Computational Physics Homework 2

Hao Li[†]

December 7, 2019

Problem 1

a)

The true solution is related to the current estimate x and the next estimate x' of the equation is

$$\begin{aligned} x^* &= (1+\omega)f(x^*) - \omega x^* \\ x^* &= x + \epsilon \\ x^* &= x' + \epsilon' \end{aligned} \tag{1}$$

Here ϵ and ϵ' are the error on the two estimates. Then Taylor expand the iteration function of the overrelaxation function about the true solution x^*

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

[†]hl3270@nyu.edu UID:N12137527

Now plug Eq. 1 into Eq. 2, we will get

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

that is

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

Back to Eq. 1

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

So the equivalent of Eq. (6.83) for the overrelaxation method is

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

b)

To calculate $x = 1 - \mathrm{e}^{-cx}$, $c = 2$ with relaxation method, the iteration function is simply

$$x' = 1 - \mathrm{e}^{-cx} \tag{7}$$

The code for relaxation method to solve (nonlinear) equations is available in "relaxation.h", attached in the folder. By inputing the starting point x_0 and the target accuracy of solution, the code returns the final answer x and the process of iteration. Here we set the starting point to be $x_0 = 1$, and the desired accuracy to be 10^{-6}. It takes 13 steps to converge to an accuracy better than the target, and the solution x by the program and the true solution x^* by Mathematica to Eq. 7 are

$$\begin{aligned} x &= 0.796812(7) \\ x^* &= 0.79681213 \end{aligned} \tag{8}$$

So the error $\epsilon \approx 7 \times 10^{-7} < 10^{-6}$. The solution given by the program reaches the desired accuracy. The process of iteration is shown in Figure. 1.

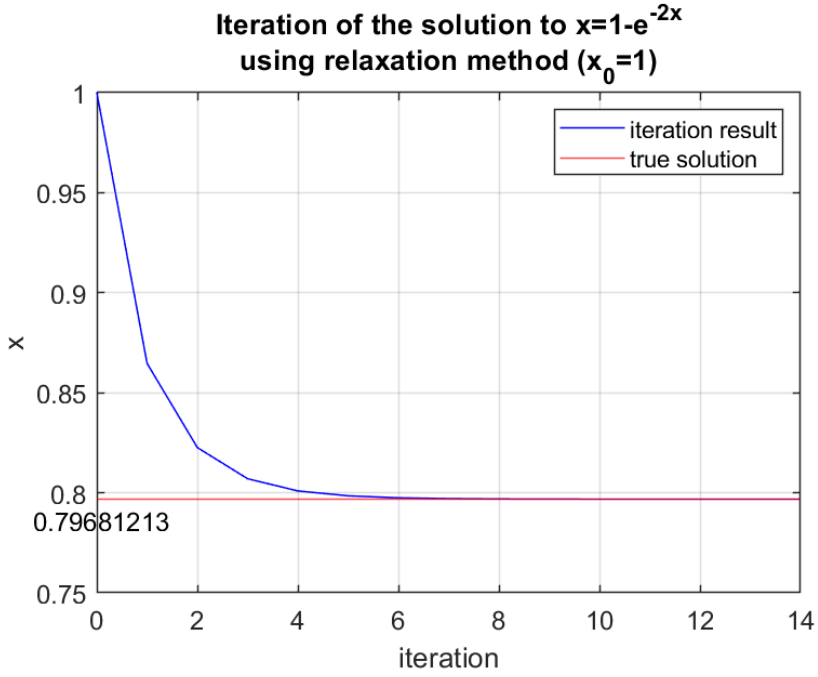

Fig. 1: *The iteration process of the solution to* $x = 1 - \mathrm{e}^{-2x}$ *using relaxation method. Starting from* $x_0 = 1$*, and target accuracy* $\epsilon = 10^{-6}$

c)

