EP3_BrunoYamada

December 10, 2016

0.1 Exercício 1:

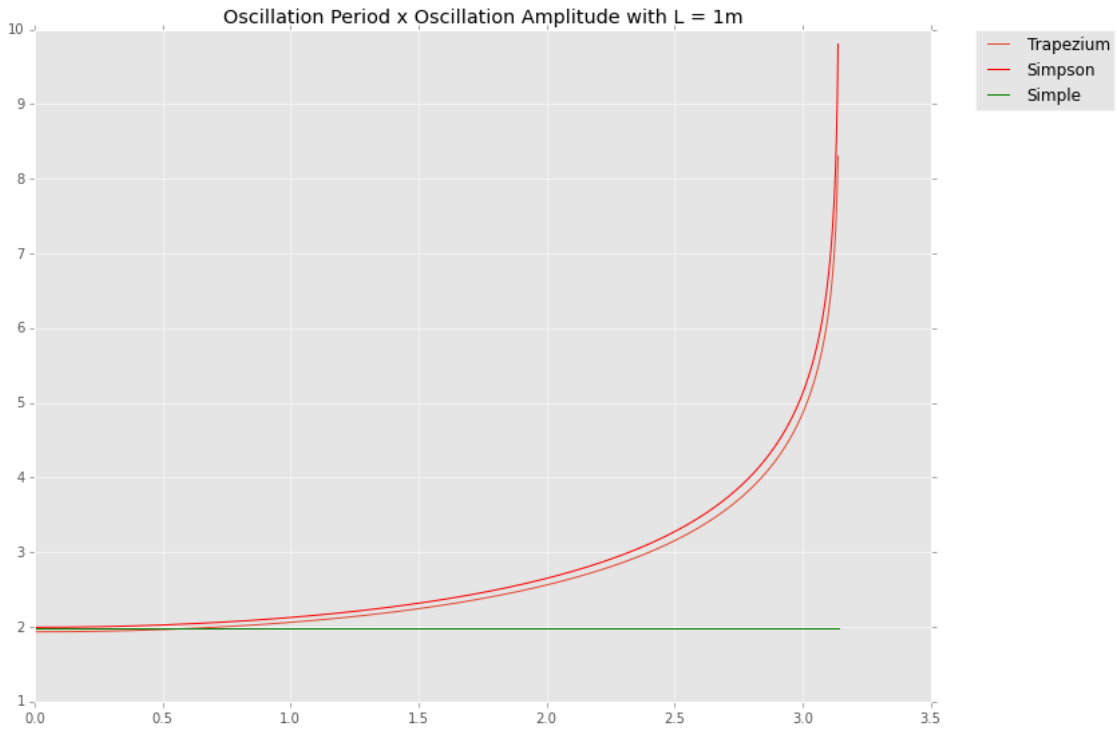
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
plt.style.use("ggplot")
import scipy.integrate as integrate
%matplotlib inline
def trapezio(f, h):
    sum = 0
    f[0] = (1.0*f[0])/2
    f[-1] = (1.0*f[-1])/2
    for i in f:
        sum += i
    return (h * sum)

def simpson(f, h):
    sum = 0
    for i in range(0, len(f)):
        if i != 0 and i != (len(f) - 1):
            if i%2 == 0:
                sum += (f[i]*1.0)/2
                #print 2
            else:
                sum += (f[i]*1.0)/4
                #print 4
        else:
            sum += f[i]
            #print 1
        #print
    return (h * sum * 8)/3

