1. 使用 Monte Carlo 方法估计超球（hypersphere）的体积。

```
3   def hypersphere(n):
4       N = 0
5
6       for i in range(1000000):
7           r = 0
8
9           for j in range(n):
10              x = np.random.rand()
11              r += x**2
12
13          if r <= 1:
14              N += 1
15
16      return (N/1000000) * (2**n)
17
18  print(hypersphere(2))
19  print(hypersphere(3)*0.75)
20  print(hypersphere(4))
21  print(hypersphere(5))
```

定义$hypersphere(n)$函数来计算 n 维超球的体积。

Monte Carlo 模拟次数为 100 万次，每次模拟采样的样本点都随机均匀落在边长为 2 的超立方体内。超球半径为 1，N 为落在超球内部的样本点总数。

由于超球是 n 维，函数中 j 指标满足 j=0,1,2,...,n-1 。样本点的坐标$x = (x_0, x_1, \dots, x_{n-1})$，样本点离圆心的距离$r = \sum_{j=0}^{n-1} x_j^2$，判断样本点是否落在超球内部，$r \leq 1$即落在内部。

这样就有：$\frac{N}{10^6} = \frac{V_{hypersphere}}{V_{hypercube}}$，由于$V_{hypercube} = 2^n$，$V_{hypersphere} = 2^n \times \frac{N}{10^6}$。

2 维超球（圆）的体积为π，3 维超球（球）的体积为$\frac{4}{3}\pi$。将$hypersphere(2)$和$\frac{3}{4}hypersphere(3)$的结果输出，与$\pi$的值作对比，并输出$hypersphere(4)$（4 维超球）和$hypersphere(5)$（5 维超球）的体积。结果如图所示：

```
3.140492
3.1418999999999997
4.924976
5.276576
```

对于 2 维，相对误差$\eta = 0.035\%$。对于 3 维，相对误差$\eta = 0.01\%$。

对于 4 维超球，$V = \frac{\pi^2}{2}$，相对误差$\eta = 0.20\%$。

对于 5 维超球，$V = \frac{8\pi^2}{15}$，相对误差$\eta = 0.24\%$。

Input: n

Output: $the\ volume\ of\ n-dimension\ hypersphere$

1. **Function** $hypersphere(n)$
2. $N \leftarrow 0$
3. **For** $i \leftarrow 1\ to\ 1000000$ **Do**
4. $r \leftarrow 0$
5. **For** $j \leftarrow 1\ to\ n$ **Do**
6. $x \leftarrow random\ number \in [0,1)$
7. $r \leftarrow r + x^2$
8. **If** $r \leq 1$ **Then**
9. $N \leftarrow N + 1$
10. $V \leftarrow 2^n \times \frac{N}{10^6}$
11. **Return** V
12. **End Function**

2. 使用 Monte Carlo 方法估计 3D Heisenberg spin model 的居里温度。

定义 1 个 Heisenberg 类，来构造 Heisenberg model：

```
4   class Heisenberg(object):
5
6       def __init__(self, size, J):
7           self.size = size
8           self.J = J
9           self.lattice = np.ones([size, size, size, 3])
```

这个类有三个初始化属性：size 表示这个立方晶格每条边上格点的个数；J 在本题中等于 1；lattice 是一个$size \times size \times size \times 3$的矩阵，表示这个立方晶格共有$size^3$个格点，每个格点的自旋都有 3 个方向的分量。

定义 2 个方法：$initialize_spin()$初始化自旋和$random_spin()$随机化自旋。

```
11      def initialize_spin(self):
12          for i in range(self.size):
13              for j in range(self.size):
14                  for k in range(self.size):
15                      self.lattice[i, j, k] = np.array([0.0,0.0,1.0])
16          return None
17
18      def random_spin(self):
19          for i in range(self.size):
20              for j in range(self.size):
21                  for k in range(self.size):
22                      vec = np.random.randn(3)
23                      vec = vec / np.linalg.norm(vec)
24                      self.lattice[i, j, k] = vec
25          return None
```

$initialize_spin()$把所有格点的自旋均设置为(0,0,1)，即全都朝向 z 轴方向。

$random_spin()$把所有格点的自旋随机化，自旋是一个模为 1 的随机向量，均匀分布在半径为 1 的球面上。随机化的方法是：先随机生成服从标准正态（standard normal）分布的 x, y, z，再将矢量$vec = (x, y, z)$归一化，这样的矢量vec就均匀分布在单位球面上，证明详见 http://corysimon.github.io/articles/uniformdistn-on-sphere/。

