The Potts Model

Yihang Chen 1700010780

December 9, 2019

Abstract

In this paper, I use the Metropolis-Hastings algorithm and the Wolff algorithm to investigate critical temperature and several critical exponents in the Potts model. Despite the fact that **my code is designed for all dimensions** d **and all** q, I only use it to investigate the cases where $d = 2, 3$, $q = 3$. I utilize Python to implement the algorithm for the sake of its flexibility, while at the expense of the efficiency. In addition, I use Weiming Teaching to perform onerous simulation.

Contents

1 Settings

The d-dimensional Potts model on the N^d square lattice with periodic boundary condition. The Hamiltonian of the q-state Potts model is defined as

$$H(\sigma) = -J \sum_{<i,j>} \delta_{\sigma_i \sigma_j} - h \sum_i \sigma_i, \quad i \in [N^d]$$

where $\sigma_i = [q]$. In this report, we only consider $d = 2.3$.
We define some quantities below:

1. Internal energy u: $u = U/N^d$, where $U =< H >$.
2. Specific heat c: $c = C/N^d$, where $C = k_B \beta^2 \mathrm{Var}(H)$
3. Magnetization m: $m = M/N^d$, where $M =< \sum_i \sigma_i >$
4. $\Gamma(k) = C(i,j)_{|i-j|=k}$, where $C(i,j) =< \sigma_i \sigma_j > - < \sigma_i >< \sigma_j >$.

Under designated conditions, we will perform a series of numerical experiments to[1]

1. For different T, plot m as function of h.
2. Assuming $h = 0$, find critical temperature $T^\star$.
3. Assuming $h = 0$, find the scaling exponents $\alpha,\ \gamma,\ \delta,\ \xi$.

We are to adopt the traditional Metropolis-Hastings algorithm and Wolff algorithm to deal with the problem. However, despite its efficiency, the Wolff algorithm requires $h = 0$. Hence, we will study $m - h$ relation only by the traditional Metropolis-Hastings Algorithm.

1.1 Comparison with the Ising model

If we set $q = 2$, then $\sigma_1 = 1, \sigma_2 = 2$. However, in the traditional Ising model require $\sigma_1' = -1, \sigma_2' = 1$. We use a substitution

$$\sigma_i' = 2\sigma_i - 3 \tag{1}$$

Hence

$$\sigma_i' \sigma_j' = 2\sigma_i \sigma_j - 1 \tag{2}$$

We can get

$$J_{Potts} = 2J_{Ising} \tag{3}$$

2 Parameter estimation method

2.1 Estimate u, c and m

Since

$$\begin{aligned} u &= \frac{< H >}{N^d} \\ c &= k_B \beta^2 \frac{< H^2 > - < H >^2}{N^d} \\ m &= \frac{< \sum \sigma_i >}{N^d} \end{aligned} \tag{4}$$

[1] Note: the order is different from the report.

We only need to estimate $< H >, < H^2 >$ and $< \sum \sigma_i >$.

$$\begin{aligned} < H > &\approx \frac{1}{T} \sum_{t=1}^{T} H(\Sigma_t), \\ < H^2 > &\approx \frac{1}{T} \sum_{t=1}^{T} H(\Sigma_t)^2 \\ < \sum \sigma_i > &\approx \frac{1}{T} \sum_{t=1}^{T} \sum (\Sigma_t)_i \end{aligned} \tag{5}$$

It is too slow to directly calculate Hamiltonian every time we get a new stage. Instead, we update it according to the local change around the flipped spin.

2.2 Estimate $\Gamma(k)$

Since $h = 0$, $< \sigma_i >= \frac{q+1}{2}$. Hence we only need to estimate $< \sigma_i \sigma_j >$.
We consider two points with distance k differing in only one coordinate. Define

$$D(\Sigma, k) = \frac{1}{2d} \sum_{(i,j)} \sigma_i \sigma_j \tag{6}$$

where the sum is taken over all pairs with distance k in one dimension. Under this approximation, we can estimate $\Gamma(k)$ as

$$\Gamma(r) \approx \frac{1}{N^d T} \sum_{t=1}^{T} D(\Sigma_t, k) \tag{7}$$

Here, we still update $\Gamma(k)$ according to the local change around the flipped spin.

2.3 Correlation length

According to [Wikipedia, 2019], for the Ising model, when the temperature is above the critical

$$\Gamma(r) \approx \frac{1}{r^{d-2+\mu}} \exp(-\frac{r}{\xi}) \tag{8}$$

The critical point is the place where $\xi = \infty$. When the temperature is below the critical

$$\Gamma(r) \approx \frac{1}{r^{d-2+\mu}} \exp(-\frac{r}{\xi}) + C \tag{9}$$

The above fact can be plotted in Figure 1.

We expect the Potts model behave similar to the Ising model. Hence, the relation $\Gamma(r) \propto \exp(-r/\xi)$ can only hold when $T > T_c$. When $T < T_c$, the relation can be reformulated as

$$\Gamma(r) = a \exp(-r/\xi) + C \tag{10}$$

We use package "scipy" to perform nonlinear regression on the result.

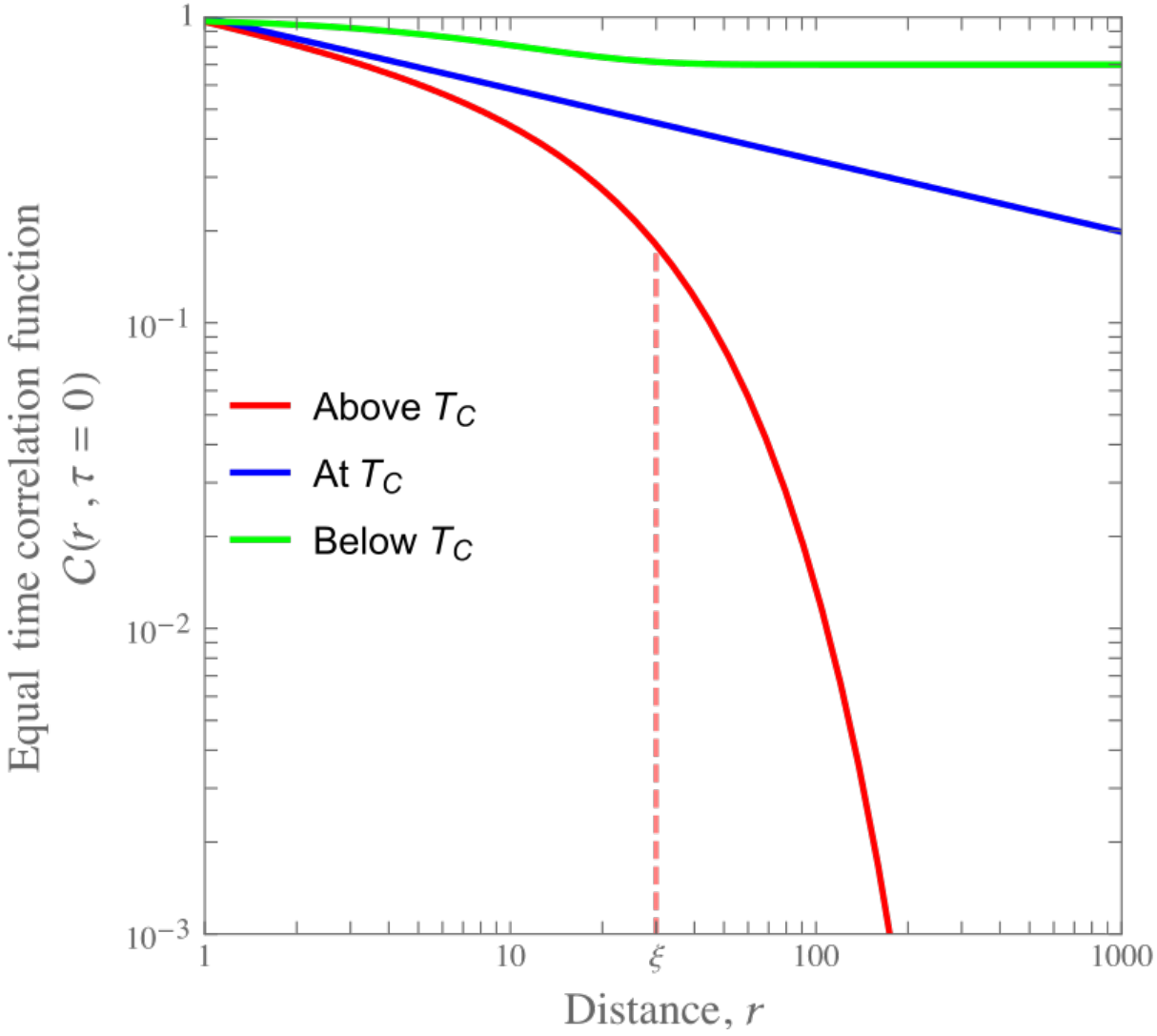


Figure 1: Correlation length, from [Wikipedia, 2019]

3 Simulation methods

3.1 Metropolis-Hastings Algorithm

We implemented the classical Metropolis algorithm with Python.