h = np.pi/1001
f = []
fe = []
x = []
for i in range(0, 1001):
    f.append(np.sin(i*h))
```

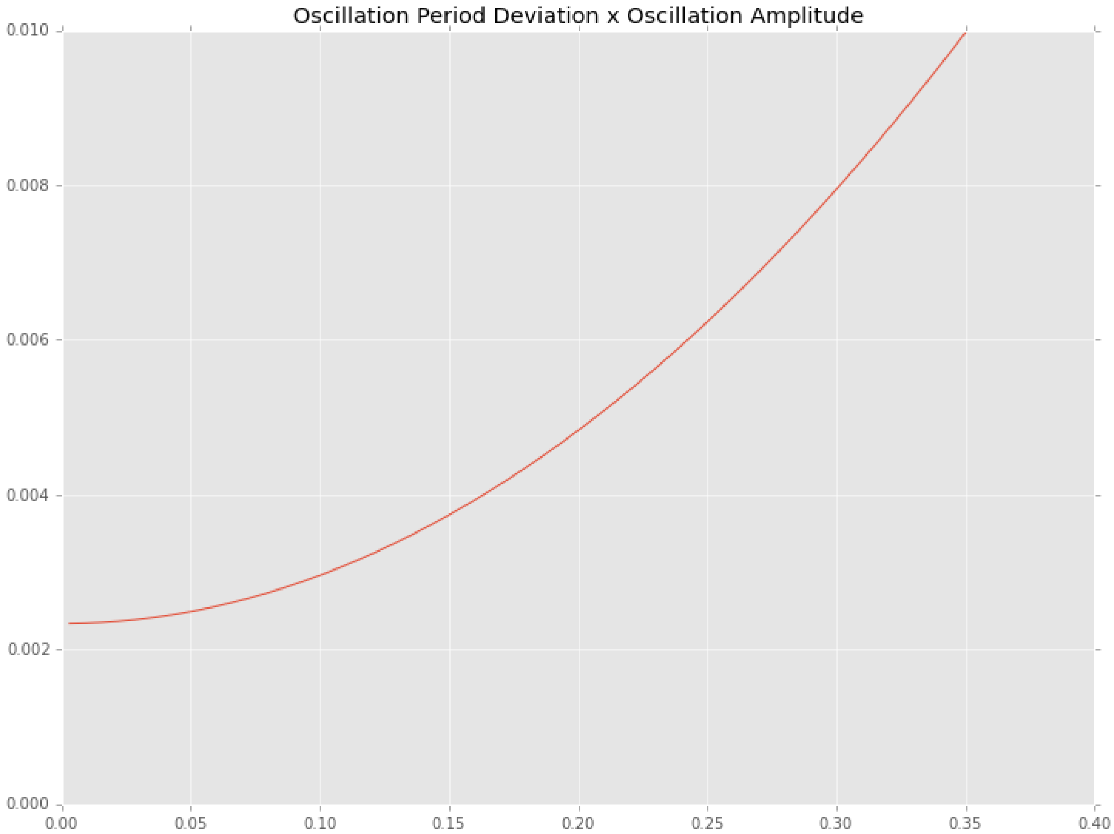
```
        fe.append(np.exp(i*h))
        x.append(i*h)
```

```
In [2]: def getT(l, g):
            return 2*np.pi*np.sqrt((1.0*l)/g)

        def getTInt1(l, g, teta):
            f = []
            x = []
            h = (1.0 * teta)/1001
            for i in range(0, 1001):
                f.append(1/np.sqrt(abs(np.cos(i*h) - np.cos(teta))))
                x.append(i*h)
            tmp = trapezio(f, h)
            #print "tmp = ", tmp
            #tmp = integrate.trapz(f, x)
            return 4*np.sqrt((1.0*l)/(2*g))*tmp

        def getTInt2(l, g, teta):
            f = []
            x = []
            h = (1.0 * teta)/1001
            for i in range(0, 1001):
                f.append(1/np.sqrt(abs(np.cos(i*h) - np.cos(teta))))
                x.append(i*h)
            tmp = simpson(f, h)
            #print "tmp = ", tmp
            #tmp = integrate.trapz(f, x)
            return 4*np.sqrt((1.0*l)/(2*g))*tmp
```

