PHY407
Lab Assignment #2: Numerical Errors, Numerical Integration

Q2 by Hayley Agler, Q1,3 by Nicholas Pavanel

September 2020

Question 1

a)

Nothing to submit.

b)

Refer to Table 1.

c)

The plot clearly shows a minimum derivative error at a step size of 10^{-8}. This value agrees with the text's optimum step size for forward difference differentiation of $\sqrt{C} \approx 10^{-8}$. Noting Equation 5.91: $\epsilon = \frac{2C|(f(x)|}{h} + \frac{1}{2}h|f''(x)|$, and the fact the the text states that the first term in equation 5.91 represents the rounding error while the second term represents the truncation error, we can determine when each dominates. When h is very small, the first term in equation 5.91 will dominate the error, therefore on the plot the rounding error dominates to the left of the minimum. Thus, to produce a turning point, the truncation error must dominate to the right of the minimum on the plot.

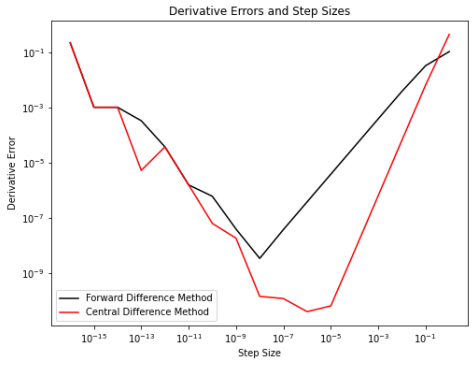

Figure 1: This plot compares the accuracy of the forward difference method and the central difference method of numerical integration when integrating the function $f(x) = e^{-x^2}$ at $x = 0.5$ for different step sizes (h).

Step Value (h)	Forward Difference Approximation	Error From Analytical Solution
10^{-16}	-1.0	2.21199217e-01
10^{-15}	-0.7777777777777778	1.02300529e-03
10^{-14}	-0.7777777777777778	1.02300529e-03
10^{-13}	-0.779134295227525	3.33512156e-04
10^{-12}	-0.7788386810258687	3.78979545e-05
10^{-11}	-0.7787991828759215	1.60019548e-06
10^{-10}	-0.778801403321787	6.20250382e-07
10^{-9}	-0.7788008236522808	4.05808759e-08
10^{-8}	-0.7788007866107726	3.53936769e-09
10^{-7}	-0.77880082202023	3.89488252e-08
10^{-6}	-0.7788011724183845	3.89346980e-07
10^{-5}	-0.7788046770076301	3.89393623e-06
10^{-4}	-0.7788397166209352	3.89335495e-05
10^{-3}	-0.7791895344301241	3.88751359e-04
10^{-2}	-0.7826298571289574	3.82907406e-03
10^{-1}	-0.8112445700037387	3.24437869e-02
10^{0}	-0.6734015585095405	1.05399225e-01

Table 1: A table displaying the numerical approximation of the integral of $f(x) = e^{-x^2}$ at $x = 0.5$ using the forward difference method. Note the analytical solution -0.7788007830714049. At optimal step size the forward difference method achieves the correct answer to 9 decimal places.

d)

Firstly note that although the central difference method (CDM) of calculating the derivative of $f(x) = e^{-x^2}$ at $x = 0.5$ isn't always more accurate than the forward difference method (FDM), with the optimal choice of step size for each method the CDM is approximately a factor of 10^2 more accurate. This is shown qualitatively in the plot as the CDM has a minimum that reaches deeper than the FDM. However, the plot also displays steps sizes where the FDM and the CDM are equally accurate, such as at steps sizes of $10^{-16}, 10^{-15}, 10^{-14}, 10^{-12}, 10^{-11}$ and somewhere between 10^{-1} and 10^{0}.

Secondly, note that the optimal step size of each method is different. Whereas we have seen that the optimal step size for the FDM occurs at 10^{-8}, for the CDM the optimal step size occurs at 10^{-6}. This is shown qualitatively in the plot as the FDM and CDM reach minimums at different x-axis locations.

Finally, note that the behaviour of the CDM error is more complex than the behaviour of the FDM. The plot of the error with the CDM has two minimums, the global minimum at 10^{-6} and a different local minimum at 10^{-13}, while the plot of the error with the FDM only has one.

