Noah Kasmanoff
nsk367

Homework 2

Computational Physics
October 11, 2019

Problem 1

10/10

Q1 code

a) 6.10

a) The program to solve the relaxation method for $x = 1 - e^{-2x}$ is given in the link above, and finds that for a tolerance of 1e-6, $x = 0.796813$

b) For values of c from 0 to 3 in steps of 0.01 we recover the following plot for x as a function of c.

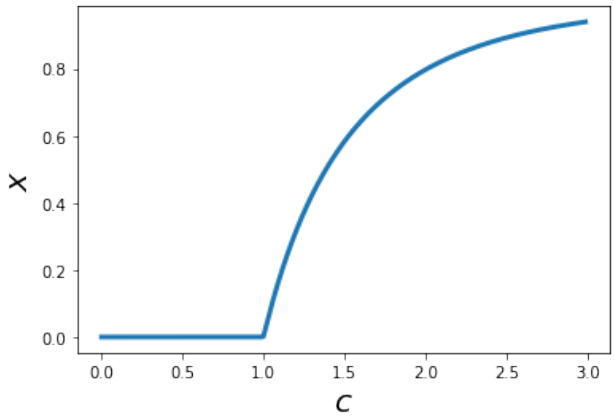


Figure 1: We observe a clear transition from a regime in which x = 0 to a regime of nonzero x. This is another example of a phase transition, and in physics this transition is known as the percolation transition; in epidemiology it is the epidemic threshold.

b) 6.11

a) Using the over-relaxation method, we recover a new formulation for the error on the new step taken after each iteration. Using what we know about how the new step x' is related to prior steps and function outputs x and $f(x)$, we can begin this process of deriving the new error by defining the following:

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

where $g(x)$ is just some newly defined value for the function at it's new value, $f(x')$.

Next, we expand the $g(x)$ about it's true value x^* through a Taylor series, giving us

$$g(x) \approx g(x^*) + (x - x^*)g'(x)$$

, where

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

. For simplicity, we set this derivative equal to this G.

Now by plugging in what we know, we find that the new formulation for x' is

$$x' = x^* + (x - x^*)G$$

now we apply the known values for error as $x^* = x + \epsilon = x' + \epsilon'$, and recover

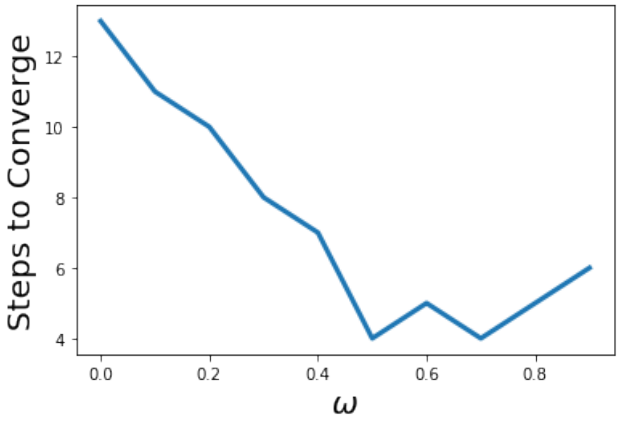
$$-\epsilon' = Gx - Gx' - G\epsilon'$$

, yielding

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

, thus giving the right quantity.

b) Taking the same operation from 6.10, we again solve $x = 1 - e^{-2x}$ using over relaxation. With different values of ω, we observe



good presentation

Figure 2: The number of steps needed to converge to the true value of x, given varying values of ω between 0 and 1.

c) And as stated in the question, there are scenarios where making $\omega < 0$ helps us find a solution than when we use ordinary relaxation.

In order for a situation like this to arise, we consider the case of when a function would not normally converge.

Generally speaking, the reason for such a failure is that the value of x at each iterative step never gets closer to it's true value. And in this same circumstance, if we allow the new value x' to be smaller than the relaxation prediction, it will gradually converge when it wouldn't. This is the utility of under-relaxation.

Problem 2

link to code

a) Applying the quotient rule to derive this function, and setting this value equal to 0, we recover the equation given.

Furthermore, we can make the relatively straightforward substitution of $x = \frac{hc}{\lambda k_b T}$, to provide a straightforward method of deriving Wien's law, formulated as $\lambda = \frac{b}{T}$ where b is in turn derived from the variable x. As for the derivative of Planck's radiation law, after substituting we are left with the far more accommodating equation $5e^{-x}+x-5 = 0$, and can compute the value for x numerically in the next question.

b) Through the implementation of the binary search method (found in q2.py), we recover $x = 4.97$ and consequentially can derive that the value of Wien's displacement constant b is equal to $b = \frac{hc}{k_b x} = 0.00289$

c) If we plug in the derived value for Wien's displacement constant above, and given that that peak wavelength for the Sun is $\lambda = 502$ nm, we recover that surface temperature of the Sun is approximately 5757 K.

