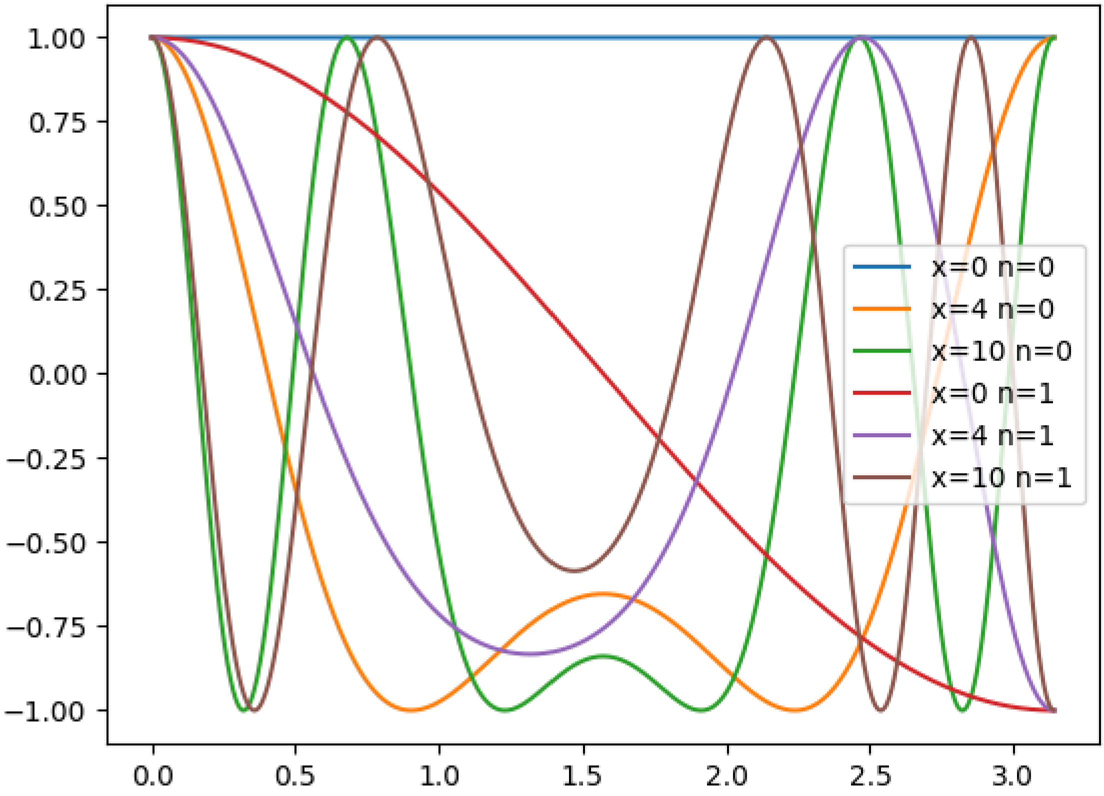
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
import pandas as pd
import matplotlib.ticker as ticker
import scipy
# import the lib we need later
```

In [ ]:
```
# Problem 1
# a)

def bessel_integrand(theta, n, x):
    """
    Returns the integrand of the Bessel function integral.
    Parameters
    ----------
    theta : float or array_like
        The angle in radians.
    n : int
        The order of the Bessel function.
    x : float
        The argument of the Bessel function.
    Returns
    -------
    J_n(x): float or array_like
        The value of the integrand at the given theta, x, and n.
    """
    return np.cos(n*theta - x*np.sin(theta))
    # by definition

xaxis = np.linspace(0,np.pi,10000)
# creat a series of x values
plt.plot(xaxis,bessel_integrand(xaxis,0,0),label = 'x=0 n=0')
plt.plot(xaxis,bessel_integrand(xaxis,0,4),label = 'x=4 n=0')
plt.plot(xaxis,bessel_integrand(xaxis,0,10),label = 'x=10 n=0')
plt.plot(xaxis,bessel_integrand(xaxis,1,0),label = 'x=0 n=1')
plt.plot(xaxis,bessel_integrand(xaxis,1,4),label = 'x=4 n=1')
plt.plot(xaxis,bessel_integrand(xaxis,1,10),label = 'x=10 n=1')
# plot the lines as asked
plt.legend()
# provide a legend such that it is easy to see
```

Out[ ]: <matplotlib.legend.Legend at 0x1d745523f50>

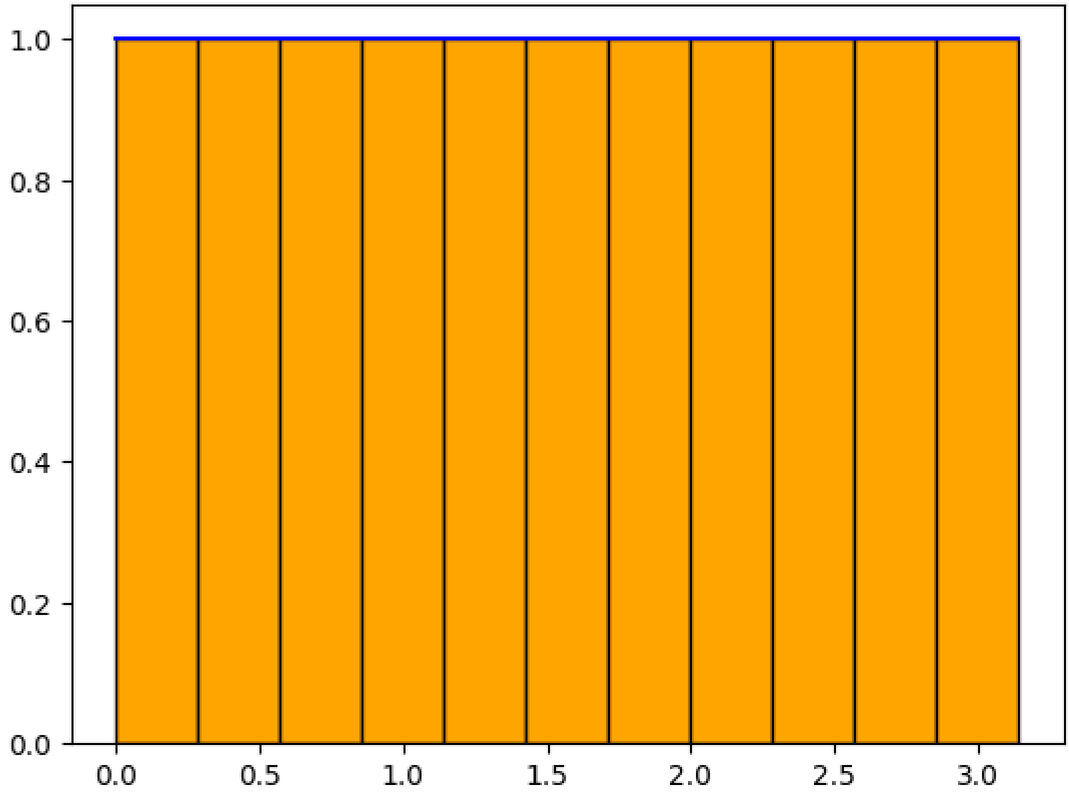


In [ ]:
```
# b)
def trapezoidal_rule(func, a, b, Ntrap, args=None, plot=True):
    """
    Compute the integral of func(x, *args) over the interval [a, b] using the tr