```
In [11]: #generate values for graphs
         #T in function of teta:
         div = 1001
         h = np.pi/(div)
         T = []
         TInt1 = []
         TInt2 = []
         deviation = []
         x = []
         for i in range(1, div):
             T.append(getT(1, 9.98))
             TInt1.append(getTInt1(1, 9.98, i*h))
             TInt2.append(getTInt2(1, 9.98, i*h))
             deviation.append((TInt2[-1] - T[-1])/(TInt2[-1]))
             x.append(i*h)
```

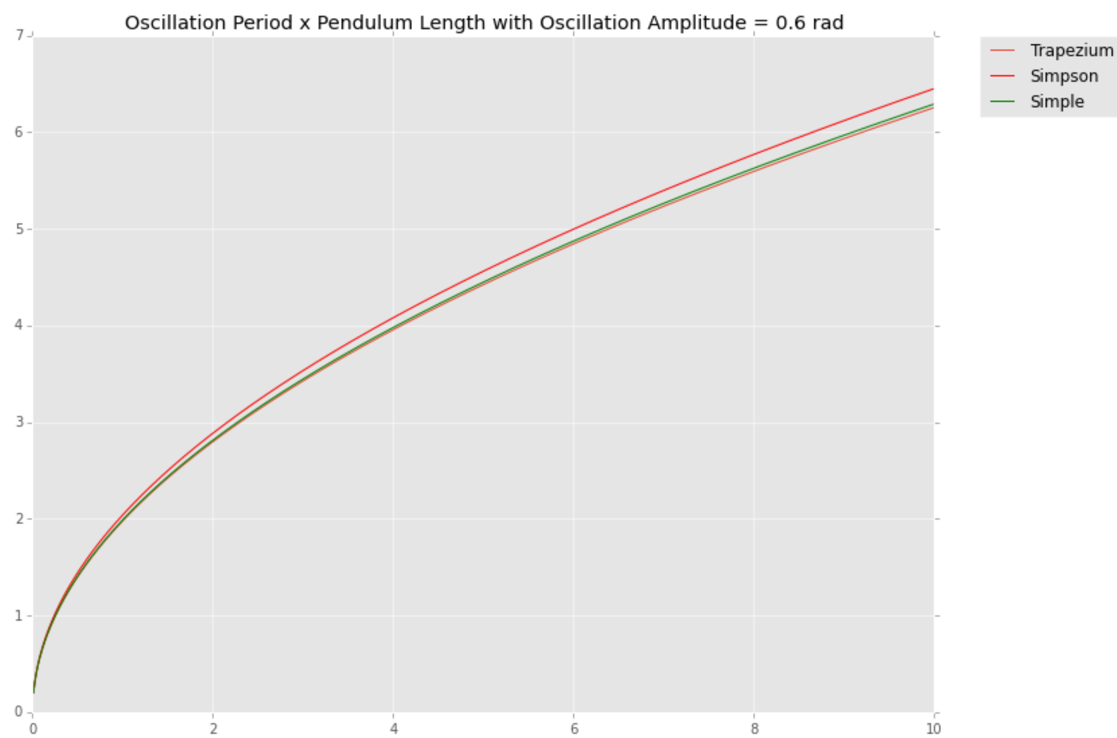
```python
#plt.figure(1)
p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolo
plt.plot(x, TInt1, label='Trapezium')
plt.plot(x, TInt2, color = 'r', label='Simpson', )
plt.plot(x, T, color = 'g', label='Simple')
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
plt.title("Oscillation Period x Oscillation Amplitude with L = 1m")
plt.show(p)