$initialize_spin()$对应低温状态，自旋有序排列；$random_spin()$对应高温状态，自旋无序随机排列。

定义 6 个方法：$x\_before(), x\_after(), y\_before(), y\_after(), z\_before(), z\_after()$。

```
    def x_before(self, x, y, z):
        if x < 1:
            return [self.size - 1, y, z]
        else:
            return [x - 1, y, z]

    def x_after(self, x, y, z):
        if x > self.size - 2:
            return [0, y, z]
        else:
            return [x + 1, y, z]
```

```
    def y_before(self, x, y, z):
        if y < 1:
            return [x, self.size - 1, z]
        else:
            return [x, y - 1, z]

    def y_after(self, x, y, z):
        if y > self.size - 2:
            return [x, 0, z]
        else:
            return [x, y + 1, z]
```

Heisenberg model 中，任意一格点均有 6 个相邻格点与其有相互作用，同时还要考虑到周期性边界条件，所以要定义 6 个函数来寻找满足周期性边界条件的“邻居”。

定义方法：$unit_E(x, y, z)$来计算坐标为$[x, y, z]$格点的相互作用能。

```
    def unit_E(self, x, y, z):
        [x1, y1, z1] = self.x_before(x, y, z)
        [x2, y2, z2] = self.x_after(x, y, z)
        [x3, y3, z3] = self.y_before(x, y, z)
        [x4, y4, z4] = self.y_after(x, y, z)
        [x5, y5, z5] = self.z_before(x, y, z)
        [x6, y6, z6] = self.z_after(x, y, z)
        spin = self.lattice[x1, y1, z1] + self.lattice[x2, y2, z2] + self.lattice[x3, y3, z3] \
            + self.lattice[x4, y4, z4] + self.lattice[x5, y5, z5] + self.lattice[x6, y6, z6]

        E = -self.J * self.lattice[x, y, z].dot(spin)
        return E
```

$[x_i, y_i, z_i]$ $(i = 1,2, \dots ,6)$分别代表 6 个相邻格点的坐标。由于相互作用能$E_{ij} = -J\overrightarrow{S_i} \cdot \overrightarrow{S_j}$，那么单个格点与其相邻格点所拥有的总相互作用能为$E_i = -J\overrightarrow{S_i} \cdot \sum_{j=1}^{6} \overrightarrow{S_j}$。函数中的变量$spin = \sum_{j=1}^{6} \overrightarrow{S_j}$，这样计算得到的$E$即为总相互作用能。

定义 4 个方法：$total\_E(), total\_M(), average\_E(), average\_M()$。

```
    def total_E(self):
        total_energy = 0

        for x in range(self.size):
            for y in range(self.size):
                for z in range(self.size):
                    total_energy += self.unit_E(x, y, z)
        return total_energy / 2

    def total_M(self):
        return np.sum(np.sum(np.sum(self.lattice,axis = 0),axis = 0),axis = 0)

    def average_E(self):
        E = self.total_E()
        return E / (self.size ** 3)

    def average_M(self):
        M = self.total_M()
        return M / (self.size ** 3)
```

$total_E()$计算体系的总能量，$total_M()$计算体系的总磁矩，$average_E()$计算体系的平均能量，$average_M()$计算体系的平均磁矩。

$total\_E = \frac{1}{2}\sum_i unit\_E$（相互作用能被计算了两次），$total\_M = \sum_{x,y,z} lattice[x,y,z]$，$average\_E = total\_E/size^3$，$average\_M = total\_M/size^3$。

定义方法：$Metropolis(temperature)$。Metropolis 方法，一次随机翻转一个 spin。由于玻尔兹曼常数k太小，在进行数值计算时，我们把kT当作一个整体T来考虑，即变量$temperature$。

```
98      # Metropolis method, flip a single spin for one time.
99      def Metropolis(self,temperature):
100
101         # Pick a spin to flip randomly.
102
103         x = np.random.randint(0, self.size)
104         y = np.random.randint(0, self.size)
105         z = np.random.randint(0, self.size)
106         original_spin = self.lattice[x, y, z].copy()
107
108         # The original energy.
109         E0 = self.unit_E(x, y, z)
110
111         vec = np.random.randn(3)
112         vec = vec / np.linalg.norm(vec)
113         self.lattice[x, y, z] = vec
114
115         # The energy after flip.
116         E1 = self.unit_E(x, y, z)
117         delta_E = E1 - E0
118
119         # Calculate the Metropolis acceptance rate.
120         if delta_E > 0:
121             if np.random.rand() > np.exp(-delta_E / temperature):
122                 self.lattice[x, y, z] = original_spin
123
124         return None
```