According to Eq. 2, the overrelaxation iteration equation is

$$x' = (1 + \omega)(1 - \mathrm{e}^{-2x}) - \omega x \tag{9}$$

The code for overrelaxation method to solve (nonlinear) equations is available in "relaxation.h", attached in the folder. By inputing the starting point x_0 and the target accuracy of solution, the code returns the final answer x for different parameter

ω, and the number of iterations for each ω with other conditions the same. Here we still set the starting point to be $x_0 = 1$, and the desired accuracy to be 10^{-6}. From Figure. 2 we can see that there is an range of optimal value of ω about $[0.5, 0.7]$ that can converge twice or three times faster than the standard relaxation method.

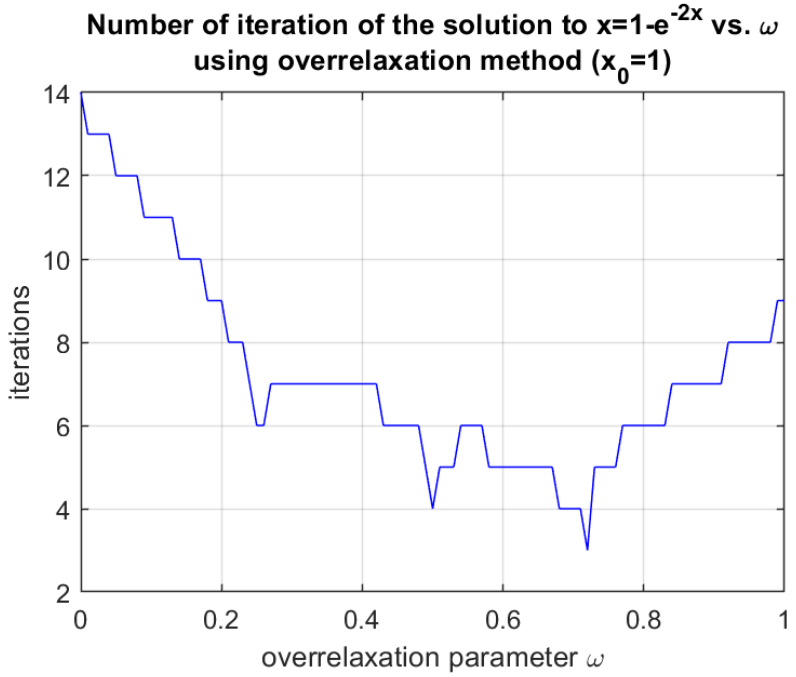

Fig. 2: *The number of iterations of the solution to* $x = 1 - \mathrm{e}^{-2x}$ *using overrelaxation method, with respect to the overrelaxation parameter* ω*. Starting from* $x_0 = 1$*, and target accuracy* $\epsilon = 10^{-6}$*, parameter* $\omega \in [0, 1]$*.*

The stop condition of the iteration is the error estimated by Eq. 6, replacing the derivative with the finite estimate

$$\epsilon' \approx \frac{x - x'}{1 - \dfrac{1}{(1+\omega)\dfrac{(f(x) - f(x'))}{x - x'} - \omega}} < \epsilon_{\text{tar}} \tag{10}$$

The final results for all ω's are plotted in Figure. 3 with respect to ω. From the plot we can see that for most of the values of $\omega \in [0, 1]$, the error is just as estimated using Eq. 10, except for a narrow range arround [0.2,0.3], there is a spike whose error $\epsilon \sim 10^{-5}$. So the break condition of the code may need further improvement.