    Parameters
    ----------
    func : callable
        The function to integrate. The first argument of this function must be t
        integration, i.e. f = func(x, *args). The other arguments are passed thr
        parameter.
    a : float
        The lower limit of integration.
    b : float
        The upper limit of integration.
    Ntrap : int
        The number of trapezoidal subintervals to use.
    args : tuple, optional
        Extra arguments besides the integration variable to pass to the function
        integrated.
    plot : bool, optional
        If True, plot the function func(x) over the interval [a, b] as well as t
        trapezoids used to compute the trapezoidal rule. Default is True.

    Returns
    -------
    answer : float
        The estimate of the integral of func(x) over the interval [a, b].

    """
    x = np.linspace(a,b,Ntrap)
    # get all the points required
    y = func(x,*args)
    # find the value of each point
    elements = (y[:-1]+y[1:])/2*(x[1]-x[0])
```

```
    # calculate the area of each element
    if(plot):
        plt.bar((x[:-1]+x[1:])/2,(y[:-1]+y[1:])/2,color = 'orange',edgecolor = '
        # create the bar chart
        plt.plot(x,y, color = 'blue')
        # draw the orignal line
    return np.sum(elements)
    # return the sum of elemtns whcih is the numerical result

print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(0,0))/np.pi)
# this and following print out the function and required intergal as asked
scipy.special.jv(0,0)
```

```
0.9999999999999999
```

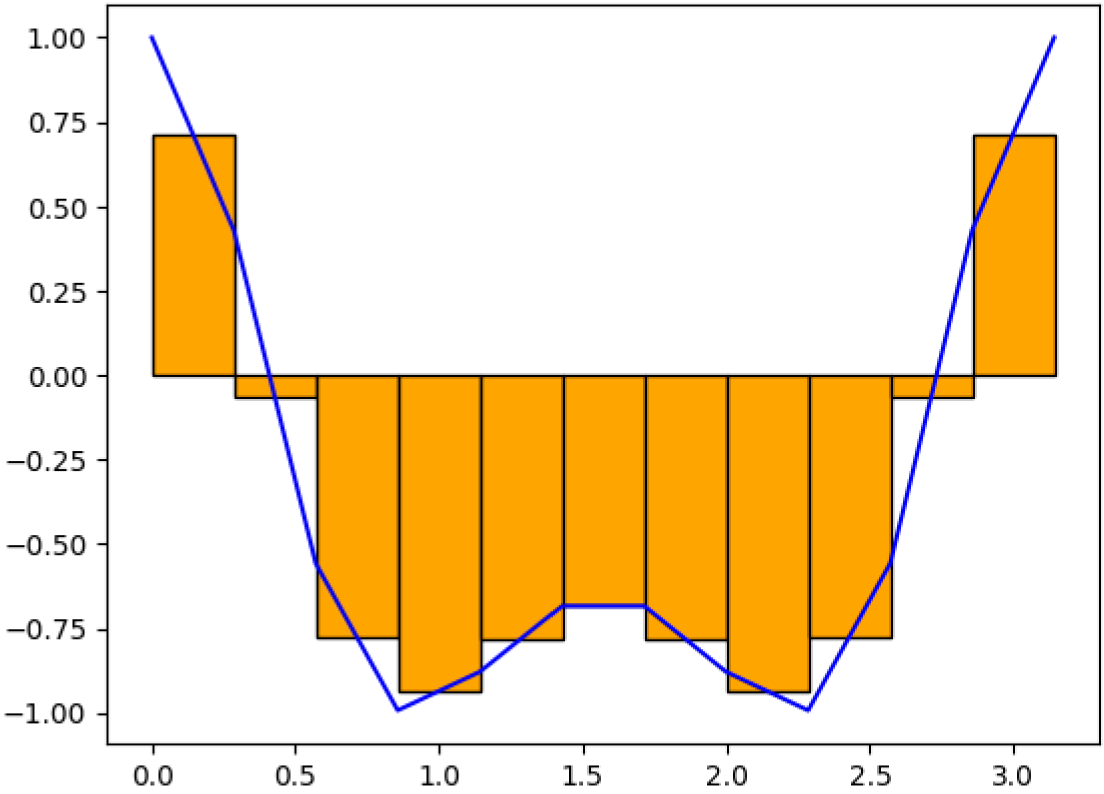
Out[ ]: 1.0



In [ ]:
```
print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(0,4))/np.pi)
scipy.special.jv(0,4)
```