x,y,z是$[0,size-1]$范围内的随机整数，这样可以随机挑选出一个待翻转的格点。把原来的 spin 值保存下来（original_spin），再把翻转前的相互作用能保存下来（E_0）。随机生成一个均匀分布在单位球面上的矢量vec，将其作为新的 spin 值赋值给原来的格点，这样就完成一次翻转。但这次翻转有一定的 acceptance rate，这个概率需要通过能量差ΔE来计算。计算翻转后的相互作用能E_1，则能量差$delta_E = \Delta E = E_1 - E_0$。

被选中的点翻转的概率 acceptance rate $P = \begin{cases} 1 & \Delta E \le 0 \\ \exp\left(-\frac{\Delta E}{kT}\right) & \Delta E > 0 \end{cases}$，因此生成一个[0,1)

的随机数，只有当$\Delta E > 0$且随机数大于$-\frac{\Delta E}{kT}$时，这次翻转不被接受。只有在这种情况下，我们需要把格点的 spin 值改回原来的 original_spin，这样就不发生翻转，其他情况下均发生翻转。

```
127  size = 32
128  temperature = 1
129  J = 1
130  steps = 10000000
131
132  h = Heisenberg(size, J)
133  h.initialize_spin()
134  iteration = []
135  M = []
136
137  for i in range(steps):
138      h.Metropolis(temperature)
139
140      if (i+1) % 10000 == 0:
141          iteration.append(i+1)
142          M.append(h.average_M())
143
144      if (i+1) % 100000 == 0:
145          print(h.average_M())
146
147  plt.plot(iteration, M)
148  plt.show()
```

最后开始我们的 Metropolis 采样。我采用的晶格大小为$32 \times 32 \times 32$，温度$temperature$可变，$J = 1$，模拟的总次数为10^7次，在居里温度附近会遇到临界慢化现象，需要采用更多的模拟次数，最多2×10^7次。

模拟开始前生成一个Heisenberg类的实例 h，调用 h 的$initialize_spin()$方法，将所有格点的自旋初始化，模拟从低温有序向高温无序状态的演化。在每次模拟过程中记录平均磁矩随模拟次数的演化，每 1 万次模拟后记录一个平均磁矩的值，每 10 万次模拟后输出一个平均磁矩，以便在模拟过程中观察系统是否达到平衡状态。

最后，程序输出一个 x, y, z 方向的磁矩大小随模拟次数的变化图。

由于初始态所有格点的磁矩都是(0,0,1)，平均磁矩的初始值即为(0,0,1)，不同温度下模拟的结果如图所示，蓝线为 x 方向的磁矩大小，橙线为 y 方向，绿线为 z 方向。

