Computational Physics

Assignment 7

Cheng Ching Lin

1. Integration

Perform Voigt integration.

$$I = \int_{\mathrm{a}}^{\mathrm{b}} \frac{e^{-y^2}}{c+y} dy$$

For a = -10, b = 10, c = 10^{-3}.

(1).Trapezoidal Method

We use Trapezoidal method with the following bins:

$$h_n = \frac{a-b}{2^n}$$

So, Trapezoidal method is like following:

$$I_i = \frac{\big(I(h_i) + I(h_{i+1})\big)h}{2}$$

$$I_t = \sum_{i=0}^{N} I_i = h\sum_{i=0}^{N} I(h_i) - \frac{h}{2}I(h_0) - \frac{h}{2}I(h_N)$$

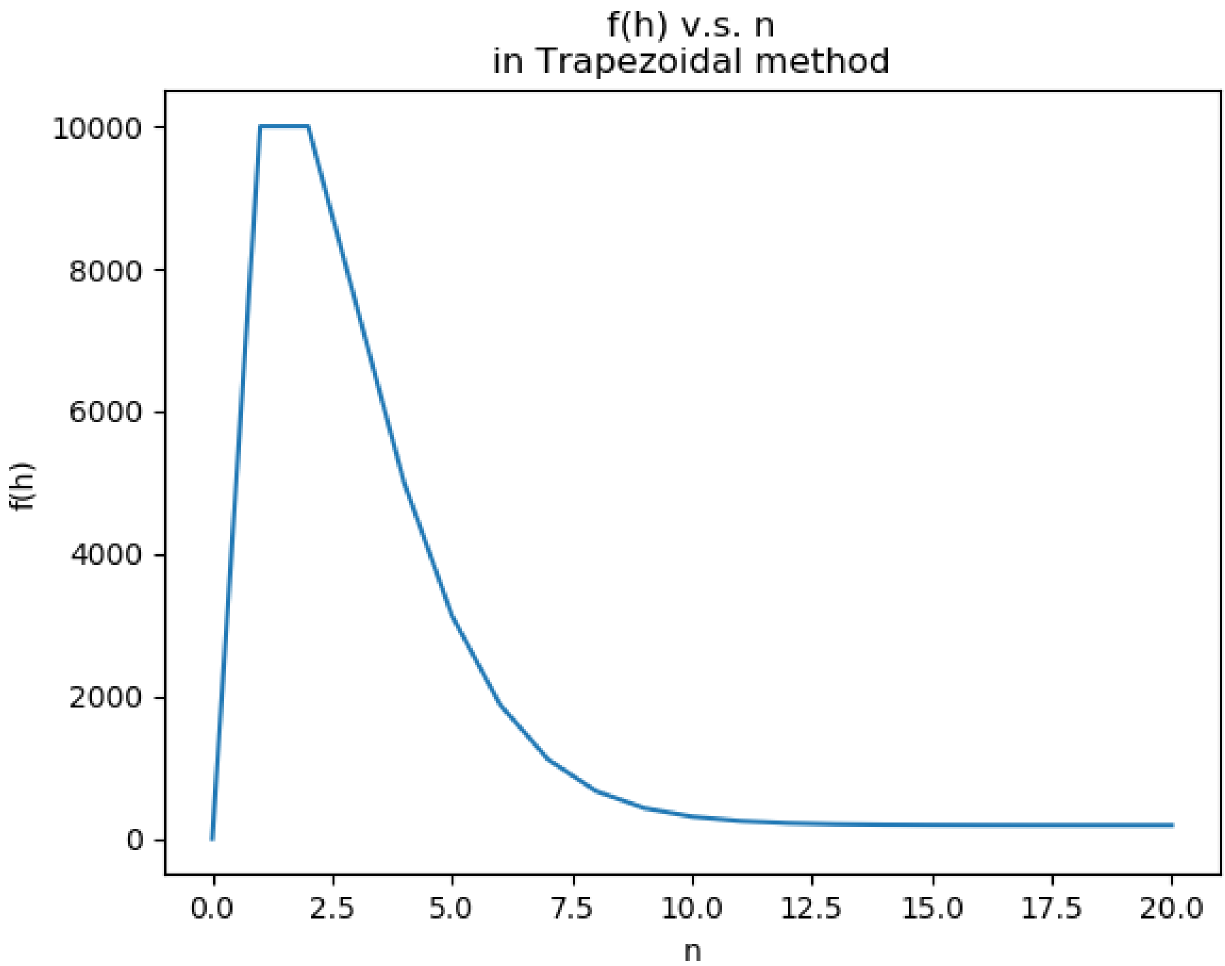
The answer will be the $\mathrm{I_t}$ in the calculation.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import math

def h(n): #create bins
    a = -10
    b = 10
    ans = (b-a)/pow(2,n)
    return ans