p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolo
plt.plot(x, deviation)
plt.xlim(0,0.4)
plt.ylim(0, 0.01)
plt.title("Oscillation Period Deviation x Oscillation Amplitude")
plt.show(p)
```

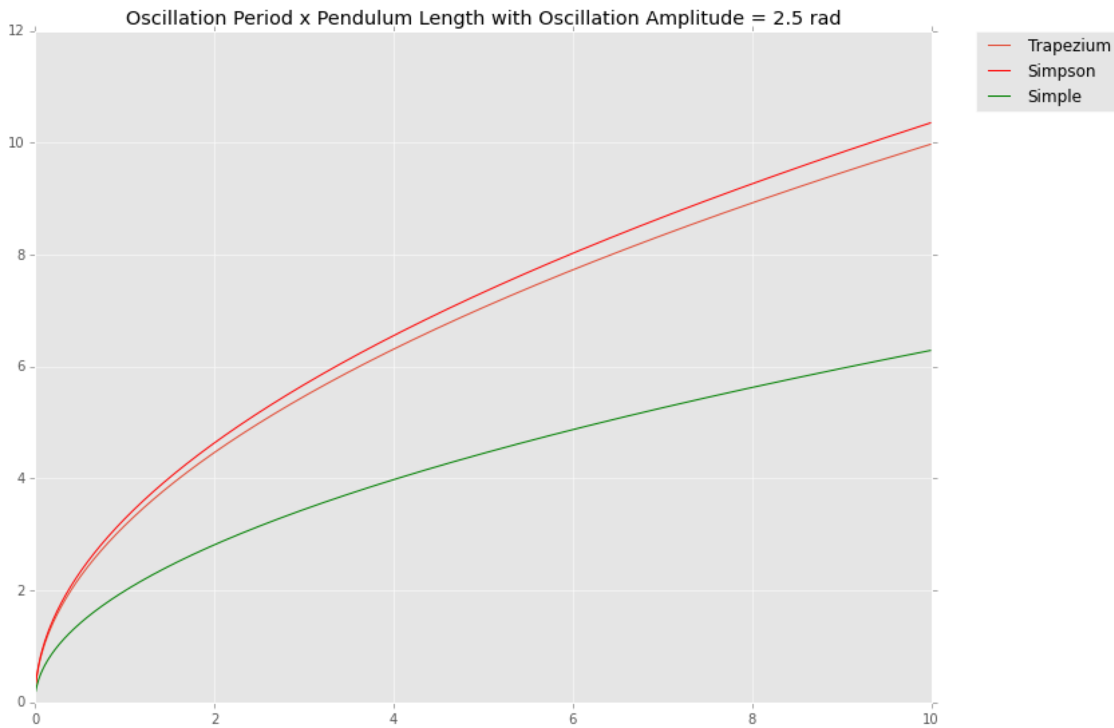




In [4]:
```
#T in function of l
#teta = 0.6 rad
div = 1000
h = 10.0/(div)
T = []
TInt1 = []
TInt2 = []
x = []

for i in range(1, div):
    T.append(getT(i*h, 9.98))
    TInt1.append(getTInt1(i*h, 9.98, 0.6))
    TInt2.append(getTInt2(i*h, 9.98, 0.6))
    x.append(i*h)

p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolor
plt.plot(x, TInt1, label='Trapezium')
plt.plot(x, TInt2, color = 'r', label='Simpson', )
plt.plot(x, T, color = 'g', label='Simple')
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
plt.title("Oscillation Period x Pendulum Length with Oscillation Amplitude
```

```
plt.show(p)