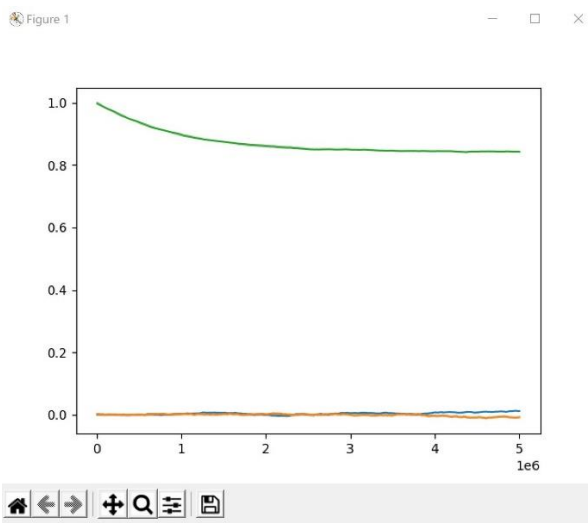


图 1 $T = 0.6$

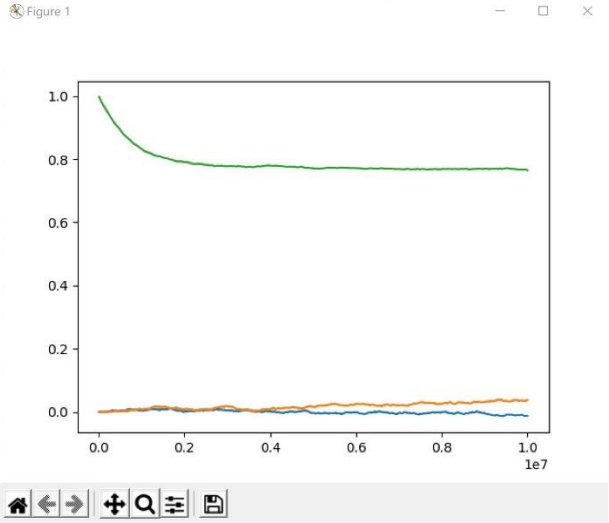


图 2 $T = 0.8$

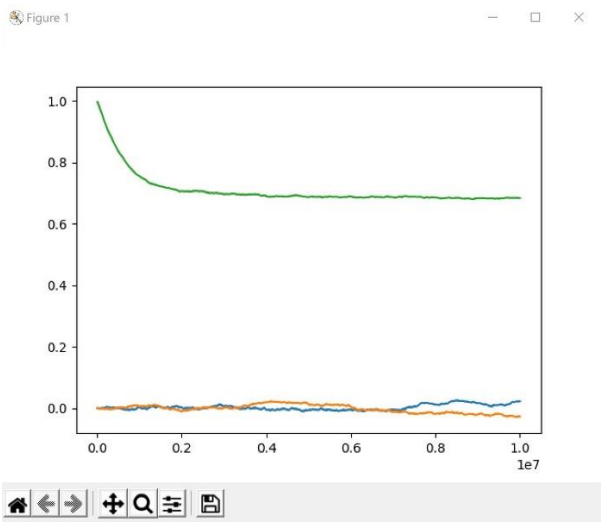


图 3 $T = 1.0$

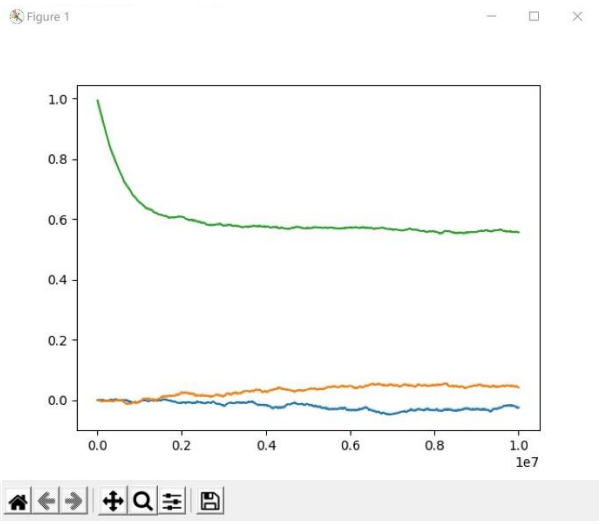


图 4 $T = 1.2$

可以看到，随着温度升高，平衡状态下的 z 方向磁矩M_z越来越小，但离居里温度还有一定的距离。居里温度下，M_z随模拟次数增多逐渐减小至 0 附近，与M_x, M_y无差别（达到各向同性）。