Question 2

i)

The value of the Dawson function

$$D(x) = e^{-x^2} \int_0^x e^{t^2} \tag{1}$$

at x=4 was computed using the Trapezoidal rule, Simpson's rule, and finally Scipy's scipy.special.dawsn with N=8 slices. The Trapezoidal rule returned a value of 0.1396, Simpson's rule gave a value of 0.1300, and scipy.special.dawsn gave a value of 0.1293 (all rounded to 4 significant digits). Simpson's rule gave an answer only 0.0007 away from the true value whereas the Trapezoidal rule gave an answer 0.0096 away from the true value, thus Simpson's rule gave a more accurate value.

ii)

Using Scipy's value of the Dawson function as the "true" value, the number of slices needed to approximate the integral with an error of $O(10^9)$ was calculated. For the Trapezoidal rule, $N = 2^{14}$ is needed to get an error of $O(10^9)$, the error being 5.14×10^{-9}. For Simpson's rule, $N = 2^{21}$ gave an error of $O(10^9)$, with the error value being -5.30×10^{-9}. Using the timing method described in Lab 1, Simpson's method was timed at 2.592×10^4 seconds, Trapezoidal method was timed at 7.072×10^4 seconds, and Scipy's method took 6.795×10^{-5} seconds. The timing code was performed multiple times to get a more accurate result.

iii)

The practical estimation of errors method was adapted from the textbook to give an estimate of the errors for both the Trapezoidal and Simpson's rule. For the Trapezoidal rule, the equation $\epsilon = \frac{1}{3}(I_2 - I_1)$ was used with $N_1 = 32$, $N_2 = 64$ to find $\epsilon = -2.546 \times 10^{-3}$. For Simpson's rule, the equation $\epsilon = \frac{1}{15}(I_2 - I_1)$ was used with the same N values to find an error of $\epsilon = -4.116 \times 10^{-5}$. The error for Simpson's rule is smaller by $O(10^2)$, making it notably more accurate than the Trapezoidal rule.

b)

To calculate the value of the Bessel function:

$$J_m(x) = \frac{1}{\pi} \int_0^\pi \cos(m\theta - x \sin\theta) \mathrm{d}\theta \tag{2}$$

Simpson's rule was used with N=1000. The values of Bessel functions J_0, J_1, J_2 were computed for x=[0,20] and plotted in Figure 2.

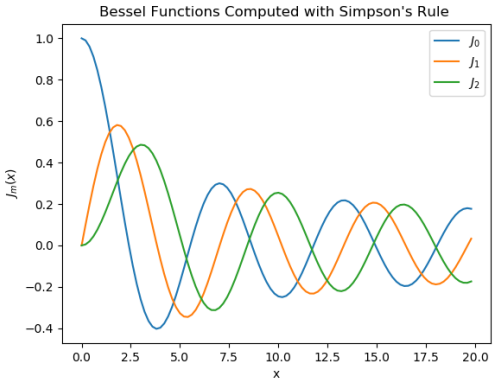

Figure 2: Bessel functions computed using Simpson's rule with N=1000.

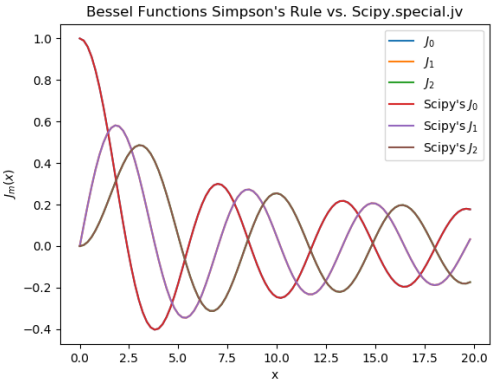

Figure 3: Bessel functions computed using Simpson's rule compared to those computed using Scipy's routine.

These results were then compared to those from scipy.special.jv in as can be seen in Figure 3. From the graph, it can be seen that our Bessel functions reproduce the Scipy routines very well as they overlap exactly. The density plot in Figure 4 shows the intensity of a point light source calculated using

$$I(r) = \left(\frac{J_1(kr)}{kr} \right)^2 \tag{3}$$

with $\lambda = 500$nm and r values ranging from 0 to 1 μm.

