计算物理第九次作业

近代物理系 张浩然 SA21004048

Question 1 ▷ Multiple Integral

Calculate the following integrals with MC method:

1. $$\int_{-\infty}^{\infty}\left(\prod_{i=1}^{4}\frac{\mathrm{d}x_i}{\sqrt{2\pi}}\right)x_1^2\mathrm{e}^{-\sum_n n^2x_n^2+\sum_{m<n}x_mx_n}\ ;$$

2. $$\int_{\Lambda/s}^{\Lambda}\frac{\mathrm{d}^3\boldsymbol{k}_1}{(2\pi)^3}\int_{\Lambda/s}^{\Lambda}\frac{\mathrm{d}^3\boldsymbol{k}_2}{(2\pi)^3}\frac{1}{(\boldsymbol{k}_1^2+\mu^2)(\boldsymbol{k}_2^2+\mu^2)((\boldsymbol{p}-\boldsymbol{k}_1-\boldsymbol{k}_2)^2+\mu^2)}\ ;$$

   with $\Lambda=1, s=1.5, \mu=0.1$, and different $\boldsymbol{p}$'s .

Solution

1.
$$\begin{aligned}
&\int_{-\infty}^{\infty}\left(\prod_{i=1}^{4}\frac{\mathrm{d}x_i}{\sqrt{2\pi}}\right)x_1^2\mathrm{e}^{-\sum_n n^2x_n^2+\sum_{m<n}x_mx_n}\\
&=\int_{-\infty}^{\infty}x_1^2\mathrm{e}^{x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4}\prod_{n=1}^{4}\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-n^2x_n^2}\mathrm{d}x_n\\
&=\int_{-\infty}^{\infty}x_1^2\mathrm{e}^{x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4}\prod_{n=1}^{4}\frac{\frac{1}{\sqrt{2}n}}{\sqrt{2\pi}\times\frac{1}{\sqrt{2}n}}\exp\left[-\frac{1}{2}\frac{x_n^2}{\left(\frac{1}{\sqrt{2}n}\right)^2}\right]\mathrm{d}x_n\\
&=\int_{-\infty}^{\infty}\frac{1}{96}x_1^2\mathrm{e}^{x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4}\prod_{n=1}^{4}\frac{1}{\sqrt{2\pi}\times\frac{1}{\sqrt{2}n}}\exp\left[-\frac{1}{2}\frac{x_n^2}{\left(\frac{1}{\sqrt{2}n}\right)^2}\right]\mathrm{d}x_n\\
&=\int_{-\infty}^{\infty}f(x_1,x_2,x_3,x_4)\prod_{n=1}^{4}p_n(x_n)\mathrm{d}x_n.
\end{aligned}\tag{1}$$

其中

$$f(x_1,x_2,x_3,x_4)=\frac{1}{96}x_1^2\mathrm{e}^{x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4},\tag{2}$$

而

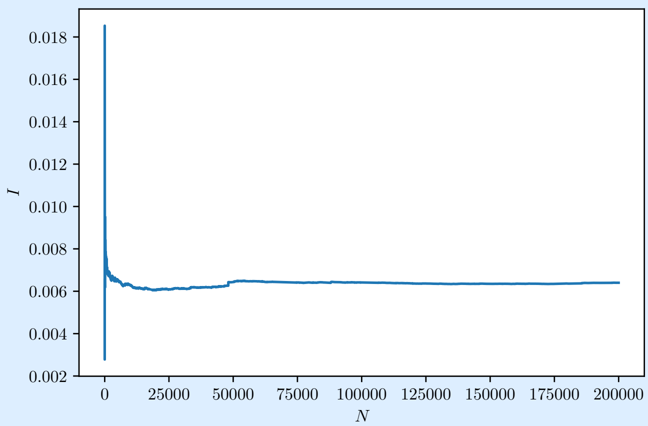
$$p_n(x_n)=\frac{1}{\sqrt{2\pi}\times\frac{1}{\sqrt{2}n}}\exp\left[-\frac{1}{2}\frac{x_n^2}{\left(\frac{1}{\sqrt{2}n}\right)^2}\right],\tag{3}$$

为 $\mu=0,\sigma=\dfrac{1}{\sqrt{2}n}$ 的正态分布.

使用python进行计算, 代码如下:

```python
import numpy as np
from numpy import sqrt, exp, pi, cos, sin
from numpy import random
import matplotlib.pyplot as plt

def f(x1,x2,x3,x4):
    return x1**2 * exp( x1 * (x2 + x3 + x4) + x2 * (x3 + x4) + x3 * x4 ) / 96

N = 200000

s = 0
v = []
for i in range(1,N+1):
    x1 = random.normal(0,1/sqrt(2))
    x2 = random.normal(0,1/sqrt(2)/2)
    x3 = random.normal(0,1/sqrt(2)/3)
    x4 = random.normal(0,1/sqrt(2)/4)
    s += f(x1,x2,x3,x4)
    v.append(s/i)
print(v[-1])
plt.plot(range(1,N+1),v);
```

积分结果为0.006400240965502292. 画出积分结果与投点次数的关系, 可以看到此时结果已经收敛.