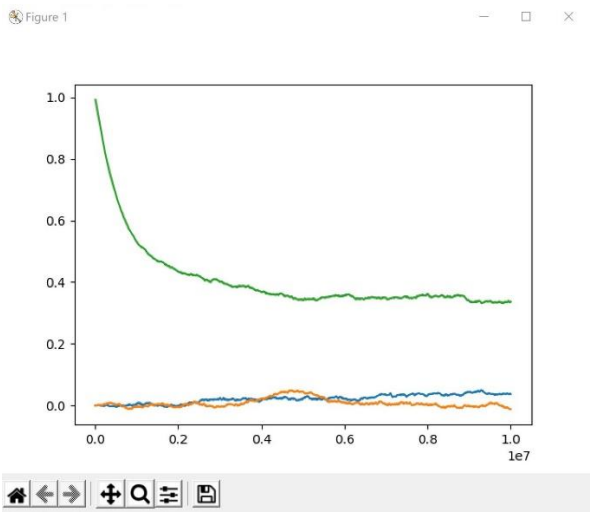


图 5 $T = 1.4$

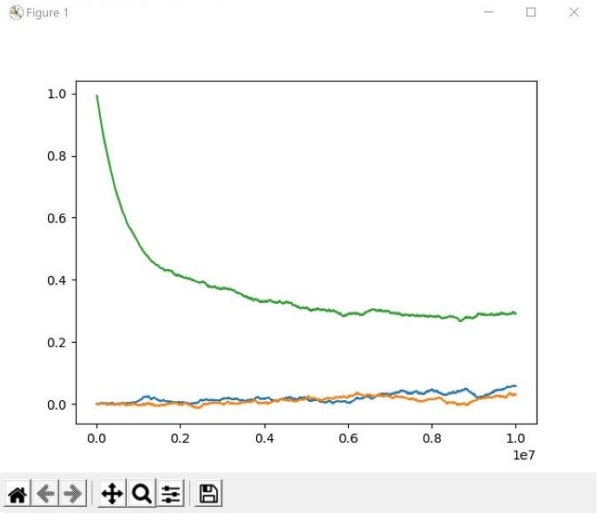


图 6 $T = 1.425$

平衡状态下M_z进一步降低，但达到平衡状态需要的模拟次数越来越多，临界慢化现象出现，需要进一步增加模拟的次数至 1500 万次。

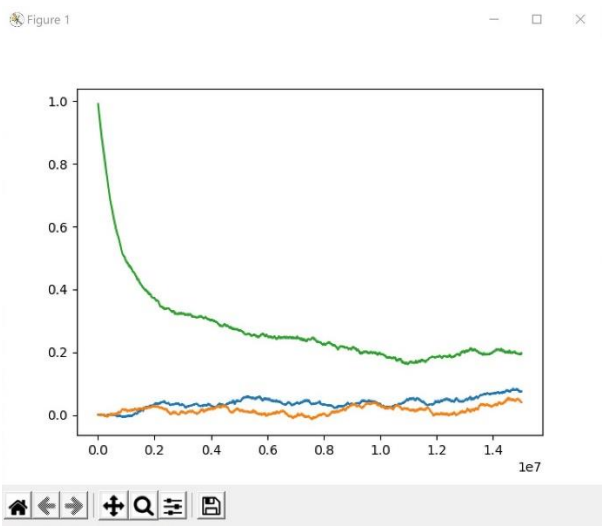


图 7 $T = 1.45$

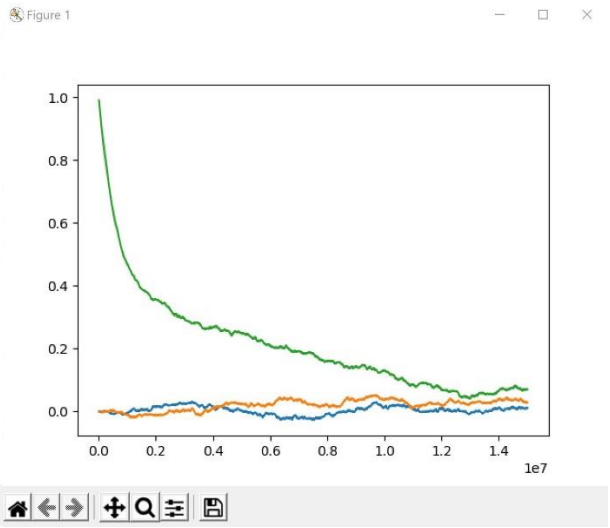


图 8 $T = 1.475$

可以看到在$T = 1.475$时，平衡状态下的M_z已经非常小（$M_z = 0.0702$），与M_x, M_y非常接近。由于临界慢化现象的存在，达到平衡所需要的模拟次数已经接近 1500 万次，对于后续模拟，需要进一步增加模拟次数至 2000 万次。

