10/10

Computational Physics, Homework 2

6

William Schiela

October 11, 2019

The files for this homework can be found at https://github.com/bschiela/comp-phys/hw2. Python code is located in the `python/` directory. Output data files and plots are located in `python/output/`. This LaTeXdocument is in the `latex/` directory.

0. Newman Exercise 6.10

Exercise 6.11 recommends completing Exercise 6.10 first. The script for this problem is `newman_6-10.py`. My implementation of the relaxation method is in `relax.py`.

a. Relaxation method

I first plot the functions x and

$$f(x) = 1 - e^{-cx} \tag{1}$$

for the particular case of $c = 2$ to obtain a graphical solution to the equation

$$x = 1 - e^{-cx} \tag{2}$$

as shown in Figure 1. The plot provides a good guide for which x values to use as starting points

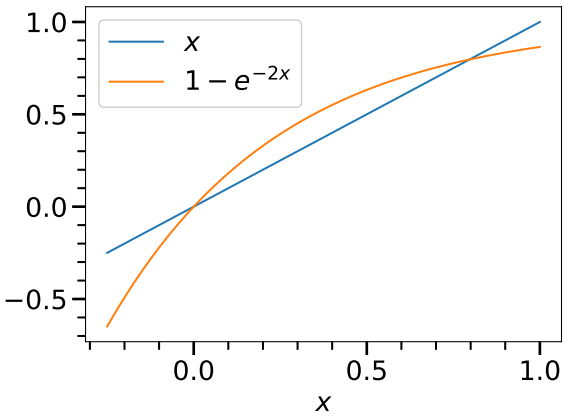

Figure 1: Graphical solution to the equation $x = 1 - \exp(-2x)$.

in the relaxation method. There is a trivial solution at $x = 0$ and a nontrivial solution near $x \approx 0.8$.

x	error
0.79681263	-5.0109078 $\times 10^{-7}$
8.4882237×10^{-7}	-8.4882381 $\times 10^{-7}$

Table 1: Numerical solutions to the equation $x = 1 - \exp(-2x)$ with accuracy 10^{-6}. The second solution is effectively 0 up to the accuracy threshold specified. Reproduced from `newman_6-10-a.txt` and truncated to 8 significant figures.

However, the relaxation method cannot be used to obtain the trivial solution to the equation in its current form, as

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=0} = 2 > 1.$$

To obtain the trivial solution using the relaxation method (for the sake of the exercise), the equation $x = f(x)$ must first be inverted for $x = -\ln(1-x)/c \equiv f^{-1}(x)$. Then, for $c = 2$,

$$\left.\frac{\mathrm{d}f^{-1}}{\mathrm{d}x}\right|_{x=0} = \frac{1}{2},$$

and the relaxation method will converge. The numerical results are shown in Table 1 where[1] $x = 0.1$ and $x = 1$ were used as starting values to obtain the trivial and nontrivial solutions, respectively.

b. Percolation transition

Figure 2 shows a plot of the solutions to Equation (2) as a function of the constant $c \in [0, 3]$, while Figure 3 shows graphical solutions for a few select values of c. The percolation transition (also

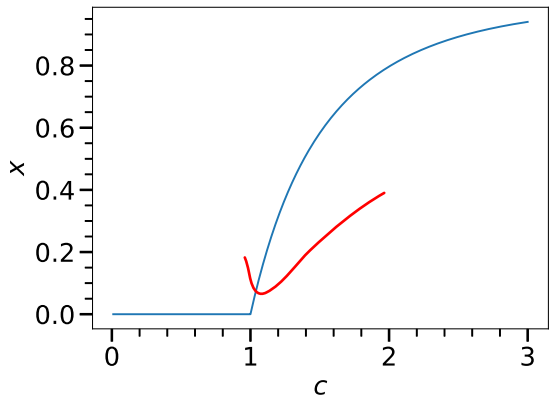

Figure 2: Solution to $x = 1 - \exp(-cx)$ as a function of c.