2.

$$
\begin{aligned}
&\int_{\Lambda/s}^{\Lambda}\frac{\mathrm{d}^3\boldsymbol{k}_1}{(2\pi)^3}\int_{\Lambda/s}^{\Lambda}\frac{\mathrm{d}^3\boldsymbol{k}_2}{(2\pi)^3}\frac{1}{(\boldsymbol{k}_1^2+\mu^2)(\boldsymbol{k}_2^2+\mu^2)[(\boldsymbol{p}-\boldsymbol{k}_1-\boldsymbol{k}_2)^2+\mu^2]}\\
=&\frac{1}{(2\pi)^6}\iint_{\Lambda/s}^{\Lambda}\mathrm{d}k_1\mathrm{d}k_2\iint_0^{\pi}\mathrm{d}\theta_1\mathrm{d}\theta_2\iint_0^{2\pi}\mathrm{d}\phi_1\mathrm{d}\phi_2\frac{k_1^2\sin\theta_1\ k_2^2\sin\theta_2}{(k_1^2+\mu^2)(k_2^2+\mu^2)[(\boldsymbol{p}-\boldsymbol{k}_1-\boldsymbol{k}_2)^2+\mu^2]}
\end{aligned}
\tag{4}
$$

积分时不妨取 $\boldsymbol{p}$ 的方向为 z 轴方向, 则有

$$
\begin{aligned}
\boldsymbol{p}\cdot\boldsymbol{k}_1 &= pk_1\cos\theta_1\\
\boldsymbol{p}\cdot\boldsymbol{k}_2 &= pk_2\cos\theta_2\\
\boldsymbol{k}_1\cdot\boldsymbol{k}_2 &= k_1k_2[\sin\theta_1\sin\theta_2(\cos\phi_1\cos\phi_2+\sin\phi_1\sin\phi_2)+\cos\theta_1\cos\theta_2]\\
&= k_1k_2[\sin\theta_1\sin\theta_2\cos(\phi_1-\phi_2)+\cos\theta_1\cos\theta_2]
\end{aligned}
\tag{5}
$$

进一步地, 可令 $\boldsymbol{k}_1$ 落在 xOz 平面上, 即 $\phi_1=0$, 则

$$
(\boldsymbol{p}-\boldsymbol{k}_1-\boldsymbol{k}_2)^2=p^2+k_1^2+k_2^2-2p(k_1\cos\theta_1+k_2\cos\theta_2)+2k_1k_2(\sin\theta_1\sin\theta_2\cos\phi_2+\cos\theta_1\cos\theta_2),\tag{6}
$$

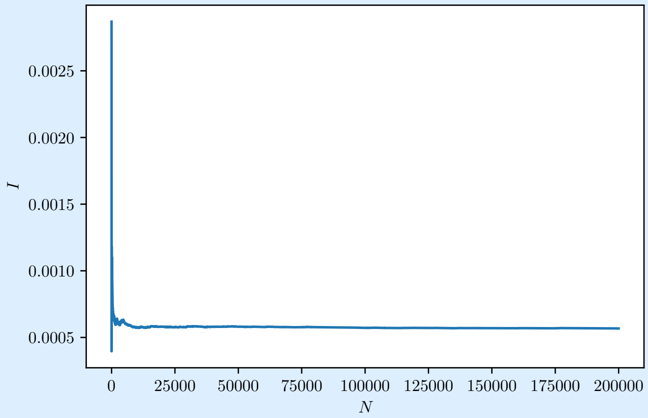
原积分化为

$$
\frac{1}{(2\pi)^6}\iint_{\Lambda/s}^{\Lambda}\mathrm{d}k_1\mathrm{d}k_2\iint_0^{\pi}\mathrm{d}\theta_1\mathrm{d}\theta_2\int_0^{2\pi}\mathrm{d}\phi_2\frac{2\pi k_1^2\sin\theta_1\ k_2^2\sin\theta_2}{(k_1^2+\mu^2)(k_2^2+\mu^2)[(\boldsymbol{p}-\boldsymbol{k}_1-\boldsymbol{k}_2)^2+\mu^2]}.\tag{7}
$$

使用python进行计算, 代码如下:

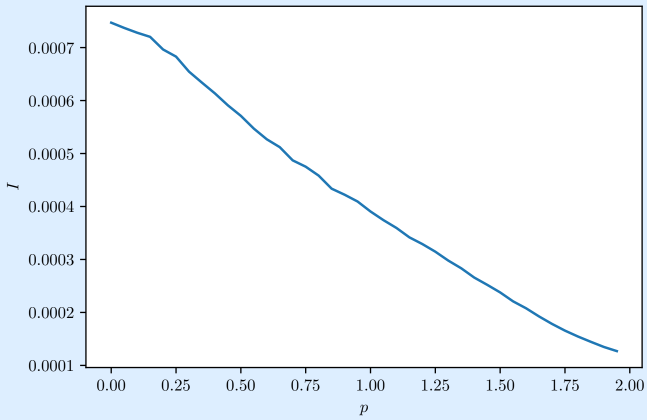
```python
def g(mu,p,k1,k2,th1,th2,ph2):
    pkk = p**2 + k1**2 + k2**2 - 2*p * (k1*cos(th1) + k2*cos(th2)) \
        + 2*k1*k2 * (sin(th1)*sin(th2)*cos(ph2) + cos(th1)*cos(th2))
    return k1*sin(th1)*k2*sin(th2) / ( (k1**2+mu**2)*(k2**2+mu**2)*(pkk+mu**2) ) / (2*pi)**5

N = 200000

mu = .1; L = 1; s = 1.5
kmin = L/s; kmax = L

p=.5

summ = 0; valu = []
reg_area = (kmax-kmin)**2 * pi**2 * 2*pi

for i in range(1,N+1):
    k1 = kmin + (kmax - kmin) * random.rand()
    k2 = kmin + (kmax - kmin) * random.rand()
    th1 = pi * random.rand()
    th2 = pi * random.rand()
    ph2 = 2 * pi * random.rand()
    summ += g(mu,p,k1,k2,th1,th2,ph2)
    temp = summ / i * reg_area
    valu.append(temp)
print(valu[-1])
plt.plot(range(1,N+1),valu);
```

当 $p=0.5$ 时, 积分结果为0.0005675542943088573. 画出积分结果与投点次数的关系, 可以看到此时结果已经收敛.



改变 p 的取值, 得到积分结果与 p 的关系:

```
mu = .1; L = 1; s = 1.5
kmin = L/s; kmax = L
reg_area = (kmax-kmin)**2 * pi**2 * 2*pi

p_lst = [0.05*i for i in range(40)]
val_lst = []

for p in p_lst:
    summ = 0
    for i in range(1,N+1):
        k1 = kmin + (kmax - kmin) * random.rand()
        k2 = kmin + (kmax - kmin) * random.rand()
        th1 = pi * random.rand()
        th2 = pi * random.rand()
        ph2 = 2 * pi * random.rand()
        summ += g(mu,p,k1,k2,th1,th2,ph2)
    val_lst.append(summ / N * reg_area)
plt.plot(p_lst,val_lst);
```



Question 2 ▷ Generate Random Variables

Generate random variables $\{x_i\} \sim P(x)$ for following each $P(x)$:

1.
$$P(x) = 0.7\mathrm{e}^{-(x+10)^2/0.1} + 0.9\mathrm{e}^{-x^4+3x^2}$$

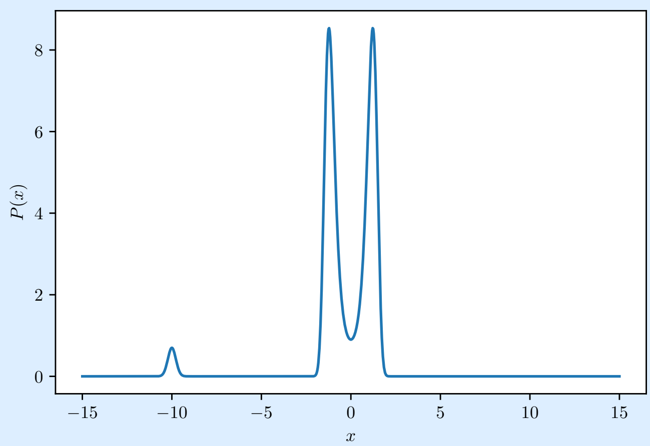
2.
$$P(x,y) = \frac{1}{x^2+m^2} + \frac{1}{y^2}$$
with $y > \epsilon \to 0$

Solution

1.

做出 $\{x\}$ 的分布图:

```
import numpy as np
from numpy import sqrt, exp, pi, cos, sin
from numpy import random
import matplotlib.pyplot as plt

def p1(x):
    return .7 * exp(-(x+10)**2/.1) + .9 * exp(-x**4 + 3*x**2)

xx = np.linspace(-15,15,1000)
pp = [p1(x) for x in xx]
plt.plot(xx,pp)
```