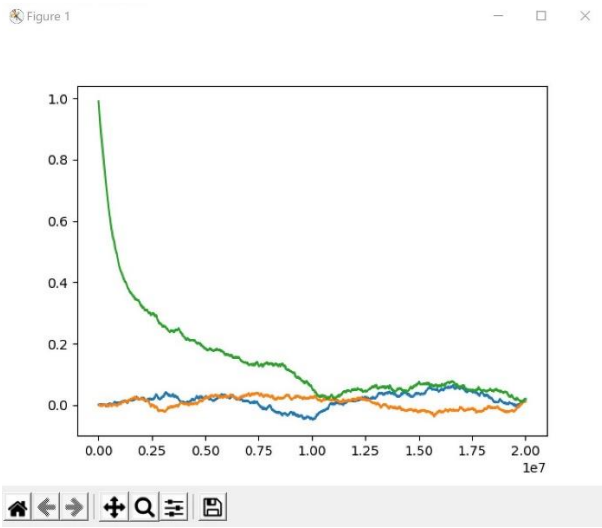


图 9 $T = 1.5$

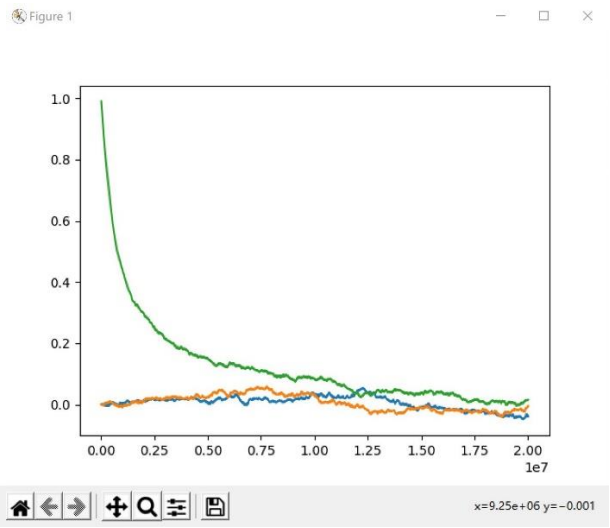


图 10 $T = 1.525$

在$T = 1.5$时，平衡状态下的$M_z = 0.0163$，已经可以认为是 0，M_z与M_x, M_y已经没有区别，体系达到各向同性。可以认为居里温度$T_c = 1.5$或居里温度在$1.475 - 1.5$之间。$T = 1.525$时，体系达到平衡时$M_z = 0.0141$，约等于 0。同样，体系也达到各向同性，M_z与M_x, M_y没有什么区别。

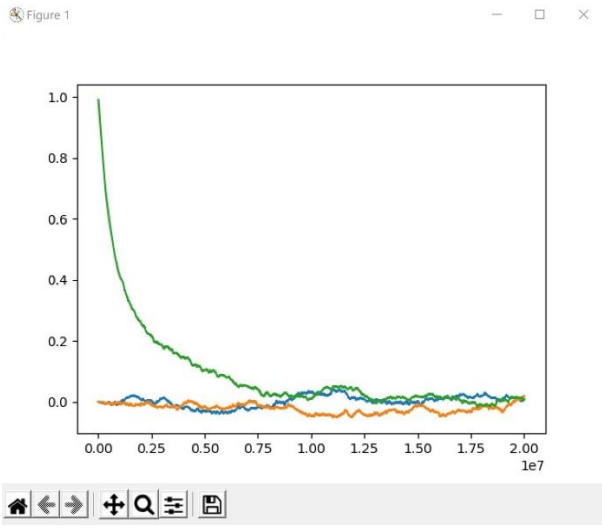


图 11 $T = 1.55$

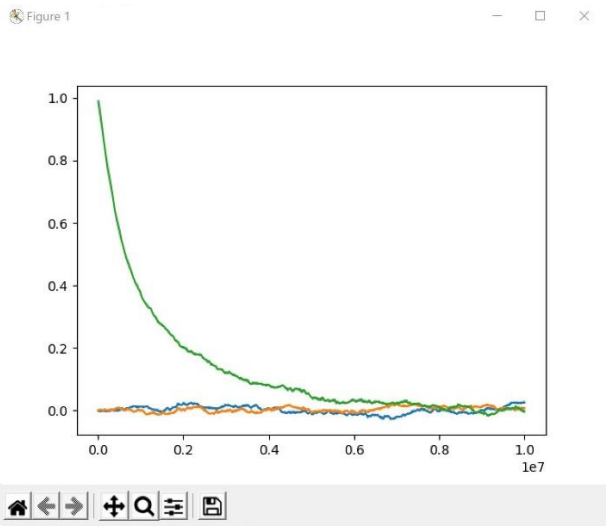


图 12 $T = 1.6$

在$T = 1.55$时，我仍然把模拟次数设为 2000 万次，但可以看到，在温度超过居里温度T_c后，体系更快地达到平衡状态。在大约 1000 万次左右的时候，我们就可以看到M_x超过了M_z，且M_x, M_y, M_z都约等于 0。所以在$T = 1.6$的情况下，我将模拟次数调整回 1000 万次。可以看到，体系更快地达到了平衡（800 万次左右），相应的，M_z也减小得更快。