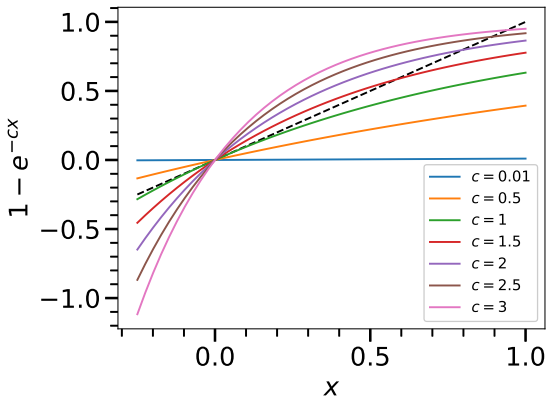

Figure 3: Graphical solutions to $x = 1 - \exp(-cx)$ for a few values of c. The dashed black line is the function x.

known as the epidemic threshold in epidemiology) occurs at $c = 1$, for which (1) is tangent to the function x at $x = 0$. Nontrivial solutions begin to appear for $c > 1$.

[1] An initial value of $x = 0$ was not used for the trivial case in order to allow the relaxation method to run a few iterations, again, for the sake of the exercise. In practice we are obviously more interested in finding the nontrivial solution.

1. Newman Exercise 6.11

My implementation of the relaxation method in `relax.py` supports overrelaxation via the function parameter `w`. For `w = 0`, this reduces to the ordinary relaxation method.

a. Overrelaxation error

In the ordinary relaxation method, the current estimate x is fed into the function $f(x)$ to obtain a new estimate,

$$x'_{\text{relax}} = f(x) \equiv x + \Delta x, \tag{3}$$

of the solution to $x = f(x)$. In the overrelaxation method, we attempt to get even closer to the true solution by modifying the shift Δx that results from the relaxtion method, yielding a new "overrelaxed" estimate,

$$x' = x + (1+\omega)\Delta x,$$

where the relaxation parameter ω must be chosen carefully. Using (3), we obtain an expression for the new overrelaxed estimate in terms of the current estimate and the current value of the function,

$$x' = g(x) \equiv (1+\omega)f(x) - \omega x. \tag{4}$$

To obtain an approximate value of the error ε' of the overrelaxed estimate, we Taylor expand (4) about the true solution x^*,

$$\begin{aligned} x' &\approx g(x^*) + (x - x^*)g'(x^*) \\ &= g(x^*) + (x - x^*)\left[(1+\omega)f'(x^*) - \omega\right] \end{aligned}$$

where higher-order terms have been neglected as we assume the current estimate is close to the true solution, i.e. $|x - x^*| \ll 1$. Now, note that (4) implies $g(x^*) = x^*$ since $f(x^*) = x^*$ by definition. Defining the errors, $\varepsilon' = x' - x^*$ and $\varepsilon = x - x^*$, of the new and current estimates, respectively, we obtain

$$\varepsilon' \approx \varepsilon\left[(1+\omega)f'(x^*) - \omega\right], \tag{5}$$

the overrelaxed analogue of Newman Eq. 6.81.

Finally, by the definitions of the errors we have $x^* = x' - \varepsilon' = x - \varepsilon$. Replacing ε via (5) and solving for ε' we obtain the overrelaxed analogue of Newman Eq. 6.83.,[2]

$$\varepsilon' \approx \frac{x' - x}{1 - 1/\left[(1+\omega)f'(x) - \omega\right]}, \tag{6}$$

where we have let $f'(x^*) \approx f'(x)$ under the previous assumption $|x - x^*| \ll 1$.

b. Number of iterations

My relaxation code in `relax.py` supports iteration counting via the return parameter `i`.

