代码文件说明：
‘simplespin1.m’　simple-spin flip 随机模拟方法的代码
‘wolff2.m’，‘wolff3.m’，‘wolff4.m’，三种不同写法实现的 wolff 算法随机模拟代码

1. 算法的解释与选择：对比 simple-spin flip 的随机模拟结果和 3 种 wolff 算法的模拟结果

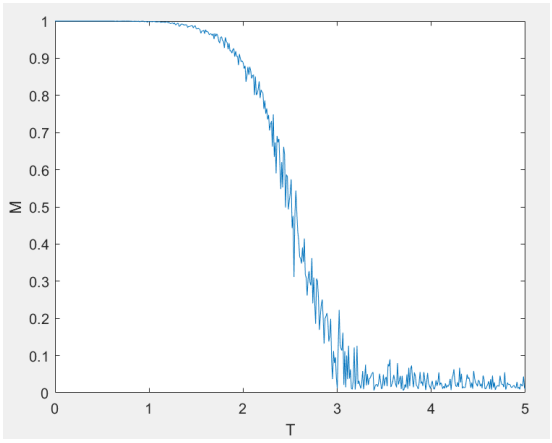
基本假设：玻尔兹曼常数$k_B = 1$，交换耦合能 J=1，T 取[0,5]，步长 0.01，初始状态取自旋全部向上的铁磁态，warmup 和 measure 次数根据每次运行时间长短进行调整，使得总体运行时间在可控范围内（基本不超过两小时），用马尔科夫-蒙特卡洛随机模拟 L=64 的二维伊辛模型，最终导出每个温度下 measure 次平均磁化强度 M，磁化率χ，平均总能量 E。其中磁化强度 M、平均总能量 E 均指对于系统最大值的相对值，*磁化率*$\chi = \frac{<M^2>-<M>^2}{k_B T}$，衡量系统涨落幅度。

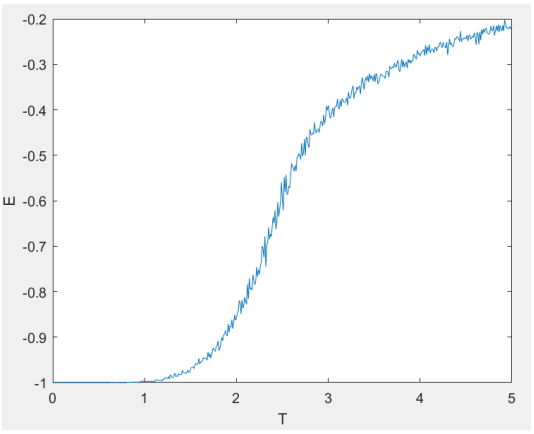
下面对 4 种算法的核心部分进行解释，并展示结果和分析

① simplespin1

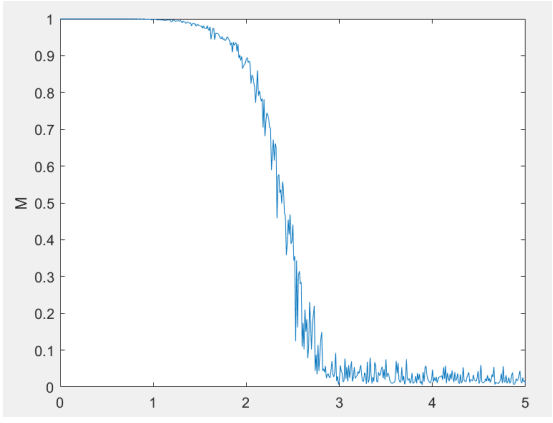
对于每一次迭代
1. 随机选取一个点
2. 对该点上下左右四个点的自旋态进行判定，计算该点与相邻点的作用能，记为 ei
3. 计算该点翻转后与相邻点的作用能，记为 ej，$de = ej - ei$
4. 如果$de < 0$，则翻转该点，如果$de > 0$，则有$p = e^{\beta * de}$概率翻转

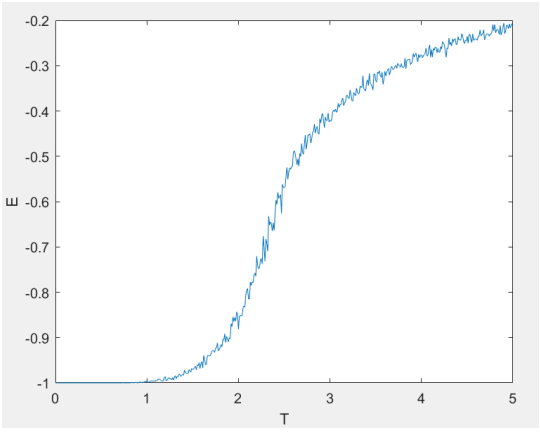
结果如下：
warmup=100000，measure=10000，耗时 20min





warmup=200000，measure=10000，耗时 30min





分析：

1. 由于 simple-spin flip 算法每次最多只改变一个电子的自旋，因此对于数量不够充分大的 measure 次数，它们的终态可能相差并不算太大，因此每个温度下平均的 M 与 E 也具有较大的波动
2. 随着 warmup 次数的增加，末态随温度由铁磁态转变为非铁磁态的过程加快，即更像是随着温度增加而突变而非渐变了，直观表述就是 M-T 图像中下降的那部分变得更陡了，但由于 warmup 次数不能无穷大，simple-spin flip 算法对于临界温度T_c的判断是很难达到准确的
3. 正如 1 中提到的，simple-spin flip 算法所 measure 的终态相差不会太大，因此测得的磁化率χ不具备太大参考价值，进一步探讨还需要用 wolff 算法

