robust golden section search. This helps when the parabolic step starts to wander off from the minimum.

I have repurposed the functions for the parabolic step and some portions of the golden section search from the lecture notebooks. Figure 1 shows the steps taken by the program when provided a bracketing interval of (0,1) with the function:

$$y = (x - 0.3)^2 \cdot e^x$$

The figure shows that the function did a good job of selecting appropriate steps and alternating between parabolic steps and golden section search as appropriate.

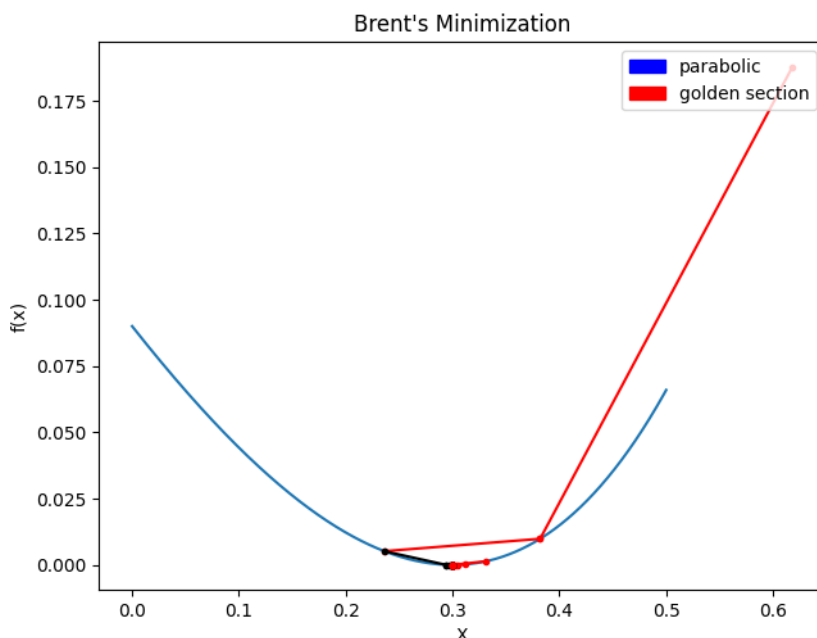


Figure 1: Steps taken in Brent

Comparing my implementation with `Scipy's`, we got results that differed in the order of $1.5e^{-6}$. Although the minimums were close, `Scipy's` Brent took only 12 iterations to find the minimum while mine took 21.

```
/Users/prithivirana/PycharmProjects/Co
Scipy's Brent:
  Minimum: 0.3000000000023735
  Iterations 12
2023-11-07 22:54:46.836 Python[3988:14
My Brent:
  Minimum: 0.3000015015529384
  Iterations 21
```

Figure 2: Comparing implementations

Problem 2. *The second problem tasks us with performing a likelihood maximization problem. We have results from a hypothetical survey with a yes or no question: “Do you recognize the phrase ‘Be Kind, Rewind’, and know what it means?” Our hypothesis is that the answer should depend on the age.*

Solution. For this problem, we will analyze the results to look for a correlation using logistic regression. We will find the maximum likelihood values and formal errors and covariance

matrix of β_0 and β_1 where we get the β values from modeling the probability as the logistic function:

$$p(x) = \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x)]}$$

Now let's derive our negative log likelihood function. First, let $z = (\beta_0 + \beta_1 x)$ for better readability.

From our logistic function, the probability of getting a response "yes" given the responder's age x_i is given by:

$$p(x_i) = \frac{1}{1 + e^{-z}} \quad (1)$$

The probability of getting a response "no" given age x_i is given by:

$$1 - p(x_i) = \frac{1}{1 + e^z} \quad (2)$$

Dividing equation (1) by (2) and taking its natural log gives us:

$$\ln \left(\frac{p(x_i)}{1 - p(x_i)} \right) = z \quad (3)$$

For a single observation (x_i, y_i) , the likelihood would be given by:

$$L(x_i, y_i) = p(x_i)^{y_i} \cdot (1 - p(x_i))^{(1-y_i)}$$

Thus, the likelihood of the full dataset is:

$$L = \prod_{i=1}^N p(x_i)^{y_i} \cdot (1 - p(x_i))^{(1-y_i)}$$

With the log-likelihood (we will multiply by a - sign at the end to get the negative log likelihood):

$$\begin{aligned} \ln L &= \sum_i y_i \ln p(x_i) + (1 - y_i) \ln[1 - p(x_i)] \\ &= \sum_i \ln[1 - p(x_i)] + y_i [\ln p(x_i) - \ln(1 - p(x_i))] \\ &= \sum_i \ln[1 - p(x_i)] + y_i \ln \left[\frac{p(x_i)}{1 - p(x_i)} \right] \end{aligned}$$