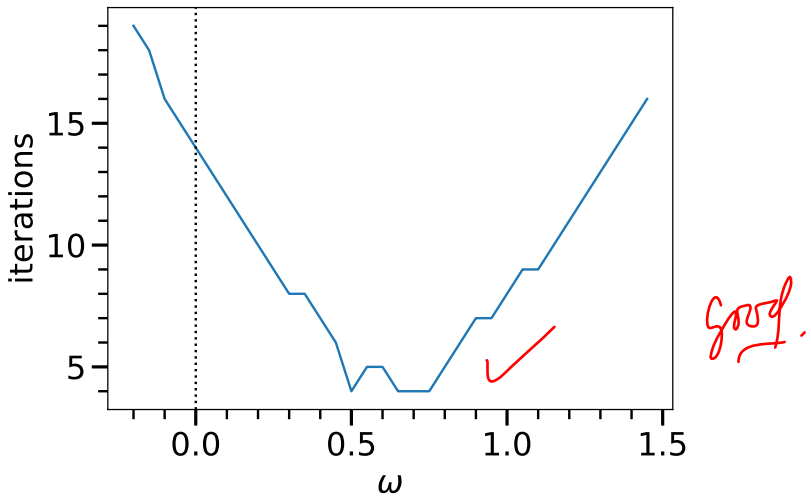

Figure 4: Effect of the relaxation parameter ω on the convergence rate. The dotted line indicates $\omega = 0$ where the overrelaxation method reduces to the ordinary relaxation method.

c. Convergence rate vs. relaxation parameter

Figure 4 shows the number of iterations, as a function of the relaxation parameter ω, required for the relaxation method to converge to within an accuracy of 10^{-6} when applied to Equation (2) with $c = 2$ and an initial guess $x = 1$. A table of results including the solution x, error, and number of iterations for each value of ω can be found in `newman_6-11-c.tsv`.

The point $\omega = 0$ corresponds to the ordinary relaxation method, while $\omega \neq 0$ corresponds to under- or overrelaxation. For $\omega < 0$ we observe underrelaxation whereby the convergence rate is negatively impacted by the use of a relaxation parameter. The optimal ω occurs in the range $\omega \in [0.5, 0.8]$ where overrelaxation achieves a minimum number iterations, 4.

One might think there should be an optimal value in the range $\omega \in [0.5, 0.8]$ for which the relaxation method finds the solution in a single iteration. Figure 5 shows the same plot with a smaller step size, and there appear to be a few "special" values of ω which deviate from the general pattern to produce rapid convergence, the absolute best being $\omega \approx 0.37112$ which converges in just 1 iteration. On inspection of the result, however, the obtained solution clearly deviates from the true solution by more than the required accuracy 10^{-6}, despite the reported error being less than 10^{-6}. Some of these points are therefore spurious and likely due either to cancellation errors in the computation of the relaxation estimate's error, or a breakdown (in early iterations) of the approximation $|x - x^*| \ll 1$ which led to Equation (6). In either case, the result is an underestimation of the error which terminates the algorithm prematurely.

d. Negative relaxation parameter

A negative relaxation parameter, $\omega < 0$, would improve the convergence rate if the slope of the function f in the equation $x = f(x)$ is negative at the true solution x^*. In that case, if the current

[2] This differs from Newman's result by an overall minus sign due to our different definitions of the error: Newman uses $x^* = x + \varepsilon$ while I use $x^* = x - \varepsilon$. In my opinion, the latter makes more sense: subtracting the error from the estimate should give you the true value. Then $\varepsilon < 0$ if the estimate is too small, and $\varepsilon > 0$ if the estimate is too big.

