9/10

Computational Physics HW2

Jinjie Zhang

October 11, 2019

please don't ever do this again. put equations as equations (enumerated).

Newman 6.11

a) Let ϵ be the error on our current estimation of the solution to the equation, that is the true solution x^* of equation $x = f(x)$ is related to the current estimate x by $x^* = x + \epsilon$. Similarly, let ϵ' be the error on the next estimate, so that $x^* = x' + \epsilon'$. Since $x' = (1+\omega)f(x) - \omega x$ and also, $x' = (1+\omega)f(x) - \omega x = (1+\omega)[f(x^*) + (x - x^*)f'(x^*) + ...] - \omega x$, we have that $\epsilon' = x^* - x' = f(x^*) - f(x^*) - \omega f(x^*) - (1+\omega)(x - x^*)f'(x^*) + O((x-x^*)^2) + \omega x = -\omega f(x^*) + \omega x - (1+\omega)(x - x^*)f'(x^*) = -\omega(x^* - x) + (1+\omega)(x^* - x)f'(x^*) = -\omega\epsilon + (1+\omega)\epsilon f'(x^*)$

We have the equivalent of (6.81) as $\boxed{\epsilon' = \epsilon[f'(x^*) + \omega f'(x^*) - \omega]}$. Then $x^* = x + \epsilon = x + \frac{\epsilon'}{f'(x^*)+\omega f'(x^*)-\omega}$, equating with $x^* = x' + \epsilon'$, we have $x - x' = \epsilon'[1 - \frac{1}{(1+\omega)f'(x^*)-\omega}]$ so $\epsilon' = \frac{x-x'}{1-\frac{1}{(1+\omega)f'(x^*)-\omega}}$, which could be simplified to $\boxed{\epsilon' = \dfrac{x-x'}{1-\frac{1}{(1+\omega)f'(x^*)-\omega}} \approx \dfrac{x-x'}{1-\frac{1}{(1+\omega)f'(x)-\omega}}}$.

b) We set err$=1e-6$ in the function below, the second tuple returned is the number of iterations.

```
def relax(f,x0,err):
x=np.float32(x0)
xnew=np.float32(f(x0))
nit=0
while np.absolute(xnew-x)>err:
    x=xnew
    print(x)
    xnew=np.float32(f(x))
    print(xnew)
    nit=nit+1
return (xnew,nit)
```

? what are your values?

c)

```
def over_relax(f,omega,x0,err):
x=np.float32(x0)
xnew=np.float32(f(x0)*(1+omega)-omega*x0)
nit=0
while np.absolute(xnew-x)>err:
    x=xnew
    print(x)
    xnew=np.float32(f(x)*(1+omega)-omega*x)
    print(xnew)
    nit=nit+1
return (xnew,nit)
```

? Show! -1

I did witness faster convergence.

d) The answer is yes. For instance, when $f'(x)$ is negative in the vicinity of the solution. For instance, solving $x = 1 + e^{-2x}$, setting ω to be -0.5 solves faster than $\omega = 0.5$. Now the key for faster convergence is to let $|1 - \frac{1}{(1+\omega)f'(x)-\omega}|$ beat(be greater than) $|1 - \frac{1}{f'(x)}|$ of normal relaxation method. So when $f'(x) < 0$ in the vicinity of the root, scale it to $f'(x^*) = -1$, then $|1 - \frac{1}{-1-\omega-\omega}| = |1 + \frac{1}{1+2\omega}|$ which decreases is greater than $|1 + \frac{1}{f'(x^*)}|$ when $\omega \in (0,1)$ and less than $|1 + \frac{1}{f'(x^*)}|$ when $\omega > 0$. Again, there is no heal-all ω.