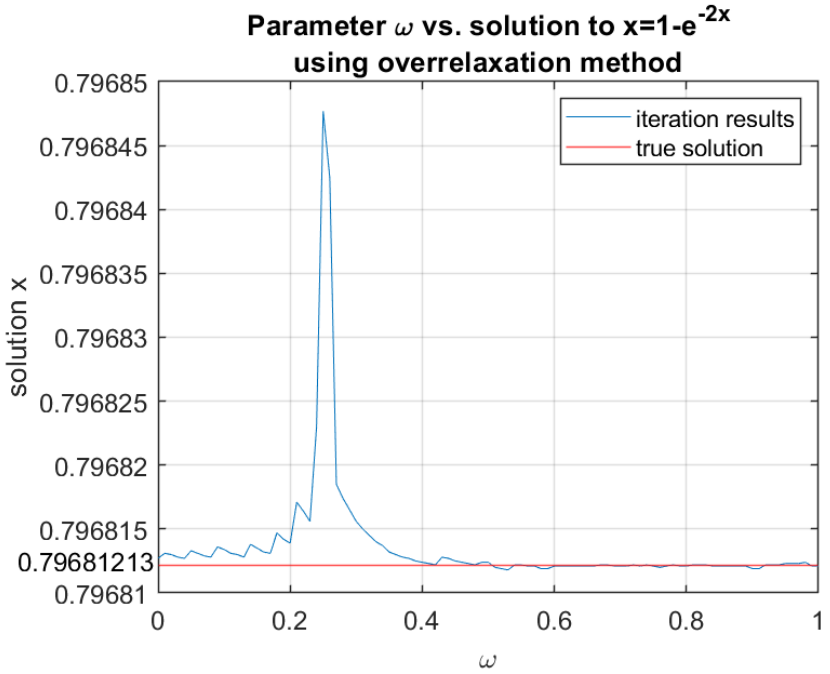

Fig. 3: *The iteration solution to $x = 1 - \mathrm{e}^{-2x}$ using overrelaxation method, with respect to the overrelaxation parameter ω. $x_0 = 1$, target accuracy $\epsilon = 10^{-6}$, parameter $\omega \in [0, 1]$. For most of the values of $\omega \in [0, 1]$, the error is just as estimated using Eq. 10, except for a narrow range arround [0.2,0.3], there is a spike whose error $\epsilon \sim 10^{-5}$.*

d)

From Eq. 6, we can see that if we expect an $\omega < 0$ could help speed up

$$|(1 + \omega)f'(x) - \omega| < |f'(x)| \tag{11}$$

The solutions are $f'(x^*) > 1$ or $-1 < f'(x^*) < 0$. For $f'(x^*) > 1$, we can modify ω so that the solution could converge. For $-1 < f'(x^*) < 0$, the underrelaxation method $\omega < 0$ could help converge faster compared with standard relaxation method.

Problem 2

a)

The intensity of radiation from a black body

$$I(\lambda) = \frac{2\pi hc^2\lambda^{-5}}{\mathrm{e}^{hc/\lambda k_B T} - 1} \tag{12}$$

The first derivative with respect to wavelenth λ

$$\frac{\mathrm{d}I(\lambda)}{\mathrm{d}\lambda} = \frac{2\pi hc^2\lambda^{-6}\mathrm{e}^{hc/\lambda k_B T}}{(\mathrm{e}^{hc/\lambda k_B T} - 1)^2}[5(\mathrm{e}^{-hc/\lambda k_B T} - 1) + \frac{hc}{\lambda k_B T}] \tag{13}$$

For the wavelenth λ at which the emmited radiation is strongest, there should be $\frac{\mathrm{d}I(\lambda)}{\mathrm{d}\lambda} = 0$, i.e.

$$5(\mathrm{e}^{-hc/\lambda k_B T} - 1) + \frac{hc}{\lambda k_B T} = 0 \tag{14}$$

Substitute $x = hc/\lambda k_B T$, then the equation is

$$5\mathrm{e}^{x} + x - 5 = 0 \tag{15}$$

Define the Wien displacement constant $b = hc/k_B x$, where x is the solution to Eq. 15, then from the substitution we can derive the Wien displacement law

$$\lambda = \frac{hc}{k_B x T} = \frac{b}{T} \tag{16}$$

b)