```
-0.3971498098638411
```

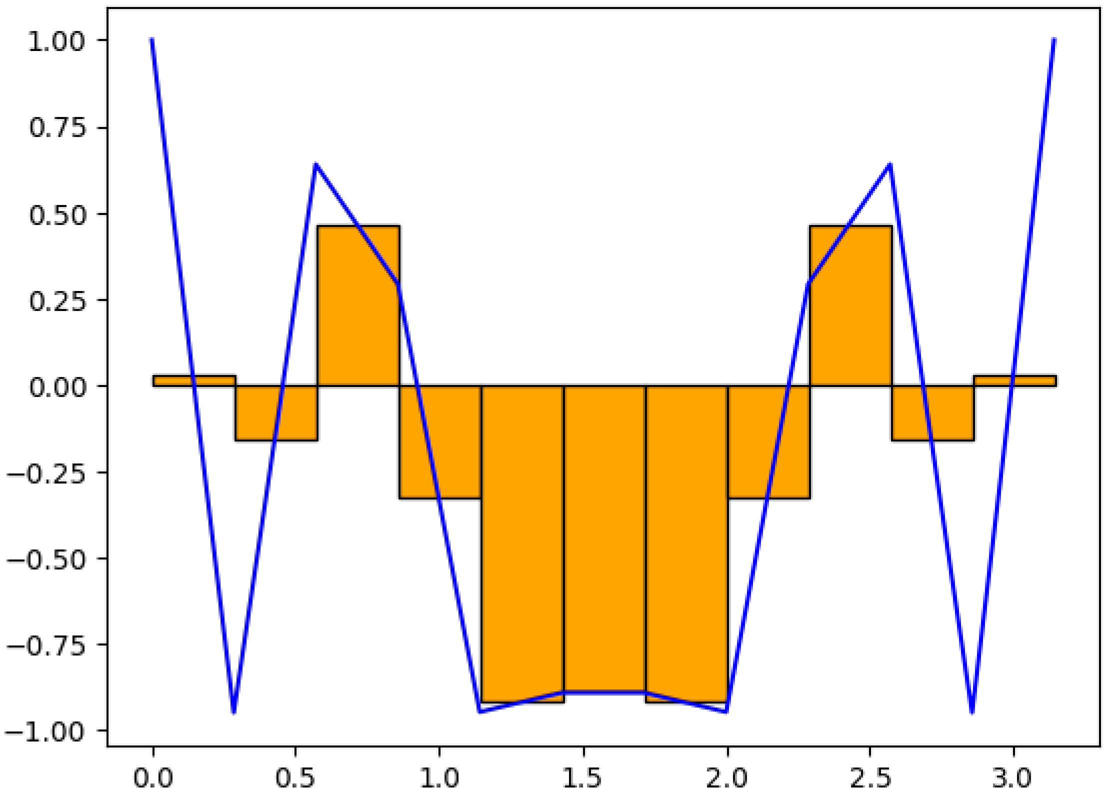
Out[ ]: -0.39714980986384746



In [ ]:
```
print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(0,10))/np.pi)
scipy.special.jv(0,10)
```