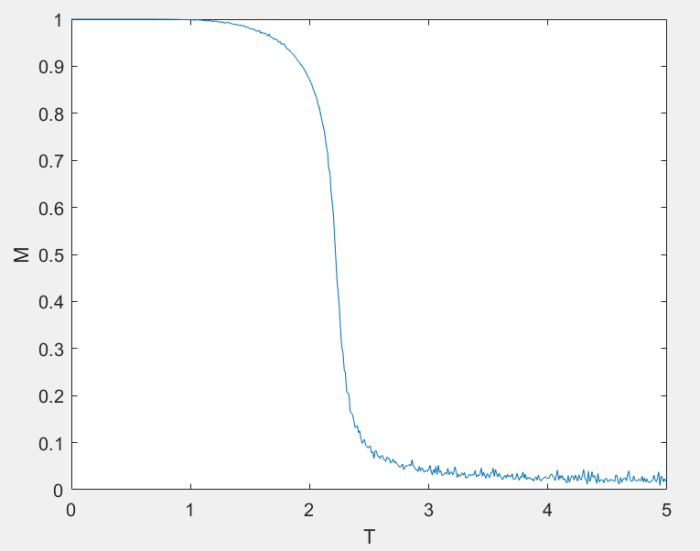
② wolff2

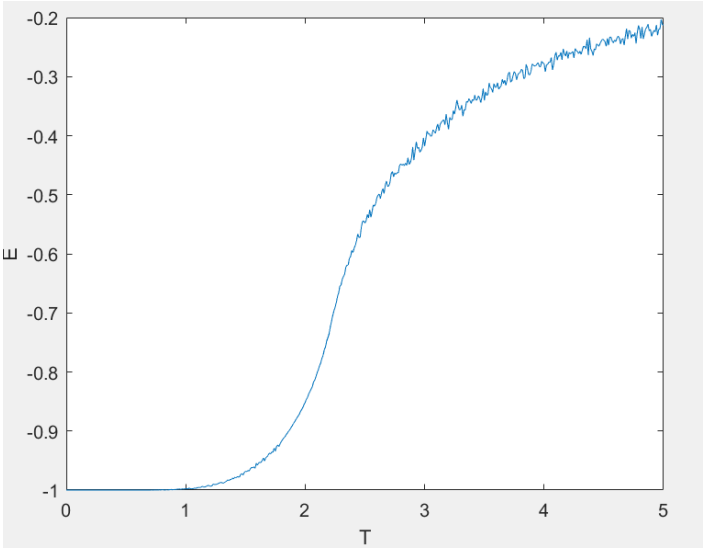
核心代码及解释如图

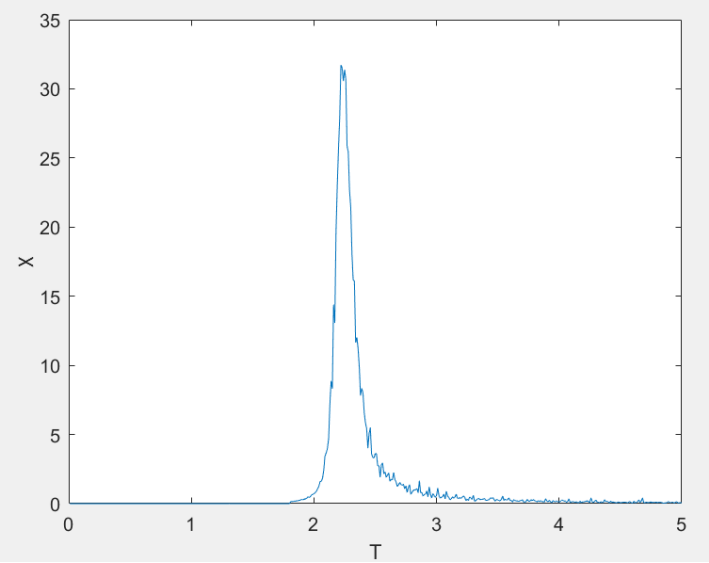
```
for i = 1:warm
    x = randi(1);
    y = randi(1);
    % 随机选点
    cluster = [x,y];
    % cluster： 所有要翻转点坐标
    new = [x,y];
    % new： 上一轮加入cluster的点坐标
    newn = 1;
    % newn： 上一轮加入cluster的点个数
    mm = zeros(1);
    mm(x,y) = 1;
    % mm： cluster的图
    while newn > 0
        neww = [];
        % neww： 这一轮加入cluster的点坐标
        for j = 1:newn
            % 对于每一个上一轮加入的坐标，判定上下左右四个邻居
            if new(j,1)+1 <= l
                % 判定邻居是否超出边界
                if mm(new(j,1)+1,new(j,2)) == 0
                    % 判定邻居是否已经在cluster里
                    if m(new(j,1)+1,new(j,2)) == m(new(j,1),new(j,2))
                    if m(new(j,1)+1,new(j,2)) == m(new(j,1),new(j,2))
                        % 判定邻居是否与本身自旋相同
                        if rand(1) < p
                            % 取P_add概率
                            neww = [neww;new(j,1)+1,new(j,2)];
                            % 将邻居坐标加入neww
                            mm(new(j,1)+1,new(j,2)) = 1;
                            % 更新cluster的图
                        end
                    end
                end
            end
...
        new = neww;
        cluster = [cluster;neww];
        if isempty(neww) == 0
            newn = length(neww(:,1));
        else
            newn = 0;
        end
        % 更新new, cluster, newn
    end

    for k = 1:length(cluster(:,1))
        m(cluster(k,1),cluster(k,2)) = 1 - m(cluster(k,1),cluster(k,2));
        % 翻转cluster中每一个坐标对应的点
    end
```