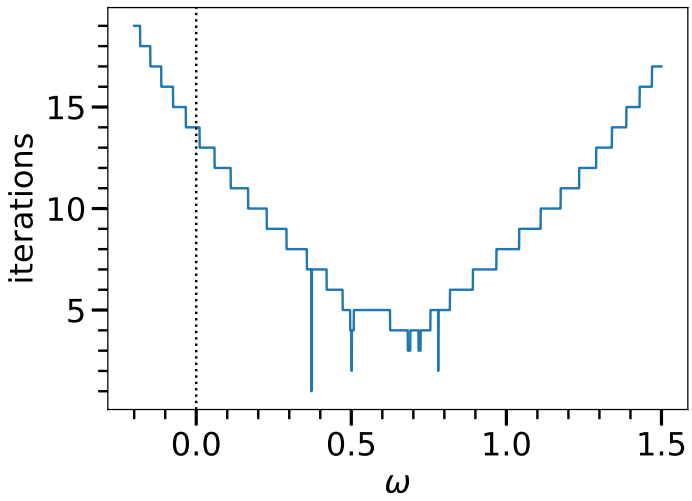

Figure 5: High-resolution version of Figure 4.

estimate $x < x^*$, then the new estimate

$$x' = f(x) \approx x^* + (x - x^*)f'(x^*) > x^*$$

and the relaxation method overshoots the true solution. Likewise if $x > x^*$ then $x' < x^*$ and the algorithm again overshoots. A negative value of ω would then reduce the overshoot and converge more quickly to x^*.

2. Newman Exercise 6.13

a. Wien displacement law

Given Planck's law for the intensity per unit area of blackbody radiation,

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

the wavelength at which the emitted radiation is strongest is obtained by extremizing with respect to λ,

$$0 = \frac{\mathrm{d}I}{\mathrm{d}\lambda} = \frac{2\pi hc^2}{\left(e^{hc/\lambda k_B T} - 1\right)^2}\left[-5\lambda^{-6}\left(e^{hc/\lambda k_B T} - 1\right) - \lambda^{-5}\left(\frac{-hc}{\lambda^2 k_B T}\right)e^{hc/\lambda k_B T}\right].$$

Discarding the nonzero prefactor and multiplying through by $\lambda^6 \exp(-hc/\lambda k_B T)$ we arrive at the expected result

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

after reordering the terms. Defining

$$x \equiv \frac{hc}{\lambda k_B T}, \tag{7}$$

this may be expressed more compactly as

$$5e^{-x} + x - 5 = 0, \tag{8}$$

and from the definition (7) we obtain the Wien displacement law

$$\lambda = \frac{b}{T}, \quad \text{with} \quad b \equiv \frac{hc}{k_B x} \tag{9}$$

where b is the Wien displacement constant in terms of x, the solution to (8).

b. Binary search for displacement constant

I have implemented a recursive binary search algorithm in `binsearch.py`.

Figure 6 shows a plot of the left-hand side of Equation (8), of which we seek the roots. Note

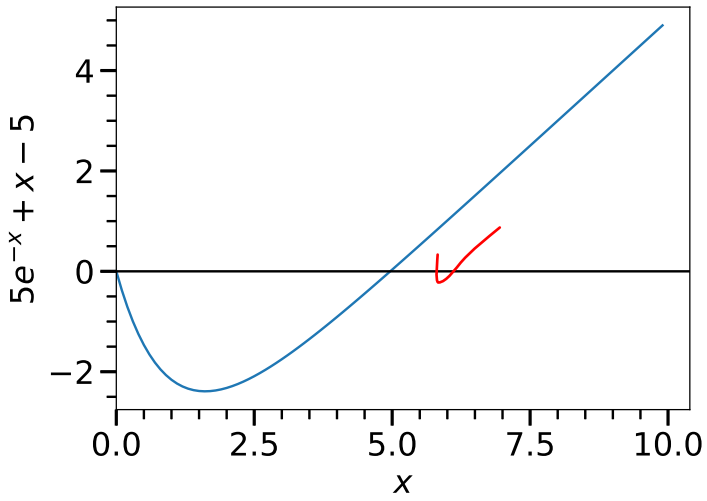

Figure 6: Graphical solution to Equation (8).