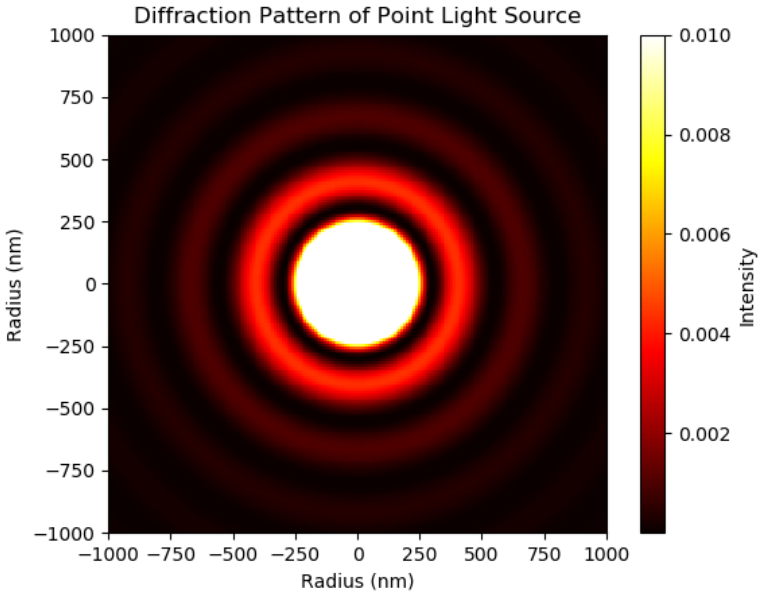

Figure 4: Density plot showing the diffraction pattern of a point light source.

Question 3

a)

Nothing to submit.

b)

Nothing to submit.

c)

When considering which integration method to use, I mainly thought about what a diffraction pattern looks like, and what our freedom of choice for number of steps meant. I think that I will use Simpson's integration as it is likely to give a higher degreen of accuracy than the trapezoid integration. Although the text states that trouble can arise when using Simpson's integration if the integrand is noisy or not smooth, I believe that the oscillatory behabiour of diffraction patterns are smooth functions, and that the text was referring to extremely noisy or piecewise behaviour. As for the number of steps that will be used, I shall repeat the process of section 5.3 in the text until I get an accuracy in the intensity that ensure that the large majority of points are above Python's 16 figure rounding limit.

```
#pseudocode
#define a function that computes (*) from above
#the function will take u and x as arguments
#define a simpson integration function that takes a position x, the grating u, integration
#bounds a and b, and the number of steps as arguments
#use the loops from the text to set up the sums: odd-for k in range(1,N,2) and
#even-for k in range(2,N,2)
#set initial conditions
#choose number of sample points with trial and error as per the text, until a desired
#accuracy is achieved.
```

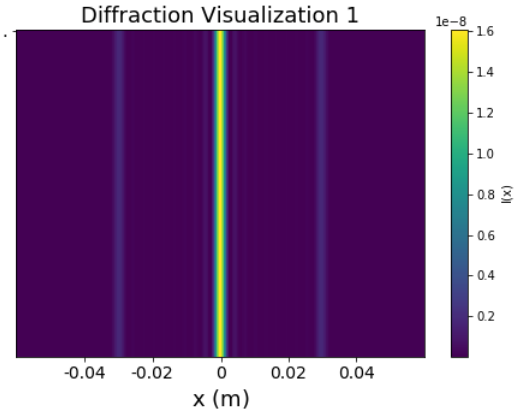

Figure 5: This plot is for question 3d). It shows a visualization of a diffraction pattern produced by incident light of 500nm that passes through a grating with 10 slits that have separation of 20μm, which produces an intensity transmission function given by $q(u) = sin^2(\alpha u)$, is then focused by a lens with a focal length of 1 meter, and is projected onto a screen that is 10cm wide.

d)

See Figure 5.

e)

See Figures 6, 7, and 8.