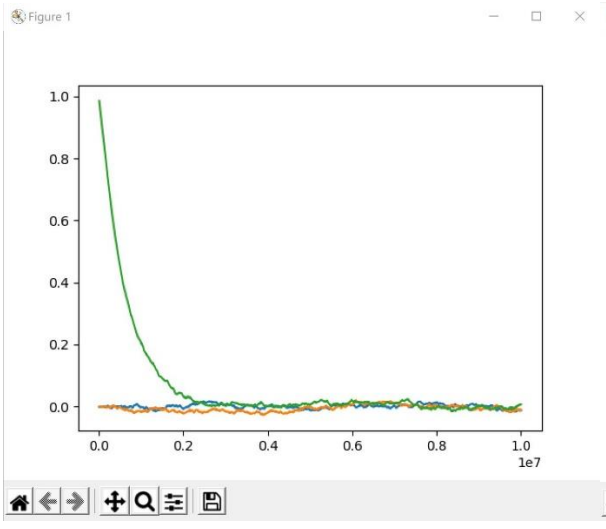


图 13 $T = 1.8$

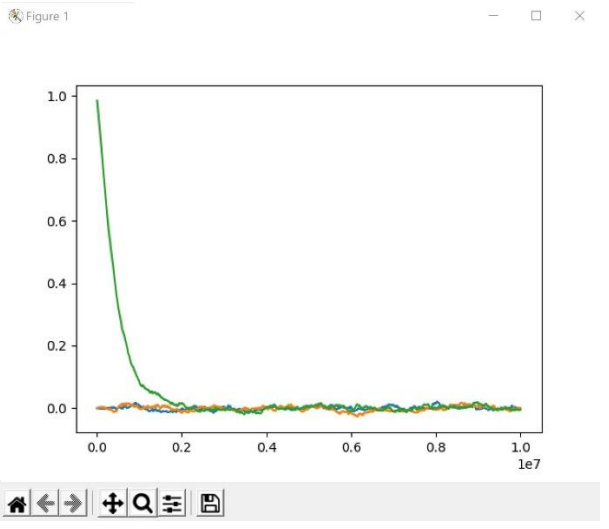
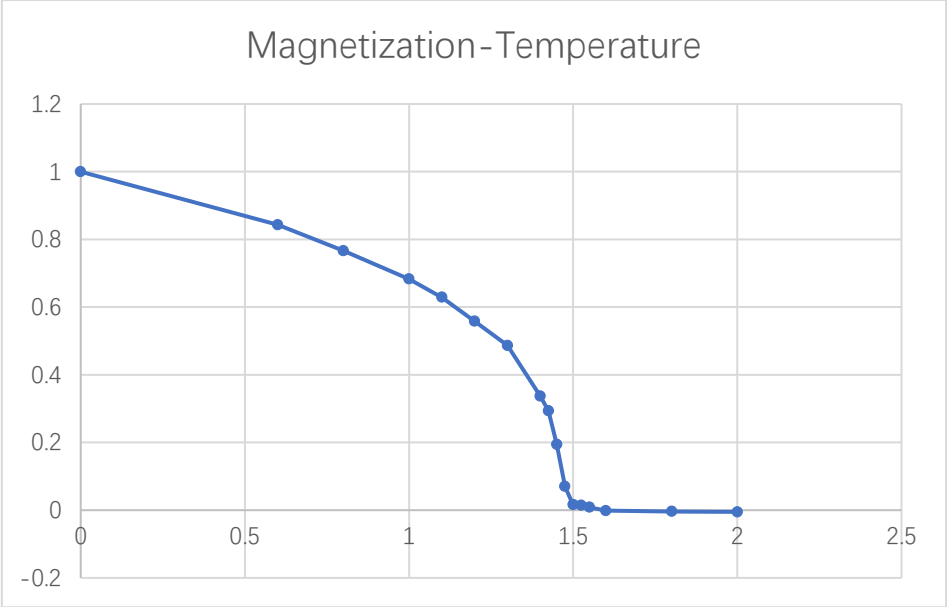