there is a trivial solution at $x = 0$ which we disregard on physical grounds as the definition (7) implies that x is strictly positive. The root we seek is near $x \approx 5$, so we can safely bracket the root with the initial domain endpoints $(x_1, x_2) = (3, 7)$. The binary search method returns $x = 4.96511$ to an accuracy of 10^{-6}, which implies a Wien displacement constant of $b = 2.897\,77 \times 10^{-3}\,\text{m·K}$ according to (9). See `newman_6-13-b.tsv` for output.[3]

c. Temperature of the sun

From the Wien displacement law (9) the temperature may be calculated as $T = b/\lambda$ where λ is the wavelength at which the emission spectrum peaks. Using the Wien displacement constant b obtained in part b. and $\lambda = 502\,\text{nm}$ for the emission peak of the Sun, we obtain[4] $T_{\text{sun}} \approx 5770\,\text{K}$.

[3] Note that although the output is double precision, the accuracy threshold of 10^{-6} only produces 5 significant digits after the decimal point in x, or 6 significant figures in total.

[4] Here the accuracy is limited to 3 significant figures due to the precision of the given value for λ.

$x_{\min}$	$y_{\min}$	$f(x_{\min}, y_{\min})$
1.99999	1.99999	$8.654\,58 \times 10^{-12}$

Table 2: Result of the gradient descent method applied to the test function (10). The value of the function at the minimum is effectively 0. Reproduced from `gradesc.tsv` and truncated to six significant figures.

3. Fitting the Schechter function via gradient descent

a. Gradient descent

My gradient descent implementation is in `gradesc.py`, and my tests are in `test_gradesc.py`. I tested my gradient descent implementation using the given function

$$f(x, y) = (x - 2)^2 + (y - 2)^2 \tag{10}$$

which can easily be differentiated partially with respect to x and y to find the location of the extremum, $(x_{\min}, y_{\min}) = (2, 2)$. The value of the function at this point is 0, and the positive curvature (given by the second partial derivatives) in both dimensions indicates that this is a minimum. Table 2 shows the result of the gradient descent algorithm for the starting point $(x_0, y_0) = (-3, -4.5)$, learning rate $\gamma = 0.1$, and error tolerance 10^{-6}. For this example, at least, the implementation finds the location of the minimum and seems to be working correctly.

As another verification, Figure 7 shows the path taken from the initial point to the minimum. The initial step is taken in an arbitrary direction a distance 1% larger than the user-specified

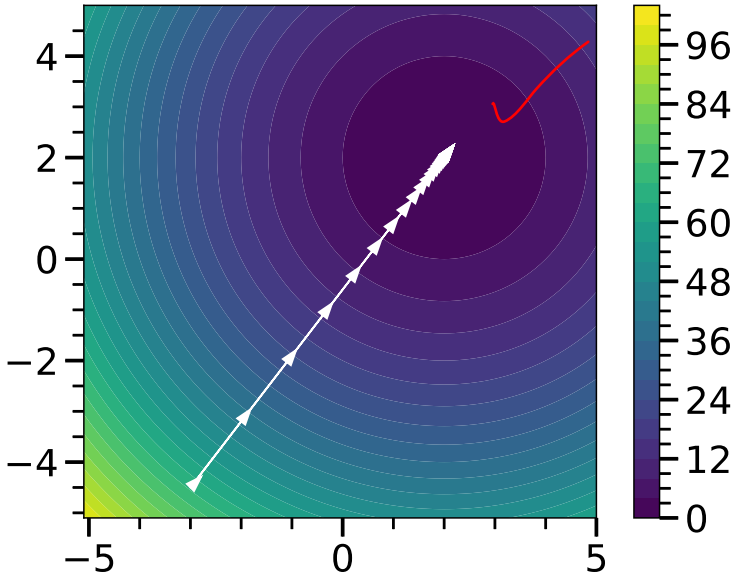

Figure 7: Path taken from the initial point to the minimum of the test function (10).