Newman 6.13

a) When $\frac{\partial I(\lambda)}{\partial \lambda}=0$, $\frac{2\pi c^2 h\left(che^{\frac{ch}{\lambda k_B t}}-5\lambda k_B t\left(e^{\frac{ch}{\lambda k_B t}}-1\right)\right)}{\lambda^7 k_B t\left(e^{\frac{ch}{\lambda k_B t}}-1\right)^2}=0$. Simplify it, we want to numerator to be 0, so $5e^{-\frac{hc}{\lambda k_B T}}+\frac{hc}{\lambda k_B T}-5=0$. Now let $x=\frac{hc}{\lambda k_B T}$, we have $5e^{-x}+x-5=0$. When we have the solution, the wavelength that maximizes $I(\lambda)$ is $\lambda=\frac{b}{T}$ where $b=\frac{hc}{k_B x}$

b)

```
def binary_search(f,x1,x2,err):
midx=np.float32((x1+x2)/2)
diff=x2-x1
while np.absolute(diff)>err:
    midx=np.float32((x1+x2)/2)
    if np.sign(f(midx))==np.sign(f(x1)):
        x1=midx
    elif np.sign(f(midx))==np.sign(f(x2)):
        x2=midx
    diff=x2-x1
return midx
```

Run the program, we get the answer $x=4.9651136$, so $b=0.00289978$.

c) As in the previous parts, $\lambda=\frac{b}{T}$ so $T\approx 5776.45$K.

Problem 3

We borrow the extrapolated difference method from HW1.

```
def grad_descent1d(f,x,err,gamma,max_iter):
i=0
while i<=max_iter:
    current_x=x
    x=current_x-gamma*extrap_diff(f,current_x,0.0001)
    step=x-current_x
    if np.abs(step)<=err:
        break
return x
```

The 1D version is trivial, bringing it into higher dimensions is essentially the same, but we need to game up the numerical differentiation. We simply borrow the extrapolated difference to 2D and 3D

```
def grad_2d(f,x,y,z,h):
x1=(f(x+2*h,y,z))
x2=(f(x+h,y,z))
x3=(f(x-h,y,z))
x4=(f(x-2*h,y,z))
y1=(f(x,y+2*h,z))
y2=(f(x,y+h,z))
y3=(f(x,y-h,z))
y4=(f(x,y-2*h,z))
return np.array([(-x1+8*x2-8*x3+x4)/(12*h),(-y1+8*y2-8*y3+y4)/(12*h)])

def grad_3d(f,x,y,z,h):
x1=(f(x+2*h,y,z))
x2=(f(x+h,y,z))
x3=(f(x-h,y,z))
x4=(f(x-2*h,y,z))
y1=(f(x,y+2*h,z))
```

```
    y2=(f(x,y+h,z))
    y3=(f(x,y-h,z))
    y4=(f(x,y-2*h,z))
    z1=(f(x,y,z+2*h))
    z2=(f(x,y,z+h))
    z3=(f(x,y,z-h))
    z4=(f(x,y,z-2*h))
    return np.array([(-x1+8*x2-8*x3+x4)/(12*h),(-y1+8*y2-8*y3+y4)/(12*h),(-z1+8*z2-8*z3+z4)/(12*h)])
```

Now with the gradient method(in $\mathbb{R}^3$) ready, we can finally develop the gradient descent method.

```
def grad_descent3d(f,x,y,z,err,gamma,max_iter):
    i=0
    pos=np.array([x,y,z])
    while i<=max_iter:
        current_pos=pos
        pos=current_pos-gamma*grad_3d(f,current_pos[0],current_pos[1],current_pos[2],0.000001)
        step=pos-current_pos
        if np.linalg.norm(step)<=err:
            break
    return pos
```

Now this function takes in a function $f : \mathbb{R}^3 \to \mathbb{R}$, a point (x, y, z) which is the starting point, error tolerance err, step size γ, and parameter max_iter which is the permitted maximum number of iteration. We can test this method on some easy to solve functions first.
Provided a convex function $f(x,y) = (x-2)^2+(y-2)^2$, with an apparent minimum 0 at $(2, 2)$. Starting at some randomly generated points in $[-10, 10] \times [-10, 10]$, now they all converge to $(2, 2)$ within steps. We attach one plot as picture proof that the method works on $f(x, y) = (x - 2)^2 + (y - 2)^2$ where we started from $(1, 3)$ with error tolerance 10^{-10}, maximum number of iterations 100, step size 0.1.