Fun = lambda x: math.exp(-pow(x,2))/(pow(10,-3)+pow(x,2)) #Voigt integration

def test_1():
    tempans = []
    nn = []
    ii = []
    ndx = 0
    err = 1
    while err > pow(10,-3): #accuracy
```

```
        nn.append(ndx)
        limi = pow(2,ndx)+1
        for idx in range(limi):
            x = -10+h(ndx)*idx
            temp = Fun(x)
            tempans.append(temp)
        tempsum = (sum(tempans)-0.5*tempans[0]-0.5*tempans[-1])*h(ndx) #Trapezoidal method
        ii.append(tempsum)
        print(tempsum)
        if ndx >3:
            err = abs((ii[ndx]-ii[ndx-1])/ii[ndx])
        ndx+=1
    print(ndx)
    plt.plot(nn,ii)
    plt.title("f(h) v.s. n\nin Trapezoidal method")
    plt.xlabel("n")
    plt.ylabel("f(h)")
    plt.show()
```

Result:



In the code, I do some accuracy correction that if error is bigger than 0.001, then n plus 1, and the max n is 21.
Also, because when n is 2 or 3, the error is smaller than accuracy, but it is not the answer we are looking for, I but an if loop to protect this thing happens.

(2).Romberg Integration
Let

$$R_{i,1} = I_t(h_i)$$

I_t is the answer of Trapezoidal method. Romberg integration is following:

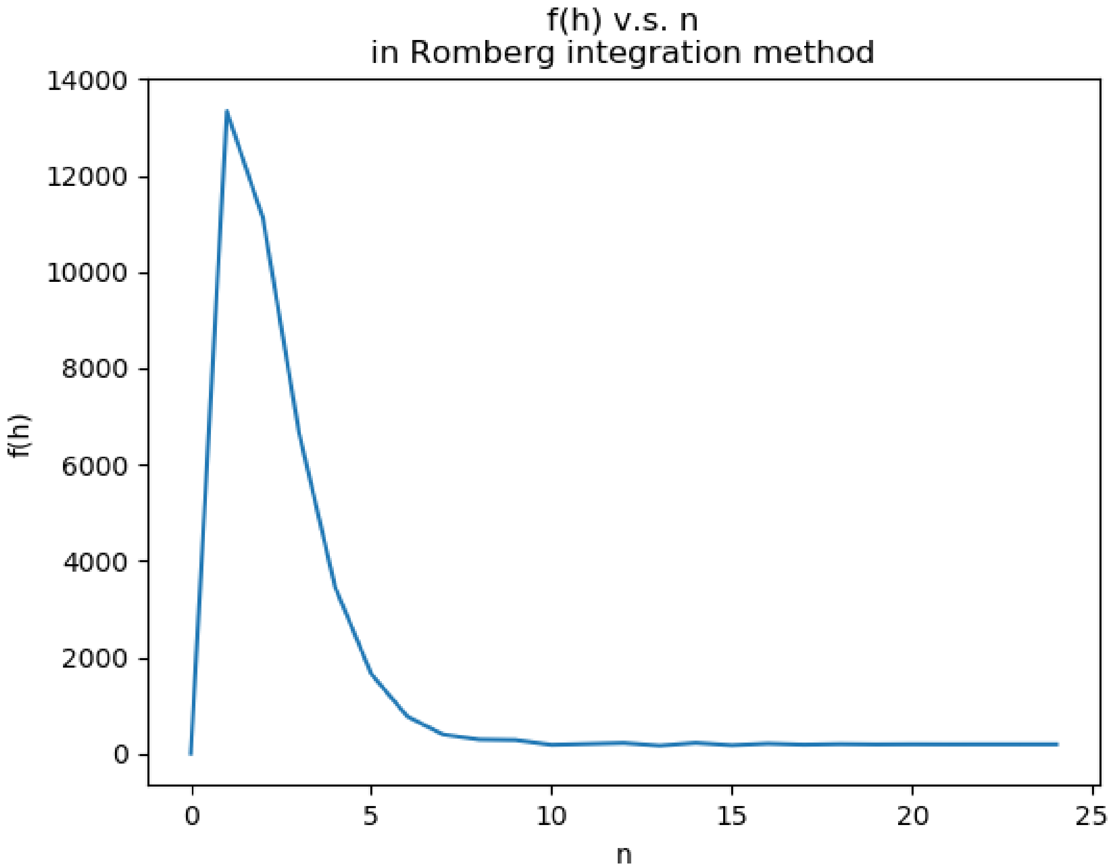
$$R_{i,j} = R_{i,j-1} + \frac{R_{i,j-1} - R_{i-1,j-1}}{4^{j-1} - 1}$$

Therefore, we know that Romberg integration base on Trapezoidal method.
Code:

```
def test_2():
    tempans = []
    nn = []
    ii = np.zeros(1000)
    rr = []
    kdx = 0
    err = 1
    while err>pow(10,-3):
        nn.append(kdx)
        num = kdx
        for ndx in range(num+1): #Trapezoidal method
            limi = pow(2, ndx) + 1
            for idx in range(limi):
                x = -10 + h(ndx) * idx
                temp = Fun(x)
                tempans.append(temp)
            tempsum = (sum(tempans) - 0.5 * tempans[0] - 0.5 * tempans[-1]) * h(ndx)
            ii[ndx] = tempsum
        while(num!=0): #Romberg method
            jj = 1
            for idx in range(1,num+1): #R
                tempr = ii[idx]+(ii[idx]-ii[idx-1])/(pow(4,jj)-1)
                ii[idx-1] = tempr
            num -=1
            jj+=1
        rr.append(ii[0])
        print(ii[0])
        if kdx!=0:
            err = abs((rr[kdx]-rr[kdx-1])/rr[kdx])
```

```python
        kdx+=1
    print(kdx)
    plt.plot(nn, rr)
    plt.title("f(h) v.s. n\nin Romberg integration method")
    plt.xlabel("n")
    plt.ylabel("f(h)")
    plt.show()
```

Result:



The max n is 25. Usually, the n of Trapezoidal method is bigger than Romberg integration. However, because Voigt integration is very sharp and narrow around the answer, and also Romberg integration is very sensitive, the answer of Romberg fluctuates around the answer.

2. Finding PI

To find PI in Monte Carlo, we suppose create a quarter of circle which radius is 1 in a 1*1 square. We can write down the equation below.

$$\frac{N_{circle}}{N_{total}} = \frac{A_{circle}}{A_{square}} = \frac{1*1*\pi*\frac{1}{4}}{1*1} = \frac{\pi}{4}$$

N is the total number of point inside the circle and square. A is area of circle and square.

Therefore, we can use the formula to find π

$$\frac{N_{circle}}{N_{total}} * 4 = \pi$$

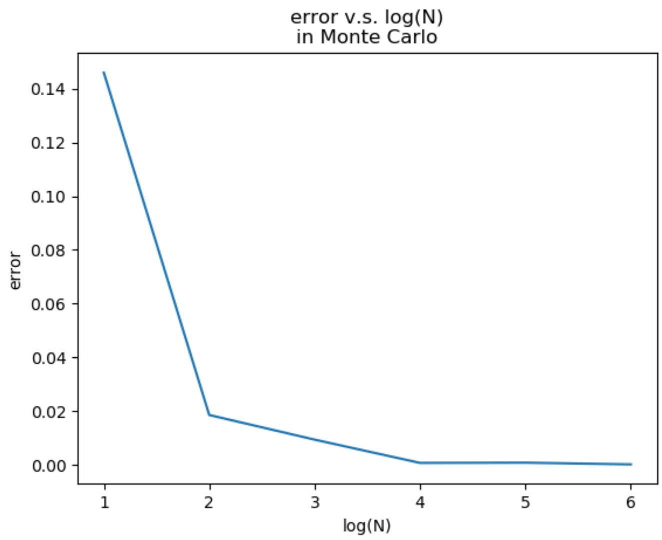
We also use spaced points to find π

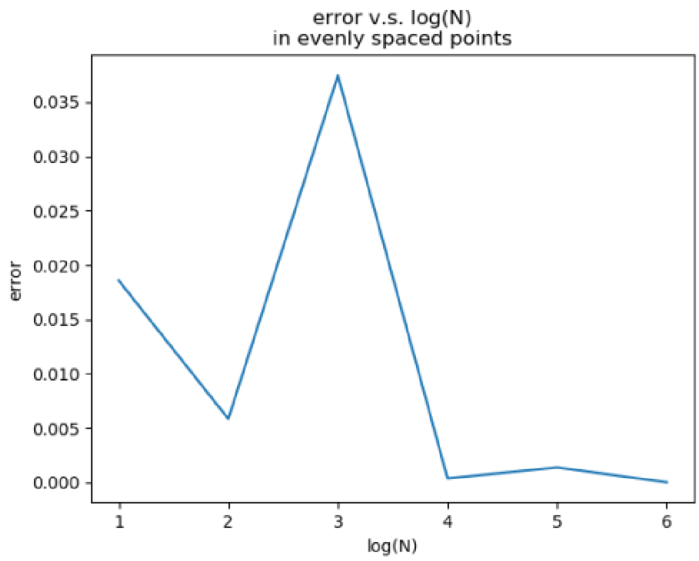
Code:

```
import matplotlib.pyplot as plt
import numpy as np
import math

def test_1(): #Monte Carlo
    nn = []
    ee = []

    for idx in range(1,7):
        i=0
        nn.append(idx)
        N = pow(10,idx)
        xx = np.random.uniform(0,1,N)
        yy = np.random.uniform(0,1,N)
        for jdx in range(N):
            if (pow(xx[jdx],2)+pow(yy[jdx],2)) <= 1:
                i+=1
        numpi = 4*i/N
        err = abs((numpi-math.pi)/math.pi)
        ee.append(err)
    plt.plot(nn,ee)
    plt.title("error v.s. log(N)\nin Monte Carlo")
    plt.xlabel("log(N)")
    plt.ylabel("error")
    plt.show()

def test_2(): #spaced points
    nn = []
    ee = []
```

```
for idx in range(1,7):
    i=0
    nn.append(idx)
    N = pow(10,idx)
    num = int(math.sqrt(N))
    for adx in range(num):
        for bdx in range(num):
            x = (adx+0.5)/num
            y = (bdx+0.5)/num
            if (pow(x,2)+pow(y,2)) <= 1:
             i+=1
    numpi = 4*i/N
    err = abs((numpi-math.pi)/math.pi)
    ee.append(err)
plt.plot(nn,ee)
plt.title("error v.s. log(N)\nin evenly spaced points")
plt.xlabel("log(N)")
plt.ylabel("error")
plt.show()
```

Result:





We compare these two. Although, at small N, the error of spaced points is smaller than Monte Carlo; at bigger N, the error of spaced points is bigger than Monte Carlo. The important thing is that we can predict error of Monte Carlo will be smaller when N is bigger, but we cannot predict how error of evenly spaced goes.

3. Maxwell Boltzmann Distribution

We create N particles with random directions and same velocity 1, and we make M collides of particles. At the end, we create the histogram of velocity.

The process of collide is following. We transform velocity from lab frame into center of mass frame.

$$v_{cm} = \frac{mv_1 + mv_2}{2m} = \frac{v_1}{2} + \frac{v_2}{2}$$

$$v_1^* = v_1 - v_{cm}$$

$$v_2^* = v_2 - v_{cm}$$

We create another random direction after colliding.

$$\hat{n} = random\ direction$$

$$v_1(new) = |v_1^*|\hat{n} + v_{cm}$$

$$v_2(new) = -|v_2^*|\hat{n} + v_{cm}$$