tips:
由于在低温状态下，p_{add}趋近于 1，铁磁态所有自旋又几乎都相同，因此每次迭代的 cluster 都几乎会扩张到整个模型的大小，非常耗费运行时间，可以适当减少 warmup 和 measure 次数。而高温状态下则反之，每次迭代的 cluster 都很小，但却需要很多次迭代才能从铁磁态转变为非铁磁态，应适当增加 warmup 次数。

结果如下:
运行时间 50 分钟







除高温状态下略有波动外结果良好，可以明显地看出，M、E 的导数最大处，χ的最大值处都在约 T=2.27 处（实际由于尺寸有限，比真实临界温度略小）

③ wolff3（失败）

由于在 wolff2 中，循环较大，运算速度不是特别理想，因此试图在 wolff2 的基础上进行改进，将其循环以矩阵形式计算。通过先将整个 cluster 的自旋相同边界判断出来，再用一个 L*L 的随机数矩阵判定出来是否要将边界加入 cluster

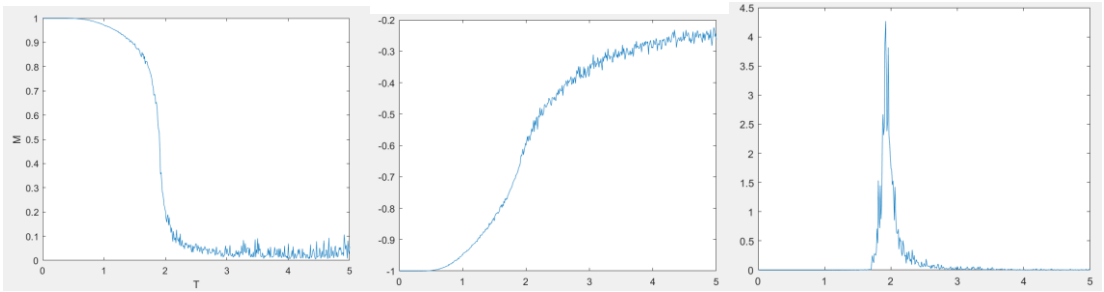
核心代码及解释如图

```
for i = 1:warm
    x = randi(l);
    y = randi(l);
    % 随机取点
    m1 = zeros(l);
    m2 = zeros(l);
    m1(x,y) = 1;
    % m1为cluster里所有点的图
    m2(x,y) = 1;
    % m2为上一轮加入的点的图
    len = 1;
    % len:  上一轮加入点个数
    while len>0
        m3 = zeros(l);
        for j = 1:len
            % 对于每个上一轮加入点，判定上下左右四个邻居
            if x(j)+1 <= l
                % 判定邻居是否超出边界
                if m(x(j),y(j)) == m(x(j)+1,y(j))
                    % 判定邻居是否与本身自旋相同
                    m3(x(j)+1,y(j)) = 1;
                    % 将邻居点画到图m3中
                end
            end
```

…

```
            m4 = m3 - m1 - rand(l)/p;
            % 图m4为m3(这一轮新加入所有邻居，可能包含这一轮前就有的点)减去m1(这一轮前的点)减去在[0,1/p]区间随机矩阵
            m4(m4 > 0) = 1;
            m4(m4 <= 0) = 0;
            % 如果m4>0则保留此邻居，反之则不保留，得到的就是这一轮要加入cluster的点
            m2 = m4;
            % 更新上一轮加入点的图
            [x,y] = find(m2==1);
            len = length(x);
            % 查找上一轮加入点的坐标并计算个数
            m1 = m4 + m1;
            % 更新整个cluster的图
        end
        m = m1-m;
        m(m == -1) = 1;
        % 将cluster的图减去原图再取绝对值，使得cluster内的点0,1互换
    end
```