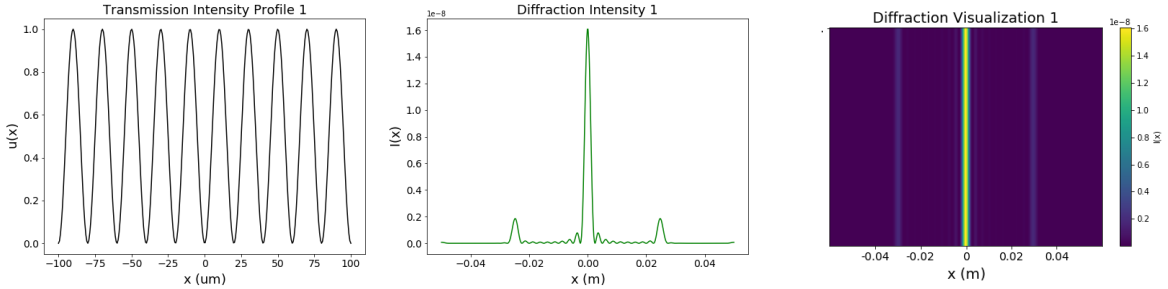

Figure 6: The left plot of this suite shows the fraction of light transmitted through a grating of total width 200μm with 10 slits separated by 20μm each. The intensity transmission function for such a grating is given by $q(u) = sin^2(\alpha u)$ where u represents the position on the grating and $\alpha = \pi/20\mu m$. The center plot shows the intensity of a diffraction pattern produced by incident light of 500nm that passes through such a grating, is focused by a lens with a focal length of 1 meter, and is projected onto a screen that is 10cm wide. The right plot shows a visualization of the diffraction pattern that would show up on the screen. Note that the visualization matches the intensity profile. That is, that the location of the most intense bands of light on the screen matches the predication that is the intensity profile.

Code

Q1

```
#import needed modules
import numpy as np
import matplotlib.pyplot as plt
```

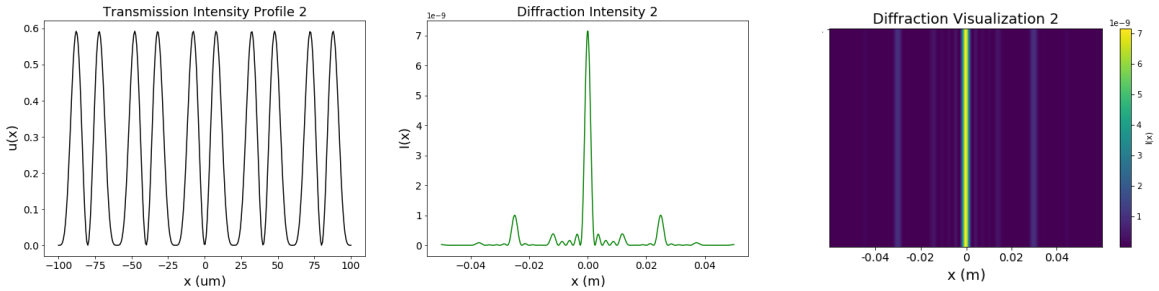

Figure 7: The left plot of this suite shows the fraction of light transmitted through a grating of total width $200\mu m$ *with 10 slits separated by* $20\mu m$ *each. The intensity transmission function for such a grating is given by* $q(u) = sin^2(\alpha u)cos^2(\beta u)$ *where* u *represents the position on the grating,* $\alpha = \pi/20\mu m$*, and* $\beta = \frac{\alpha}{2}$*. The center plot shows the intensity of a diffraction pattern produced by incident light of 500nm that passes through such a grating, is focused by a lens with a focal length of 1 meter, and is projected onto a screen that is 10cm wide. The right plot shows a visualization of the diffraction pattern that would show up on the screen. Note again that the visualization matches the intensity profile. This suite of plots shows that two gratings with an equal number of slits can produce different diffraction patterns. It should be clear that the diffraction patterns in this figure and Figure 6 are very similar, differing most with the addition of two extra peaks at approximately* $\pm 0.1m$*.*

