

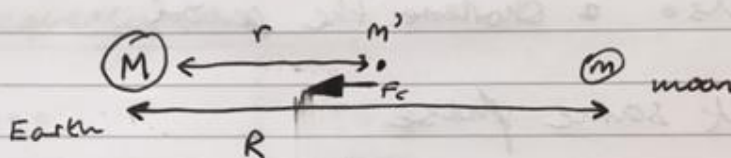
Programming Coursework.

Candidate number: 21642

Question 1 - Using the secant method to find L1

Part a)

1 a)



at r , there may be an object of mass m' which feels an equal pull by the Earth and the Moon in opposing directions, which results in a period of orbit which is the same as the Moon's.

$\frac{GMm'}{r^2} = \frac{Gmm'}{(R-r)^2}$ The centripetal force keeping the satellite in orbit is therefore:

$$F_c = \frac{GMm'}{r^2} - \frac{Gmm'}{(R-r)^2} = \frac{m'v^2}{r}$$

resultant centripetal force for circular orbit of radius r and velocity v (standard result).

equivalent.

$$\left(\frac{m'}{m'}\right) \rightarrow \frac{GM}{r^2} - \frac{Gm}{(R-r)^2} = \omega^2 r$$

Part b)

I designed this program to contain the equation found in part a) where the resultant force is set such that the period of the satellite is the same as the period of the moon. One function returns the value of this equation, another calculates a numerical derivative of this function at a certain distance and uses this to find the root of the equation in part a) using the secant method of differentiation.

```
1. import numpy as np
2.
3. # define physical constants and measurements
4. G = 6.6741 * 10**-11 # gravitation constant in m3kg-1s-2
5. M = 5.9722*10**24 # mass of the earth in kg
6. m = 7.3420*10**22 # mass of the moon in kg
```

```

7. R = 3.8440 * 10**8 # earth-moon distance in m
8. w = 2.6617*10**-6 # angular velocity in radians per second
9.
10. # this function returns the value of the expression in part 1 a).
11. def polynomial(x):
12.     return G*M*(x**-2) - G*m*(R - x)**-2 - x*w**2
13.
14. # returns the derivitive of a polynomial using the second method
15. def secantmethod(x1, x2):
16.     return x2 - (polynomial(x2)*(x2 - x1))/(polynomial(x2)-polynomial(x1))
17.
18. precision = 6 # ~ the number of decimal places needed
19. done = False
20. # make guesses for x1 and x2. A newton raphson search could be made for the first v
    alue
21. # but using a guess here is simpler and fast.
22. x1 = 10000
23. x2 = x1 * 2
24. x3 = 0
25.
26. # continually looks for a more precise derivative until
27. # one of the required precision is found.
28. while True:
29.     x3 = secantmethod(x1, x2)
30.     if np.abs(polynomial(x3)) < 1/10**precision:
31.         break
32.     x1 = x2
33.     x2 = x3
34.
35.
36. print('L1: {} m'.format(int(np.round(x3, -4)))) #rounds and prints L1

```

Output:

L1: 326050000 m

This value for L1 is similar to values cited in articles online.

Question 2 - Diffraction

Part a)