From Equation (3)

$$\begin{aligned} &:= \sum_i \ln \left[\frac{1}{1 + e^z} \right] + y_i \cdot z \\ &= \sum_i -\ln(1 + e^z) + y_i z \end{aligned}$$

This gives us the negative log likelihood function:

$$-\ln L = \sum_i \ln(1 + e^z) - y_i z \quad (4)$$

We can easily see that $(1 + e^z) \geq 1$. So, reworking our equation this way ensures that we are never taking the log of 0. I implemented this function in my program as follows:

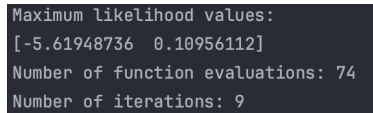
```
#negative log likelihood function
def negloglike(beta,x,y):
    b0 = beta[0]
    b1 = beta[1]
    z = b0 + (b1 * x)
    lnpi = jnp.log(1 + jnp.exp(z)) - (y * z)
    nll = jnp.sum(lnpi)
    return nll
```

I used the `jax.numpy` module for all NumPy functionalities in this program.

After reading in the data from the csv file, I set initial guesses for β_0 and β_1 to 0 and ran SciPy's `optimize.minimize` function with the call:

```
optimize.minimize(fun = negloglike, x0 = initial_betas, jac=nll_grad,
args= (x,y), method='BFGS', tol=1e-6)
```

This function uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. `nll_grad` is the gradient of the negative log likelihood function, obtained using `jax.grad`.



```
Maximum likelihood values:
[-5.61948736 0.10956112]
Number of function evaluations: 74
Number of iterations: 9
```

Figure 3: Maximum likelihood values

This gives us maximum likelihood values, $\beta_0 = -5.6195$ and $\beta_1 = 0.1096$ after 74 function evaluations and 9 iterations. β_0 has an error of 1.057 and β_1 has an error of 0.208 (Figure 4).

```
Standard errors:
[1.0574917  0.02084758]
```

Figure 4: Errors

The errors were obtained by first getting the hessian using `jax.jacfwd` and passing the gradient we obtained earlier as the argument. This was assigned to a variable `h`. We then obtain the hessian matrix by calling `h(β_0, β_1)`. Inverting this using `jnp.linalg.inv` gives us the covariance matrix in Figure 5. Taking the square root of the diagonal elements of the covariace matrix using `jnp.sqrt(jnp.diag(cov))` gives us the standard errors. Plotting

```
Covariance matrix:
[[ 1.1182886e+00 -2.1114409e-02]
 [-2.1114409e-02  4.3462167e-04]]
```

Figure 5: Covariance matrix

the survey data and the logistic model with the maximum likelihood values of β_0 and β_1 we obtained earlier gives us the plot in Figure 6. The logistic model makes sense because it shows that as the age increases, the probability of the surveyed responding "yes" increases. We can see the logistic model crosses at $p = 0.5$ at $\frac{-\beta_0}{\beta_1} = 51.29$. Although this survey is hypothetical, let's dive into these results and see if lines up with historical events. A quick google search shows that a movie titled "Be Kind, Rewind" starring Jack Black released in 2008. A 18-20 year old in 2008 would be around 34 now so the probability increasing around then would make sense. Urban dictionary states the phrase "was used by video rental employees after handing customers their video tape." Reddittor [JuggaloMason](#) comments "Both Blockbuster and Movie Gallery/Movie Time had those stickers" and Blockbuster expanded internationally in the 1990s so it makes sense that people who saw those stickers would be 50 or over now!

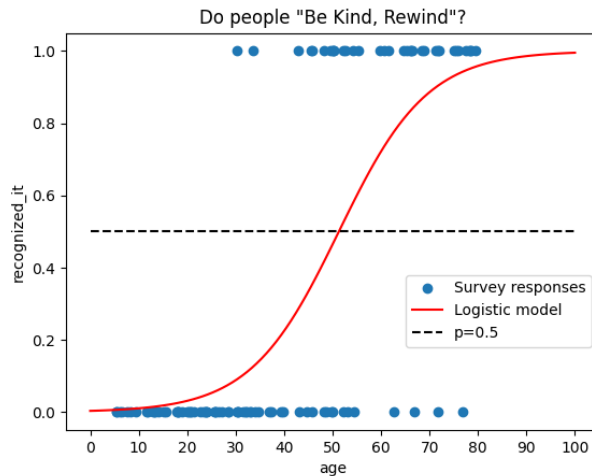


Figure 6: Logistic model vs answers