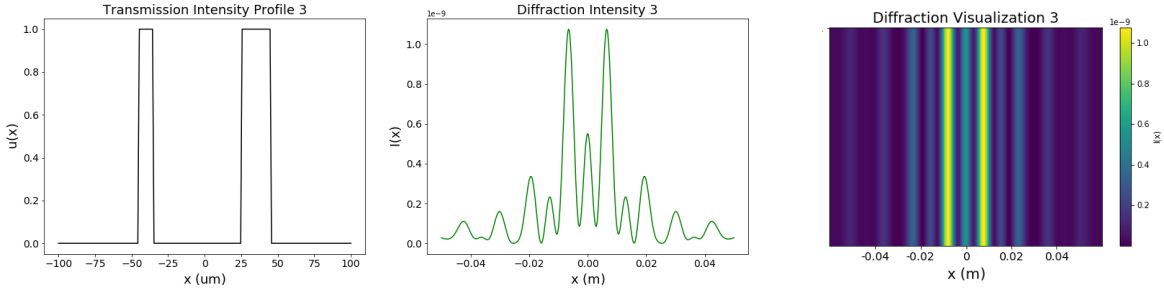

Figure 8: The left plot of this suite shows the fraction of light transmitted through a grating of total width $200\mu m$ *with 2 non-symmetrical 'square slits, one of width* $10\mu m$ *the other of width* $20\mu m$ *separated by* $60\mu m$*; 'square' refers to the fact that the transmission fraction of light is zero everywhere on the grating, and one at locations of the slit. The intensity transmission function for such a grating is shown in the left plot. The center plot shows the intensity of a diffraction pattern produced by incident light of 500nm that passes through such a grating, is focused by a lens with a focal length of 1 meter, and is projected onto a screen that is 10cm wide. The right plot shows a visualization of the diffraction pattern that would show up on the screen. Note the extremely unique diffraction pattern. Two slit diffraction usually results in a diffraction pattern that has a central peak. It seems to be the case that the 'square' profile of the slits creates a diffraction pattern that instead has two large peaks. It seems also possible that the unique sizes of the slits plays into the results.*

```
#define forward differnce method
def fwd_dif(x,hs):
    results=[]
    for i in hs:
        results.append((np.exp(-(x+i)**2) - np.exp(-(x)**2))/((x+i) - x))
    return results

#define h values
hs=[10**(-16),10**(-15),10**(-14),10**(-13),10**(-12),10**(-11),10**(-10),10**(-9),10**(-8),10**(-7),10

#run forward differnce method at x=0.5
```

```
results=fwd_dif(0.5,hs)

#compute analytical answer for comparison
analytical_answer=-2*(0.5)*np.exp(-(0.5)**2)

#find error
derivative_error=np.abs(np.array(results)-analytical_answer)

#plot fwd dif error
plt.figure(figsize=[8,6])
plt.plot(hs,derivative_error)
plt.title('Derivative Errors and Step Sizes')
plt.xlabel('Step Size')
plt.ylabel('Derivative Error')
plt.yscale('log')
plt.xscale('log')
plt.savefig('1_c')
plt.show()

#define central difference method
def cntrl_dif(x,hs):
    results=[]
    comp=[]
    for i in hs:
        results.append((np.exp(-(x+i)**2) - np.exp(-(x-i)**2))/((x+i) - (x-i)))
        comp.append((np.exp(-(x+i)**2) - np.exp(-(x-i)**2), (x+i) - (x-i)))
    return results, comp

#run central dif method at x=0.5
results_cntrl,comp=cntrl_dif(0.5,hs)

#find error
derivative_error_cntrl=np.abs(np.array(results_cntrl)-analytical_answer)

#plot both fwd and cntrl difference
plt.figure(figsize=[8,6])
plt.plot(hs,derivative_error,label='Forward Difference Method',color='black')
plt.plot(hs,derivative_error_cntrl,label='Central Difference Method',color='red')
plt.title('Derivative Errors and Step Sizes')
plt.xlabel('Step Size')
plt.ylabel('Derivative Error')
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.savefig('1_d')
plt.show()
```