The program to solve the Eq. 15 is available in "root_finding.h", attched in the folder. Starting with the interval $[x_1, x_2] = [4, 6]$, with an accuracy of $\epsilon = 10^{-6}$, as shown in Figure. 4, the root of the equation is

$$x = 4.965114 \tag{17}$$

which agrees with the true solution

$$x^* = 4.9651142317 \tag{18}$$

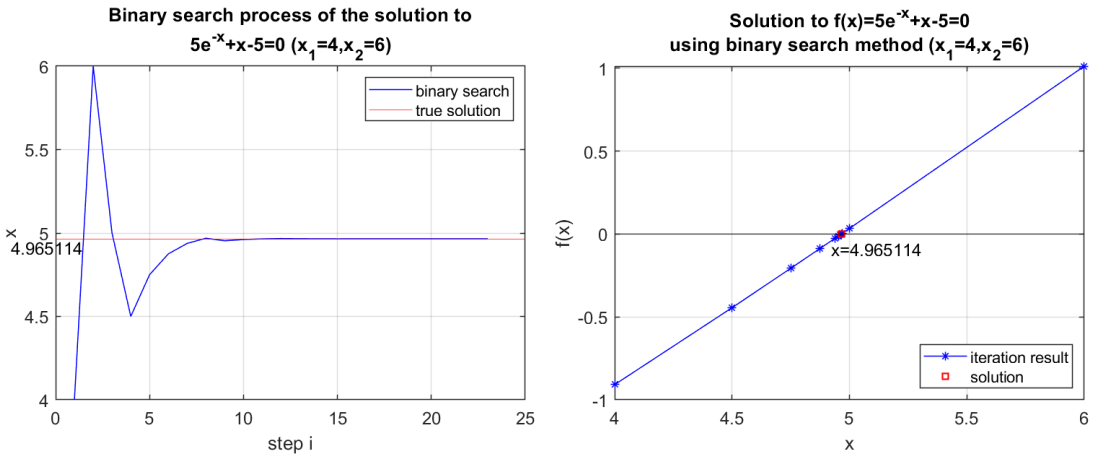

Fig. 4: *The binary search process and result of solution to the equation* $5\mathrm{e}^{-x} + x - 5 = 0$. *Starting with the interval* $[x_1, x_2] = [4, 6]$, *target accuracy* $\epsilon = 10^{-6}$. *The solution given by the program is* $x = 4.965114$, *which agrees with the true solution.*

c)

From Eq. 16 we have

$$T = \frac{b}{\lambda} = \frac{hc}{k_B x \lambda} \tag{19}$$

Plug in the result in b) and $\lambda = 5.2$nm, an estimate for the tempreture of the Sun is

$$T \approx 5.77 \times 10^3 \text{K} \tag{20}$$

Problem 3

a)

To find the minimum of a multi dimensional function $f(\mathbf{x})$ using the gradient descent method, we start from an initial position $\mathbf{x}_0$, $f(\mathbf{x})$ decreases fastest in the negative gradient $\nabla f(\mathbf{x})$. That is, for small enough step size γ

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \nabla f(\mathbf{x_n}) \tag{21}$$

gives $f(\mathbf{x}_{n+1}) < f(\mathbf{x}_n)$. For complecated or unknown expression function, we use numerical derivatives to calculate the gradient and plug in Eq. 21. The loop stops when the maximum difference in $\mathbf{x}$ (or some other difference) reaches the target accuracy.

For further usage in fitting the Schechter function, first test the code on a simple function

$$f(x, y) = (x - 2)^2 + (y - 2)^2 \tag{22}$$

The code that implement the gradient descent method to find the minimum of a multi dimensional function, using numerical derivatives, is available in "descent.h". By inputing the function $f(\mathbf{x})$ and its dimension, the initial position $\mathbf{x}_0$, step size γ, the target accuracy ϵ, and the maximum number of loops to avoid infinite calculation, the code saves the process of calculation and returns the final result.