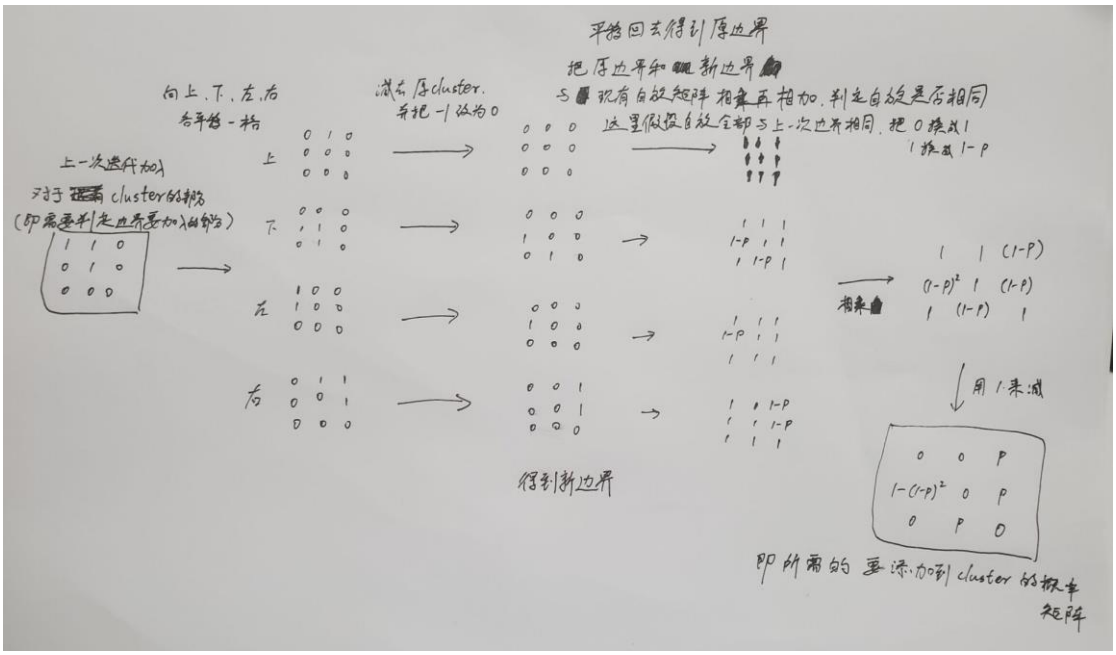
结果如下：(运行时间 10 分钟)



发现，在 T=2 的时候末态就从铁磁态转化为了非铁磁态，与实际情况的$T_c = 2.27$不符，想了很久终于发现原因：对于一个同时与多个 cluster 内电子相邻的电子(即 cluster 呈 L 形或 U 形时的边界)，正常 wolff 算法对这个电子应该判定多次决定是否加入 cluster，而我的 wolff2 算法里只判定了一次。在此基础上改进出了完全矩阵化的算法 wolff4，正是这个概率的不同导致了临界温度的错误和所需迭代次数的增加

④ wolff4

对于新加入 cluster 的判定完全使用矩阵而非循环的方法。对于上一次加入到 cluster 的需要再判断其边界是否要加入的部分，通过矩阵的平移和加减得到其上下左右边界的矩阵，判定自旋是否相同后将可能加入的点设为 1-p，不会加入的点设为 1，把上下左右边界矩阵点乘起来再用 1 减，即是要添加到 cluster 的概率矩阵，再生成一个 L*L 的随机数矩阵判定
示例如下



代码如下

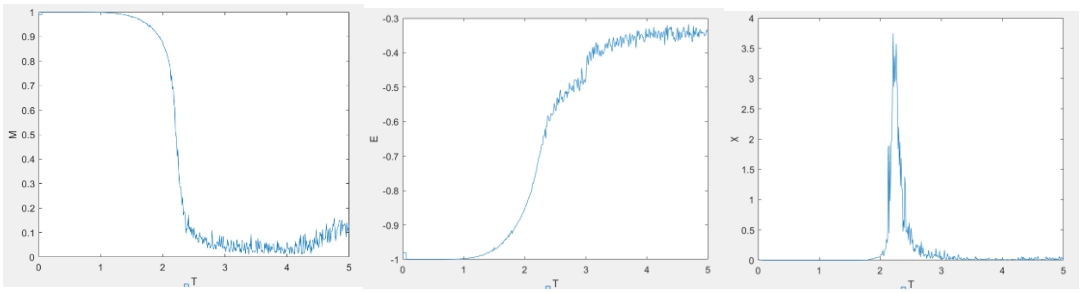
```
for i = 1:warm
    m0 = m;
    m0(m0 == 0) = -1;
    % m0:  原图转为1与-1的图
    x = randi(l);
    y = randi(l);
    % 随机取点
    m1 = zeros(l);
    m2 = zeros(l);
    m1(x,y) = 1;
    % m1为cluster里所有点的图
    m2(x,y) = 1;
    % m2为上一轮加入的点的图
    is = 1;
    % is:  上一轮是否有新点加入
    while is == 1
        mm1 = [m2(2:end,:);zeros(1,l)];
        % 将上一轮新加入的点的图向上下左右平移一格
        mm1 = mm1 - m1;
        mm1(mm1 == -1) = 0;
        % mm1:  平移后边界图
        mm11 = [zeros(1,l);mm1(1:end-1,:)];

       % mm11:  平移前边界图
       mm1 = mm1 .* m0;
       mm11 = mm11 .* m0;
       % 将前后边界转化为1,-1的边界(表示自旋)
       mm111 = [mm11(2:end,:);zeros(1,l)];
       % 将带有自旋信息的平移前边界再次平移
       mm1 = mm1 + mm111;
       mm1(mm1 == 2) = q;
       mm1(mm1 == -2) = q;
       mm1(mm1 == 0) = 1;
       % 将2和-2(平移前后自旋相同)转为1-p，将0转为1
```