Q2

a)

```
#lab 2 question 2 a)
#import necessary modules
import numpy as np
```

```
from scipy import special
from time import time
# save start time
start_trap=time()

#define function we want to integrate
def f(x):
    return np.exp(x**2)


#trapezoidal rule of integration
 #number of slices
N=32
a=0
b=4
h=(b-a)/N

s_trap = 0.5*f(a) + 0.5*f(b)
for k in range(1,N):
    s_trap += f(a+k*h)

#multiply by the other part of function
I1_trap=s_trap*h*np.exp(-4**2)

# save the end time
end_trap=time()
# the difference is the elapsed time (in seconds)
diff_trap=end_trap-start_trap

#compute and time scipy's dawson function
start_scipy=time()
a2=special.dawsn(4)
end_scipy=time()
diff_scipy=end_scipy-start_scipy

#perform integration with simpsons rule and time
start_simp=time()

s_simp=f(a)+f(b)
#odd terms
for k in range(1,N,2):
    s_simp += 4*f(a+k*h)
#even terms
for k in range(2,N,2):
    s_simp += 2*f(a+k*h)

#multiply by other part of function
I1_simp=s_simp*(1/3)*h*np.exp(-4**2)

#compute time
end_simp=time()
diff_simp=end_simp-start_simp

#print out values
```

```
print("Simpsons time",diff_simp )
print("Trapezoidal time",diff_trap )
print("Scipys time",diff_scipy )
#%%

# practical estimation of error
#define new N2 and corresponding h2
N2=2*N
h2=(b-a)/N2

#redo the integral using trapezoidal method with the new value of N2
s2_trap = 0.5*f(a) + 0.5*f(b)
for k in range(1,N2):
    s2_trap += f(a+k*h2)

I2_trap=s2_trap*h2*np.exp(-4**2)

#find the error using textbook eqn
error_trap=1/3*(I2_trap-I1_trap)

#redo the integral using simpsons method with the new value of N2
s2_simp=f(a)+f(b)
#odd terms
for k in range(1,N2,2):
    s2_simp += 4*f(a+k*h2)
#even terms
for k in range(2,N2,2):
    s2_simp += 2*f(a+k*h2)

I2_simp=s2_simp*(1/3)*h2*np.exp(-4**2)

#find the error using textbook eqn
error_simp=1/15*(I2_simp-I1_simp)


#%% 2ii

#define trapezoidal error function
def err_trap(c,d):
    return 1/12*h**2*(c-d)

#derivative of dawson at a, b wrt x
#c=f'(a)
c=1
#d=f'(b)
d=1-8*special.dawsn(4)
#print the value
print(err_trap(c,d))

#N=2**14 gives 5.14e-09


#define simpsons error fn
def err_simp(e,f):
```

```
    return 1/90*h**4*(e-f)

#derivative of dawson at a, b wrt x
#c=f'''(a)
e=2*np.exp(a**2)*(2*np.exp(a**2)*(2+7*a**2)+np.sqrt(np.pi)*a*(3+2*a**2)*special.dawsn(a))
#d=f'''(b)
f=2*np.exp(b**2)*(2*np.exp(b**2)*(2+7*b**2)+np.sqrt(np.pi)*b*(3+2*b**2)*special.dawsn(b))

#print error value
print(err_simp(e,f))
#N=2**21
```

b)

```
#lab2 q2 b)
#Import necessary modules
import numpy as np
import matplotlib.pyplot as plt
from scipy import special
from matplotlib import cm

#define arrays for the values of m and x
m=np.array([0, 1, 2])
x=np.arange(0, 20, 0.2)

#define the bessel function that will perform integration with Simpsons rule
def J(m,x):
    #simpsons rule
    N=1000
    a=0
    b=np.pi
    h=(b-a)/N
    #define function to be integrated
    def f(theta):
        return np.cos(m*theta-x*np.sin(theta))

    s=f(a)+f(b)
    #odd terms
    for k in range(1,N,2):
        s += 4*f(a+k*h)
        #even terms
    for k in range(2,N,2):
        s += 2*f(a+k*h)

        #compute integral
        I1=s*(1/3)*h
        #multiply result by 1/pi because thats part of bessel functions
    return I1*1/np.pi

#plot the functions
plt.figure(1)
```

```
plt.title("Bessel Functions Simpson's Rule vs. Scipy.special.jv")
plt.xlabel("x")
plt.ylabel("$J_m(x)$")
plt.plot(x, J(m[0], x), label="$J_0$")
plt.plot(x, J(m[1], x), label="$J_1$")
plt.plot(x, J(m[2], x), label="$J_2$")
plt.plot(x, special.jv(0,x), label="Scipy's $J_0$")
plt.plot(x, special.jv(1,x), label="Scipy's $J_1$")
plt.plot(x, special.jv(2,x), label="Scipy's $J_2$")
plt.legend()

#%%

# b in textbook
#define arrays of values in micrometers
x=np.arange(-1000, 1010, 10)
y=np.arange(-1000, 1010, 10)
#define an empty 2d array for radius values to be stored in
r=np.zeros([len(x), len(x)])

#calculate radii
for i in range(len(x)):
    for j in range(len(x)):
        r[i,j]=np.sqrt(x[i]**2+y[j]**2)
#define values given in question, l=lambda
l=500 #nm
k=(2*np.pi)/l  #units 1/r

#define the intensity function
def intensity(r):
    return (J(m[1], k*r)/(k*r))**2

IN=intensity(r)
#get rid of infinity caused by r=0 points
IN[np.isinf(IN)]=0.25

#plot density graph
#ser vmax= low number to show more detail, and extent makes the axes right
plt.imshow(intensity(r),  vmax=0.01, cmap=cm.hot, extent=(-1000,1000,-1000,1000))
plt.colorbar(label="Intensity")
plt.figure(1)
plt.title("Diffraction Pattern of Point Light Source")
plt.xlabel("Radius (nm)")
plt.ylabel("Radius (nm)")
```