Algorithm 1 Metropolis-Hastings Algorithm

Require: J, h, k_B, T

1: $\beta = \frac{1}{k_B T}$
2: Initialize spins s_i within $[q]$ equally randomly.
3: Define $H(\sigma) = -J \sum_{<i,j>} \sigma_i \sigma_j - h \sum_i \sigma_i$.
4: **repeat**
5: Propose a state σ'.
6: Compute $\Delta H = H(\sigma') - H(\sigma_n)$, $A = \min\{1, \exp(-\beta \Delta H)\}$.
7: Generate R.V. $r \sim \mathcal{U}[0, 1]$.
8: If $r \leq A$, then $\sigma_{n+1} = \sigma'$; else $\sigma_{n+1} = \sigma_n$.
9: **until** convergence

3.2 Wolff's algorithm

The traditional Metropolis-Hastings algorithm is inefficient when $T < T_c$ due to high probability of rejection. Hence, the Metropolis-Hastings algorithm tend to converge to one specific framework when the temperature is low. To estimate the desired value with low variance, we need multiple workers. Hence, we propose a rejection-free model to overcome the obstacle. By the following algorithm, we only use one worker to perform simulation.
To get an stimulus for the algorithm, we rewrite $\pi(\sigma)$ below

$$\begin{aligned} \pi(\sigma) &\propto \exp(-\beta H(\sigma)) \\ &\propto \prod_{<ij>} \exp(\beta J \delta_{\sigma_i \sigma_j}) \end{aligned} \tag{11}$$

If we introduce an auxiliary variable u on each edge such that

$$\pi(\sigma, u) \propto \prod_{ij} I[0 \leq u_{ij} \leq \exp(\beta J \delta_{\sigma_i \sigma_j})] \tag{12}$$

The marginal distribution of σ is the Gibbs distribution. Conversely, u affects σ only through the event $I[u_{ij} > 1]$. Hence, we from the bond value 1 conditioned on the fact that $\sigma_i = \sigma_j$ and $u_{ij} > 1$. Since u_{ij} evenly distributed in the interval $[0, \exp(\beta J)]$. We set the bond value to 1 with probability $1 - \exp(-\beta J)$.
Some modifications need to be adopted to accommodate the Ising limit $q = 2$ case. The main difference is that in the Ising case, the choice of new cluster spin is obvious, but in the Potts case, the additional step of randomly choosing a new spin must be taken.

Algorithm 2 Wolff's algorithm

Require: $\beta, J, h = 0$

Initialize spins s_i within $[q]$ equally randomly.
repeat
    Randomly picks a spin s. Construct a set $C = \{s\}$. Denote $\hat{C}$ as the complementary of C in the lattice.
    repeat
        for each unchecked edge connecting C to $\hat{C}$ **do**
            Form the bond on the edge a bond value of 1 with probability $1 - \exp(\beta J)$ if two connected spins are the same, and a bond value of 0 otherwise.
            If the bond value equals 1, then add the spin to the cluster C.
        end for
    until all neighbours are checked.
    Flip all the spins corresponding to the sites in the final set to a new state (randomly chosen from the remaining $q - 1$ choices).
until convergence

We prove the algorithm satisfy the detailed balance condition.
Suppose in a configuration σ, a cluster C of spins a will be flipped to the cluster C' of spins b, denoted as configuration σ', and the neighbouring spins of C has m spins a and n spins b. Then, since the cluster is flipped iff new spins are not be added into C

$$P(\sigma \to \sigma') = (1-p)^m \tag{13}$$

Similarly

$$P(\sigma' \to \sigma) = (1-p)^n \tag{14}$$

Since $|C| = |C'|$

$$\frac{P(\sigma \to \sigma')}{P(\sigma' \to \sigma)} = (1-p)^{(m-n)} = \exp(-\beta J(m-n)) \tag{15}$$

Besides, (assuming $h = 0$)

$$\Delta H(\sigma) = H(\sigma') - H(\sigma) = -J(n-m) \tag{16}$$

Hence

$$\frac{P(\sigma \to \sigma')}{P(\sigma' \to \sigma)} = \exp(-\beta \Delta H(\sigma)) \tag{17}$$

3.3 Simulation settings

Here we propose a stage by uniformly randomly flipping spins. Before performing any measure measurement. We aimless perform 5×10^6 transformations as warming up for temperature below the critical temperature and 5×10^5 transformations for higher temperature. For each simulation, we calculate U, C, M and $\Gamma(k)$ for designated k. And measure the difference between two consecutive states, and we keep track of the number of the transformations (without warming up). If the number of transformations is larger than 10^6, or the number is larger than 5×10^3 and the difference mentioned above is smaller than 0.001, then we stop the loop and output the result.
The code is implemented in the function "Metroplis(args: dict)" and "Wolff(args: dict)"

4 Results and Analysis

4.1 Critical temperature

According to the survey [Wu, 1982], the critical temperature for $d = 2$ is

$$T_c = \frac{J}{\log(1 + \sqrt{q})} \tag{18}$$

Notice that it matches the Ising case. Explicit form of the critical temperature when $d = 3$ is unknown, but numerical results shows that $T_c = 1.82$. From the figures in the appendix, we can clearly observe that there is a drastic change of internal heat, and a clear peak, around the critical point.

Metropolis-Hastings algorithm When $d = 2$, we set $N = 20$, Figure 2 and Figure 3 separately depict the internal energy and specific heat; When $d = 3$, we set $N = 8$, Figure 6 and Figure 7 separately depict the internal energy and specific heat.

Wolff algorithm When $d = 2$, we set $N = 20$, Figure 18 and Figure 19 separately depict the internal energy and specific heat; When $d = 3$, we set $N = 8$, Figure 20 and Figure 21 separately depict the internal energy and specific heat.

The phenomenon can be explained as follows. When T is close to zero, after warming up, all states tend to be the same. Hence in future transformations, it is unlikely that the worker will escape such state due to large β. Hence, the variance is low. When T is large, despite the high variance of H, owing to small β, specific heat c is relatively small. However, around the critical temperature, β is not so small and due to relatively high temperature, the variance of H is also considerable, which together leads to the peak in the specific heat figure.
Since the specific heat is the derivative of the internal heat, we can observe a relatively significant change in the internal heat around the critical temperature. From this plot, we can clearly recognize the phase transition of the Potts model.

4.2 Estimate the magnetization

In this part, we only use Metropolis-Hastings algorithm.
Due to symmetry, we expect the $m - h$ relation is symmetry with respect to the point $(0, \frac{q+1}{2})$. As the T decreases, the jump of m around $h = 0$ becomes more drastic.
We set $N = 15, T = numpy.linspace(0.8, 2.0, 41)$. The result is depicted in Figure 4. When $d = 3$. we set $N = 6, T = numpy.linspace(1.2, 3.0, 41)$. The result is depicted in Figure 5, we rotate the figure to represent it clearly. The program takes hours to simulate.
The phenomenon can be explained as follows. If T is close to zero, the value β is very large. Hence the distribution will be concentrated in the state whose hamiltonian is the lowest. The distribution will concentrate on the state where all spins are 1 if $h < 0$, and the state where all spins are q if $h > 0$. When $h = 0$, the state will evenly concentrate on the state where all spins are the same (q types in total). Hence, given $h \neq 0$, as T increases, the expectation should approach the middle line $\frac{q+1}{2}$.

4.3 Correlation length and Critical exponents estimation

We use Python package "sklearn" to perform linear regression. The date we use are from previous simulations.
Empirically, we find that when $T < T_c$, the model's behavior is very turbulent. Hence, **we only estimate the critical exponents when** $T > T_c$.
In the following, some variables are defined below:

$$\begin{aligned} \epsilon &= |1 - T/T_c| \\ u &\sim u_0 \epsilon^{-\omega} \\ c &\sim c_0 \epsilon^{-\gamma} \\ \xi &\sim \xi_0 \epsilon^{-\delta} \\ m &\sim m_0 \epsilon^{\alpha} \end{aligned} \tag{19}$$

4.3.1 γ

Metropolis-Hastings algorithm When $d = 2$, we set $N = 20$. Figure 10 and Figure 11 separately represents the regression of γ and ω. Notice that the theoretical result of . When $d = 3$, we set $N = 12$. Figure 12 and Figure 13 separately represents the regression of γ and ω. The coefficient is the desired critical exponent, and R^2 measures the accuracy of the regression.