I made this program to graph the bessel functions of $m = 0, 1$ and 2 . By using the trapezium rule the program numerically integrates the integrand in for x values from 0 to 20 and stores the values of the bessel function value in an array to be graphed later.

```

1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. # set N
5. N = 10000
6.
7. # the function to be integrated
8. def integrand(theta, m, x):
9.     return np.cos(m*theta - x*np.sin(theta))
10.
11. # this function returns an integral using the trapezium rule
12. def trapezium(a, b, m, x):
13.     area = 0
14.     h = (b-a)/N
15.     for n in range(0, N):
16.         area = area + (h * ((integrand(h*n, m, x) + integrand(h*(n+1), m, x)) / 2))

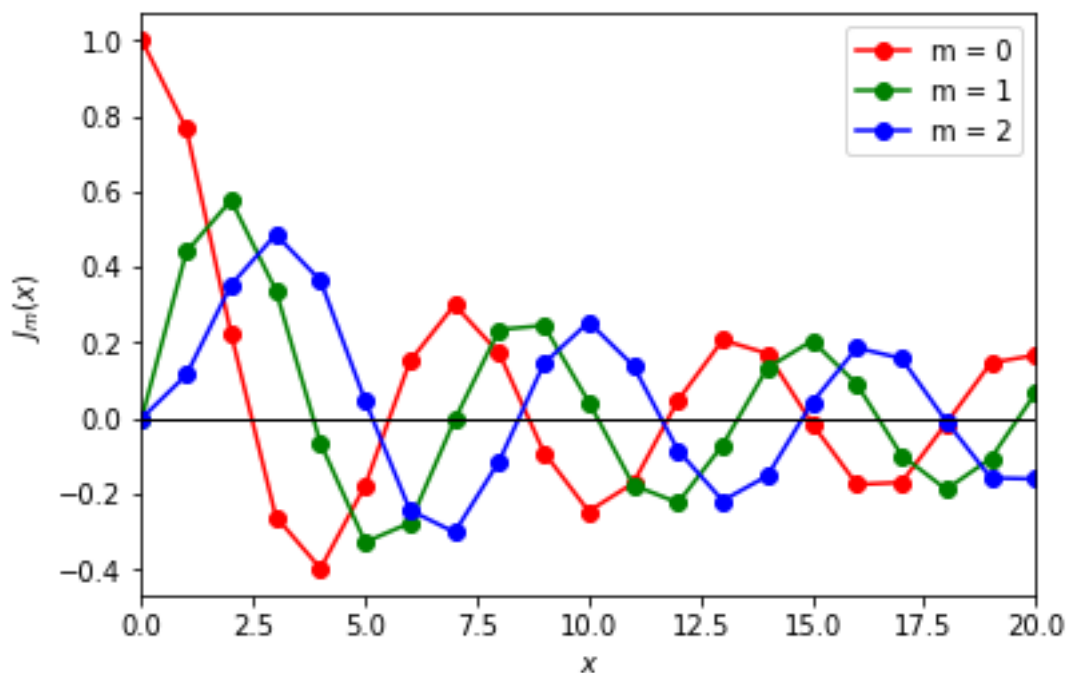
```

```

17.     return area
18.
19. # the bessel function for a specified m and x value
20. def J(m, x):
21.     return 1/np.pi * trapezium(0, np.pi, m, x)
22.
23. # define empty arrays to contain bessel function at various x-values later
24. J0 = [0]*21
25. J1 = [0]*21
26. J2 = [0]*21
27.
28. #m = 0
29. for x in range(0, 20+1): # 20+1 because range is not inclusive at the upper end
30.     J0[x] = J(0, x) # fills the array with the bessel function at each x-
        value between 0 and 20
31.
32. #m = 1
33. for x in range(0, 20+1):
34.     J1[x] = J(1, x)
35.
36. #m = 2
37. for x in range(0, 20+1):
38.     J2[x] = J(2, x)
39.
40. # plot each bessel function on the same graph
41. plt.plot(np.linspace(0, 20, num=21), J0, 'ro-', label='m = 0')
42. plt.plot(np.linspace(0, 20, num=21), J1, 'go-', label='m = 1')
43. plt.plot(np.linspace(0, 20, num=21), J2, 'bo-', label='m = 2')
44.
45. plt.axhline(y=0, color='k', linewidth=1) # x-axis line
46. plt.xlim(0, 20)
47. plt.xlabel('$x$')
48. plt.ylabel('$J_m(x)$')
49. plt.legend()

```

Output:



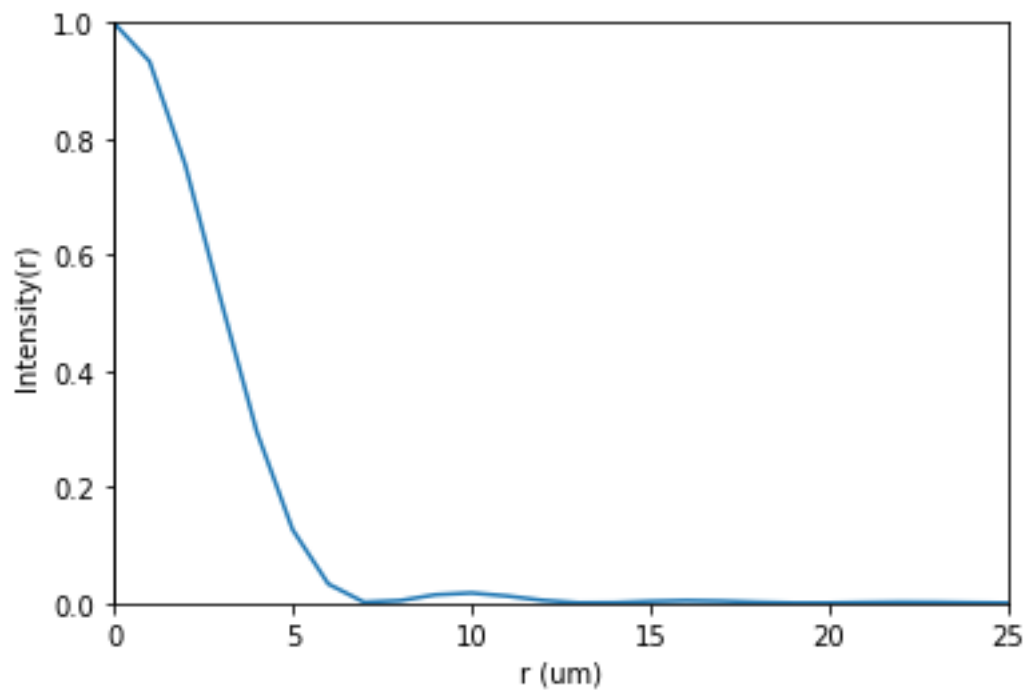
This output is similar to what I have seen in a textbook.

Part b)

This program builds off the previous question but only uses the bessel function for $m = 1$. The wavelength of light is estimated as 600 nm and a graph is plotted of r -values from 0 to 25 micrometers.

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. # set N
5. N = 10000
6. m = 1 # we only need J1(x) in the program, so it can be defined globally.
7. wavelength = 600 * 10**-9 # approximately the wavelength of visible light
8.
9. def integrand(theta, x):
10.     return np.cos(m*theta - x*np.sin(theta))
11.
12. # integrates the integrand using the trapezium rule
13. def trapezium(a, b, x):
14.     area = 0
15.     h = (b-a)/N
16.     for n in range(0, N):
17.         area = area + (h * ((integrand(h*n, x) + integrand(h*(n+1), x)) / 2))
18.     return area
19.
20. #bessel function for m = 1 at x
21. def J(x):
22.     return 1/np.pi * trapezium(0, np.pi, x)
23.
24. # calculates intensity at r in terms of I0
25. def intensity(r):
26.     I0 = 1
27.     x = (1/10 * np.pi * (1/wavelength) * (r*10**-6))
28.     return I0 * (2*J(x)/x)**2
29.
30. J1 = [0]*26
31. I = [0]*26
32.
33. # loops through i values from r = 0 to r = 25 micrometers
34. for r in range(0, 25+1):
35.     if r == 0:
36.         I[r] = 1 # the sinc function is 1 and x = 0,
37.         # doing it manually avoids a division by zero error.
38.     else:
39.         I[r] = intensity(r)
40.
41. # plots intensity against r
42. plt.plot(np.linspace(0, 25, 25+1), I)
43. plt.ylabel('Intensity(r)')
44. plt.xlabel('r (um)')
45. plt.xlim(0, 25)
46. plt.ylim(0,1)
47. plt.axhline(y=0, color='k', linewidth=1)
48. plt.axvline(x=0, color='k', linewidth=1)
```

Output:



This is what I expected to yield as it is the intensity pattern seen in diffraction experiments for light.