Q3

```
#import needed modules
import numpy as np
import matplotlib.pyplot as plt
```

```
#3c

#define alpha
omega=10*20 #total width of the grating in um
alpha=np.pi/(20) #slit separation of 20um
grating=np.linspace(-omega/2,omega/2,300)
screen=np.linspace(-0.05,0.05,500)
#define the intensity transmission function of the diffraction grating at distance u from the central a
def q(u):
    return (np.sin(alpha*u))**2
plt.figure(figsize=[8,6])
plt.plot(grating,q(grating),color='black')
plt.title('Transmission Intensity Profile 1',fontsize=18)
plt.ylabel(r'$\rm u(x) $',fontsize=18)
plt.xlabel(r'$\rm x \ (um)$',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('3c_grating')
plt.show()

#define a function that computes (*)
def f(x,u):
    return np.sqrt(q(u))*np.exp((1j*2*np.pi*x*u)/(wl*fl))

#define needed constants - all units in micrometers
wl=0.5 #wavelength: 500nm in um
fl=1 #focal length: 1m in m

def simpson(xs, u, a, b, n):
    intensity=[]
    f_a=0
    f_b=0
    for i in u:
        f_a+=f(a,i)
        f_b+=f(b,i)

    for k in xs:
        h=(b-a)/n
        odd_sum=0.0
        even_sum=0.0

        step=a + h
        for i in range(1,n,2):
            odd_sum += f(k,step)
            step += 2*h

        step = a + 2*h
        for i in range(2,n,2):
            even_sum += f(k,step)
            step += 2*h

        simp_sum = (h/3)*(f_a+f_b+4*odd_sum+2*even_sum)
        intensity.append(np.abs(simp_sum)**2)
```

```
    return np.array(intensity) * 10**(-12)

intensity=simpson(screen,grating,-omega/2,omega/2,250)

plt.figure(figsize=[8,6])
plt.plot(screen,intensity,color='green')
plt.ylabel(r'$\rm I (x) $',fontsize=18)
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.title('Diffraction Intensity 1',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('q3_c_1')

#3d

vis=np.tile(intensity,(400,1))

plt.figure(figsize=[8,5])
plt.imshow(vis)
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.yticks([0],'.')
plt.xticks([83,166,250,333,416],labels=['-0.04','-0.02','0','0.02','0.04'])
plt.title('Diffraction Visualization 1',fontsize=18)
plt.colorbar(label=r'$\rm I(x) $')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('q3_d')

#3ei

#define alpha
omega=10*20 #total width of the grating in m
alpha=np.pi/(20) #slit separation of 20um in m
beta=alpha/2
grating=np.linspace(-omega/2,omega/2,300)
screen=np.linspace(-0.05,0.05,500)
#define the intensity transmission function of the diffraction grating at distance u from the central a
def q_2(u):
    return np.sin(alpha*u)**2 * np.cos(beta*u)**2
plt.figure(figsize=[8,6])
plt.plot(grating,q_2(grating),color='black')
plt.title('Transmission Intensity Profile 2',fontsize=18)
plt.ylabel(r'$\rm u(x) $',fontsize=18)
plt.xlabel(r'$\rm x \ (um)$',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('3ei_grating')
plt.show()

#define a function that computes (*)
def f_2(x,u):
    return np.sqrt(q_2(u))*np.exp((1j*2*np.pi*x*u)/(wl*fl))
```

```
def simpson_2(xs, u, a, b, n):
    intensity=[]
    f_a=0
    f_b=0
    for i in u:
        f_a+=f_2(a,i)
        f_b+=f_2(b,i)

    for k in xs:
        h=(b-a)/n
        odd_sum=0.0
        even_sum=0.0

        step=a + h
        for i in range(1,n,2):
            odd_sum += f_2(k,step)
            step += 2*h

        step = a + 2*h
        for i in range(2,n,2):
            even_sum += f_2(k,step)
            step += 2*h

        simp_sum = (h/3)*(f_a+f_b+4*odd_sum+2*even_sum)
        intensity.append(np.abs(simp_sum)**2)

    return np.array(intensity) * 10**(-12)

intensity2=simpson_2(screen,grating,-omega/2,omega/2,250)