...

```
            m3 = ones(l) - (mm1 .* mm2 .* mm3 .* mm4);
            % m3为新加入点的概率图
            m4 = m3 - rand(l);
            m4(m4 > 0) = 1;
            m4(m4 <= 0) = 0;
            % m4为新加入点图
            m2 = m4;
            % 更新上一轮加入点的图
            m1 = m1 + m4;
            % 更新整个cluster的图
            is = ismember(1,m4);
            % 判定是否存在新加入点
        end
        m = m1-m;
        m(m == -1) = 1;
        % 将cluster的图减去原图再取绝对值，使得cluster内的点0,1互换
    end
```

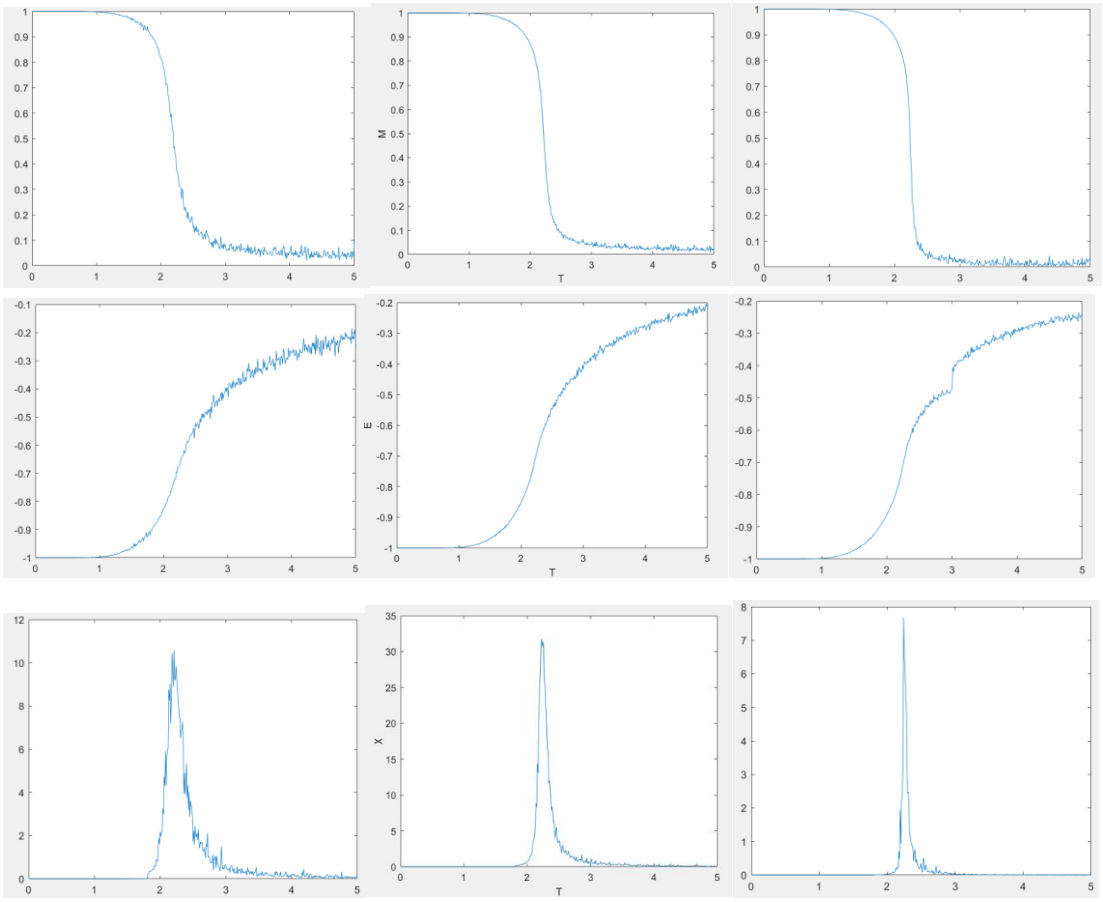
实际结果表明，其运行速度不如算法 wolff2，可能是由于矩阵操作太多导致的。(至少在这个问题里，矩阵化后的速度不如做循环)

不过结果还是对的，（由于速度过慢，设置 measure 次数较少，高温状况波动严重） 运行时间 30 分钟



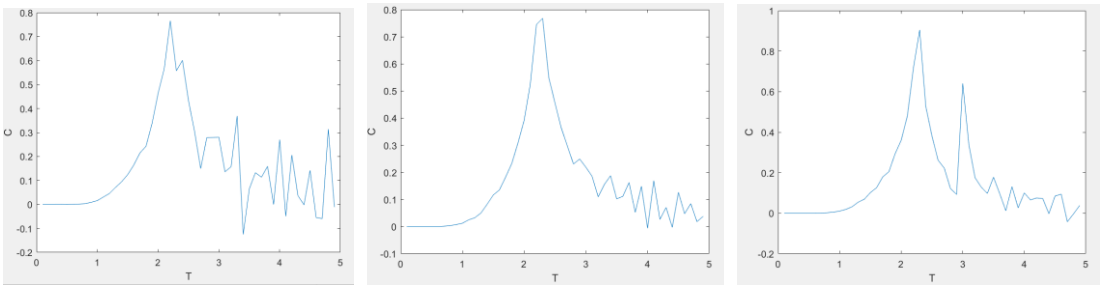
综上，实际结果表明，wolff2 是四个算法中性能最优良的一个，之后的探讨也会基于 wolff2 的结果来进行

