Jiachen Wan N19996964 | Computaitonal Physics | October 11, 2019

Homework 2

Problem 1

1. To start with, we know that:

$$x' = (1+\omega)f(x) - \omega x$$

Now apply Taylor expansion at $x = x^*$, and ignoring higer orders gives:

$$x' = [(1+\omega)f(x^*) - \omega x^*] + (x - x^*)[(1+\omega)f'(x^*) - \omega] + O((x-x^*)^2)$$

by definition, $f(x^*) = x^*$, so:

$$x' = (1+\omega)x^* - \omega x^* + (x - x^*)[(1+\omega)f'(x^*) - \omega]$$

$$= x^* + (x - x^*)[(1+\omega)f'(x^*) - \omega]$$

rearranging gives:

$$x' - x^* = (x - x^*)[(1+\omega)f'(x^*) - \omega]$$

Using $x^* = x' + \epsilon'$, we have:

$$\epsilon' = x^* - x' = (x^* - x)[(1+\omega)f'(x^*) - \omega] \tag{1}$$

and substitude in $x^* = x + \epsilon$ gives:

$$\epsilon' = \epsilon[(1+\omega)f'(x^*) - \omega] \tag{2}$$

and

$$x^* = x + \epsilon = x + \frac{\epsilon'}{(1+\omega)f'(x^*) - \omega} \tag{3}$$

Also since $x^* = x' + \epsilon'$:

$$x + \frac{\epsilon'}{(1+\omega)f'(x^*) - \omega} = x' + \epsilon' \tag{4}$$

$$x - x' = \epsilon' - \frac{\epsilon'}{(1+\omega)f'(x^*) - \omega} \tag{5}$$

So:

$$\epsilon' = \frac{x - x'}{1 - \frac{1}{(1+\omega)f'(x^*)-\omega}} \tag{6}$$

Since x is close to x^*:

$$\epsilon' = \frac{x - x'}{1 - \frac{1}{(1+\omega)f'(x)-\omega}} \tag{7}$$

2. A code for relaxation is written using $x' = f(x)$. The staring position is x=1. It took 13 steps to converge to an accuracy of 10^{-6}.

```
In [1041]: #relaxation
           #define constant
           c=2.
           #define the requred function
           def f(x):
               global c
               return 1-e**(-c*x)

           #define a starting position
           x=1.0
           #define a starting epsilon
           epsilon=abs(f(x)-x)
           #define n
           n=0

           while epsilon >= 10**-6:
               # x'=f(x)
               x=f(x)
               n+=1
               epsilon=abs(x-f(x))

           print('the number of steps is '+str(n))

           the number of steps is 13
```

Figure 1: Screenshot of the code and output

3. The code from the previous part is edited according to $x' = (1+\omega)f(x) - \omega x$. ω is set to 0.5 by trial and error. Using overrelaxation, the number of steps it takes to converge is largely reduced, to only 4 steps.

```
In [1043]: #overrelaxation
           #define constant
           c=2.
           #define the requred function
           def f(x):
               global c
               return 1-e**(-c*x)

           #define a starting position
           x=1.0
           #define learning rate
           omega=0.5
           #define a starting epsilon
           epsilon=abs(f(x)-x)
           #define n
           n=0

           while epsilon >= 10**-6:
               # x'=f(x)
               x=(1.+omega)*f(x)-omega*x
               n+=1
               epsilon=abs(x-f(x))

           print('the number of steps is '+str(n))

           the number of steps is 4
```

Figure 2: Screenshot of the code and output

4. If the next iteration would overshot, then an $\omega < 0$ would make convergence faster. That is, if $x < x^*$ but $f(x) > x^*$, then it is a good idea to make the step smaller by setting $\omega < 0$.

Problem 2

1. Starting with equaiton:

$$I(\gamma) = \frac{2\pi hc^2\lambda^{-5}}{e^{hc/\lambda k_B T} - 1}$$

Taking the derivative of $I(\lambda)$:

$$\frac{dI}{dt} = \frac{[e^{hc/\lambda k_B T} - 1](-5)(2\pi hc^2\lambda^{-6}) - (\frac{-hc}{K_B T}\frac{1}{\lambda^2}e^{hc/\lambda k_B T})(2\pi hc^2\lambda^{-5})}{[e^{hc/\lambda k_B T} - 1]^2} \tag{8}$$

so the maximum or minimum is at:

$$\frac{dI}{dt} = 0 = \frac{[e^{hc/\lambda k_B T} - 1](-5)(2\pi hc^2\lambda^{-6}) - (\frac{-hc}{K_B T}\frac{1}{\lambda^2}e^{hc/\lambda k_B T})(2\pi hc^2\lambda^{-5})}{[e^{hc/\lambda k_B T} - 1]^2} \tag{9}$$

$$0 = [e^{hc/\lambda k_B T} - 1](-5)(2\pi hc^2\lambda^{-6}) - (\frac{-hc}{K_B T}\frac{1}{\lambda^2}e^{hc/\lambda k_B T})(2\pi hc^2\lambda^{-5}) \tag{10}$$

$$0 = 5e^{-hc/\lambda k_B T} + \frac{hc}{K_B T\lambda} - 5 \tag{11}$$

Now let $x = hc/\lambda k_B T$, we have the equation:

$$5e^{-x} + x - 5 = 0 \tag{12}$$

And its solution can be used to calculate the wavelength or temperature, using the Wien displacement law:

$$\lambda = \frac{b}{T} \tag{13}$$

where:

$$b = hc/k_B x \tag{14}$$

2. A code for binary search is written, with tolerance 10^{-6}. The starting and ending points are $a = 0.1$ and $b = 10$, so that f(a) and f(b) have different sign. The result is:

$$x = 4.9651 \tag{15}$$

3. Given that $\lambda = 502nm$, $c = 299792458m/s$, $h = 6.62607004 * 10^{-34}m^2kg/s$, $k_B = 1.38064852 * 10^{-23}m^2kgs^{-2}K^{-1}$, and $x = 4.9651$:

$$T = \frac{b}{\lambda} \tag{16}$$

$$T = \frac{hc}{\lambda k_B x} \tag{17}$$

plugging the constants in,

$$T = 5772.4726K \tag{18}$$

Which is the surface temperature of the Sun. According to google, the true value is 5778 K, so my estimate is quite reasonable.

Problem 3

1. First of all, I have written the code for gradient descent. The basic formula is:

$$x_3 = x_2 - \gamma \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{19}$$

but written in 2 and 3 dimensions, of course. To demonstrate it works, I am to find the minimum of:

$$f(x,t) = (x-2)^2 + (y-2)^2 \tag{20}$$

Its minimum is located at (2,2). In my code, ϵ is defined as the distance between x_2 and x_1, so when ϵ approaches 0, we know we are close to the local minimum. Ideally, after many iterations, ϵ should approach 0. The solution is thus converged to the position of local minimum.

So to find the minimum of the above equation, I have set the starting point at (1.1,1.1), the initial δx as (0.1,0.1), and the learning rate γ to be 1.0