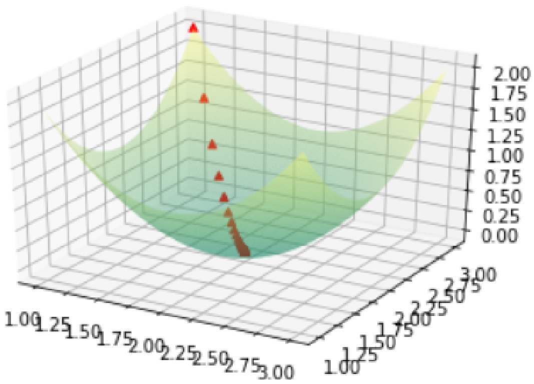


Figure 1: gradient descent converging to $(2, 2)$

Now we apply this method to the fitting of Schechter function

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

We are trying to minimize

$$\chi^2 = \sum_{i=1}^{12}(\frac{n_i - n_i^{\text{fit}}}{\text{err}_i})^2$$

over the dataset which contains 12 points.
After some try and eror, we know that ϕ^* is at about 0.001, M_* is at about 10^{11}, α is at about -1, now

since M_* is large and ϕ^* is small, we want to evaluate n is $\log 10$ base. Thus

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

Here we replaced M_* by 10^M and ϕ^* by 10^{ϕ}. Our purpose is to avoid calculations of large values($> 10^{11}$) when varying the variables.

```
grad_descent3d(lambda phi,m_star,alpha:np.sum((((10**phi)*(((10**m_gal)/(10**m_star))
((alpha)+1))*np.exp(-(10**m_gal)/(10**m_star))*np.log(10)-n)**2)/(err_n**2)),-2.44,
10.9,0.5,1e-8,0.00001,100000)
```

Set the starting position at $\phi^* = 10^{-2.44}$, $M_* = 10^{10.9}$, $\alpha = 0.5$, step size $\omega = 0.00001$, tolerance 10^{-8}, we get the result
$\phi^* = 10^{-2.56821249}$
$M_* = 10^{10.97622458}$
$\alpha = -1.0093873$.
With $\chi^2 = 2.90805586523984$. Now the degrees of freedom is $12 - 3 = 9$, we have p-value $\boxed{0.967831}$. We can plot the fitted result in log base along with original data and error bars.

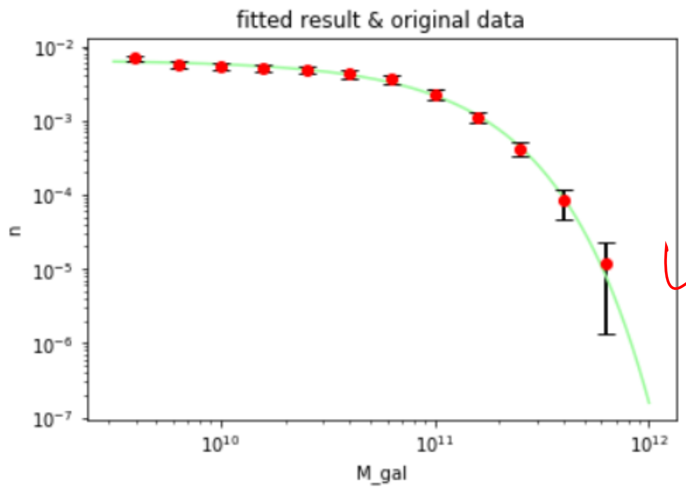
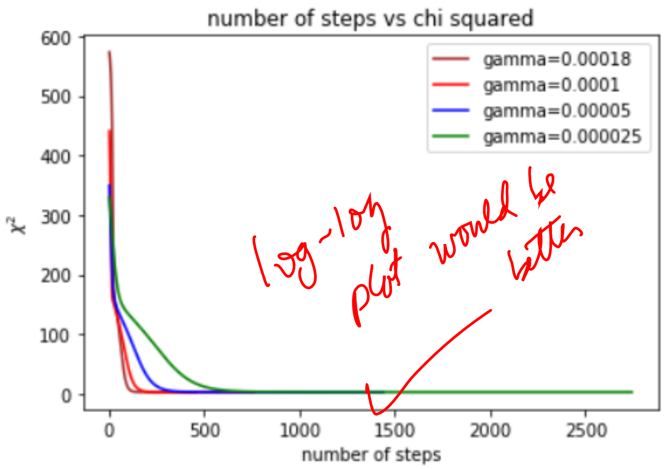


Figure 2: data vs results

We can play with step size to see the relation between number of steps to convergence and χ^2 values.

Larger step sizes were not chosen because they led to divergent results.



log-log plot would be better

Figure 3: convergence of the method