Homework 2

Jatin Abacousnac

October 11, 2019

Problem 1

Problem 6.11 (Newman)

The overrelaxation method is one that is employed to solve a system of linear equations. It is based on the simpler relaxation method:

- recast the equation in the form $x = f(x)$
- guess a value of x
- plug in that value of x on the left hand side and evaluate right hand side
- repeat until convergence.

This time, an iteration is performed where each successive term x_i is evaluated as

$$x_{i+1} = (1 + \omega)f(x_i) - \omega x_i, \tag{1}$$

where ω is a quantity that controls how much we are overshooting our calculated value from our initial guess, hoping it gets us closer to the final solution.

Consider the equation

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

Upon inspection we note that 0 is a root. The curve $y = 1 - e^{-2x}$ is also intersected by the line $y = x$ at $x = 0.797$. Upon implementation of the relaxation method, it is found that

for an initial value of 1, it takes 13 iterations for convergence to occur within a margin of 6 significant figures.
For the same function, with the same initial value of 1, the overrelaxation method is implemented. Some interesting observations can be made from above. First, $\omega = 0$ corresponds

ω	n_{it}
0	13
1	9
2	46
0.5	5
0.1	12
-1	3

to our original relaxation algorithm, and we noticed that for small positive values of ω, the overrelaxation method works better, with the best performance achieved when ω was set to 0.5 (note that I attempted a non-exhaustive list of ω values). When ω = -1, underrelaxation happens, and in this case the algorithm fails, as the values obtained were [0, 1, 1], failing to converge to the true value.
Let us explore the underrelaxation method, i.e. when $\omega < 0$. This time, we should be undershooting, and my guess would be that this algorithm might work best when dy/dx changes rapidly such that overshooting may send us spiraling away from our true value, but undershooting is a more cautious approach in these instances. Piecewise functions, or functions with 'sharp' changes may benefit from underrelaxation.

Problem 2

Problem 6.13 (Newman)

Blackbody radiation is expressed as:

$$I(\lambda) = \frac{2\pi h^2 \lambda^{-5}}{e^{hc/\lambda k_B T} - 1}, \tag{3}$$

where I is the intensity of the radiation, λ is the wavelength of radiation, and h, c, k_B are the usual fundamental constants: Planck's constant, speed of light, and the Boltzmann constant, respectively. To find the maximum value of I for a wavelength λ_{max}, differentiate the function I wrt λ: Using the quotient rule,

$$\frac{dI}{dt} = \frac{(e^{hc/\lambda k_B T} - 1)(-10\pi h^2 \lambda^{-6}) - (2\pi h^2 \lambda^{-5})(-hc/k_B T \lambda^2)(e^{hc/\lambda k_B T})}{(e^{hc/\lambda k_B T - 1})^2}. \tag{4}$$

Set it equal to 0 and group the terms,

$$\frac{dI}{dt} = e^{hc/\lambda k_B T}\Big[- 10\pi h^2 \lambda^{-6} - (2\pi h^2 \lambda^{-5})(-hc/k_B T \lambda^2)\Big] + 10\pi h^2 \lambda^{-6} = 0, \tag{5}$$

and divide the LHS by $e^{hc/\lambda k_B T}$

$$(1 - e^{-hc/\lambda k_B T})(-5\pi h \lambda^{-6}) - (-hc/k_B T \lambda^2)(\pi h^2 \lambda^{-5}) = 0. \tag{6}$$

Rearrange, and obtain

$$5e^{-hc/\lambda k_B T} + hc/k_B T \lambda - 5 = 0. \tag{7}$$

If we let $x = (hc/k_B T \lambda)$,

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

and I find the roots to this equation using what is known as the binary search method.

Binary search method

The binary search method begins by picking an initial guess x_0, and two values x_1 and x_2 that are far apart, where x_0 is the midpoint of these two. Then, we iteratively check the function at the left edge, i.e. calculate $f(x_1)$, and if the sign does not change between $f(x_1)$ and $f(x)$, we know a root lies in the interval $[x, x_2]$, and we make x_1 become our new x,

and recalculate x as the midpoint of x_2 and what is now our new x_1. And so on, until the interval is small enough.
This method was used for solving equation 8, and the root was found to be 4.965114 for a target of 1×10^{-6}.
Wien's displacement constant b is defined as hc/k_Bx. Thus, the value obtained for $b = 0.002897\,\mathrm{m}\cdot\,\mathrm{K}$, which is comparable to the known value.
Having obtained this value, the temperature at the surface of the sun (assuming it is a perfect blackbody) can be determined as $T_\odot = b/\lambda$, where $\lambda = 502\,\mathrm{nm}$. The value obtained is $T_\odot = 5772\,\mathrm{K}$. This value is comparable to a commonly-cited value of $5800\,\mathrm{K}$.

Problem 3

In this problem, we use the multivariable gradient descent method, a technique that is used to find the minima of a function. Say a function f depends on two parameters a and b. The algorithm goes as follows for the $i+1$-th value of our parameters a and b:

$$a_{i+1} = a_i - \gamma * \frac{f(a_i, b_{i-1}) - f(a_{i-1}, b_{i-1})}{a_i - a_{i-1}}, \tag{9}$$

and

$$b_{i+1} = a_i - \gamma * \frac{f(a_{i-1}, b_i) - f(a_{i-1}, b_{i-1})}{b_i - b_{i-1}}. \tag{10}$$

It iteratively finds the values of a and b for which our function $f(a, b)$ is minimized, by updating the parameters such that the product of the gradient (wrt that parameter) and γ is subtracted from the current value. γ is known as the learning rate, upon which the algorithm is usually very sensitive.