```
-0.24593437071432359
```

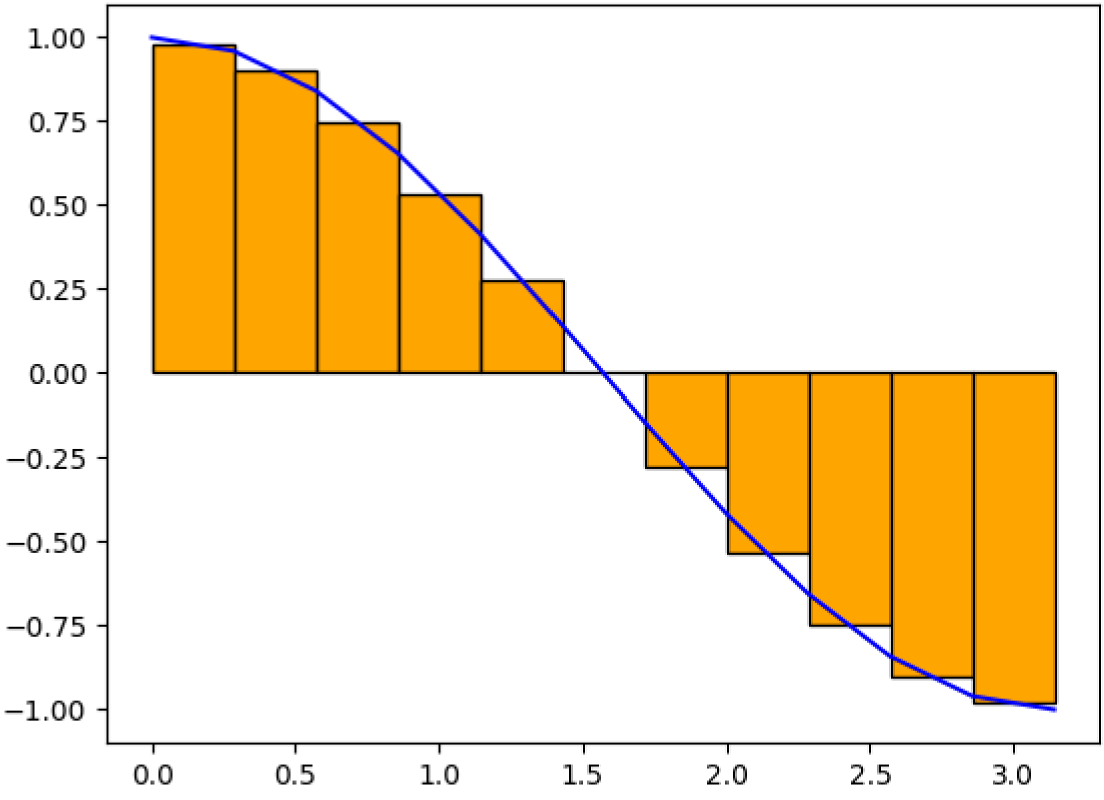
Out[ ]:
```
-0.24593576445134832
```



In [ ]:
```
print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(1,0))/np.pi)
scipy.special.jv(1,0)
```

```
3.533949646070574e-17
```

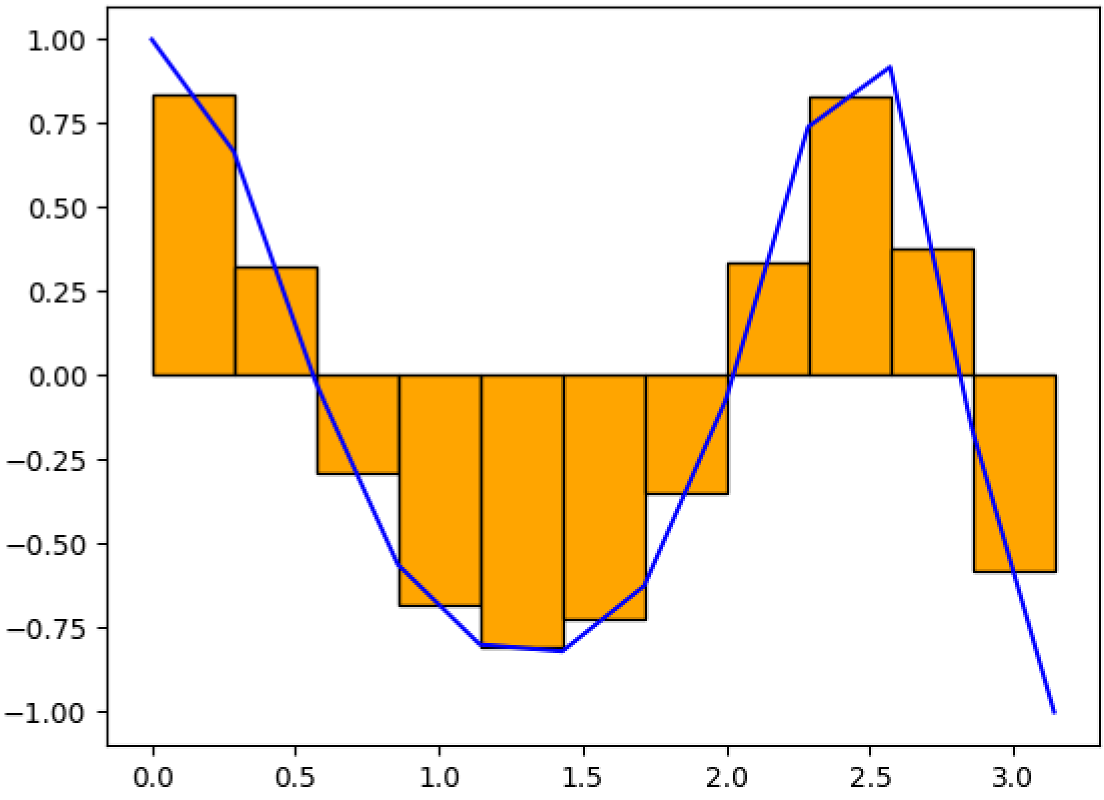
Out[ ]: 0.0



In [ ]:
```
print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(1,4))/np.pi)
scipy.special.jv(1,4)
```