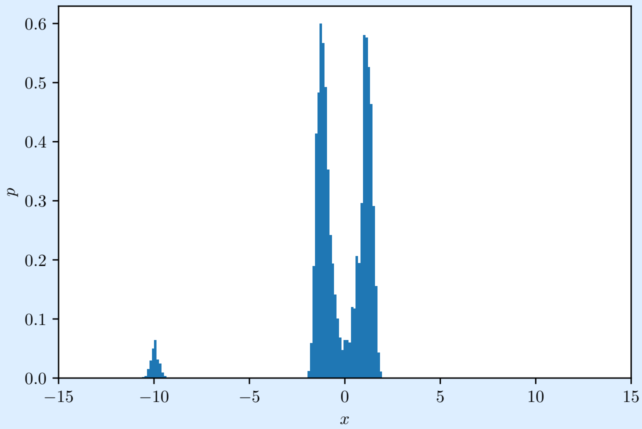


使用 Metropolis 方法进行采样:

```
N = 100000
x0 = 0

x = x0; x_lst = [x0]
for i in range(N):
    xnew = 20 * (2* random.rand() - 1)
    r = p1(xnew) / p1(x)
    if random.rand() < r :
```

```
        x_lst.append(xnew)
        x = xnew
    else:
        x_lst.append(x)
        plt.hist(x_lst,bins=100,density=True);
```

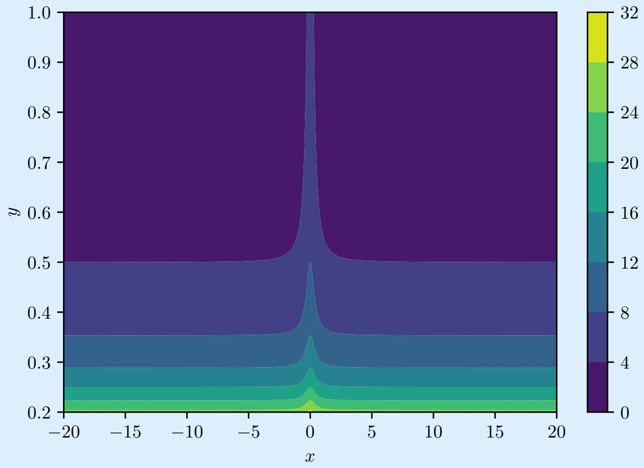
得到的样本的分布如下图所示, 与原分布基本符合.



2.

取 $m = 0.5$, 为避免发散, 令 $y > \epsilon = 0.2$, 做出 $\{(x, y)\}$ 的分布图:

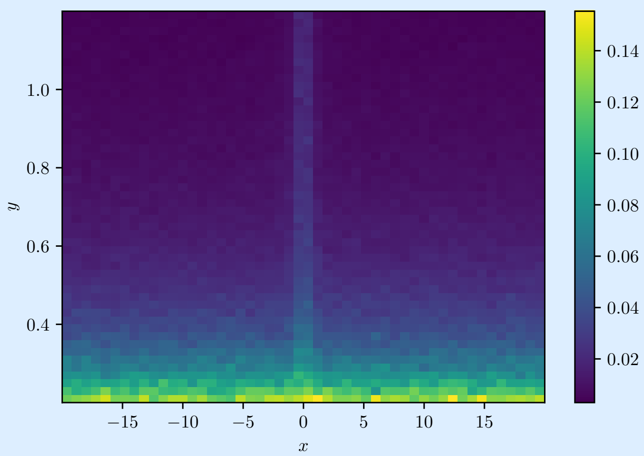
```
import numpy as np
from numpy import sqrt, exp, pi, cos, sin
from numpy import random
import matplotlib.pyplot as plt

def p2(x,y,m):
    return 1/(x**2+m**2) + 1/(y**2)

m = .5
xx = np.linspace(-20,20,1000)
yy = np.linspace(.2,1,1000)
X,Y = np.meshgrid(xx,yy)
plt.contourf(X,Y,p2(X,Y,m))
plt.colorbar()
```



使用 Metropolis 方法进行采样:

```
m = .5
N = 1000000
x0 = 0; y0 = 1

x = x0; x_lst = [x0]
y = y0; y_lst = [y0]

for i in range(N):
    xnew = 20 * (2* random.rand() - 1)
    ynew = 1 * random.rand() + .2
    r = p2(xnew,ynew,m) / p2(x,y,m)
    if random.rand() < r :
        x = xnew; x_lst.append(x)
        y = ynew; y_lst.append(y)

    else:
        x_lst.append(x)
        y_lst.append(y)
```

得到的样本的分布如下图所示, 与原分布基本符合.