error tolerance in each dimension, to obtain a second point with which to compute the numerical derivative. After that, the algorithm takes much larger steps along the gradient (perpendicular

to the contours), with the step size decreasing along with the slope as it makes a beeline for the minimum of this simple paraboloid of revolution. Viewed another way, Figure 8 shows the deviations of the current values of x and y at each iteration from their true values at the minimum, $x_{\min} = y_{\min} = 2$. Again, we observe an initial small step which does little to move us towards

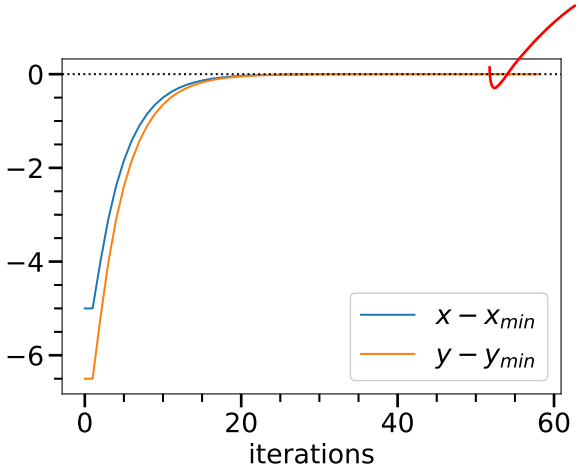

Figure 8: Absolute error in the estimate of the location of the minimum for each dimension in the domain of the test function (10).

the minimum but serves to obtain a second point with which to compute the numerical derivative. Immediately thereafter, the absolute error decreases rapidly in magnitude towards zero. This expected behavior gives us further confidence that the implementation is working correctly.

b. Fitting the Schechter function

My code for this problem is in `schechter.py`.

The Schechter function is given as

$$n(M^{\text{gal}}) = \phi^* \left(\frac{M^{\text{gal}}}{M_*}\right)^{\alpha+1} \exp\left(-\frac{M^{\text{gal}}}{M_*}\right) \ln 10, \tag{11}$$

and we determine the free parameters $\vec{p} = (\phi^*, M_*, \alpha)$ by minimizing the χ^2 error

$$\chi^2 = \sum_i \left(\frac{n(M_i^{\text{gal}}, \vec{p}) - n_i}{\sigma_{n_i}}\right)^2$$

via gradient descent in the domain of $\vec{p}$, where n_i is the empirically determined value of the galaxy stellar mass function at M_i^{gal}, and σ_{n_i} is the associated error of the measurement.

The primary challenge I faced when fitting the given data to this model was due to the vastly differing orders of magnitude of the fitting parameters. To cope with this I modified my gradient descent algorithm to allow for the specification of different error tolerances and γ factors along each dimension. The most difficult part was tuning the tolerances and γ factors to get all dimensions to converge at roughly the same rate. A common problem I faced was that one dimension would converge quickly to a local minimum, and subsequent iterations of the gradient descent would stop moving along that dimension. This would cause divide-by-zero errors in the calculation of the

gradient along that dimension, as the denominator in

$$x_{i+1} = x_i - \gamma \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

goes to zero. I thus had to ensure that all three dimensions converged to within the error threshold specified before any one dimension converged to double precision. I also found several local minima in all three-dimensions depending on the initial starting point I chose, and often χ^2 would diverge to infinity. Though I was able to get a good fit for a few different starting points, my gradient descent implementation is not particularly robust and requires significant manual tuning of the user-defined parameters with feedback from trial and error.

Figures 9, 10, 11, and 12, show fits of the given data to the Schechter function model (11) for various starting points and γ factors. The input and output parameters to the gradient descent

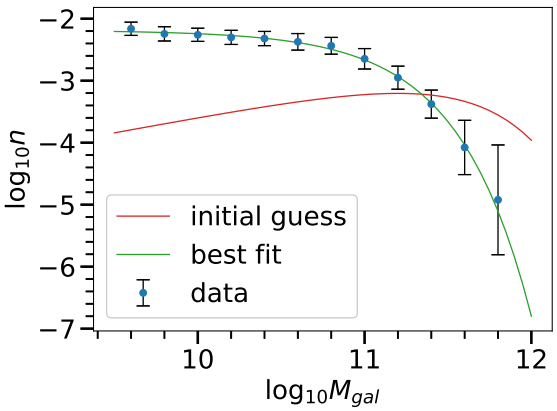

Figure 9: First best-fit obtained.