Let us consider the function $f(x, y) = (x-2)^2 + (y-2)^2$. Its minimum, for which $f = 0$ occurs at $x = y = 2$. An implementation of the gradient descent method should lead to convergence of x and y to being equal to 2. This algorithm was implemented using MATLAB (code attached in Github repository). In my implementation, the values of x and y were updated until their subsequent value was different from the previous by a tolerance of 0.00001. To illustrate that the algorithm works, the following two figures are shown. In the first, the parameters x and y are plotted against the number of iterations, and it is seen that they converge to the being equal to 2.
In the second figure, the function itself is shown to minimize to 0.

This method can be implemented to minimize more complicated functions. The Schechter function is defined as:

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

where n is the number of galaxies with a certain mass M_{gal}, α is the slope of the function at low masses, M^* is the "characteristic mass scale", and ϕ^* is some amplitude to the function.

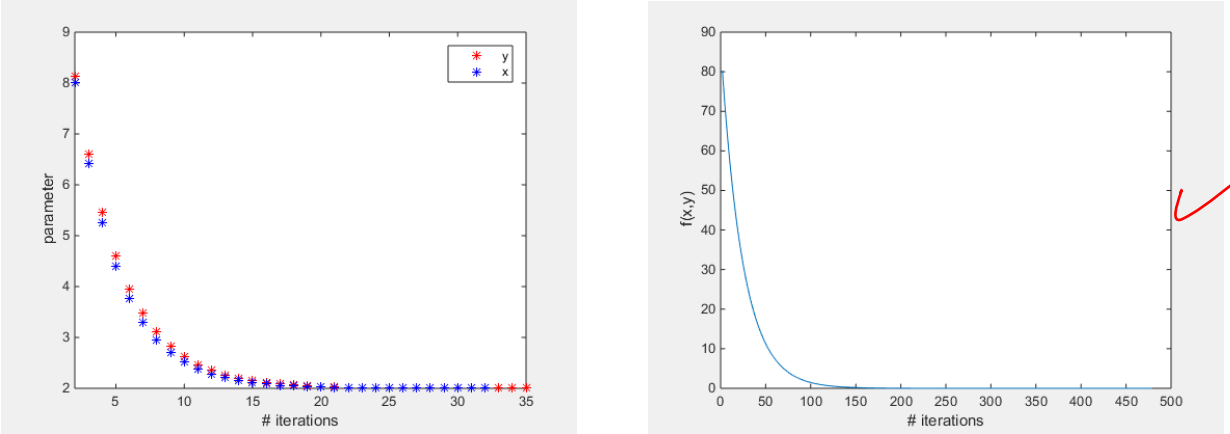

Figure 1: On the left, the plot shows values of x and y as we are attempting to minimize the function $f(x, y) = (x - 2)^2 + (y - 2)^2$ (shown on the right as a function of number of iterations). In the plot on the left, I show values of x- and y- that are at different iterations, so the convergence of each could be shown more clearly. Both x and y converge to 2, as expected.

In this problem, I obtain observed data with values for $n(M_{gal})$, M_{gal} and the error in the $n(M_{gal})$ values. Then, this fit is compared to the theoretical model in equation 11, and we obtain the chi-squared value which gives a metric for how good the fit is. Recall the definition of the chi-squared function for data obtained from observations (n_{obs}) vs. data obtained from a model (n_{mod}):

$$\chi^2 = \sum_i^N \frac{(n_{obs,i} - n_{mod,i})^2}{n_{mod,i}}. \quad (12)$$

this not what you and I supposed to use.

Here, the gradient descent method is implemented to minimize the χ^2 function.

I implemented it by creating a file (titled "final_attempt.m"[1]). This function takes α, M^* and ϕ^* as inputs, and for every M_gal, it calculates the output from the Schechter function and obtains a value from an interpolation of the data provided. It calculates a chi-squared value at that value of M_gal, and the final output is the actual χ^2 value over a range of M_{gal}. Here, that range is logarithmically spaced between $[10^{9.5}, 10^{11.5}]$.

[1]As you may be able to tell, it was a last-ditch attempt at solving the problem after repeated tries.

In another function (titled "finalfinalattempt.m"[2]), for starting values of α, M^* and ϕ^* and some dx by which they increment in the first iteration, I implemented the gradient descent method by minimizing the chi-squared value obtained from final_attempt.m. The values for α, M^* and ϕ^* were expected to converge.

Attempt to solve issues

I attempted to debug my code through several ways. The initial values I tried for the parameters α, M^* and ϕ^* were close to the solution, which would make it easier to converge. However, that did not work. When the code is run, within a few iterations, M^* shoots off to NaN, which may mean it blows up to ∞. Since M^* fails to converge, and the other parameters depend on an evaluation of the derivative at M^*, they also fail to converge, and the entire algorithm fails. All my code is attached, for further perusal. Since I do not have plots of the chi-square function after multiple iterations, a table is created with the values obtained for each parameter and chi-square before the parameters blow up. I also attempted different values of the initial step size (different signs and magnitudes for $\Delta(M^*)$), and also changed the parameter γ, but to no avail.

α	ϕ^*	M*	χ^2
-0.9018	-0.1598	8.9125e+10	197.5968
-0.8998	1.0389	8.9125e+10	1.2211e+03
-0.5996	0.1819	NaN	NaN

Did you try stepping in $\log M^*$ as discussed in class?

−2

[2]Successor to finalattempt.m