```
-0.06604332802358304
```

Out[ ]: -0.06604332802354924

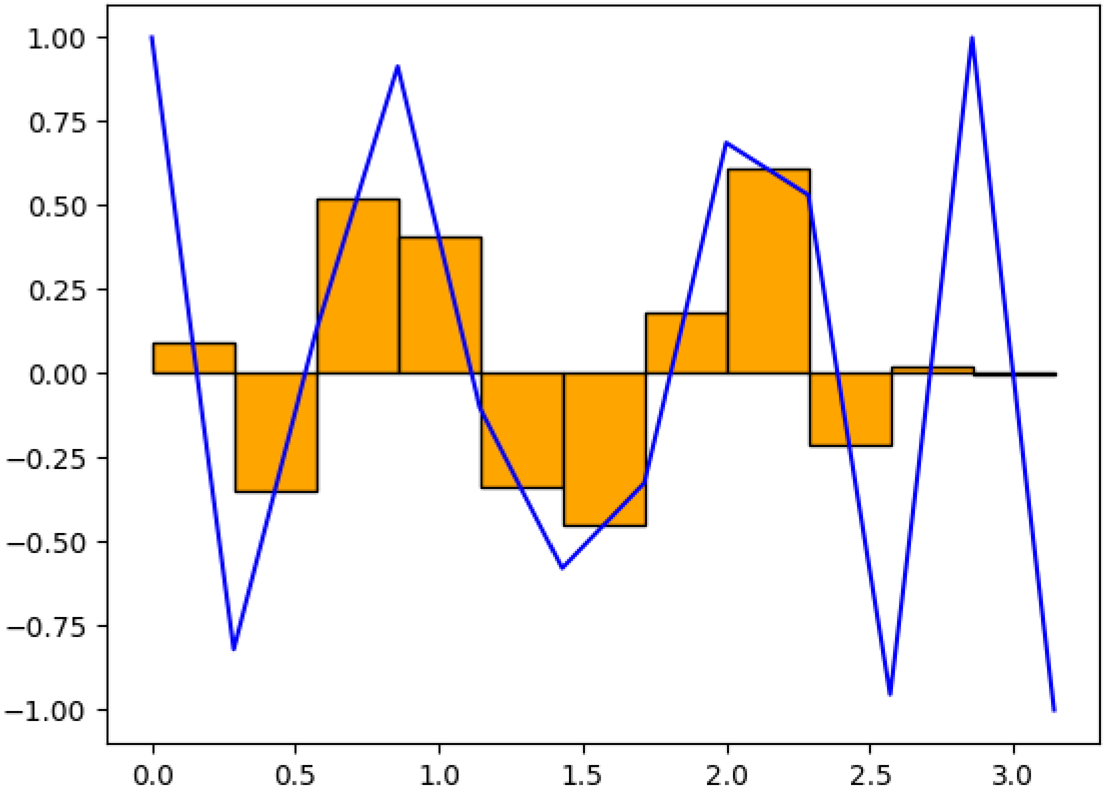


In [ ]:
```
print(trapezoidal_rule(bessel_integrand,0,np.pi,12,(1,10))/np.pi)
scipy.special.jv(1,10)
```

```
0.04346999799138204
```

Out[ ]:

```
0.0434727461688616
```



In [ ]:

```
# c)
import scipy.integrate

def bessel_jn(n, x):
    """
    Returns the Bessel function J_n(x) computed from the scipy.integrate.quad fu

    Parameters
    ----------
    n : int
        The order of the Bessel function.
    x : float
        The argument of the Bessel function.

    Returns
    -------
    J_n(x) : array_like
        The value of the Bessel function at the given x and n.
    """
    return (scipy.integrate.quad(bessel_integrand,0,np.pi,(n,x)))[0]/(np.pi)
    # calculate the result as asked
```

In [ ]:

```
# d)
import scipy.special

x = np.linspace(0,10,100)
# create a linespace from 0 to 10 with 100 stpes
bessel_j0_quad = np.zeros(100)
for i in range(100):
    bessel_j0_quad[i] = bessel_jn(0,x[i])
# get the array of j0 asked
epsilon_quad = abs(bessel_j0_quad - scipy.special.jv(0,x))
```

```
# get the error as asked
print(np.min(epsilon_quad))
print(np.max(epsilon_quad))
# print the value as asked
```

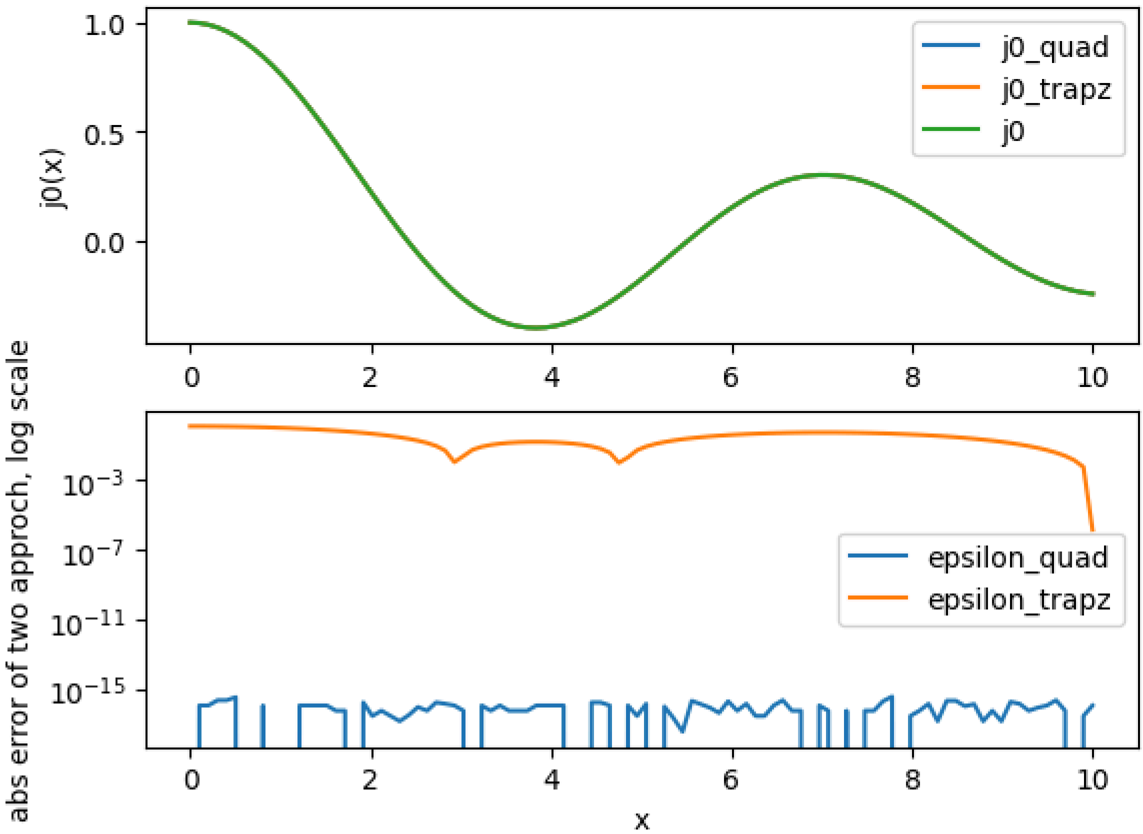
```
0.0
3.608224830031759e-16
```

In [ ]:
```
# e)