Problem 3

link to code

a) To validate the success of the gradient descent method, we can try and apply to an analytically solvable question, which will be the function $f(x, y) = (x-2)^2 + (y-2)^2$, with the known solution of having a global minima of x=2, and y=2, and $f(x, y) = 0$.

By running the code linked above for question 3a, we recover the following diagram demonstrating how gradient descent eventually converges to the correct solution, returning the correct values based on our prior analysis of this easily solvable problem.

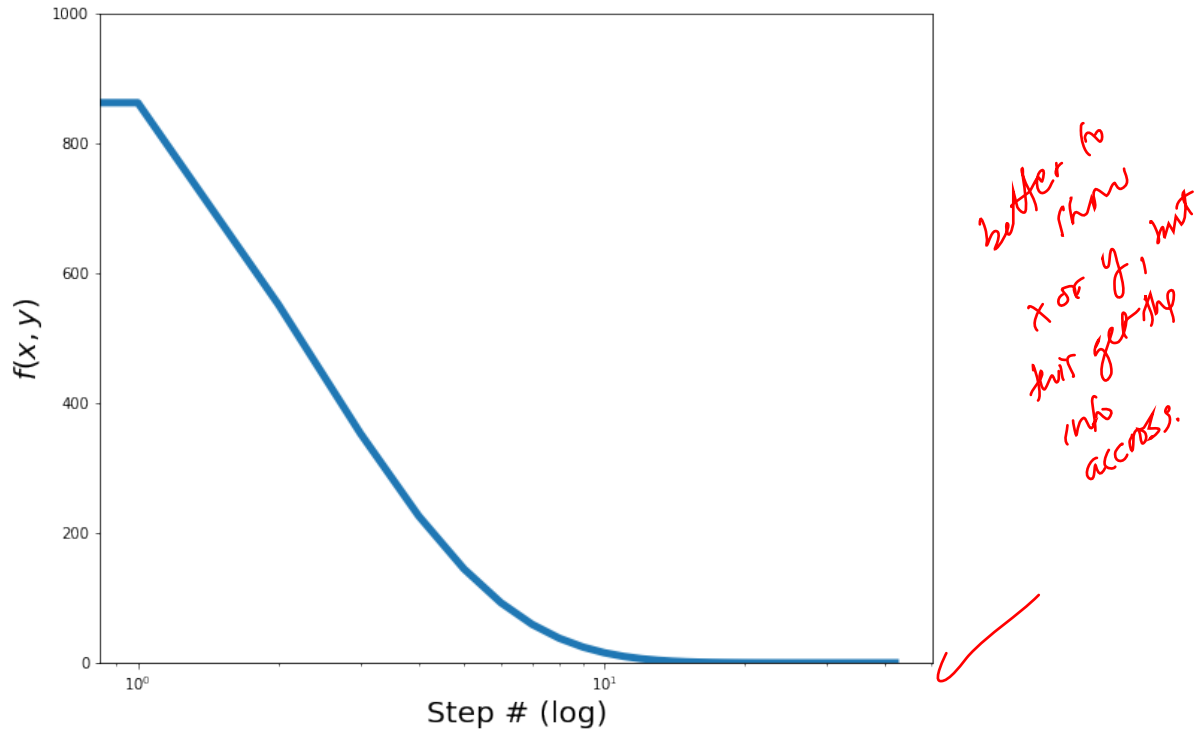


Figure 3: After each step in the gradient descent, we see that even despite starting so far away, the function gradually converges to it's minimum.

b) Next, we will try and fit the three free parameters of the Schecter function to this collected data courtesy of the COSMOS galaxy survey.

The Schecter function is defined as

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

,

where the number of galaxies n is equal to this function of the mass of the galaxy tuned by the free parameters of characteristic mass scale M_*, amplitude ϕ_*, and exponential cutoff α.

To apply gradient descent to this method, we have two options, whether to approach this problem linearly (or applying the gradient descent method directly to the parameters, which we will do to quickly illustrate the flaw in this method, especially when dealing with datasets and parameters of very high orders of magnitude.

Alternatively, we can "transform" these parameters into logarithm units.

To do so, we set each parameter equal to a base10 term, where for instance $\phi_* = 10^{\phi_{mag}}$ and now we can apply gradient descent to minimize our desired function (in this case, chi-squared) for ϕ_{mag} and the other likewise values.

We apply the logarithmic parameter method below, and obtain the following fit.

