Solución Asignación 4
FISI 6510

Guillermo Fidalgo

2 de marzo de 2021

Heat capacity of a solid: Debye's theory of solids gives the heat capacity of a solid at temperature T to be

$$C_V = 9V\rho k_B \left(\frac{T}{\theta_D}\right)^3 \int_0^{\theta_D/T} \frac{x^4 e^x}{(e^x - 1)^2}\, dx,$$

where V is the volume of the solid, ρ is the number density of atoms, k_B is Boltzmann's constant, and θ_D is the so-called *Debye temperature*, a property of solids that depends on their density and speed of sound.

(a) Write a Python function `cv(T)` that calculates C_V for a given value of the temperature, for a sample consisting of 1000 cubic centimeters of solid aluminum, which has a number density of $\rho = 6.022 \times 10^{28}\,\mathrm{m}^{-3}$ and a Debye temperature of $\theta_D = 428\,\mathrm{K}$. Use the trapezoidal rule to evaluate the integral with $N = 1000$ sample points. Hint: The value of the integrand at $x = 0$ is zero.

(b) Use your function to make a graph of the heat capacity as a function of temperature from $T = 5\,\mathrm{K}$ to $T = 500\,\mathrm{K}$.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
# plt.rcParams.update({
#     "text.usetex": True,
#     "font.family": "sans-serif",
#     "font.sans-serif": ["Helvetica"]})
# for Palatino and other serif fonts use:
plt.rcParams.update({
    "text.usetex": True,
    "font.family": "serif",
    "font.serif": ["Palatino"],
})
```

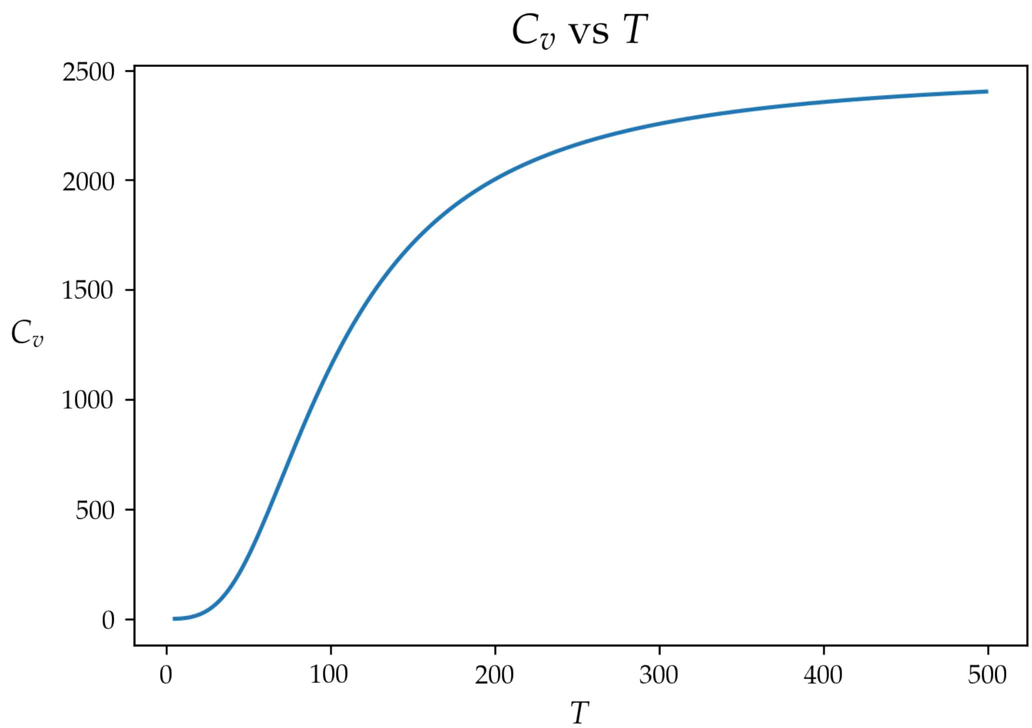
a)

[2]:
```
k_B=1.38064852e-23
def f(x):
    func=x**4*np.exp(x)/(np.exp(x)-1)**2
    return func



def cv(T,V=0.001,rho=6.022e28,theta_D=428,N=1000):
    constantes=9*V*rho*k_B*(T/theta_D)**3
    a=0
    b=theta_D/T
    h=(b-a)/N
    fa=0
    fb=.5*f(b)
    integral=0
    for k in range(1,N):
        integral=f(a+k*h)+integral
    integral=integral+ .5*fa + .5*fb
    C_v=constantes*integral*h
    return C_v
```

b)

[3]:
```
T=np.arange(5,500)
Cv=[cv(x) for x in T]
```

[5]:
```
plt.figure(dpi=250)
plt.plot(T,Cv)
plt.title(r"$C_v$ vs $T$",size=16)
plt.xlabel("$T$",size=12)
# plt.yscale("log")
plt.ylabel("$C_v$",size=12,rotation=0,labelpad=13)
# plt.savefig("figure1.png",dpi=250)
plt.show()
```


C_v vs T
C_v
T
2500
2000
1500
1000
500
0
0
100
200
300
400
500

Simpson's rule:

(a) Write a program to calculate an value for the integral $\int_0^2 (x^4 - 2x + 1)\,dx$ from Example 5.1, but using Simpson's rule with ten slices instead of the trapezoidal rule.

(b) Run the program and compare your result to the known correct value of 4.4. What is the fractional error on your calculation?

(c) Modify the program to use a hundred slices instead, then a thousand. Note the improvement in the result. How do the results compare with those from Example 5.1 for the trapezoidal rule with the same number of slices?

section*{a)}

[5]:
```
def f(x):
    func=x**4-2*x+1
    return func

def simp(func,N=10,a=0,b=2):
    h=(b-a)/N
    fa=f(a)
    fb=f(b)
    integral=0
    for k in range(1,N,2):
        integral=4*f(a+k*h)+integral
    for k in range(2,N-1,2):
        integral=2*f(a+k*h)+integral

    integral=(1/3)*h*(integral+fa+fb)
    I=integral
    return I
```

b)

[6]:
```
I_simp=simp(f)

error= (I_simp-4.4)/4.4
print("El error fraccionario es de :",error)
print("En porciento es de {:.4%}".format(error))
```

```
El error fraccionario es de : 9.696969696972666e-05
En porciento es de 0.0097%
```

c)

[7]:
```
I_simp100= simp(f,N=100)
I_simp1000= simp(f,N=1000)
```

[9]:
```
print(I_simp100,I_simp1000)
error100= (I_simp100-4.4)/4.4
error1000=(I_simp1000-4.4)/4.4
print("Sus errores son de {:.6%} y {:.10%} respectivamente".
 ↪format(error100,error1000))
```

```
4.400000042666667 4.400000000004266
Sus errores son de 0.000001% y 0.0000000001% respectivamente
```

Estos resultados son órdenes de magnitud más precisos que al utilizar el método del trapezoide.