For the simple function Eq. 22, two initial points $(x_0, y_0) = (0, 1), (3, 1)$ with two step sizes $\gamma = 10^{-2}, 10^{-3}$, and accuracy $\epsilon = 10^{-7}$, are tested respectively to see if the final solution is the same.

Figure. 5 shows that different initial points with different step sizes will give the same result within some error range. The four results of the four initial conditions are (1.999995,1.999997), (1.999950,1.999970), (2.000006,1.999997), (2.000059,1.999970), are clearly the correct results $(x, y) = (2, 2)$ in a relatively high accuracy.

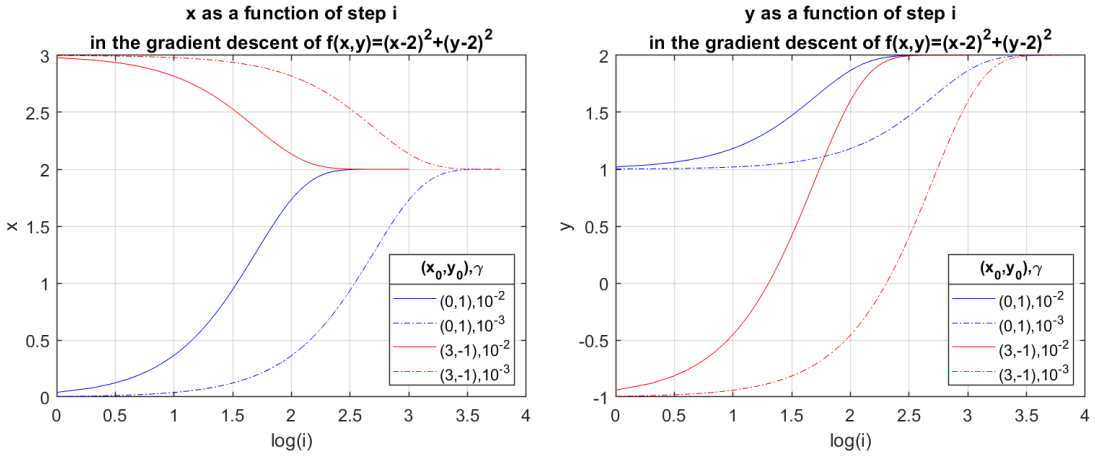

Fig. 5: *Finding the minimum of the function* $f(x, y) = (x - 2)^2 + (y - 2)^2$ *using gradient descent method. Initial points* $(x_0, y_0) = (0, 1), (3, 1)$*, step sizes* $\gamma = 10^{-2}, 10^{-3}$*, accuracy* $\epsilon = 10^{-7}$*. it confirms that the code can give the same result with different initial conditions, and is thus robust within at least a small vincinity of the true minimum.*

The evolution of the position and the function value in the four processes above are given in Figure. 6. From the test result, it is obvious that the code is robust within at least a small vincinity of the true minimum.