Question 3 ▷ Phase Transition in Ising Model

Solution

二维情况

求解 50×50 的二维格子. 初始值分别取 $\dfrac{3}{4}$ 的格子为 $|\uparrow\rangle$ 态, 以及 $\dfrac{3}{4}$ 的格子为 $|\downarrow\rangle$ 态这两种情况.

```python
import numpy as np
from numpy.random import random
import numba
from numba import njit
from scipy.ndimage import convolve, generate_binary_structure
import matplotlib.pyplot as plt
```

使用二维数组来存储各个格点的spin, 并完成初始化:

```python
init_random = random((N,N))
lattice_n = np.zeros((N,N))
lattice_n[init_random>=0.75] = 1
lattice_n[init_random<0.75] = -1

init_random = random((N,N))
lattice_p = np.zeros((N,N))
lattice_p[init_random>=0.25] = 1
lattice_p[init_random<0.25] = -1
```

为了计算给定 spin 构型的能量, 需要获得每个格点的近邻关系. 这一步可以用一个形如 $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ 的数组与存储 spin 的二维数组卷积得到. 这样的数组可以通过科学计算函数库 **scipy** 中图像处理包 **ndimage** 的 **generate_binary_structure** 函数方便地得到:

```python
nbr = generate_binary_structure(2, 1).astype(int)
nbr[1][1] = 0
print(nbr)
```

nbr 数组的输出结果为:

```
[[0 1 0]
 [1 0 1]
 [0 1 0]]
```

定义计算给定 spin 构型能量的函数:

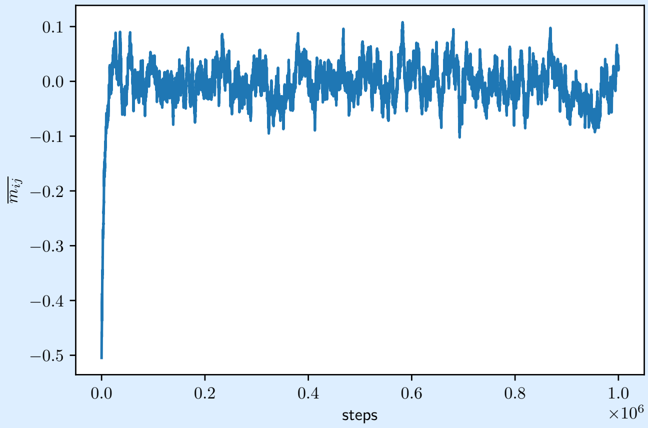
```python
def get_energy(spins):
    E = -spins * convolve(spins, nbr, mode='constant')
    return E.sum()
```

由于计算量较大, 使用 **numba** 函数库的 **jit** 函数对 Metropolis 过程进行加速. 被 **numba_metropolis** 装饰的函数只支持个别基本函数, 自己定义的函数无法使用, 因此需要将 Metropolis 过程写成两个函数:

```python
def metropolis(spins, times, beta):
    spins = spins.copy()
    energy = get_energy(spins)
    return numba_metropolis(spins, times, beta, energy)

@numba.njit(nogil=True)
def numba_metropolis(spins, times, beta, energy):

    net_spins = np.zeros(times-1)
    net_energy = np.zeros(times-1)

    for t in range(0,times-1):

        x = np.random.randint(0,N); y = np.random.randint(0,N)
        spin_i = spins[x,y]; spin_f = - spin_i

        E_i = 0; E_f = 0
        if x > 0:
            E_i += -spin_i*spins[x-1,y]; E_f += -spin_f*spins[x-1,y]
        if x < N-1:
            E_i += -spin_i*spins[x+1,y]; E_f += -spin_f*spins[x+1,y]
        if y > 0:
            E_i += -spin_i*spins[x,y-1]; E_f += -spin_f*spins[x,y-1]
        if y < N-1:
            E_i += -spin_i*spins[x,y+1]; E_f += -spin_f*spins[x,y+1]

        dE = E_f-E_i
        if ( dE > 0 and random() < np.exp(-beta*dE) ) or ( dE <= 0 ):
            spins[x,y]=spin_f
            energy += dE

        net_spins[t] = spins.sum()
        net_energy[t] = energy

    return net_spins, net_energy
```

首先试验 $\beta = 0.2$ 时达到平衡所需的步数.

```python
spins, energies = metropolis(lattice_n, 1000000, 0.2)
plt.plot(spins/N**2)
```