Wolff algorithm When $d = 2$, we set $N = 20$. Figure 25 and Figure 27 separately represents the regression of γ and ω. Notice that the theoretical result of . When $d = 3$, we set $N = 12$. Figure 28 and Figure 30 separately represents the regression of γ and ω. The coefficient is the desired critical exponent, and R^2 measures the accuracy of the regression.
In the left part of the critical point, we only use Wolff algorithm to simulate. Modify the codes a little, i.e. set T_lst = numpy.linspace(1.75, 1.89, 15), we get the estimated γ on the left is approximately 1.0, which is plotted in 26 for $d = 2$. For $d = 3$, we set T_lst = numpy.linspace(1.4, 1.76, 10), we get the estimated γ on the left is approximately 2.4, which is plotted in 29. The right side and left side of critical exponent γ is identical.

Analysis In brief, when $d = 2$, $\gamma = 1.0$; when $d = 3$, $\gamma = 0.8 \sim 1.0$. Here we only consider the results whose correlations are larger than 0.9. Since in total the Wolff algorithm flips more spins, its figure looks smoother.

4.3.2 α

Metropolis-Hastings algorithm Since $h = 0$, by symmetry, $m = \frac{q+1}{2}$ for all T. Hence, theoretical result of α is zero. The configuration is the same as the Section 4.2. Despite the low R^2, Figure 14 shows that $\alpha \approx 0.04$, which is clearly close to zero. The low correlation can be explained by the fact that the deviation is caused by numerical error.

Wolff algorithm We restrict our attention on the case where $h = 0$.
When $d = 2$, we set $N = 20$, T_lst = numpy.linspace(0.85,1.25,21). Results are plotted in the Figure 24. We still observe that $\alpha = 0$.
We do not expect the case when $d = 3$ to be different. Thus, we postpone the experiment.

4.3.3 ξ, δ - above the critical

In this part, we only use temperature above the critical temperature.

Metropolis-Hastings algorithm Since the change in the correlations is not significant, we need stricter condition to stop the Monte Carlo. Specifically, we set the threshold is 0.001, and use 10 workers under such situation.
When $d = 2$, the result of $\Gamma(k)$ with k under different temperature T is depicted in the Figure 8, the relation of $\xi - T$ is depicted in the right part of the Figure 16 [2], corresponding regression results are summarized in the Table 1. The δ is estimated in the Figure 16.
When $d = 3$, the result of $\Gamma(k)$ with k under different temperature T is depicted in the Figure 9, the relation of $\xi - T$ is depicted in the right part of 17, corresponding regression results are summarized in the Table 2. The δ is estimated in the Figure 17.

Wolff algorithm Since the Wolff algorithm will not be trapped in the local minima, we only use one worker to simulate. However, the case where $d = 3$ still takes about 8 hours to simulate. The settings are the same as above. When $d = 2$, the result of $\Gamma(k)$ with k under different temperature T is depicted in the Figure 22, the relation of $\xi - T$ is depicted in the right part of the Figure 31, corresponding regression results are summarized in the Table 3. The δ is estimated in the Figure 31.
When $d = 3$, the result of $\Gamma(k)$ with k under different temperature T is depicted in the Figure 23, the relation of $\xi - T$ is depicted in the right part of the Figure 32, corresponding regression results are summarized in the Table 4. The δ is estimated in the Figure 32.

Analysis It is not difficult to realize that the Wolff algorithm significantly outperforms the Metropolis-Hastings algorithm in estimating δ.[3]. However, notice that when the temperature is below the critical point, the estimation is both inaccurate and time-consuming. Hence, we will first examine the case where the temperature is above the critical.
Comparing the result from the Table 1 and 3, the Table 2 and 4, the Figure 16 and 31, the Figure 17 and 32, we can only use the results by the Wolff algorithm.
In brief, when $d = 2, \delta = 0.78$; when $d = 3, \delta = 0.37$.

4.3.4 ξ, δ - below the critical

Since from above, the Wolff's algorithm will not trapped in the local minima below the critical temperature. In this case, we only use Wolff algorithm to simulate. We use scipy.optimize.curve_fit to perform nonlinear approximation.
Empirically, we find that under this case, larger N needs to be applied. Hence, we use $N = 32, k = 15$ when $d = 2$; and $N = 16, k = 7$ when $d = 3$. Performing the simulation on a laptop is onerous, hence I register on the Weiming Teaching I take advantage of parallel computation. Example "run.slurm" file is presented in the appendix. When $d = 2$, the result is depicted in Figure 33, numerical results are presented in Figure 5. Performing linear regression w.r.t. $\log(\xi)$, $\log(\epsilon)$, we get the result 34.
When $d = 3$, the result is depicted in Figure 35, numerical results are presented in Table 6. Performing linear regression w.r.t. $\log(\xi)$, $\log(\epsilon)$, we get the result in Figure 36.
Since R^2 of above regressions are above 0.95, the accuracy is guaranteed. In brief, when $d = 2$, $\delta = 0.44$; when $d = 3$, $\delta = 0.19$.
Hence, **Estimated δs in two sides of the critical point are not identical.**

[2] Real solutions are blue dots and not connected

[3] Here "outperform" does not means convergence in shorter time, but in fewer flips, since one flip by the Wolff algorithm is more time-consuming than the Metropolis-Hastings algorithm

References

[Wikipedia, 2019] Wikipedia (2019). Correlation function (statistical mechanics) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Correlation%20function%20(statistical%20mechanics)&oldid=922950462. [Online; accessed 02-December-2019].

[Wu, 1982] Wu, F. Y. (1982). The potts model. *Rev. Mod. Phys.*, 54:235–268.