bessel_j0_trapz = np.zeros(100)
for i in range(100):
    bessel_j0_trapz[i] = trapezoidal_rule(bessel_integrand,0,np.pi,12,(0,x[i]),F
# general the other j0 as asked, the x is generated in last question
epsilon_trapz = abs(bessel_j0_trapz - scipy.special.jv(0,x[i]))
# calculate the error
print(np.min(epsilon_trapz))
print(np.max(epsilon_trapz))
# print the result
```

```
1.3937370247352199e-06
1.2459357644513482
```

In [ ]:
```
# f)

plt.subplot(2, 1, 1)
# create the upper plot
plt.plot(x, bessel_j0_quad,label = 'j0_quad')
plt.plot(x,bessel_j0_trapz,label = 'j0_trapz')
plt.plot(x,scipy.special.jv(0,x),label = 'j0')
# plot all three lines as asked
plt.ylabel('j0(x)')
# give the y a label of j0(x)
plt.legend()
# provide the legend

plt.subplot(2,1,2)
# create the lower plot
plt.plot(x,epsilon_quad,label = 'epsilon_quad')
plt.plot(x,epsilon_trapz,label = 'epsilon_trapz')
# plot the lines as asked
plt.yscale('log')
# make y to log scale
plt.ylabel('abs error of two approch, log scale')
# give y a label
plt.xlabel('x')
# give e a label
plt.legend()
# show legend
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x1d746998e90>
```



f)

I am kind of surprised by how accurate we get. The Trajpz method get accurate is becaus it is choosing the mean of two nearby values. I used to think it will be larger than 5 is becuase the line we get is so not like the acutal curve.

Problem 2

a)

From last week's homework, we can find out that there is and only one max on the whole plot. Thus the ν_{max} occurs at the $\frac{dF(\nu)}{d\nu} = 0$

Sine we know $F(\nu) = \frac{2\pi h\nu^3}{c^2}\frac{1}{\exp(h\nu/k_\mathrm{B}T)-1}$

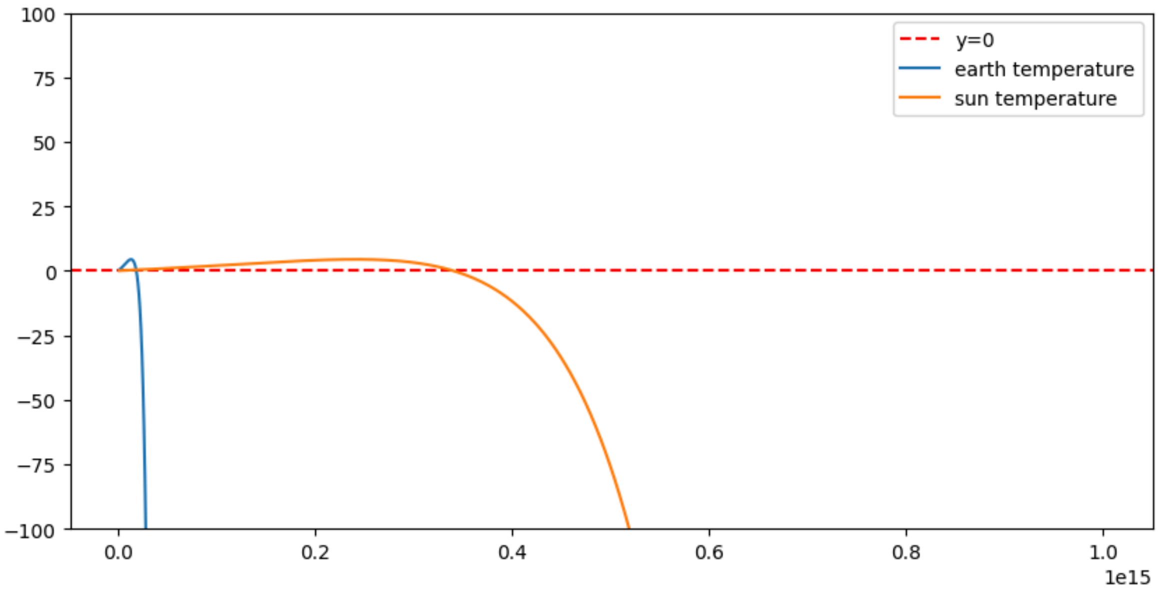
$$\frac{dF(\nu)}{d\nu} = \frac{6\pi h\nu^2}{c^2}\frac{1}{\exp(h\nu/k_\mathrm{B}T)-1} + \frac{2\pi h\nu^3}{c^2}\frac{-1}{(\exp(h\nu/k_\mathrm{B}T)-1)^2}h/k_\mathrm{B}T\exp($$

since we know that at $\nu = 0$ the function do not exist, the exsit value is when $3\exp(h\nu/k_\mathrm{B}T) - 3 - h\nu/k_\mathrm{B}T\exp(h\nu/k_\mathrm{B}T) = 0$, which is $\left(3 - \frac{h\nu}{k_\mathrm{B}T}\right)\exp\left(\frac{h\nu}{k_\mathrm{B}T}\right) - 3 = 0$ and the solution of ν is ν_max

In [ ]:
```
# b)