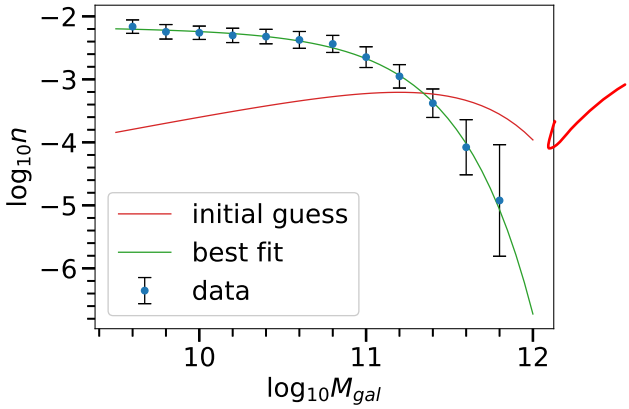

Figure 10: Same starting point but different γ factors.

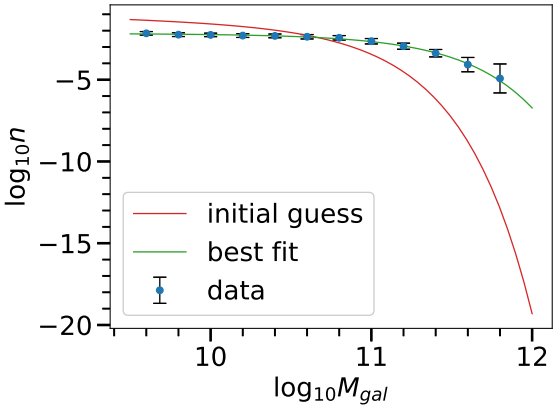

Figure 11: Different starting point.

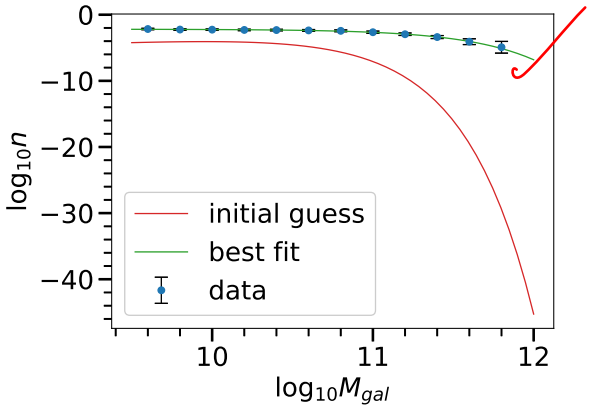

Figure 12: Another different starting point.

algorithm for Figure 9 are given in Table 3, and the rest can be found in `schechter_fit-2.tsv`, `schechter_fit-3.tsv`, and `schechter_fit-4.tsv`, respectively. In each case we obtain a minimum $\chi^2 < 3$ (corresponding to the 3 degrees of freedom in $\vec{p}$), indicating a good fit.

Finally, Figures 13–16 show χ^2 at each iteration of the gradient descent algorithm, corresponding

	initial	**final**	γ
$\log_{10}\phi^*$	-3.2	-2.56882	10^{-10}
$\log_{10} M_*$	11.5	10.9766	10^{19}
α	-0.5	-1.00995	10^{-4}
χ^2	1224.50	2.90814	

Should use same γ for each, but this seemed to work well!