5 Appendix A: Figure Collections

5.1 Metropolis-Hastings Algorithm

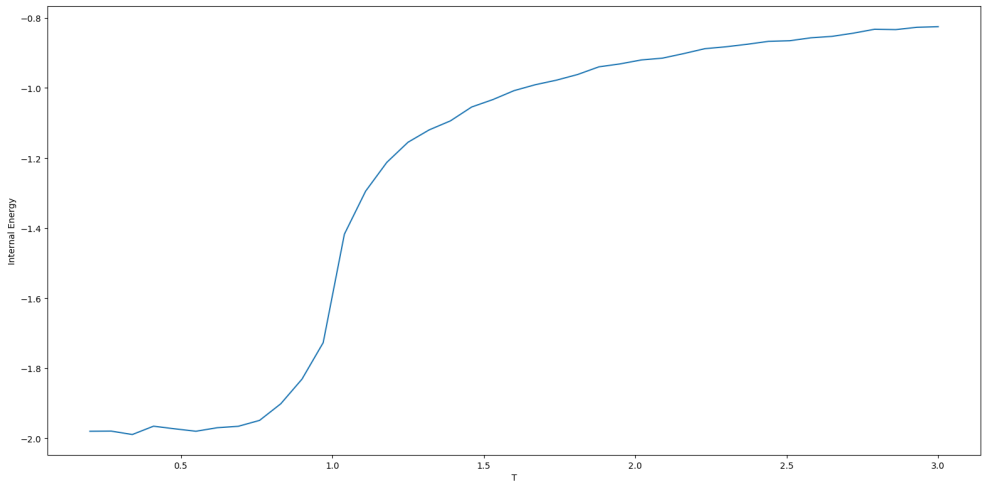


Figure 2: Internal energy

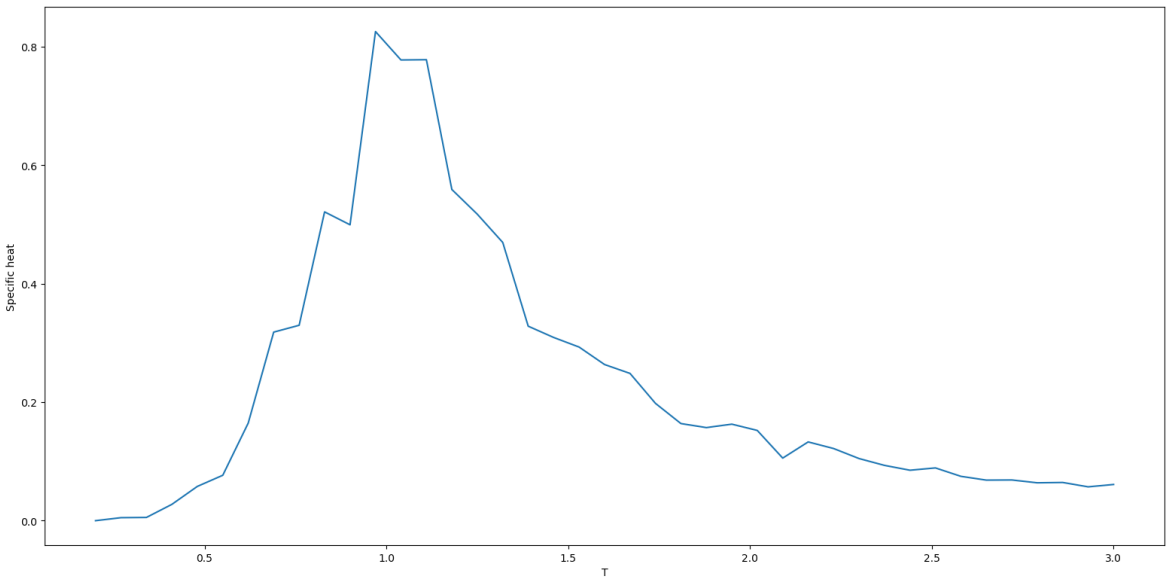
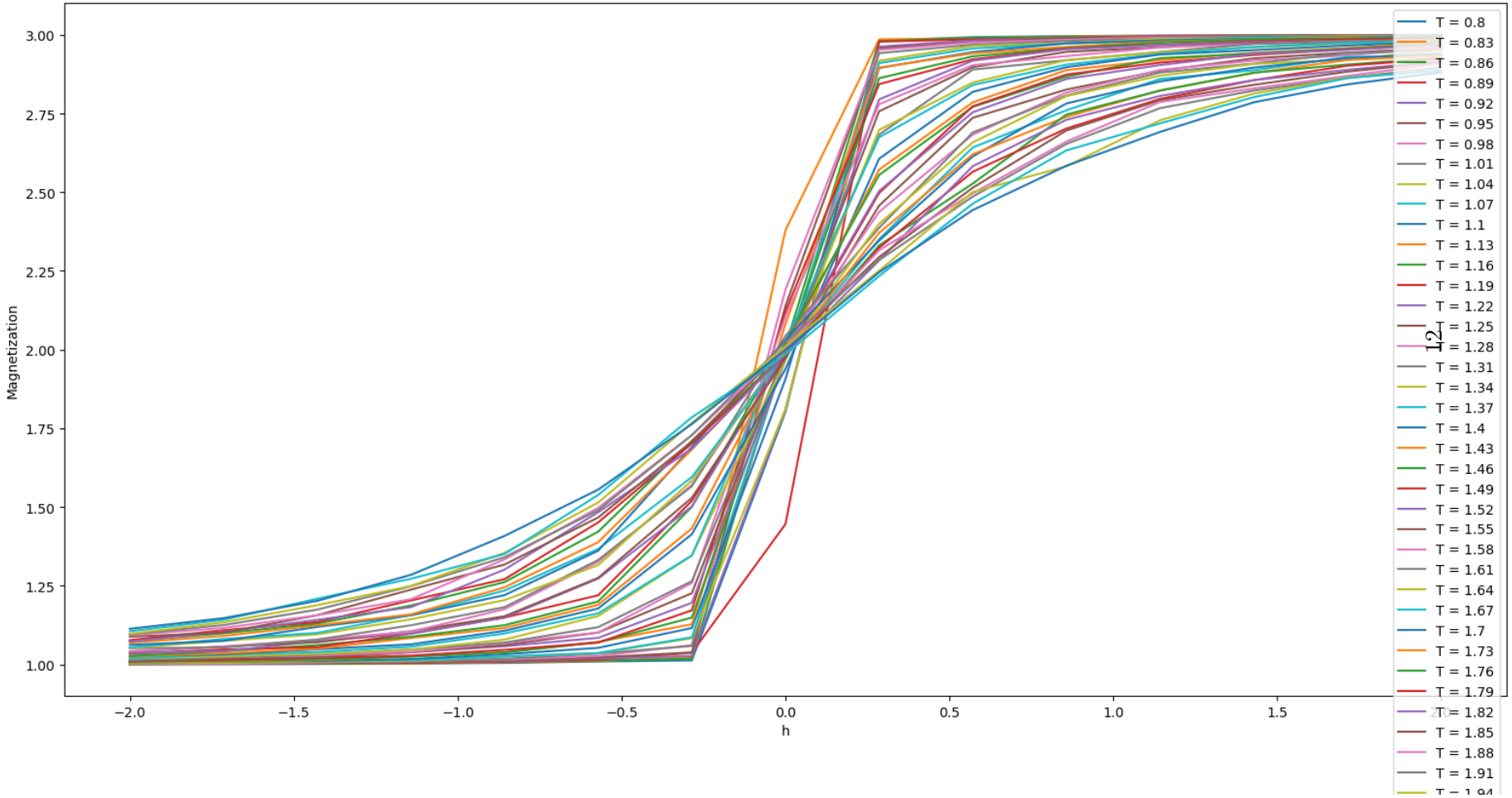
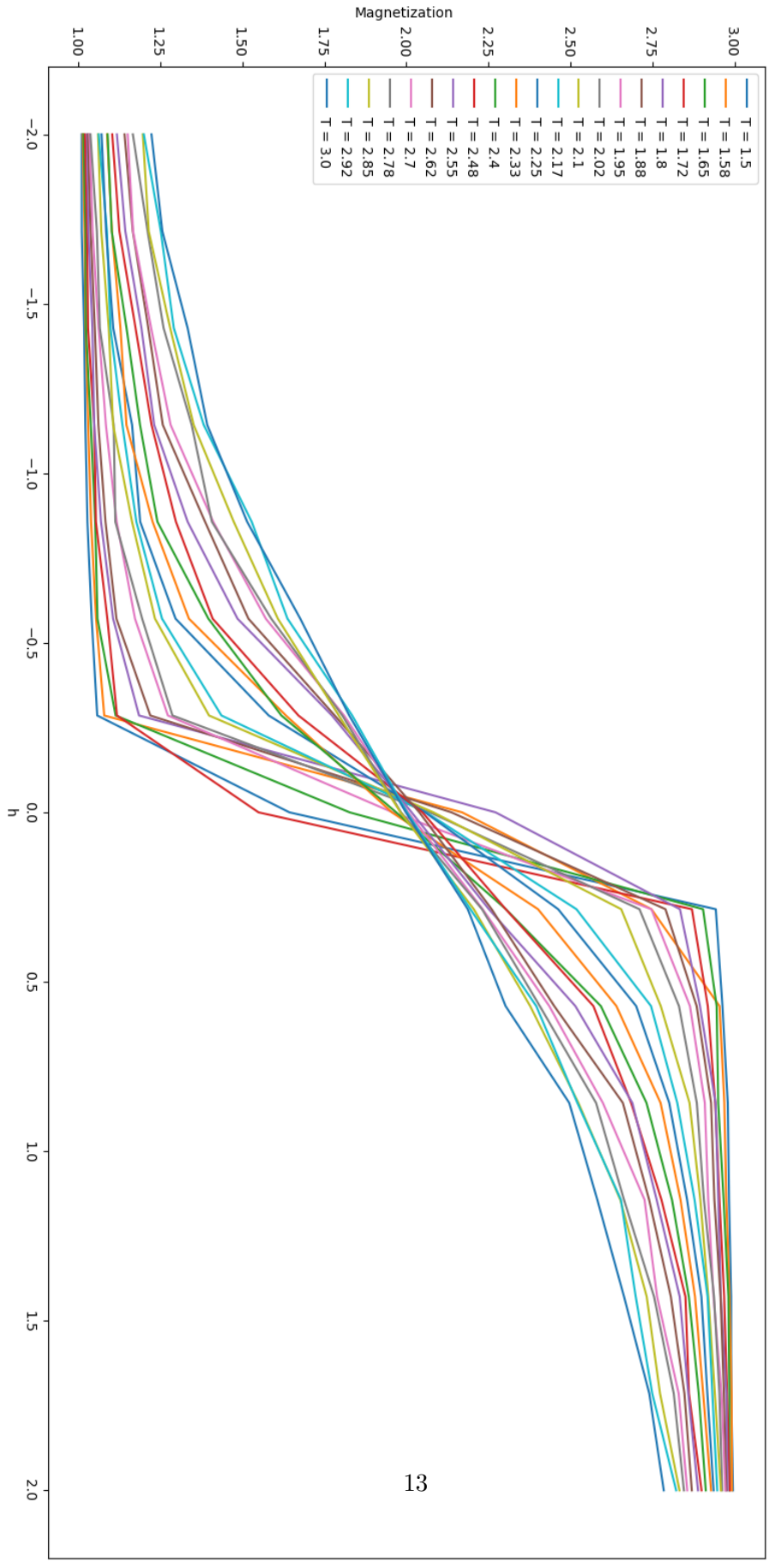