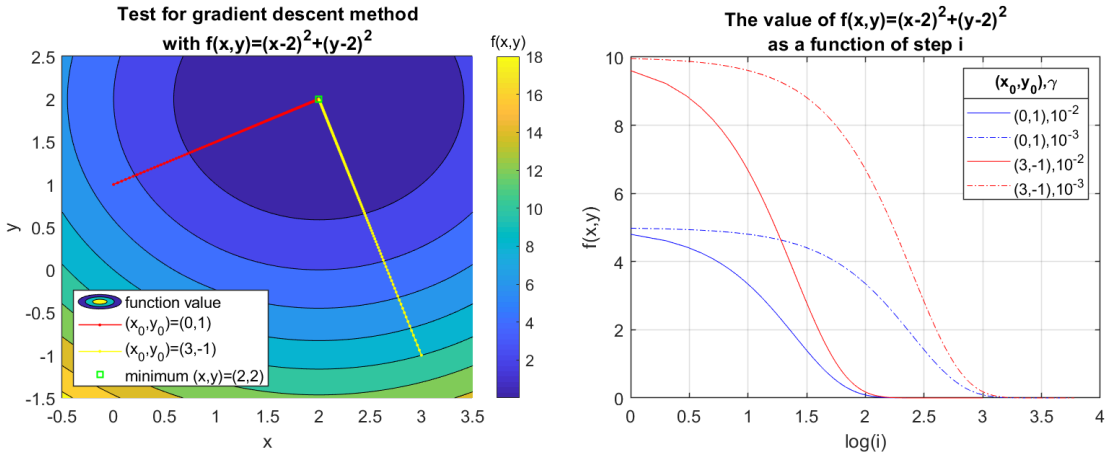

Fig. 6: *The evolution of the position* (x, y) *and the value of the function* $f(x, y) = (x-2)^2 + (y-2)^2$ *in the gradient descent process, with initial points* $(x_0, y_0) = (0, 1), (3, 1)$*, step sizes* $\gamma = 10^{-2}, 10^{-3}$*, and accuracy* $\epsilon = 10^{-7}$*. it is obvious that he four different initial conditions will lead to the same true minimum, thus confirms the robustness of the code.*

b)

Now that the code works, it is going to be applied to the problem of fitting the Schechter function

$$n(M_{\text{gal}}) = \phi^* \left(\frac{M_{\text{gal}}}{M^*}\right)^{\alpha+1} \exp\left(-\frac{M_{\text{gal}}}{M^*}\right) \ln(10) \tag{23}$$

From the data file "smf_cosmos.dat"we can see that M_{gal} is rather large while $n(M_{\text{gal}})$ is small. And it can be guessed that the exponent α cannot be large, so the variables used in fitting are $\log(\phi^*)$, $\log(M^*)$, and α since they are close to 1.

Guessing from the data, four initial points are chosen to verify the robustness of the code: $(\log(\phi^*), \log(M^*), \alpha) = (-5, 9.5, -1.5), (-3, 10.5, -1), (-3, 10, 0), (-4, 9, -0.5)$. Step size are the same $\gamma = 10^{-4}$, and the target accuracy $\epsilon = 2 \times 10^{-7}$.

The function to minimize is the χ^2 of the model

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{n_{i,\mathrm{fit}} - n_{i,\mathrm{data}}}{\sigma} \right)^2 \tag{24}$$

The evolution of the three parameters $(\log(\phi^*), \log(M^*), \alpha)$ are as given in Figure. 7. And the final result of the fitting is

$$\begin{aligned} \log(\phi^*) &= -2.5682(1) \\ \log(M^*) &= 10.9762(3) \\ \alpha &= -1.0093(9) \end{aligned} \tag{25}$$

The minimum χ^2 is

$$\chi^2 = 2.90806(5) \tag{26}$$

The evolution of χ^2 as a function of i is shown in Figure. 8.

However, it can be seen from Figure. 7 and 8 that for initial points far from the minimum, the value of the variables and the function might stay nearly unchanged for quite a few steps, may be because it is far, and may be because the derivative may be close to zero, i.e. the function is nearly flat. In this circumstance, it will take exponentially longer time to finish calculation and converge to the minimum. In practice, when starting with $\log(M_{\mathrm{gal}}) \approx 14$, or $\alpha \approx 1$, it may remain unconverged in over 8000 steps, with χ^2 or some parameters staying nearly the same , so I am not sure if it could work for arbitrary initial points in single precision.

Finally, Figure. 9 shows the comparison of the best-fit Schechter function to the data in log-log plot. The fit result agrees quite well with the data.