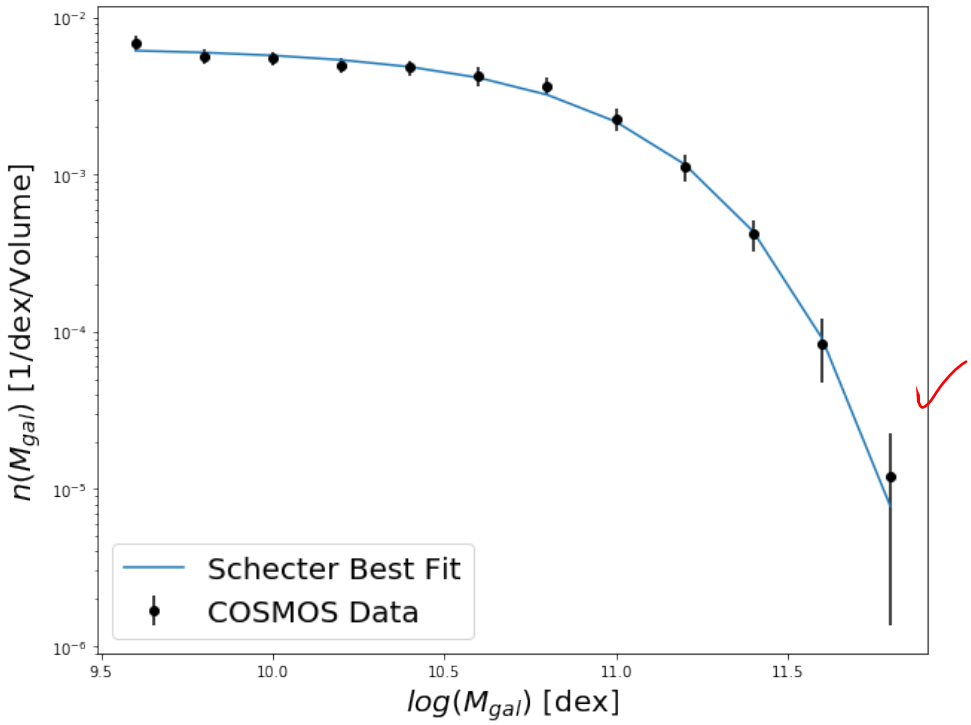


Figure 4: The best fit of the Schecter function.

In this model, we find that the best fit for these parameters are

$$\phi_* = 10^{-2.5664}$$
$$M_* = 10^{10.9750}$$
$$\alpha = -1.0076$$

Additionally, we need not fear that this result is a consequence of extremely well picked first parameters. To confirm this method's robustness, an important test is to see if the initial step size is too important and perfectly picked, so now we begin by plotting the convergence of different initial steps and confirming this hyper-parameter did not completely determine this method's convergence.

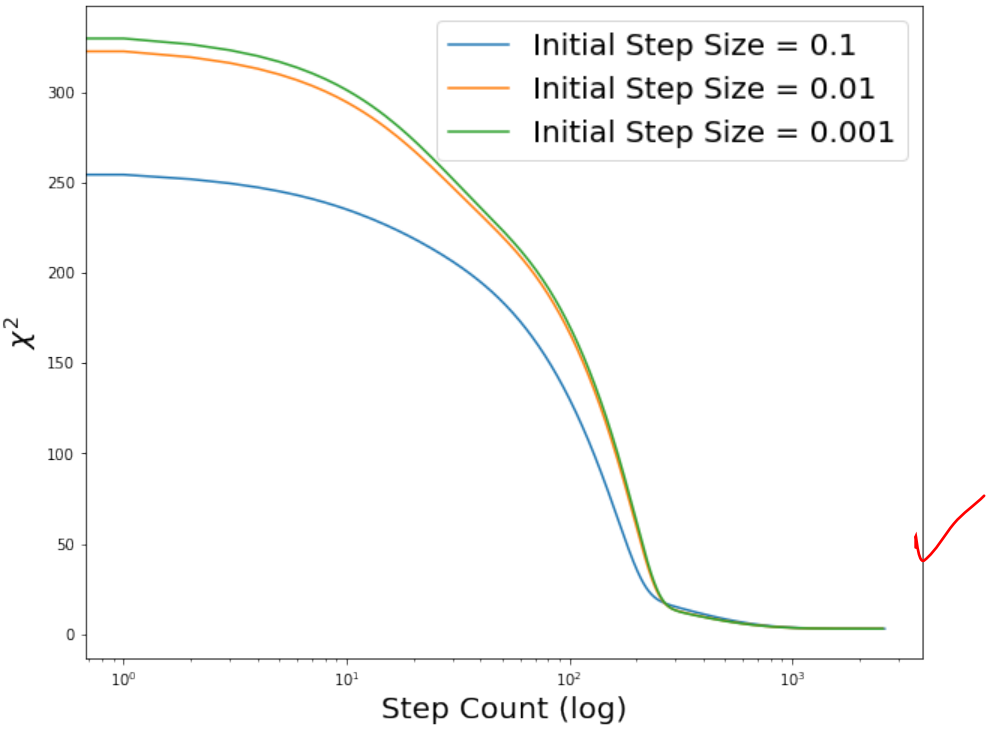


Figure 5: The χ^2 value for minimizing the Schecter function converges, regardless of our initial step size.