Figure 3: Specific heat


Magnetization
h
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
T = 0.8
T = 0.83
T = 0.86
T = 0.89
T = 0.92
T = 0.95
T = 0.98
T = 1.01
T = 1.04
T = 1.07
T = 1.1
T = 1.13
T = 1.16
T = 1.19
T = 1.22
T = 1.25
T = 1.28
T = 1.31
T = 1.34
T = 1.37
T = 1.4
T = 1.43
T = 1.46
T = 1.49
T = 1.52
T = 1.55
T = 1.58
T = 1.61
T = 1.64
T = 1.67
T = 1.7
T = 1.73
T = 1.76
T = 1.79
T = 1.82
T = 1.85
T = 1.88
T = 1.91


Magnetization
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
h
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
T = 1.5
T = 1.58
T = 1.65
T = 1.72
T = 1.8
T = 1.88
T = 1.95
T = 2.02
T = 2.1
T = 2.17
T = 2.25
T = 2.33
T = 2.4
T = 2.48
T = 2.55
T = 2.62
T = 2.7
T = 2.78
T = 2.85
T = 2.92
T = 3.0

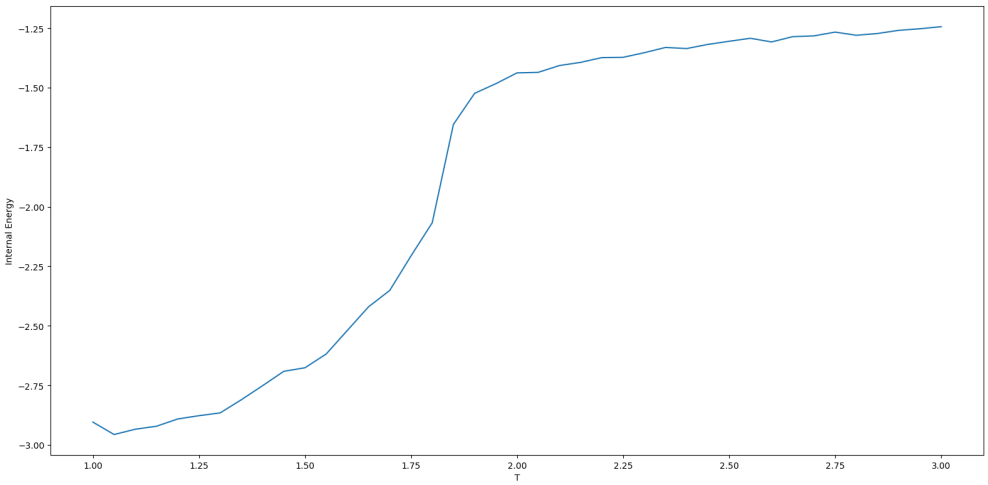


Figure 6: Internal energy

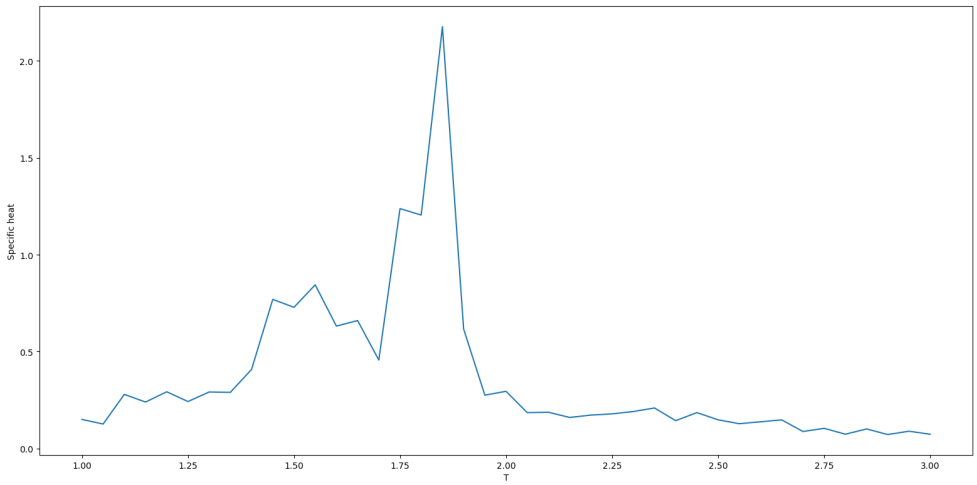


Figure 7: Specific heat

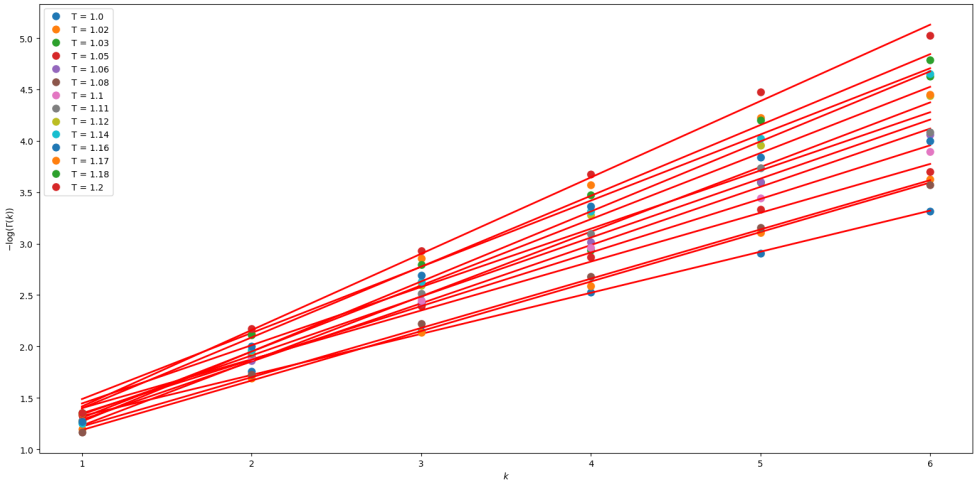


Figure 8: $\Gamma(k) - k$ w.r.t. different $T > T_c$,$d = 2$

T	$1/\xi$	R^2
1.0	0.54095851	0.9971754438859441
1.02	0.44251808	0.9992613510609634
1.03	0.45244452	0.9990618240072892
1.05	0.55423823	0.9971826216357939
1.06	0.60229418	0.9970182488835609
1.08	0.51598231	0.9996543363005579
1.1	0.67129211	0.9979463492199687
1.11	0.71397665	0.9992149186523895
1.12	0.53787788	0.9939953633328179
1.14	0.64948966	0.9993188473138691
1.17	0.72596495	0.9931220132925702
1.18	0.73913456	0.9964691339686178
1.2	0.66492158	0.9875325396250086

Table 1: Regression result in Figure 8

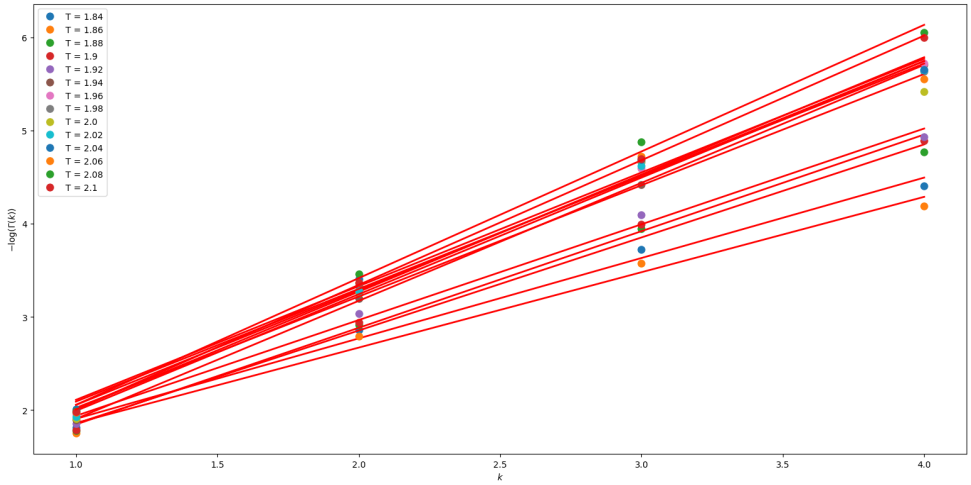


Figure 9: $\Gamma(k) - k$ w.r.t. different T, $d = 3$

T	ξ	R^2
1.84	1.158785	0.991052
1.86	1.236063	0.986143
1.88	1.001243	0.994967
1.9	0.964265	0.997029
1.92	0.973051	0.994148
1.94	0.78965	0.999871
1.96	0.794697	0.998115
1.98	0.812888	0.994334
2	0.837273	0.983384
2.02	0.799042	0.995973
2.04	0.819616	0.993506
2.06	0.826883	0.987565
2.08	0.735683	0.997374
2.1	0.747462	0.999819