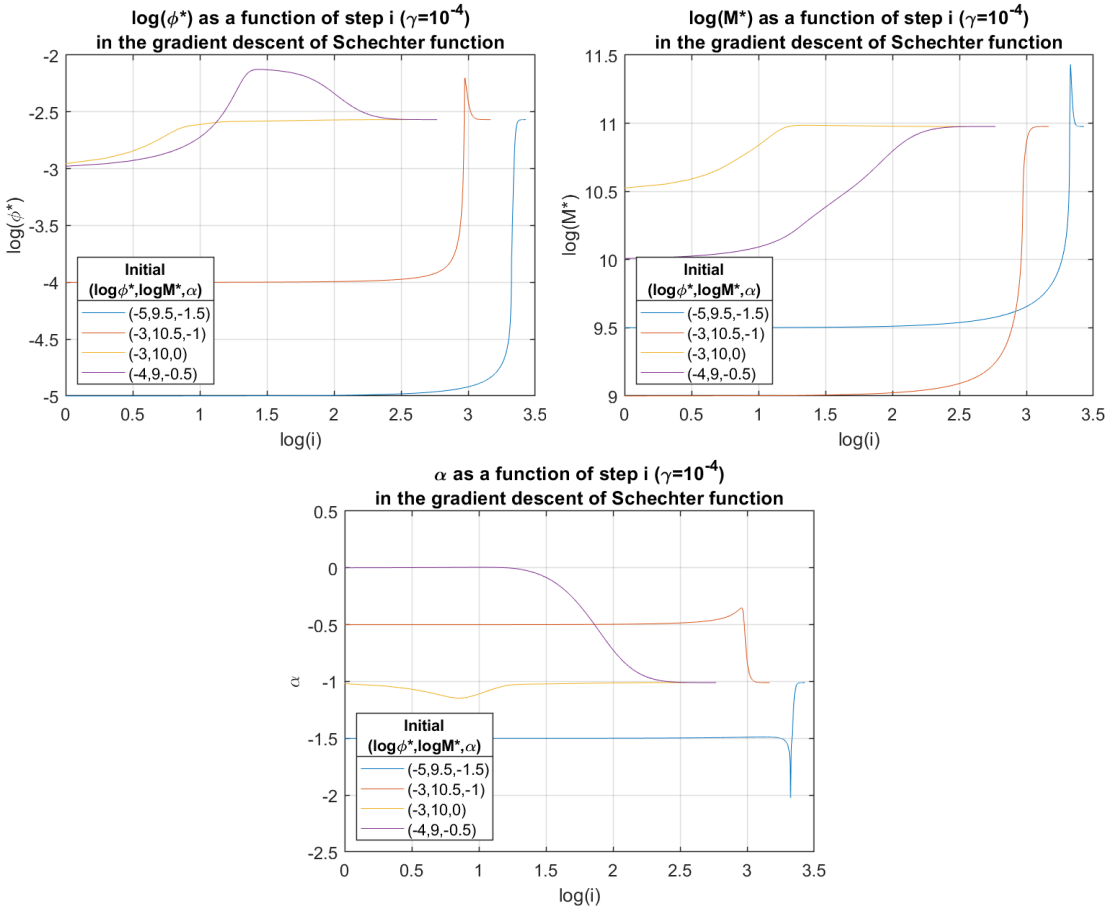

Fig. 7: *The evolution of the parameters* $(\log(\phi^*), \log(M^*), \alpha)$. *Initial points* $(\log(\phi^*), \log(M^*), \alpha) = (-5, 9.5, -1.5), (-3, 10.5, -1), (-3, 10, 0), (-4, 9, -0.5)$, *and accuracy* $\epsilon = 10^{-7}$. *The results are all* $(-2.5682, 10.9762, -1.0094)$ *with high accyracy of about* 10^{-5}, *thus demonstrating that the result is robust.*