def Fprime(v,T):
    # define the function
```

```
    h = 6.62607015e-34
    kb = 1.380649e-23
    # set the constants
    b = (h*v)/(kb*T)
    return (3-b)*np.exp(b)-3
    # return the value of the function

x = np.linspace(1e12,1e15,1000)
# create the points of x
y_earth = np.zeros(1000)
y_sun = np.zeros(1000)
# set up for y
for i in range(1000):
    y_earth[i] = Fprime(x[i],310.15)
    y_sun[i] = Fprime(x[i],5778)
    # calculate the values of all y
plt.figure().set_figwidth(10)
# make the figure wider, so that it is easer to read the value
plt.axhline(y=0,c = 'red',linestyle = '--',label = 'y=0')
plt.plot(x,y_earth,label = 'earth temperature')
plt.plot(x,y_sun,label = 'sun temperature')
# plot all the lines
plt.ylim(-100,100)
# set y to a range close to 0, such that it is easier to find 0
plt.legend()
# show the legend
```

Out[ ]: <matplotlib.legend.Legend at 0x1d74b4ca2d0>



I guess the value should be 0.005e15 for earth temperature, and 0.38e15 for sun temperature.

In [ ]:
```
# c)

import scipy.optimize


print(scipy.optimize.bisect(Fprime,1e13,1e15,(310.15)))
print(scipy.optimize.bisect(Fprime,1e13,1e15,(5778)))
# use the function asked to solve for v max
```

```
18233488237341.637
339684330276833.44
```

In [ ]:
```
# d)

print(scipy.optimize.newton(Fprime,5e13,args=(310.15,)))
print(scipy.optimize.newton(Fprime,4e14,args=(5778,)))
# use the function asked to solve for v max
```

```
18233488237341.625
339684330276833.56
```

In [ ]:
```
# e)

print(scipy.optimize.root_scalar(Fprime,(310.15),'bisect',(1e13,1e15)))
print(scipy.optimize.root_scalar(Fprime,(5778),'bisect',(1e13,1e15)))
# use the function asked to solve for v max
```

```
      converged: True
           flag: converged
 function_calls: 58
     iterations: 56
           root: 18233488237341.637
         method: bisect
      converged: True
           flag: converged
 function_calls: 54
     iterations: 52
           root: 339684330276833.44
         method: bisect
```

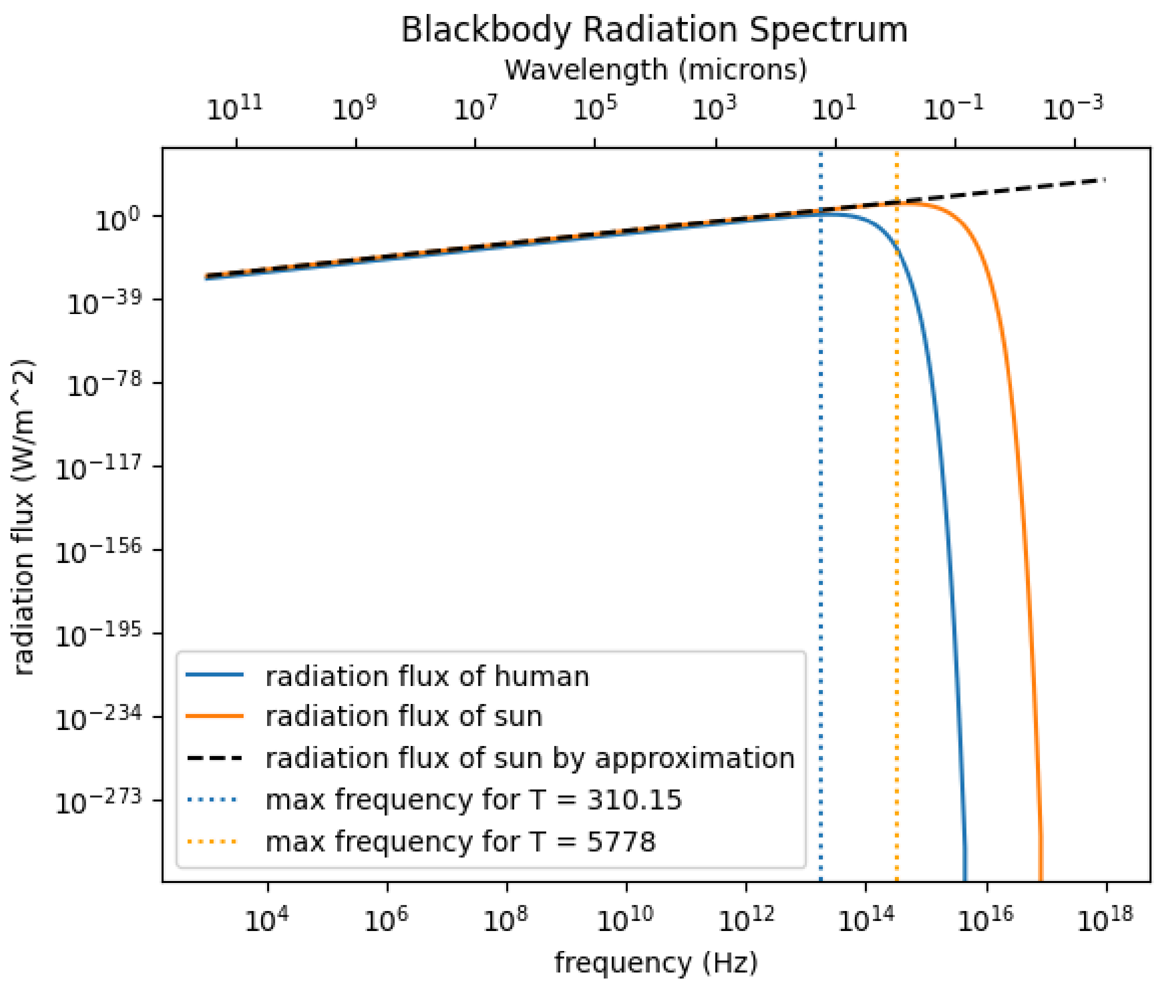
In [ ]:
```
# f)