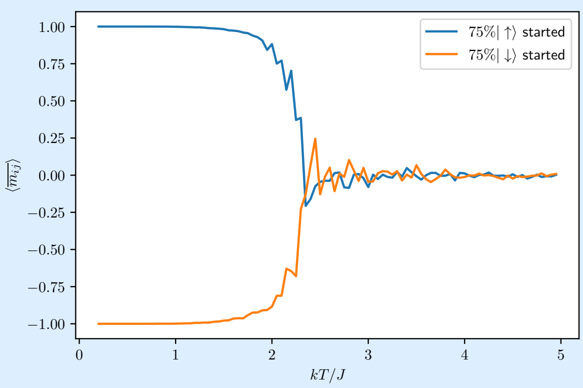


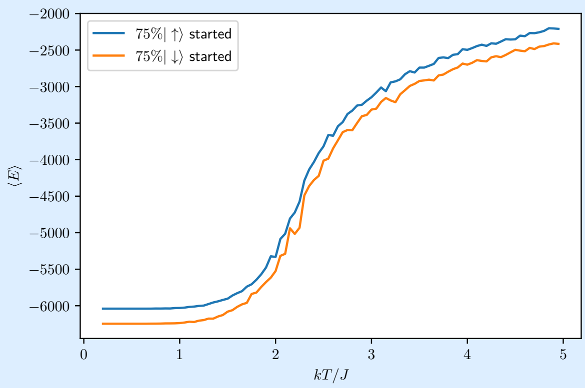
可以看到, 当 Metropolis 过程的步数超过10万次后, 体系趋于平衡. 因此可以将总步数设为100万, 并对最后10万次的结果进行平均.

```python
def get_magm_energy(lattice, beta_lst):
    magm_lst = np.zeros_like(beta_lst)
    E_lst = np.zeros_like(beta_lst)
    for i in tqdm(range(len(beta_lst))):
        spins, energies = metropolis(lattice, 1000000, beta_lst[i])#, get_energy(lattice))
        magm_lst[i] = spins[-100000:].mean() / N**2
        E_lst[i] = energies[-100000:].mean()
        E_stds[i] = energies[-100000:].std()
    return magm_lst, E_lst
```

计算 $kT = \dfrac{1}{\beta} \in [0.2J, 5J]$ 范围内的磁化强度和能量, 得到结果如图所示.

```python
beta_lst = 1 / np.arange(0.2, 5, 0.05)
magm_lst_n, E_lst_n = get_magm_energy(lattice_n, beta_lst)
magm_lst_p, E_lst_p = get_magm_energy(lattice_p, beta_lst)

plt.figure()
plt.plot(1/beta_lst, magm_lst_p, label=r'$75\%␣|\uparrow\rangle$␣started')
plt.plot(1/beta_lst, magm_lst_n, label=r'$75\%␣|\downarrow\rangle$␣started')

plt.figure()
plt.plot(1/beta_lst, E_lst_p,label=r'$75\%␣|\uparrow\rangle$␣started')
plt.plot(1/beta_lst, E_lst_n,label=r'$75\%␣|\downarrow\rangle$␣started')
```





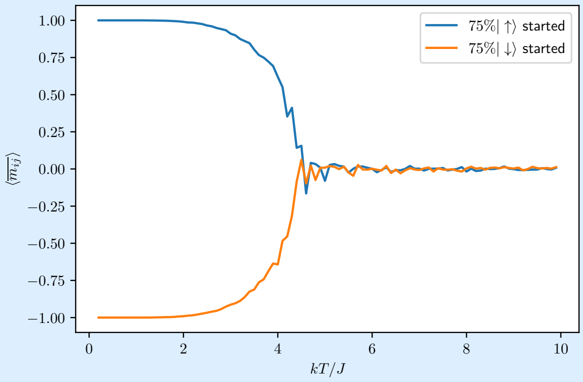
可以看到, 相变点大约为 $kT = 2.5J$.

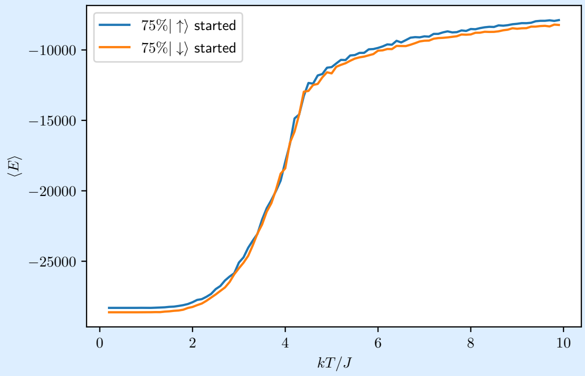
三维情况

三维情况与二维情况大致相同, 体系大小选为$20 \times 20 \times 20$, 代码与计算结果如下:

```
import numpy as np
from numpy.random import random,randint
import numba
from numba import njit
from scipy.ndimage import convolve, generate_binary_structure
import matplotlib.pyplot as plt

N = 20

init_random = random((N,N,N))
lattice_n = np.zeros((N,N,N))
lattice_n[init_random>=0.75] = 1
lattice_n[init_random<0.75] = -1

init_random = random((N,N,N))
lattice_p = np.zeros((N,N,N))
lattice_p[init_random>=0.25] = 1
lattice_p[init_random<0.25] = -1

nbr = generate_binary_structure(3, 1).astype(int)
nbr[1][1][1] = 0

def get_energy(spins):
    nbr = generate_binary_structure(3, 1)
    nbr[1][1][1] = False
    E = -spins * convolve(spins, nbr, mode='constant')
    return E.sum()

def metropolis(spins, times, beta):
    spins = spins.copy()
    energy = get_energy(spins)
    return numba_metropolis(spins, times, beta, energy)

@numba.njit(nogil=True)
def numba_metropolis(spins, times, beta, energy):

    net_spins = np.zeros(times-1)
    net_energy = np.zeros(times-1)

    for t in range(0,times-1):

        x = randint(0,N); y = randint(0,N); z = randint(0,N);
        spin_i = spins[x,y,z]; spin_f = - spin_i