采用 wolff2 算法，对 32*32, 64*64, 128*128 的模型分别计算，结果如下：
（从左到右依次是 32*32, 64*64, 128*128，运行时间分别为 3min,50min,90min 从上到下依次是平均磁化强度 M，平均总能量 E，磁化率χ）



可以看到，随着模型尺寸的增大，磁化强度更容易趋近于 0,高温状态下的能量更难趋近于 0，波动变小（说明尺寸大的模型在高温状态未态还可能存在一定成块的现象，虽然总磁化强度趋于 0，但不是完全杂乱无章），磁化率峰宽变窄，（说明尺寸大的模型在临界温度时突变更陡）
注意到在 128*128 的 T=3 处平均总能量 E 发生突变，是因为我在 T=3 处增大了 warmup 次数，反过来说明在那张图 T=3 的前一截，warmup 是不够充分的

把 E 的结果每 10 个一组求平均，然后求差值，即每个步长下 E 随 T 的变化值，即可估测出热容$C = \frac{\partial E}{\partial T}$
从左到右依次是 32*32, 64*64, 128*128 的热容



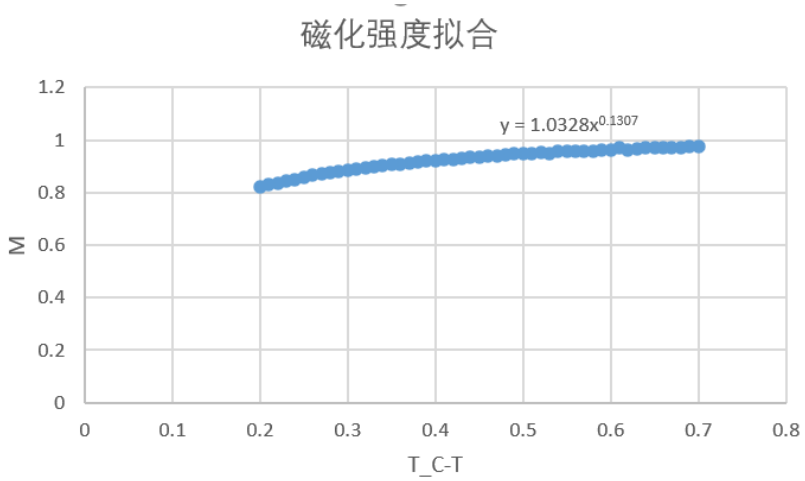
由于步长取得较大，也没有做函数拟合，因此波动较大，（L=128 的第二个峰是因为上面提到的 warmup 次数不足导致的）不过还是可以很清晰的得出热容最大点，即**真实临界温度位于 2.27±0.02 处**

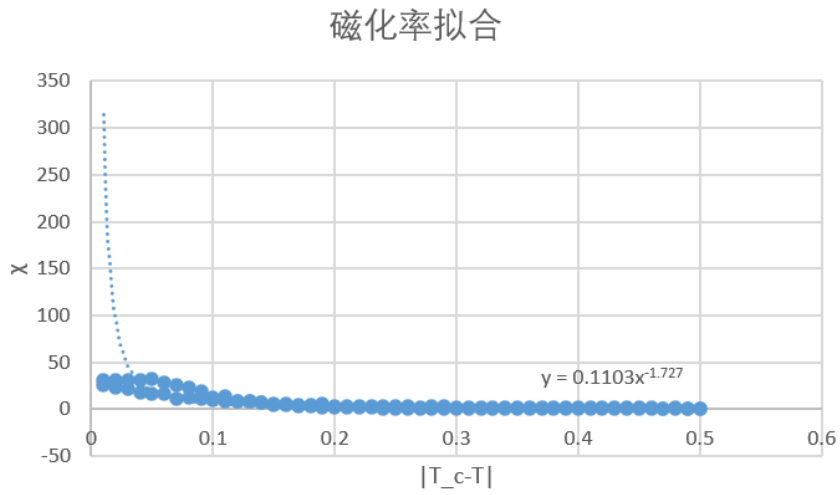
2. 在临界温度附近拟合磁化强度 M、磁化率χ

$$M(T) \sim (T_c - T)^{\beta} \text{ for } T < T_c$$
$$\chi(T) \sim |T - T_c|^{-\gamma}$$

用 L=64 的结果进行拟合，将 代入，取临界温度$T_c = 2.27$

拟合结果如图所示（当温度过分趋近于临界温度时，即$T_C - T$趋近于 0 时，M 趋近于 0，χ 趋近于正无穷。但在随机模拟中，由于模型尺寸和迭代次数不能无限，也不可能完全拟合出 M=0 和χ无穷大的结果，因此在拟合中去掉那一部分过分接近于临界温度的温度）