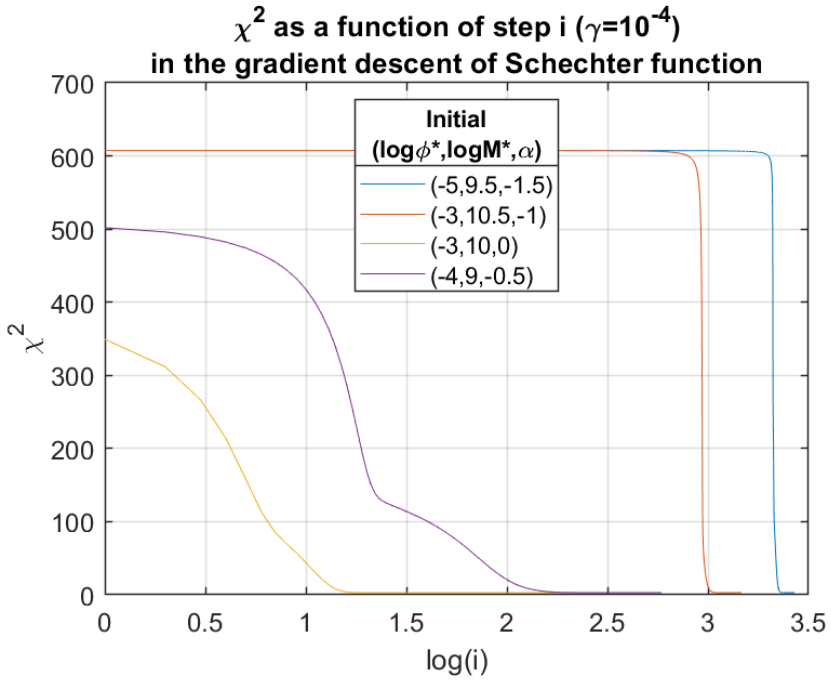

Fig. 8: χ^2 *of the Schechter function as a function of step* i*. Initial points* $(\log(\phi^*), \log(M^*), \alpha) = (-5, 9.5, -1.5), (-3, 10.5, -1), (-3, 10, 0), (-4, 9, -0.5)$*, and accuracy* $\epsilon = 10^{-7}$*. The results are all* $(-2.5682, 10.9762, -1.0094)$ *with high accyracy of about* 10^{-5}*, thus demonstrating that the result is robust.*

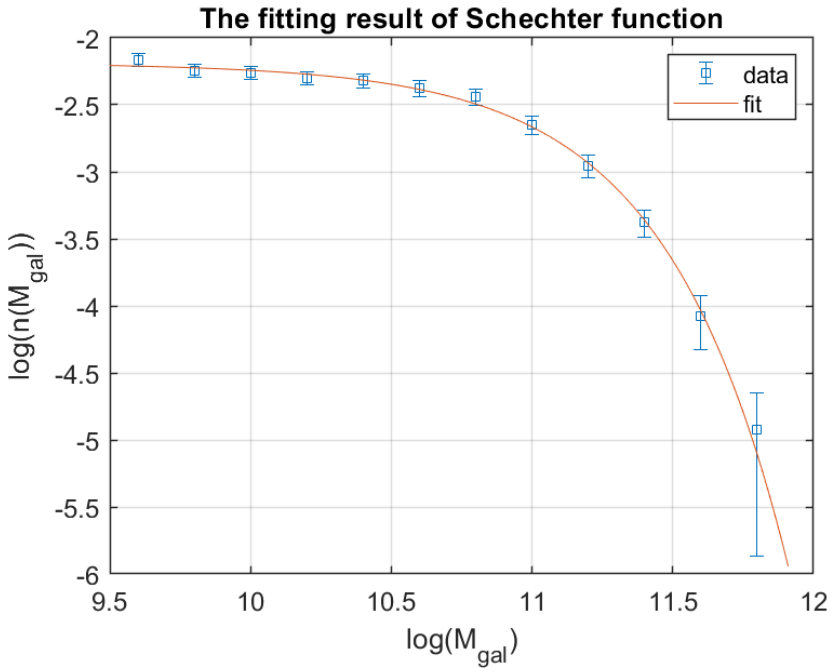

Fig. 9: *The comparison of the best-fit Schechter function to the data in log-log plot.*