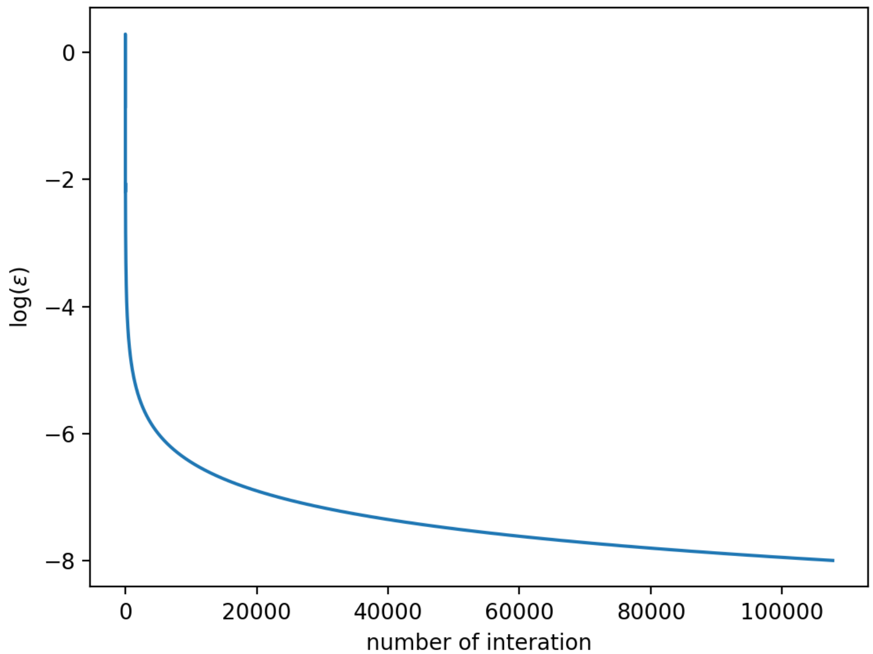

Figure 3: Plot of ϵ vs the of number of iteration

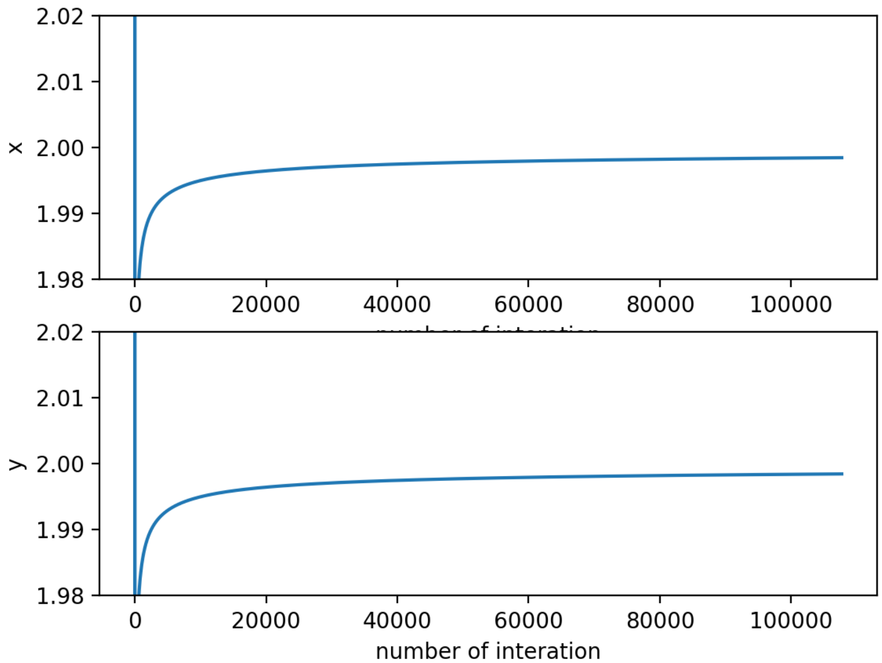

Figure 4: Plot of x and y vs the of number of iteration

From figure 3, it is clearly seen that ϵ converges to 0. I have set the tolerance to be 10^{-8}, so the iteration stops once ϵ reaches it. From figure 4, it is obvious that the solution converges to (2,2), which means gradient descent is working as expected.

I will now edit my gradient descent function to work in 3d, and apply it to the given data set to fit the Schechter function. The function to be fitted is:

$$n(M_{gal}) = \phi^*(\frac{M_{gal}}{M_*})^{\alpha+1}e^{-\frac{M_{gal}}{M_*}}ln(10) \tag{21}$$

Where the M_*, ϕ^*, and α are free variables to be find. For a given set of M_*, ϕ^*, and α, I can calculate the chi-square of the given data set. To make the fitting process easier, I have changed the parameter M_*, which has an order of magnitude of 10, to $M_* * 10^{10}$, so that M_* isn't way too large. So:

$$n(M_{gal}) = \phi^*(\frac{M_{gal}}{M_*10^{10}})^{\alpha+1}e^{-\frac{M_{gal}}{M_*10^{10}}}ln(10) \tag{22}$$

After tweaking parameters and starting points, the results are listed below:

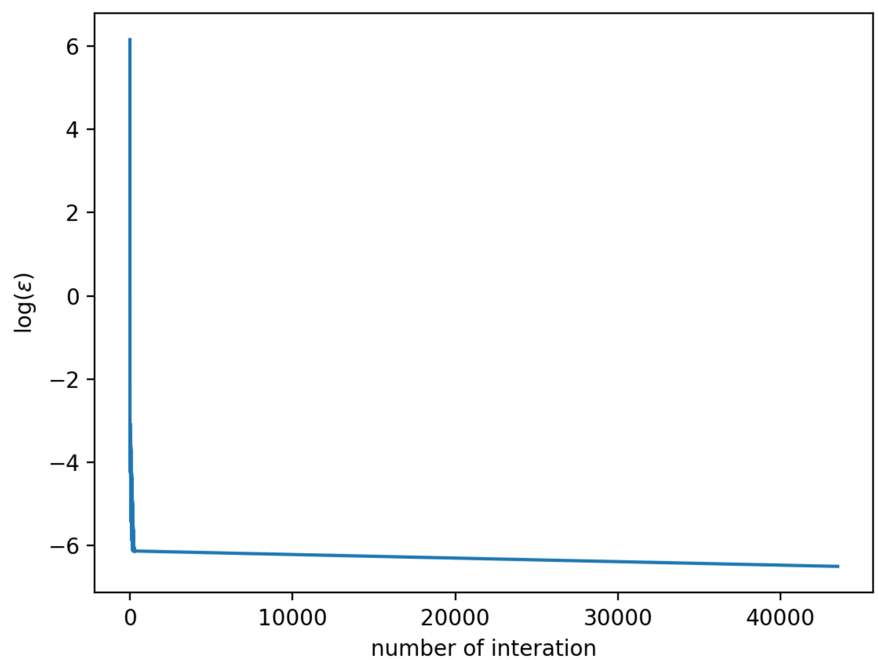

Figure 5: Starting points are: $M_* = 10.39$, $\phi^* = 0.003$, and $\alpha = -1.03$, learning rate=10^{-8}

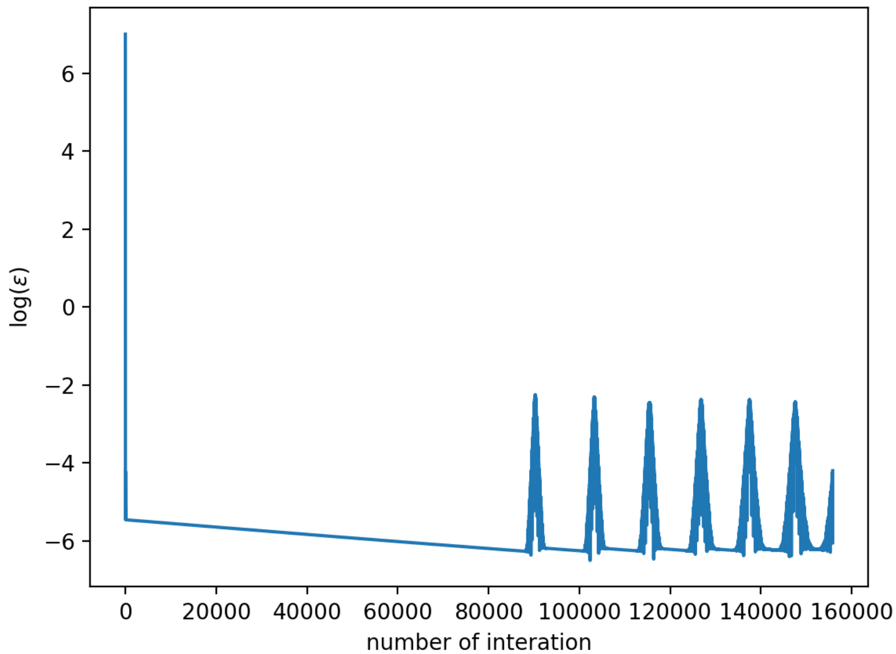

Figure 6: Starting points are: $M_* = 10.55$, $\phi^* = 0.001$, and $\alpha = -1.0$, learning rate=10^{-8}