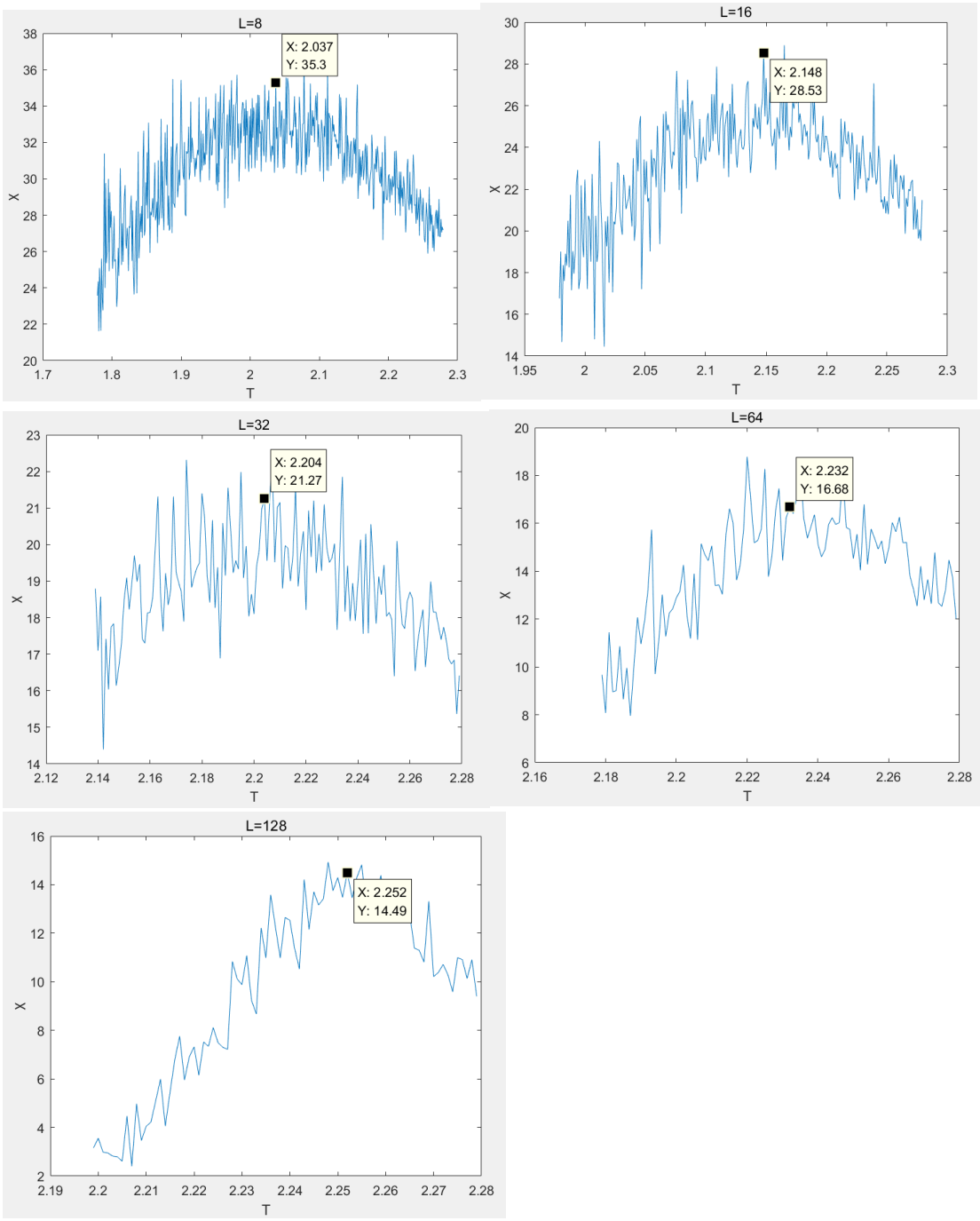


拟合结果为β=0.1307，γ=1.727

与理论值β=1/8，γ=7/4 存在一定误差

3. 模型大小与逼近临界温度的关系

首先，对于五个不同尺寸的模型（L=8,16,32,64,128），在临界温度附近对于磁化率χ做更精细步长更短的模拟，找出使得磁化率χ达到最大所对应的温度T_{max}，T_{max}应该略小于T_C（这个差值是由于模型尺寸有限，在相变点附近，关联长度ξ=$|T-T_C|^{-\nu}$。对于尺寸越大的模型，这个差值应该越小），模拟结果如下



由于步长很小，磁化率χ的绝对大小又很大，因此此处图像相对看起来波动较大，手动选取模拟结果弧顶处对应的温度值T_{max}（而非实际磁化率χ最大对应的 T 值，可能是模拟误差导致的细微偏移）（同理，后面要取的χ(T_{max})也是手动估计）