Table 2: Regression result in Figure 9

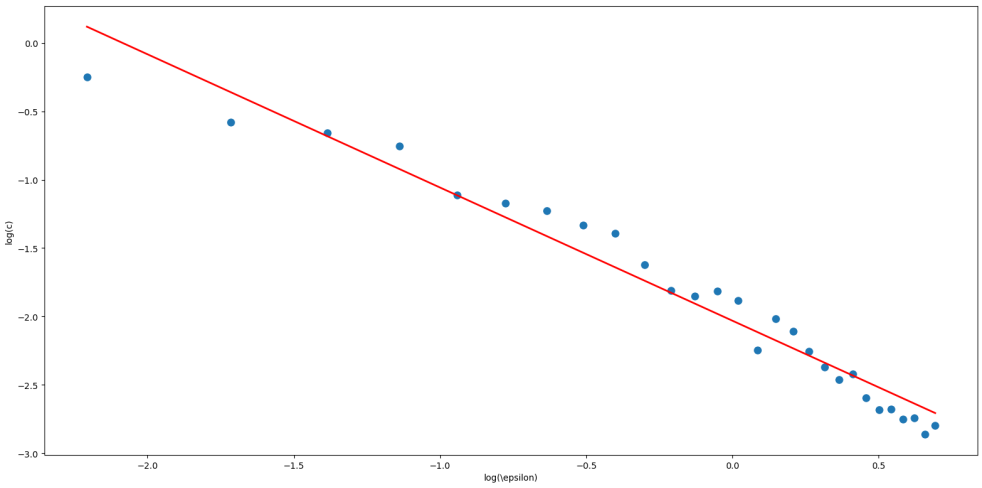


Figure 10: Coefficient: -0.97386151, R^2= 0.9592415188185291

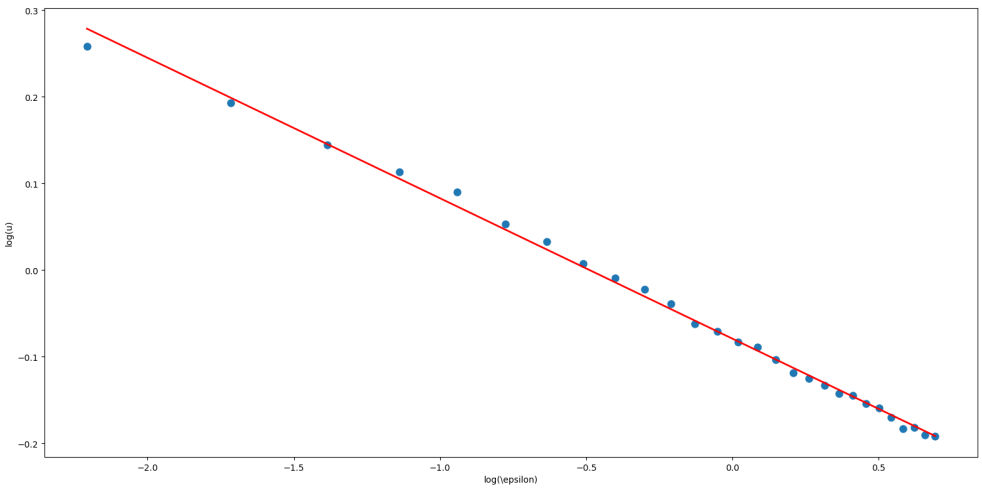


Figure 11: Coefficient: -0.16215771, R^2 = 0.9969122564585634

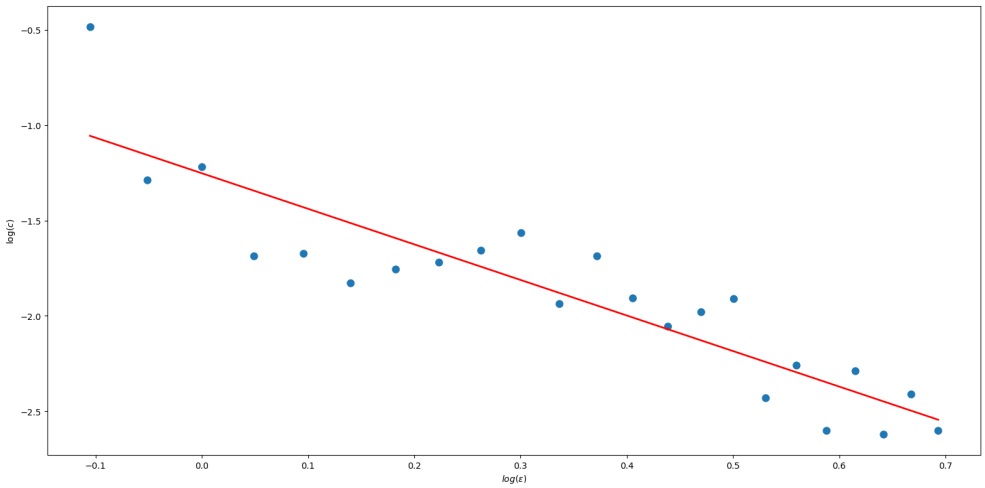


Figure 12: Coefficient: -1.86553276, R^2= 0.810862008785798

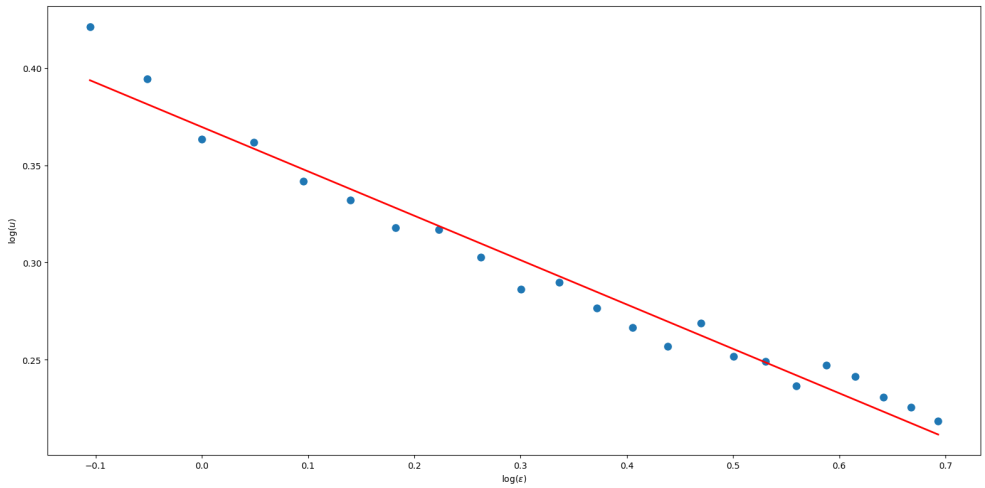


Figure 13: Coefficient: -0.22825604, R^2 = 0.9668487539325992

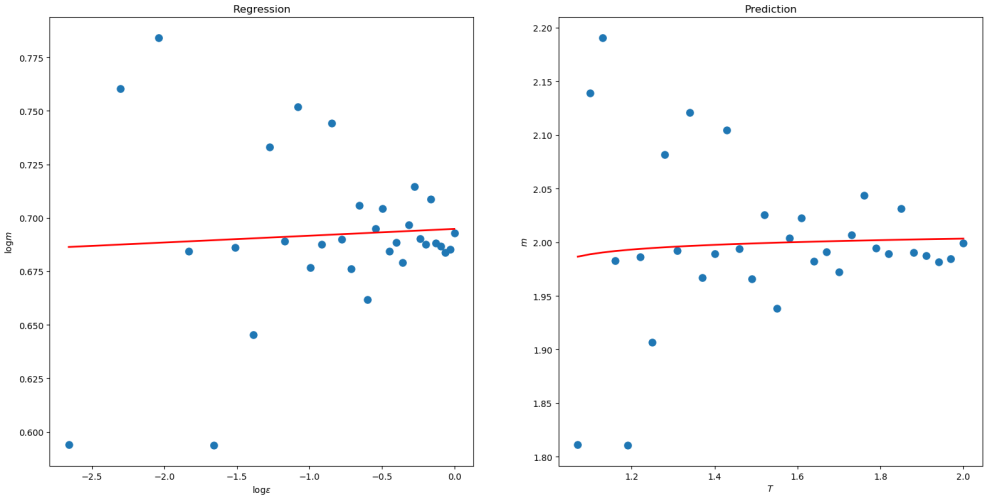


Figure 14: Coefficient: 0.03816214, $R^2 = 0.20798593255516906$

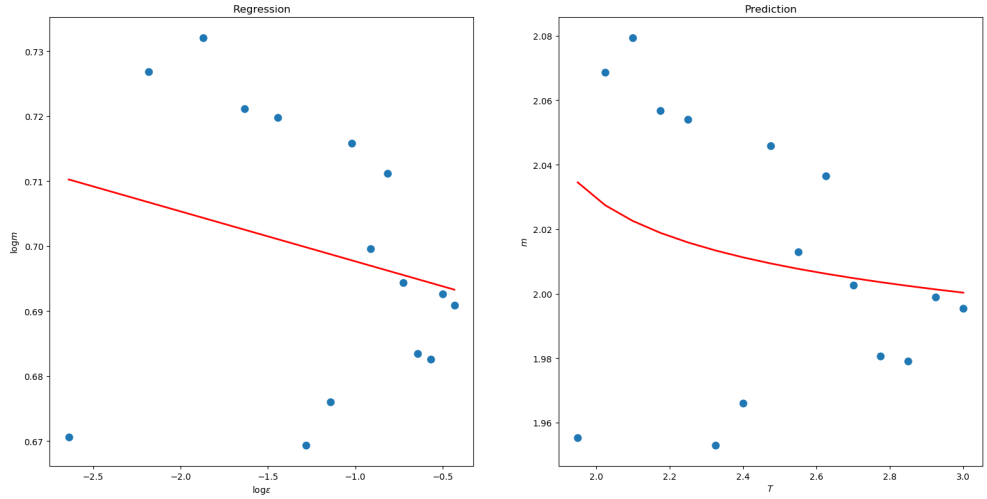


Figure 15: Coefficient: -0.00768954, $R^2 = 0.058457982365614176$

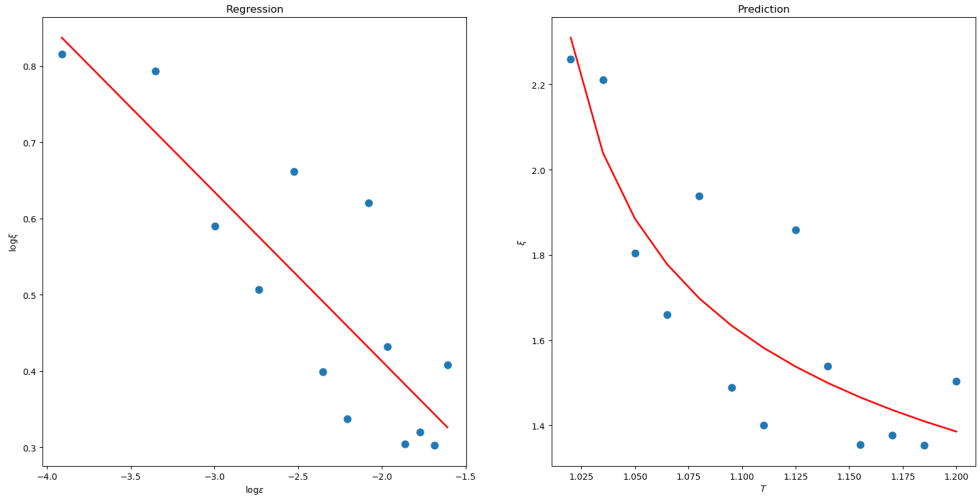


Figure 16: Coefficient: -0.22186352, R^2= 0.7271143433953376