#teta = 2.5 rad
div = 1000
h = 10.0/(div)
T = []
TInt1 = []
TInt2 = []
x = []
for i in range(1, div):
    T.append(getT(i*h, 9.98))
    TInt1.append(getTInt1(i*h, 9.98, 2.5))
    TInt2.append(getTInt2(i*h, 9.98, 2.5))
    x.append(i*h)
p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolor
plt.plot(x, TInt1, label='Trapezium')
plt.plot(x, TInt2, color = 'r', label='Simpson', )
plt.plot(x, T, color = 'g', label='Simple')
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
plt.title("Oscillation Period x Pendulum Length with Oscillation Amplitude
plt.show(p)
```





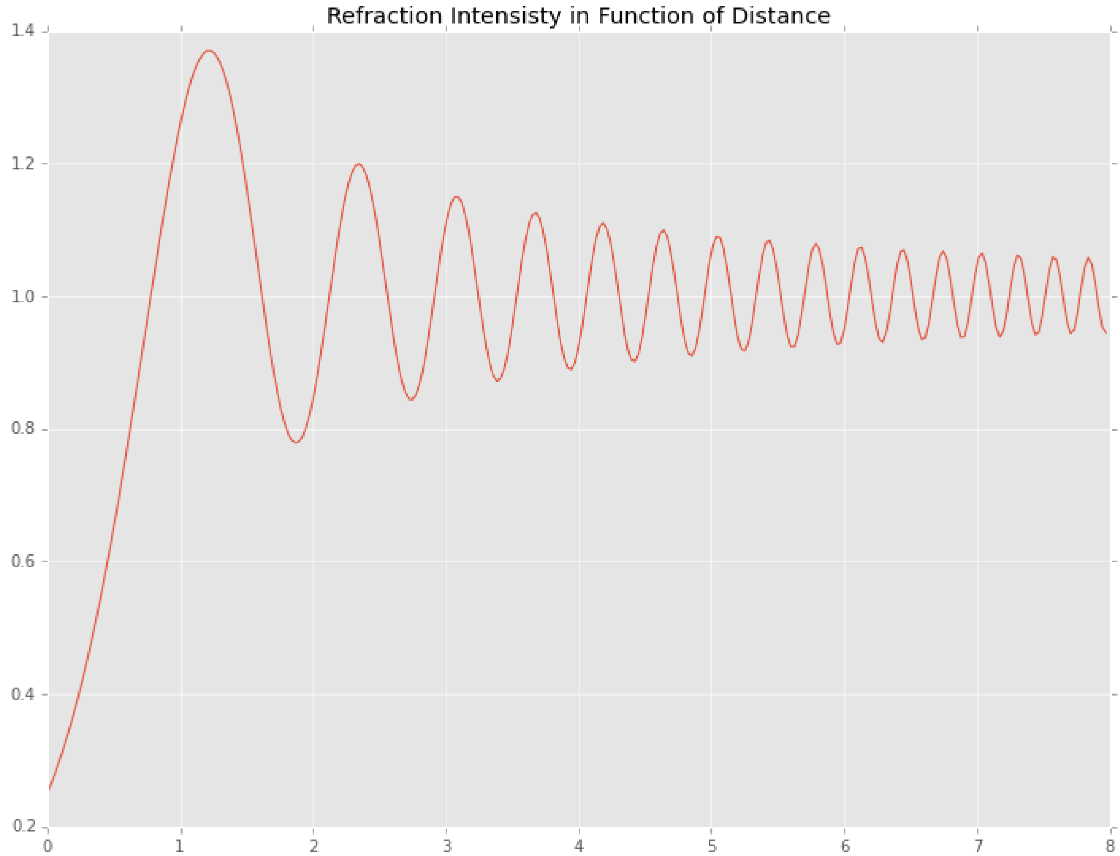
0.2 Exercício 2:

```
In [5]: def Romberg(f, a, b, N, M):
            def h(n):
                return (b-a)/(pow(2, n)*1.0)

            def soma(n):
                ans = 0
                for k in range(1, pow(2, n-1) + 1):
                    ans += f( a + (2*k-1)*h(n) )
                return ans

            def R(n, m):
                maps = np.zeros([n+1,m+1])
                values = np.empty([n+1, m+1])
                if(maps[n][m] != 0):
                    return values[n][m]

                if(n == 0 and m == 0):
                    ans = h(1)*(f(a)+f(b))

                elif(m == 0):
                    ans = (R(n-1, 0)/2.0 + h(n)*soma(n) )
```

```
        else:
            ans = 1.0/(pow(4, m)-1) * (pow(4, m) * R(n, m-1) - R(n-1, m-1))

        if(maps[n][m] == 0):
            maps[n][m] = 1
            values[n][m] = ans
        return ans