```

```
        E_i = 0; E_f = 0
        if x > 0:
            E_i += -spin_i*spins[x-1,y,z]; E_f += -spin_f*spins[x-1,y,z]
        if x < N-1:
            E_i += -spin_i*spins[x+1,y,z]; E_f += -spin_f*spins[x+1,y,z]
        if y > 0:
            E_i += -spin_i*spins[x,y-1,z]; E_f += -spin_f*spins[x,y-1,z]
        if y < N-1:
            E_i += -spin_i*spins[x,y+1,z]; E_f += -spin_f*spins[x,y+1,z]
        if z > 0:
            E_i += -spin_i*spins[x,y,z-1]; E_f += -spin_f*spins[x,y,z-1]
        if z < N-1:
            E_i += -spin_i*spins[x,y,z+1]; E_f += -spin_f*spins[x,y,z+1]

        dE = E_f-E_i
        if ( dE > 0 and random() < np.exp(-beta*dE) ) or ( dE <= 0 ):
            spins[x,y,z]=spin_f
            energy += dE

        net_spins[t] = spins.sum()
        net_energy[t] = energy

    return net_spins, net_energy

def get_magm_energy(lattice, beta_lst):
    magm_lst = np.zeros_like(beta_lst)
    E_lst = np.zeros_like(beta_lst)
    for i in tqdm(range(len(beta_lst))):
        spins, energies = metropolis(lattice, 1000000, beta_lst[i])#, get_energy(lattice))
        magm_lst[i] = spins[-100000:].mean() / N**3
        E_lst[i] = energies[-100000:].mean()
    return magm_lst, E_lst

beta_lst = 1 / np.arange(0.2, 10, 0.1)
magm_lst_n, E_lst_n = get_magm_energy(lattice_n, beta_lst)
magm_lst_p, E_lst_p = get_magm_energy(lattice_p, beta_lst)

plt.figure()
plt.plot(1/beta_lst, magm_lst_p, label=r'$75\%␣|\uparrow\rangle$␣started')
plt.plot(1/beta_lst, magm_lst_n, label=r'$75\%␣|\downarrow\rangle$␣started')

plt.figure()
plt.plot(1/beta_lst, E_lst_p,label=r'$75\%␣|\uparrow\rangle$␣started')
plt.plot(1/beta_lst, E_lst_n,label=r'$75\%␣|\downarrow\rangle$␣started')
```





三维 Ising 模型的相变点大约为 $kT = 4.5J$.

Question 4 ▷ Van der Waals Equation

相互作用气体方程为

$$\left(p+\frac{a}{V^2}\right)(V-b)=k_BT,$$

求相变点附近 ΔV 、 $\kappa=-\frac{1}{V}\left(\frac{\partial V}{\partial p}\right)_T$ 与 $T-T_c$ 的关系.

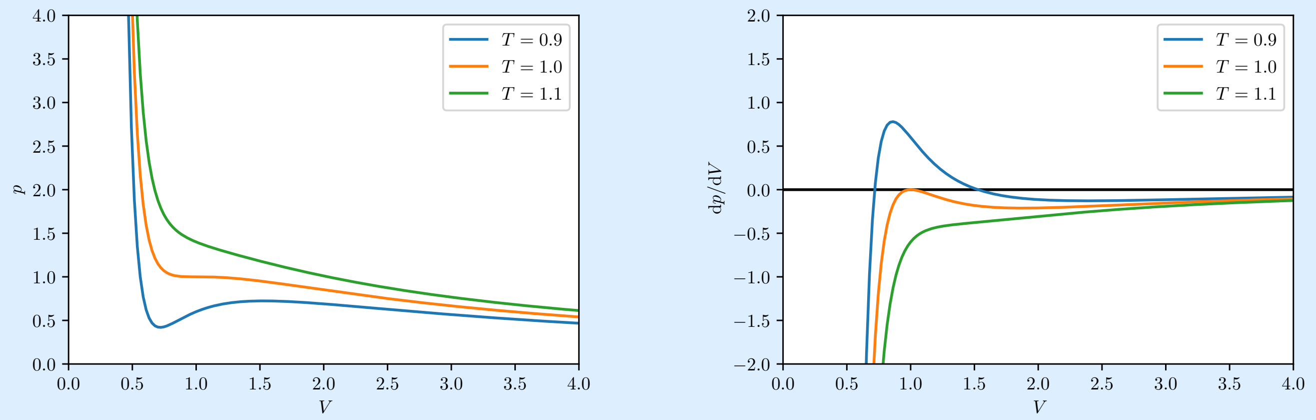
Solution

为简化计算, 直接使用约化的 Van der Waals 方程形式:

$$\left(p+\frac{3}{V^2}\right)\left(V-\frac{1}{3}\right)=\frac{3}{8}T. \tag{8}$$

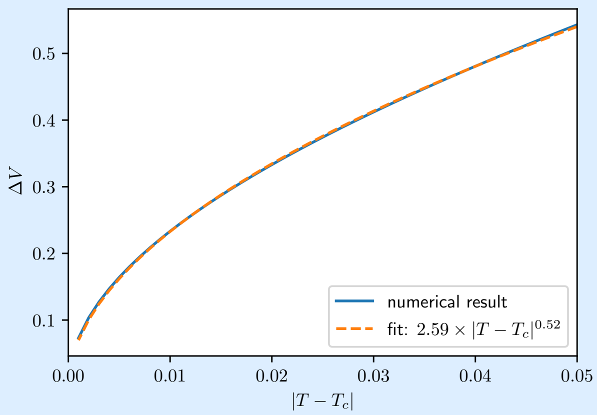
在该单位制下, 临界温度刚好为 $T_c=1$. 为直观, 做出 T_c 附近不同温度下的 $p-V$ 图:

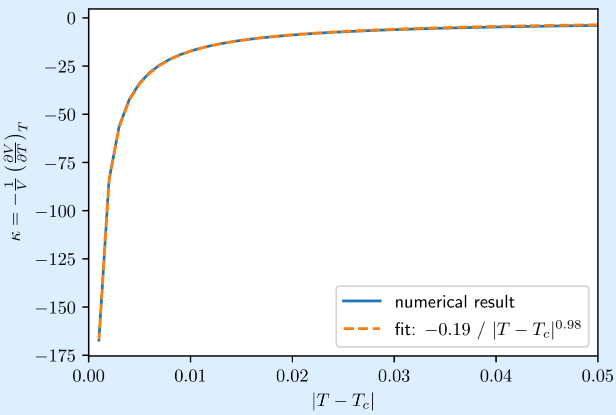
```python
import numpy as np
from numpy import sqrt, exp, pi, cos, sin
from numpy import random
from scipy.optimize import fsolve,curve_fit

kB = 8/3;
a = 3; b = 1/3

def p(V,T):
    return kB*T / (V-b) - a/V**2

def dp(V,T):
    return - kB*T / (V-b)**2 + 2*a/V**3

def kappa(V,T):
    return -1 / V / dp(V,T)

fig,axes = plt.subplots(1,2)
VV = np.linspace(0.4,5,200)
for T in [.9,1,1.1]:
    axes[0].plot(VV, p(VV,T),label=r'$T=%.1f$'%T)
    axes[1].plot(VV,dp(VV,T),label=r'$T=%.1f$'%T)
```



在 $0.9 < T < 1$ 区间内计算两极值点的位置差 $V_2 - V_1$ 作为 ΔV, 计算 $\dfrac{V1+V2}{2}$ 作为相变时的体积, 由此计算 $\kappa = -\dfrac{1}{V}\left(\dfrac{\partial V}{\partial p}\right)_T = -\dfrac{1}{V}\dfrac{1}{\left(\dfrac{\partial p}{\partial V}\right)_T}$ 并作图:

```python
T_lst = [.95+.001*i for i in range(50)]
DeltaT_lst = [1-T for T in T_lst]
DeltaV_lst, kappa_lst = [], []
for T in T_lst:
    V1 = fsolve(dp_fixT,.6)[0]
    V2 = fsolve(dp_fixT,1.2)[0]
    DeltaV_lst.append(V2-V1)
    V0 = (V1+V2)/2
    kappa_lst.append(kappa(V0,T))

def fit_power(x,C,power):
    return C* x**power

popt_DV,_ = curve_fit(fit_power,DeltaT_lst,DeltaV_lst);
print(popt_DV)
plt.plot(DeltaT_lst,DeltaV_lst);
plt.plot(DeltaT_lst,fit_power(DeltaT_lst,*popt_DV),'C1--');

def fit_power_inv(x,C,power):
    return -C/ x**power

popt_kappa,_ = curve_fit(fit_power_inv,DeltaT_lst,kappa_lst);
print(popt_kappa)
plt.plot(DeltaT_lst,kappa_lst);
plt.plot(DeltaT_lst,fit_power_inv(DeltaT_lst,*popt_kappa),'C1--');
```





对得到的 ΔV 和 κ 与 $|T - T_c|$ 的关系曲线进行拟合, 得到

$$\begin{aligned} \Delta V &\propto |T - T_c|^{0.5231}, \\ \kappa &\propto \frac{1}{|T - T_c|^{0.9837}}. \end{aligned} \tag{9}$$

与理论值基本符合.