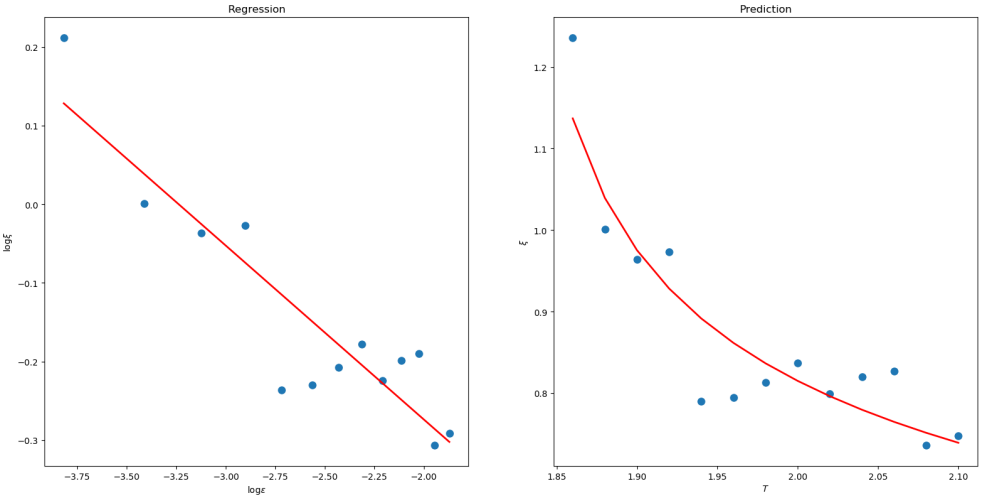


Figure 17: Coefficient: -0.22136172, R^2= 0.8314771571382932

5.2 Wolff Algorithm

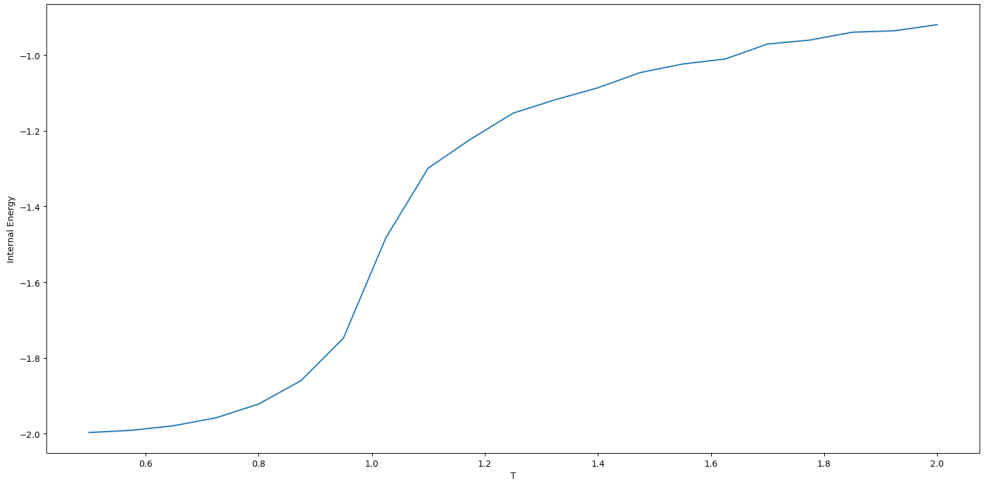


Figure 18: Internal energy

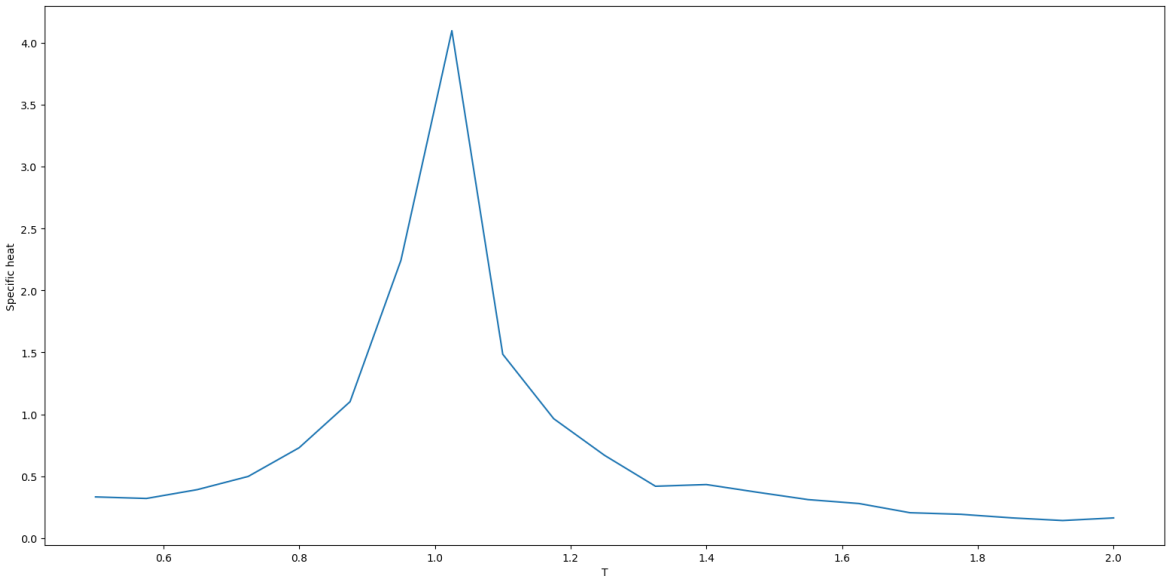


Figure 19: Specific heat

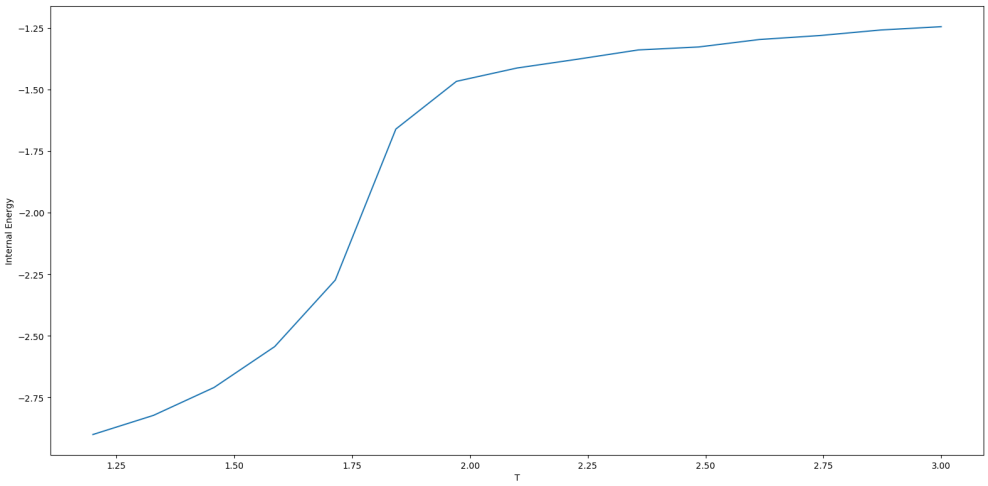


Figure 20: Internal energy

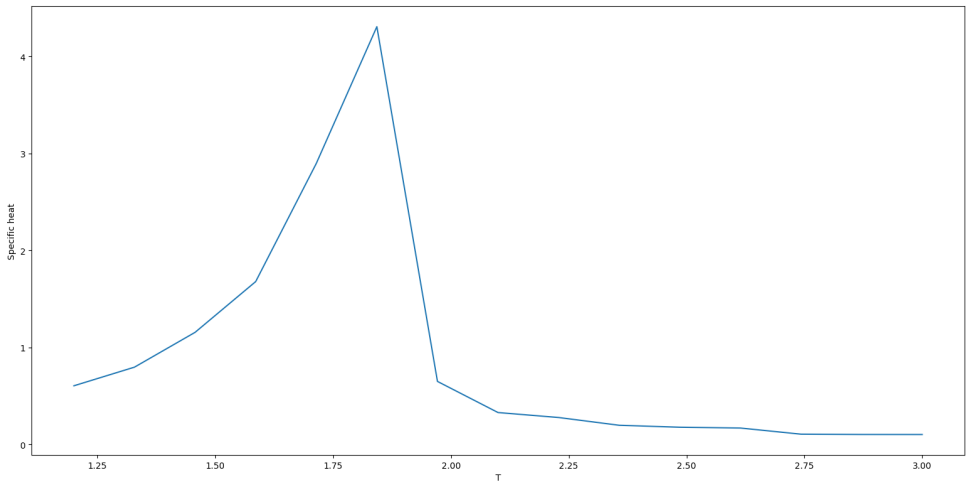


Figure 21: Specific heat

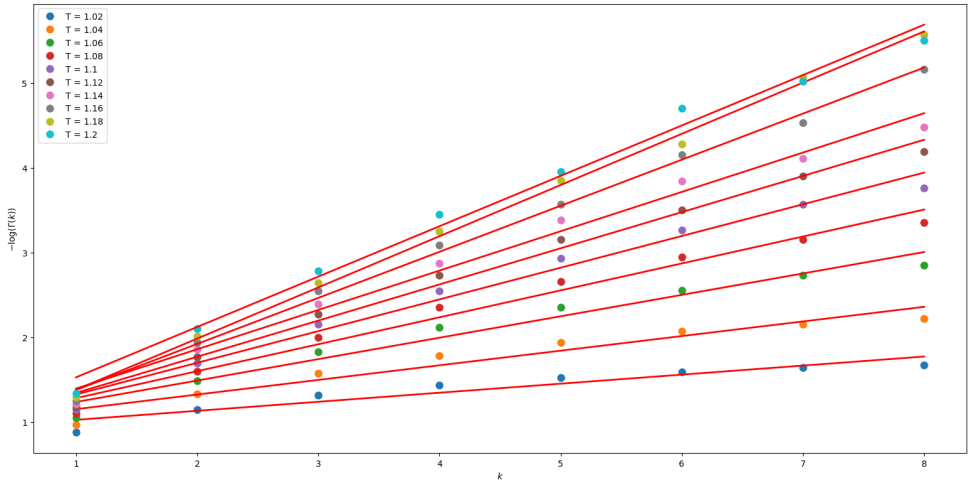


Figure 22: $\Gamma(k) - k$ w.r.t. different $T > T_c$, $d = 2$

T	$1/\xi$	R^2
1.02	0.10674214	0.9338947208861894
1.04	0.19187199	0.9576076574972642
1.06	0.25279367	0.9810820587460465
1.08	0.31784363	0.984912400631948
1.0	0.37403852	0.9870665838423055
1.12	0.42673085	0.993957023316214