vis2=np.tile(intensity2,(400,1))

plt.figure(figsize=[8,6])
plt.plot(screen,intensity2,color='green')
plt.ylabel(r'$\rm I (x) $',fontsize=18)
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.title('Diffraction Intensity 2',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('q3_ei_1')
plt.show()
plt.close()
plt.figure(figsize=[8,5])
plt.imshow(vis2)
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.yticks([0],'.')
plt.xticks([83,166,250,333,416],labels=['-0.04','-0.02','0','0.02','0.04'],fontsize=14)
plt.title('Diffraction Visualization 2',fontsize=18)
plt.colorbar(label=r'$\rm I(x) $')
plt.savefig('q3_ei_2')

#3eii

#define alpha
omega=10*20 #total width of the grating in m
```

```
grating=np.linspace(-omega/2,omega/2,200)
screen=np.linspace(-0.05,0.05,500)
#to get an idea, i make an example grating
u3=np.zeros(len(grating)) #create an array of zeros the length of the grating
u3[55:65]=1 #55 um of no transmission up to the first slit of 10um with transmission set = 1
u3[65:125]=0 #60um of transmission between slits
u3[125:145]=1 #second slit of width 20um
#define transmission function
def q_3(a):
    if a > grating[55] and a < grating[65]:
        return 1
    elif a > grating[125] and a < grating[145]:
        return 1
    else:
        return 0

#define alpha
omega=10*20 #total width of the grating in m
grating=np.linspace(-omega/2,omega/2,200)
screen=np.linspace(-0.05,0.05,500)
#to get an idea, i make an example grating
u3=np.zeros(len(grating)) #create an array of zeros the length of the grating
u3[55:65]=1 #55 um of no transmission up to the first slit of 10um with transmission set = 1
u3[65:125]=0 #60um of transmission between slits
u3[125:145]=1 #second slit of width 20um
#define transmission function
def q_3(a):
    if a > grating[55] and a < grating[65]:
        return 1
    elif a > grating[125] and a < grating[145]:
        return 1
    else:
        return 0

plt.figure(figsize=[8,6])
plt.plot(grating,u3,color='black')
plt.title('Transmission Intensity Profile 3',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.ylabel(r'$\rm u(x) $',fontsize=18)
plt.xlabel(r'$\rm x \ (um)$',fontsize=18)
plt.savefig('3eii_grating')
plt.show()

#define a function that computes (*)
def f_3(x,u):
    return np.sqrt(q_3(u))*np.exp((1j*2*np.pi*x*u)/(wl*fl))

def simpson_3(xs, u, a, b, n):
    intensity=[]
    f_a=0
    f_b=0
    for i in u:
        f_a+=f_3(a,i)
```

```
        f_b+=f_3(b,i)

    for k in xs:
        h=(b-a)/n
        odd_sum=0.0
        even_sum=0.0

        step=a + h
        for i in range(1,n,2):
            odd_sum += f_3(k,step)
            step += 2*h

        step = a + 2*h
        for i in range(2,n,2):
            even_sum += f_3(k,step)
            step += 2*h

        simp_sum = (h/3)*(f_a+f_b+4*odd_sum+2*even_sum)
        intensity.append(np.abs(simp_sum)**2)

    return np.array(intensity) * 10**(-12)

intensity3=simpson_3(screen,grating,-omega/2,omega/2,250)
vis3=np.tile(intensity3,(400,1))

plt.figure(figsize=[8,6])
plt.plot(screen,intensity3,color='green')
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.ylabel(r'$\rm I(x) $',fontsize=18)
plt.title('Diffraction Intensity 3',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('q3_eii_1')
plt.show()
plt.close()
plt.figure(figsize=[8,5])
plt.imshow(vis3)
plt.xlabel(r'$\rm x \ (m)$',fontsize=18)
plt.yticks([0],'.')
plt.xticks([83,166,250,333,416],labels=['-0.04','-0.02','0','0.02','0.04'],fontsize=14)
plt.title('Diffraction Visualization 3',fontsize=18)
plt.colorbar(label=r'$\rm I(x) $')
plt.savefig('q3_eii_2')
```