def F_function(frequency,temperature):
    # the function of the blackbody radiation
    c = 299792458
    kb = 1.380649e-23
    h = 6.62607015e-34
    # set constant
    return 2*np.pi*h*(frequency**3)/(c**2)*1/(np.exp(h*frequency/(kb*temperature
    # return the fucntion

def Approximate_F(frequency,temperature):
    # the function of the approximation of blackbody raidtion
    c = 299792458
    kb = 1.380649e-23
    # set constant
    return 2*np.pi*(frequency**2)*kb*temperature/(c**2)
    # return the value of the function

def flux_F(func,temperature,frequency_low=1e3,frequency_high=1e18,n=5000):
    # the function to calculate the flux of function put into with given tempera
    # the strating point and end point of frequcy is chosen by testing
    dv = np.logspace(np.log10(frequency_low),np.log10(frequency_high),n)
    # generate all the points which is even distributed on a log graph
    flux = np.zeros(n-1)
    # set the array to store the flux
    for i in range(n-1):
        # gonging over the series to calculate the flux
        flux[i] = (func(dv[i],temperature) + func(dv[i+1],temperature))*(dv[i+1]
        # since the flux is denfied as an integral of the function, thus, I am u
```

```
    return flux,dv
    # return the flux and the point of x

line_human,x = flux_F(F_function,310.15)
line_sun,x = flux_F(F_function,5778)
line_sun_approxiamtion,x = flux_F(Approximate_F,5778)
# get the x,and y of the plot, but since we are using the same range and points

plt.plot(x[:-1],line_human,label = 'radiation flux of human')
plt.plot(x[:-1],line_sun, label = 'radiation flux of sun')
plt.plot(x[:-1],line_sun_approxiamtion,'k--',label = 'radiation flux of sun by a
# plot the line as asked

plt.axvline(scipy.optimize.bisect(Fprime,1e13,1e15,(310.15)),1e-273,10,linestyle
plt.axvline(scipy.optimize.bisect(Fprime,1e13,1e15,(5778)),1e-273,10,linestyle =
# draw the vertical max frequecy line

plt.xscale('log')
plt.yscale('log')
# change x and y to the log scale

plt.xlabel('frequency (Hz)')
plt.ylabel('radiation flux (W/m^2)')
plt.title('Blackbody Radiation Spectrum')
plt.gca().secondary_xaxis('top', functions=(lambda nu: 299792458 / nu / 1e-6, la
# show the notes asked to show

plt.legend()
# print the legend
```

```
C:\Users\botao\AppData\Local\Temp\ipykernel_17268\2838333906.py:9: RuntimeWarnin
g: overflow encountered in exp
  return 2*np.pi*h*(frequency**3)/(c**2)*1/(np.exp(h*frequency/(kb*temperature))-
1)
```

Out[ ]: `<matplotlib.legend.Legend at 0x1d74d98cc50>`

```
C:\Users\botao\AppData\Local\Temp\ipykernel_17268\2838333906.py:54: RuntimeWarnin
g: divide by zero encountered in divide
  plt.gca().secondary_xaxis('top', functions=(lambda nu: 299792458 / nu / 1e-6, l
ambda lam: 299792458 / (lam * 1e-6))).set_xlabel('Wavelength (microns)')
```


Blackbody Radiation Spectrum
Wavelength (microns)
10^{11} 10^{9} 10^{7} 10^{5} 10^{3} 10^{1} 10^{-1} 10^{-3}
radiation flux (W/m^2)
10^{0} 10^{-39} 10^{-78} 10^{-117} 10^{-156} 10^{-195} 10^{-234} 10^{-273}
radiation flux of human
radiation flux of sun
radiation flux of sun by approximation
max frequency for T = 310.15
max frequency for T = 5778
10^{4} 10^{6} 10^{8} 10^{10} 10^{12} 10^{14} 10^{16} 10^{18}
frequency (Hz)