Probability density function of Maxwell Boltzmann distribution is following:

$$\sqrt{\frac{2}{\pi}}\left(\frac{x^2 e^{\frac{-x^2}{2a^2}}}{a^3}\right)$$

$$\mu = 2a\sqrt{\frac{2}{\pi}}$$

$$\mathrm{a} = \sqrt{\frac{\mathrm{K_T}}{m}}$$

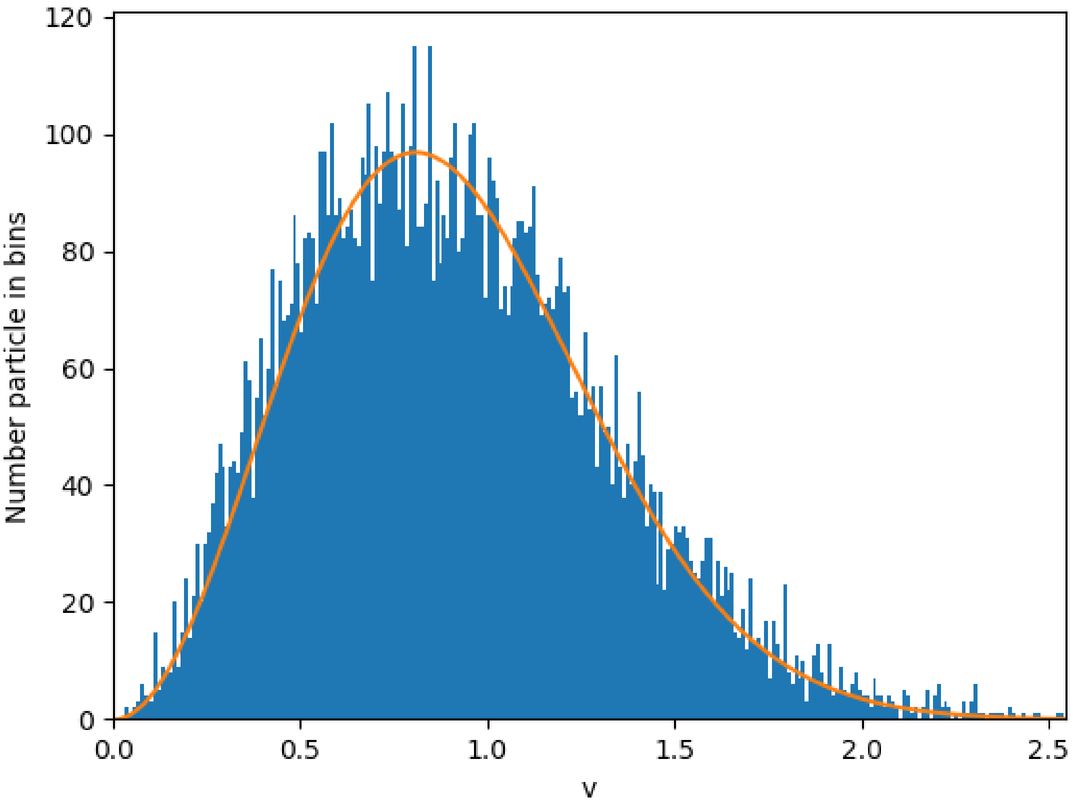
For μ is the mean number of data.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import statistics


def test_1():
    N = 10000 #10000 particles
    vx = []
    vy = []
    vz = []
    phi = np.random.uniform(0,2*math.pi,N) #randomly create direction
    c = np.random.uniform(-1,1,N)
    M = 100*N #number of collides
    for jdx in range(N):
        vx.append(math.cos(phi[jdx])*math.sqrt(1-pow(c[jdx],2)))
        vy.append(math.cos(phi[jdx]) * c[jdx])
        vz.append(math.sin(phi[jdx]))
```

```
for idx in range(M):
    v1 = random.randint(0,N-1) #randomly pick 2 particles
    v2 = random.randint(0,N-1)
    if v1 == v2:
        idx -=1
        continue
    vxcm = 0.5 * vx[v1] + 0.5 * vx[v2] #center of mass
    vycm = 0.5 * vy[v1] + 0.5 * vy[v2]
    vzcm = 0.5 * vz[v1] + 0.5 * vz[v2]
    v1_xcm = vx[v1]-vxcm
    v1_ycm = vy[v1] - vycm
    v1_zcm = vz[v1]-vzcm
    v1_cm = math.sqrt(pow(v1_xcm,2)+pow(v1_ycm,2)+pow(v1_zcm,2))
    nphi = np.random.uniform(0,2*math.pi,1)
    nc = np.random.uniform(-1,1,1)
    v1fx = v1_cm*math.cos(nphi)*math.sqrt(1-pow(nc,2))
    v1fy = v1_cm*math.cos(nphi)*nc
    v1fz = v1_cm*math.sin(nphi)
    vx[v1] = v1fx+vxcm
    vy[v1] = v1fy+vycm
    vz[v1] = v1fz+vzcm
    vx[v2] = -v1fx + vxcm
    vy[v2] = -v1fy + vycm
    vz[v2] = -v1fz + vzcm

v_fin = []
for kdx in range(N):
    value = math.sqrt(pow(vx[kdx],2)+pow(vy[kdx],2)+pow(vz[kdx],2))
    v_fin.append(value)

step = 0.01
x = np.arange(0,max(v_fin),step)
mean = statistics.mean(v_fin)
a = mean*math.sqrt(math.pi*0.5)*0.5
dis = 60*np.sqrt(np.pi*0.5)*x**2*np.exp(-x**2/(2*a**2))/a**3
plt.xlim(0,max(v_fin))
plt.hist(v_fin,bins = x)
plt.ylabel("Number particle in bins")
plt.xlabel("v")
plt.plot(x,dis)
plt.show()
```

Result:



The orange line is probability density function of Maxwell Boltzmann distribution. The random result fits in the function.