    return R(N, M)
```

In [6]:

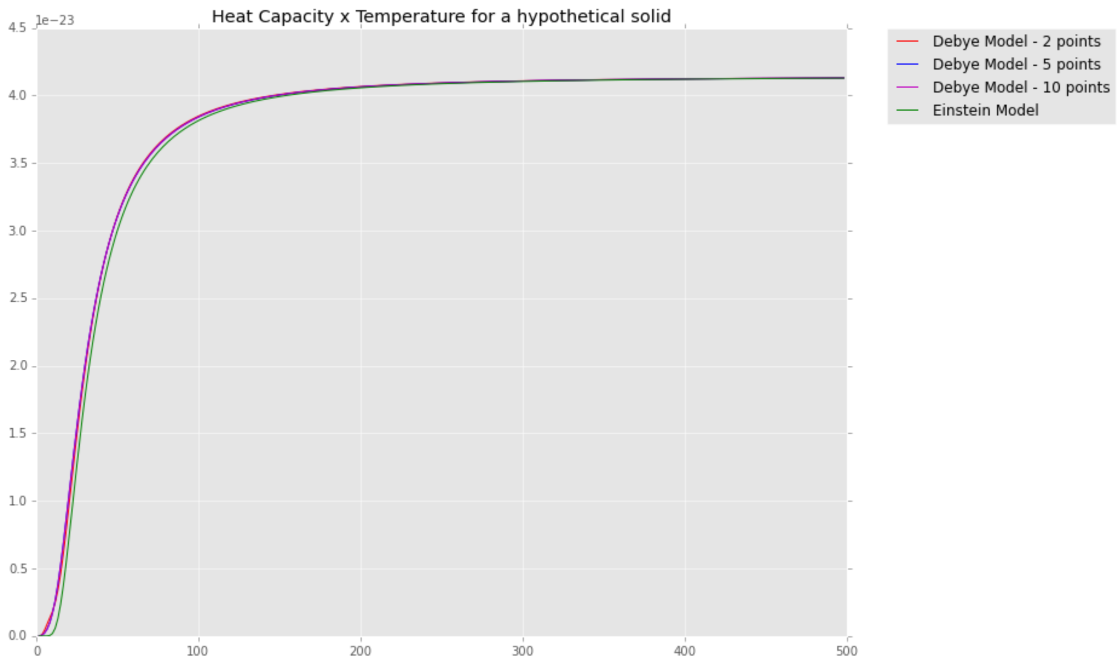
```
def c(x):
    return np.cos((np.pi * (x**2))/2.0)
def s(x):
    return np.sin((np.pi * (x**2))/2.0)

def C(x):
    #return integrate.romberg(c, 0, x, divmax = 30)
    #return integrate.quadrature(c, 0, x, maxiter = 1000)[0]
    return Romberg(c, 0, x, 10, 3)
def S(x):
    #return integrate.romberg(s, 0, x, divmax = 30)
    #return integrate.quadrature(s, 0, x, maxiter = 1000)[0]
    return Romberg(s, 0, x, 10, 3)

def I(x, I):
    return (I/2.0)*(((C(x)+0.5)**2) + ((S(x)+0.5)**2))