由结果可以外推出，当 L 趋近于正无穷时，T_{max}会趋近于临界温度T_C，即实际情况

由关系式$|T_{max}-T_C|^{-\nu} \propto L$可推导出

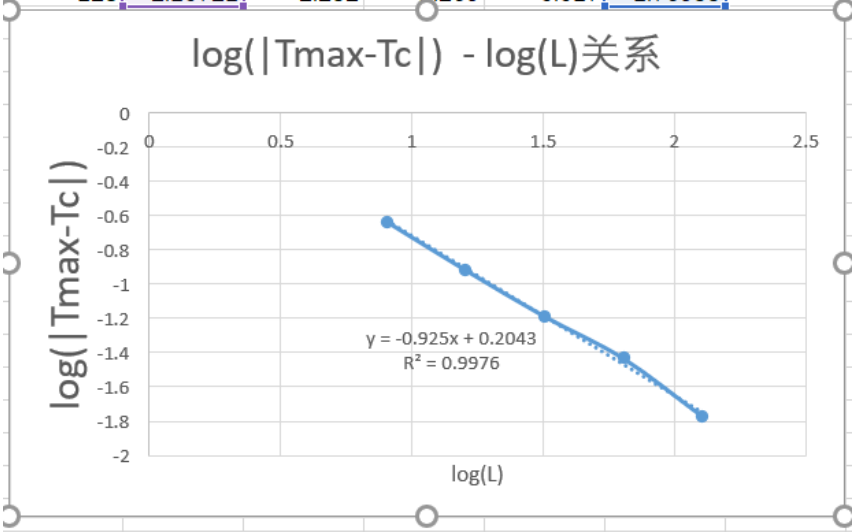
$$|T_{max}-T_C| = Constant * L^{-\frac{1}{\nu}}$$

两边取对数得

$$log|T_{max}-T_C| = Constant - \frac{1}{\nu}\log(L)$$

取临界温度$T_C = 2.269$，对五组 L 和T_{max}进行拟合，结果如图所示

L	log(L)	Tmax	Tc	|Tmax-Tc|	log(|Tmax-Tc|)
8	0.90309	2.037	2.269	0.232	-0.63451
16	1.20412	2.148	2.269	0.121	-0.91721
32	1.50515	2.204	2.269	0.065	-1.18709
64	1.80618	2.232	2.269	0.037	-1.4318
128	2.10721	2.252	2.269	0.017	-1.76955



v=-1/-0.925=1.0811

线性性非常良好，计算得到$\nu = 1.0811$

$$M(T) \sim (T_c - T)^{\beta} \propto L^{-\beta/\nu} \text{ for } T < T_c$$
$$\chi(T) \sim |T - T_c|^{-\gamma} \propto L^{-\gamma/\nu}.$$

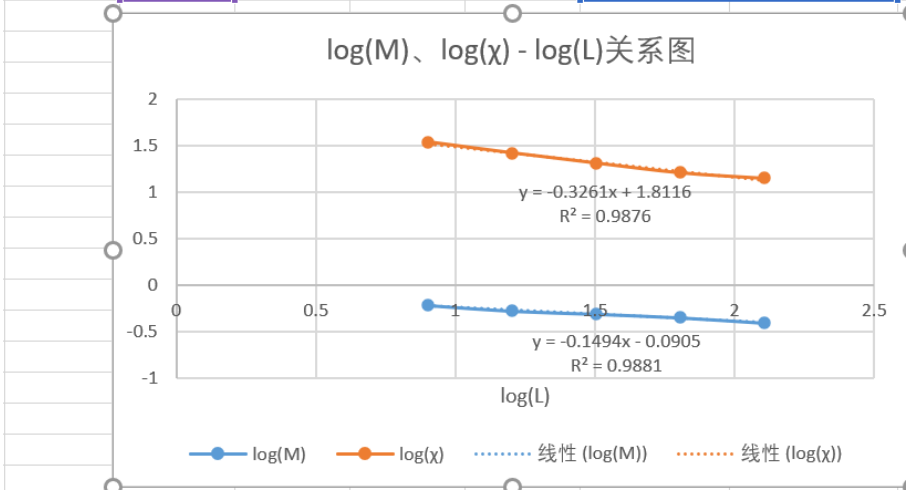
由关系式 可以推导出

$$log(\mathrm{M}(T_{max})) = Constant - \frac{\beta}{\nu}\log(L)$$

$$log(\chi(T_{max})) = Constant - \frac{\gamma}{\nu}\log(L)$$

将T_{max}代入结果图得到$\mathrm{M}(T_{max})$，根据图像手动估测$\chi(T_{max})$，并将其与 L 拟合，结果如下

L	log(L)	Tmax	M(Tmax)	χ(Tmax)	log(M(Tmax))	log(χ(Tmax))
8	0.90309	2.037	0.6027	34	-0.2198988	1.531478917
16	1.20412	2.148	0.5228	26	-0.2816644	1.414973348
32	1.50515	2.204	0.4857	20.3	-0.3136319	1.307496038
64	1.80618	2.232	0.4444	16	-0.3522259	1.204119983
128	2.10721	2.252	0.3895	14	-0.4094925	1.146128036



$-\frac{\beta}{\nu} = -0.1494$，$-\frac{\gamma}{\nu} = 0.3261$，代入$\nu = 1.0811$，得到β=0.1615，误差较小，γ=0.3525 误差较大，原因有待进一步探索。