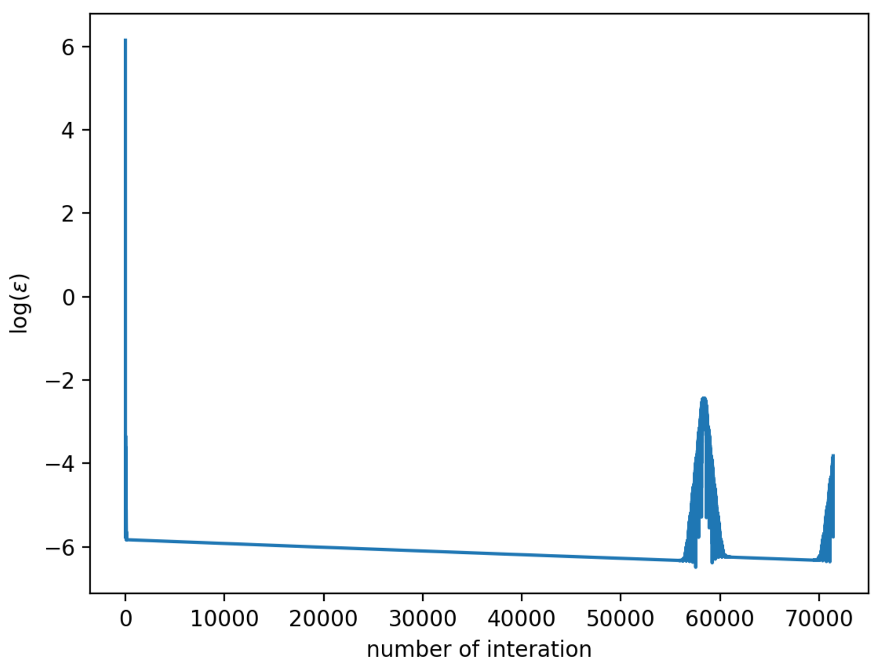

Figure 7: Starting points are: $M_* = 10.5$, $\phi^* = 0.002$, and $\alpha = -1.0$, learning rate=10^{-8}

The fitted parameters are:

$$(\phi^*, M_*, \alpha) = (0.00241565435, 10.5458867, -1.03735076) \tag{23}$$

$$(\phi^*, M_*, \alpha) = (0.00245046871, 10.3981070, -1.04158915) \tag{24}$$

$$(\phi^*, M_*, \alpha) = (0.00236733456, 10.5053719, -1.03968105) \tag{25}$$

And their chi-square are:

$$\chi^2 = 4.917404188810928 \tag{26}$$

$$\chi^2 = 4.02626351852366 \tag{27}$$

$$\chi^2 = 5.1050618753276416 \tag{28}$$

As demonstrated, the calculated minimums are quite close to each other, so the solution is likely to be accurate. Now lets choose the set of parameters that gives the lowest χ^2 and see how well it fits the data:

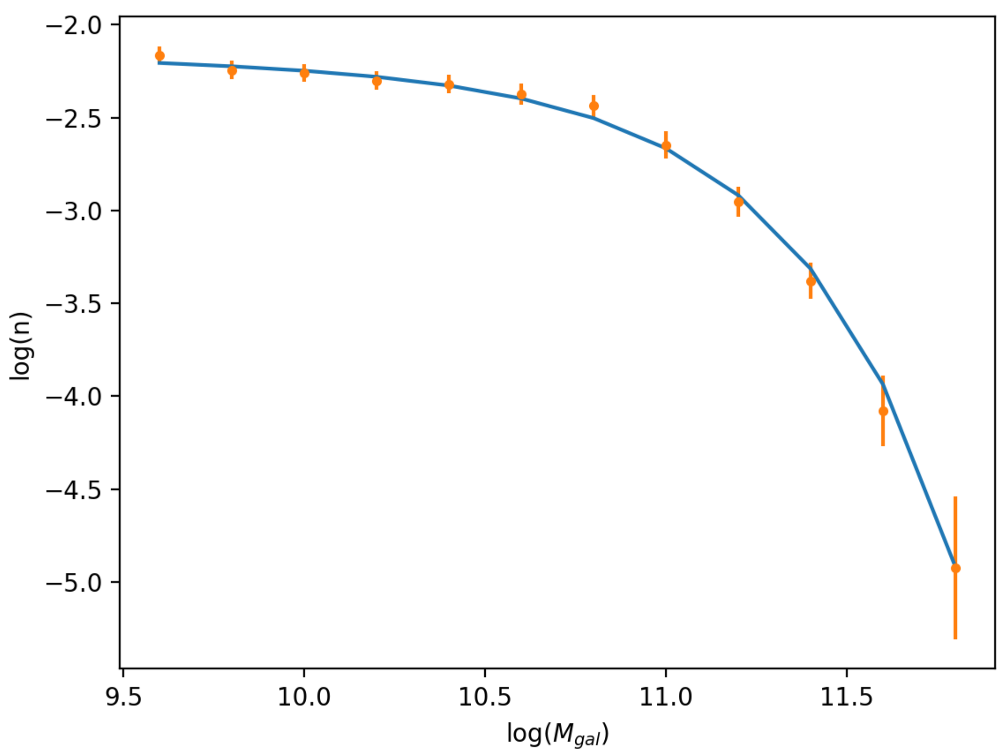

Figure 8: $(\phi^*, M_*, \alpha) = (0.00245046871, 10.3981070, -1.04158915)$, fitted line

The error bar is calculated using the following error propagation:

$$\delta log_{10}(x) = \frac{\delta x}{x ln(10)} \tag{29}$$

and I think it is a fairly good fit. However, it is important to note that ϵ from figure 6 and 7 are not smoothly converged. It is probably due to the fact that the equation we are fitting is non-linear.