To further test for the robustness of gradient descent , we can observe the convergence rate of the routine, given different starting points, and different step sizes.

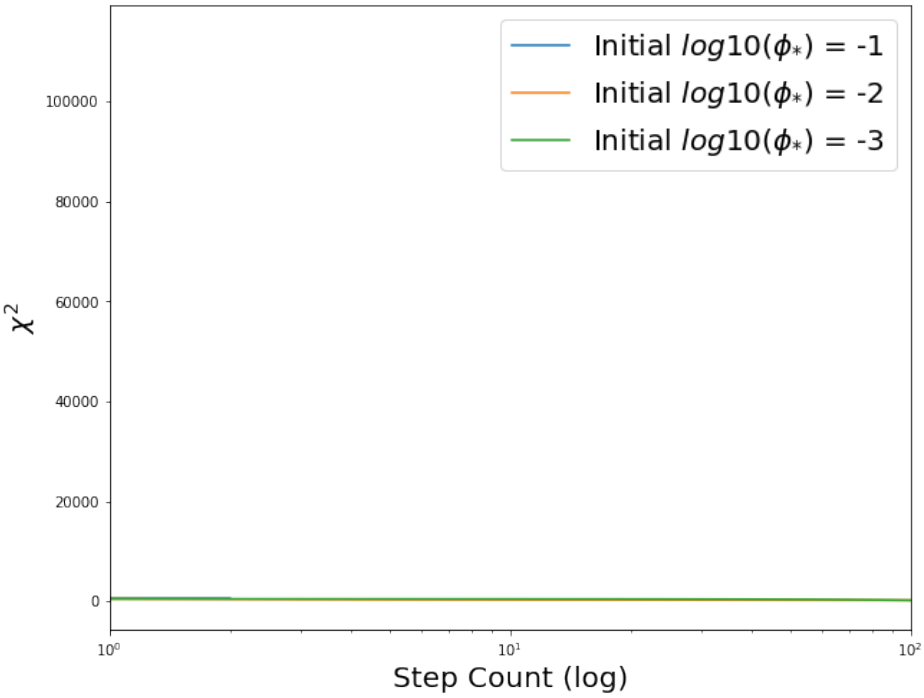


Figure 6: The χ^2 value for $\phi_* = .1$ never converges, and we are left with a bad plot!

Surprisingly, this test for robustness fails. When we set the initial parameter of $\phi_* = 10^{-1}$, we fail to find a converged function and instead see how this function fails relative to different starting ϕ_*s .

We can assert the gradient descent routine's robustness on the basis tweaking that algorithm's hyper-parameters, but we see that for the Schecter function fitting, picking a reasonable starting point is mandatory. If the distance between the "true" parameter and initial is too large, we fail to see the χ^2 value converge.

Additionally, the application of "logarithmic" stepping proved to be a valuable measure to greatly expedite (and even allow) for gradient descent to yield a good fit. If we were to try and fit this function through ordinary steps, we would never find a minima near what we have here.

Due to the extremely high order of magnitude of $M_* \approx 10^{11}$ and low magnitude of $\phi_* \approx 10^{-2.5}$, beginning at an an initial parameter of $M_* = 10^9$ for example would never yield a result as good as done through leveraging the logarithm.

To further emphasize this point we cannot even produce a plot of the evolution of the chi squared value in this regime, as the chi squared error becomes NaN in fewer than 3 iterations. As an appendix to this assignment I have created an extra file called

"lineargraddescent.py", along with the printed output of this file below, to demonstrate how poor of a method this is.

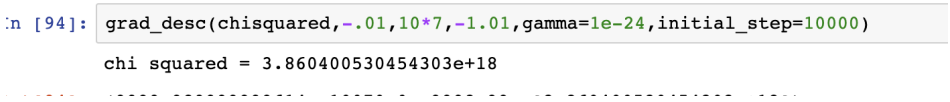


Figure 7: The sole output for a linear gradient descent in this problem. After this value, only NaN values are returned as the χ^2 overloads the memory.