#plot for I = 1 xMax = 10
div = 300
h = 8.0/div
x = []
y = []
for k in range(0, div):
    x.append(k*h)
    y.append(I(k*h, 1))
p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolor
plt.plot(x, y)
plt.title("Refraction Intensisty in Function of Distance")
plt.show(p)
```

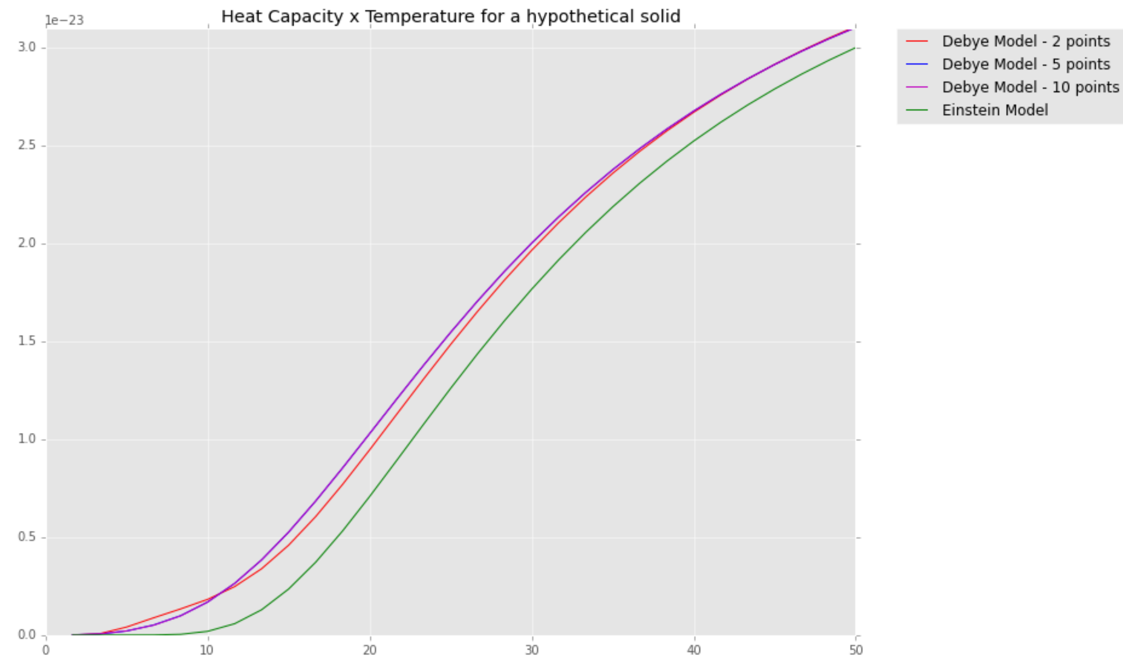


0.3 Exercício 3

```
In [7]: def CEin(t, n = 1, te = 100, k = 1.38064852e-23):
            return 3 * n * k * (((1.0*te)/t)**2) * ((np.exp((te*1.0)/t))/(np.exp((t

        def gTmp(x):
            return ((x**4)*np.exp(x)*1.0)/((np.exp(x)-1)**2)

        def g(x, n = 1, td = 124.07, k = 1.38064852e-23, points = 2):
            return 9*n*k*(1.0/(((1.0*td)/x)**3))*integrate.fixed_quad(gTmp, 0, ((1.

        div = 300
        h = 500.0/div
        x = []
        y2 = []
        y5 = []
        y10 = []
        yEin = []
        for k in range(1, div):
            x.append(k*h)
```

```
    y2.append(g(k*h))
    y5.append(g(k*h, points = 5))
    y10.append(g(k*h, points =10))
    yEin.append(CEin(k*h))
p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolor
plt.plot(x, y2, color="r", label="Debye Model - 2 points")
plt.plot(x, y5, color="b", label="Debye Model - 5 points")
plt.plot(x, y10, color="m", label = "Debye Model - 10 points")
plt.plot(x, yEin, color="g", label="Einstein Model")
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
plt.title("Heat Capacity x Temperature for a hypothetical solid")
plt.show()

p = plt.figure(num=None, figsize=(12, 9), dpi=160, facecolor='w', edgecolor
plt.plot(x, y2, color="r", label="Debye Model - 2 points")
plt.plot(x, y5, color="b", label="Debye Model - 5 points")
plt.plot(x, y10, color="m", label = "Debye Model - 10 points")
plt.plot(x, yEin, color="g", label="Einstein Model")
plt.xlim(0, 50)
plt.ylim(0, 3.1e-23)
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
plt.title("Heat Capacity x Temperature for a hypothetical solid")
plt.show()
```




Heat Capacity x Temperature for a hypothetical solid
1e−23
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
Debye Model - 2 points
Debye Model - 5 points
Debye Model - 10 points
Einstein Model

```
In [ ]:
```