图 14 $T = 2.0$

在$T = 1.8$和$T = 2.0$时，体系迅速达到平衡状态（200 万次左右），M_x, M_y, M_z已看不出任何区别，体系达到各向同性。

可以得到结论，3D Heisenberg spin model 的居里温度$T_c = 1.5$。将平均磁矩M_z随温度T的变化关系作图，得到结果如图所示：



详细数据见下表：

T	0.6	0.8	1	1.1	1.2	1.3	1.4	1.425
M	0.8432	0.7664	0.683	0.629	0.5579	0.4867	0.3369	0.2935
T	1.45	1.475	1.5	1.525	1.55	1.6	1.8	2
M	0.1945	0.0702	0.0163	0.0141	0.0086	-0.0014	-0.0036	-0.0049

讨论：

用 Python 编写 Heisenberg model 并进行 Metropolis 采样，总体来说运行速度较慢。一个温度下进行 1000 万次采样需要花费 10 多分钟，遇到临界慢化现象采样 2000 万次更需要 20 多分钟，效率不高。但是 Python 在数据可视化方面具备一定的优势，可以更直观地判断体系是否达到平衡，也可以清晰地观察到临界慢化现象。由于数据量整体来说不大，所以最终还是选择了 Python 来完成模拟。

Input: $size,\ J,\ temperature$

Output: $the\ average\ magnetization\ of\ 3D\ Heisenberg\ model$

1. $lattice \leftarrow size \times size \times size \times 3\ matrix$
2. **Function** $initialize\_spin()$
3. **For** $i \leftarrow 1\ to\ size$ **Do**
4. **For** $j \leftarrow 1\ to\ size$ **Do**
5. **For** $k \leftarrow 1\ to\ size$ **Do**
6. $lattice[i, j, k] \leftarrow (0,0,1)$
7. **Return** None
8. **End Function**
9.
10. **Function** $x\_before(x, y, z)$
11. **If** $x < 1$ **Then**
12. **Return** $[size, y, z]$
13. **Else**
14. **Return** $[x - 1, y, z]$
15. **End If**
16. **End Function**
17. /* Similarly, we can get $x\_after(), y\_before(), y\_after(), z\_before(), z\_after()$ */
18.
19. **Function** $unit\_E(x, y, z)$
20. $[x_1, y_1, z_1] \leftarrow x\_before(x, y, z)$
21. $[x_2, y_2, z_2] \leftarrow x\_after(x, y, z)$
22. $[x_3, y_3, z_3] \leftarrow y\_before(x, y, z)$
23. $[x_4, y_4, z_4] \leftarrow y\_after(x, y, z)$
24. $[x_5, y_5, z_5] \leftarrow z\_before(x, y, z)$
25. $[x_6, y_6, z_6] \leftarrow z\_after(x, y, z)$
26. $spin \leftarrow (0,0,0)$
27. **For** $i \leftarrow 1\ to\ 6$ **Do**
28. $spin \leftarrow spin + lattice[x_i, y_i, z_i]$

29. $E \leftarrow -J \times lattice[x,y,z] \cdot spin$
30. **Return** E
31. **End Function**
32.
33. **Function** $average\_M()$
34. $M \leftarrow (0,0,0)$
35. **For** $i \leftarrow 1$ to $size$ **Do**
36. **For** $j \leftarrow 1$ to $size$ **Do**
37. **For** $k \leftarrow 1$ to $size$ **Do**
38. $M \leftarrow M + lattice[i,j,k]$
39. **Return** $M/size^3$
40. **End Function**
41.
42. **Function** $Metropolis(temperature)$
43. $x,y,z \leftarrow random\ integer \in [1, size]$
44. $original\_spin \leftarrow lattice[x,y,z]$
45. $E_0 \leftarrow unit\_E(x,y,z)$
46. $x',y',z' \leftarrow random\ number \in standard\ normal\ distribution$
47. $lattice[x,y,z] \leftarrow (x',y',z')/|(x',y',z')|$
48. $E_1 \leftarrow unit\_E(x,y,z)$
49. $\Delta E \leftarrow E_1 - E_0$
50. **If** $\Delta E > 0$ **Then**
51. $n \leftarrow random\ number \in [0,1)$
52. **If** $n > \exp\left(-\frac{\Delta E}{temperature}\right)$ **Then**
53. $lattice[x,y,z] \leftarrow original\_spin$
54. **Return** None
55. **End Function**