Table 3: Initial and final values of χ^2 and the fitting parameters corresponding to Figure 9, along with the γ factors used along each dimension in the gradient descent algorithm. The γ column indicates the learning parameters used for the fitting parameters themselves, *not* their logarithms. Reproduced from `schechter_fit.tsv` and truncated to six significant figures.

respectively to Figures 9–12. In particular, Figures 13 and 14 show the sensitivity of my gradient descent implementation to the γ factors chosen along each dimension. In these two figures, the same initial starting point was provided, but the γ factors differ along the M_* and α dimensions.

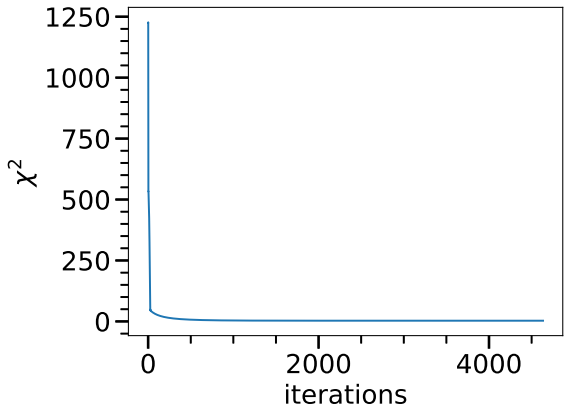

Figure 13: First best-fit obtained.

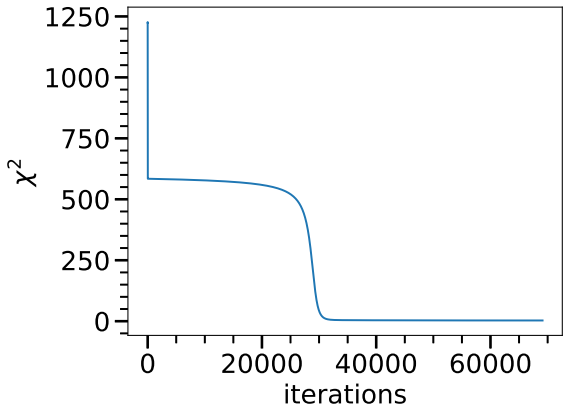

Figure 14: Same starting point but different γ factors.

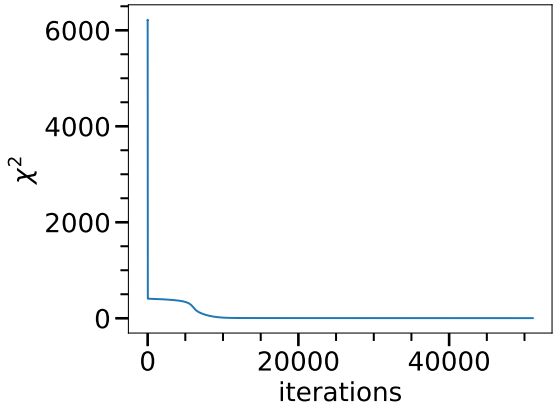

Figure 15: Different starting point.

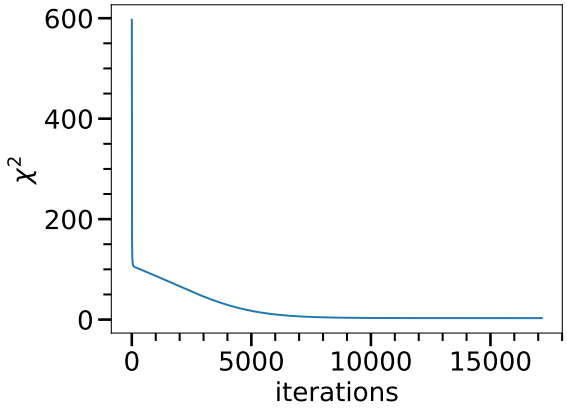

Figure 16: Another different starting point.

References

[1] Mark Newman. *Computational physics*. CreateSpace, revised and expanded edition, 2012. ISBN 978-1480145511.