1.14	0.46423571	0.9930850616072405
1.16	0.54355456	0.9930367794820021
1.18	0.60412984	0.9992565749651644
1.2	0.59450194	0.9986264357625233

Table 3: Regression result in Figure 22

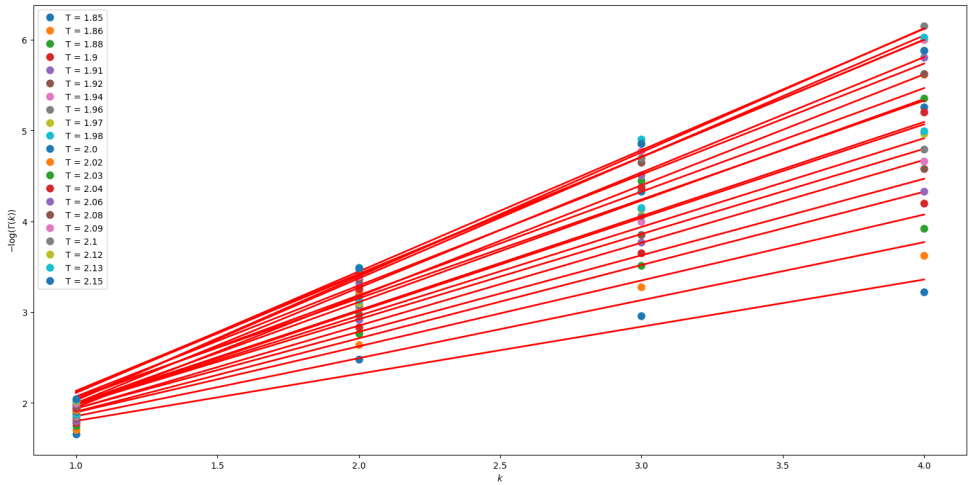


Figure 23: $\Gamma(k) - k$ w.r.t. different $T > T_c$, $d = 3$

T	$1/\xi$	R^2
1.85	0.51969026	0.9437440928075225
1.86	0.63956107	0.959860490793278
1.88	0.72652159	0.9653000332351946
1.9	0.80789391	0.980980134910335
1.91	0.84369434	0.9791977904534493
1.92	0.91377634	0.9894753638811147
1.94	0.93953858	0.9843859473186448
1.96	0.97854435	0.9883963029537196
1.97	1.03101604	0.9927539704286986
1.98	1.03737505	0.9924526326378508
2.0	1.11966014	0.994910856352704
2.02	1.22370561	0.9993929670800514
2.03	1.1416312	0.9929492891977173
2.04	1.09005078	0.9896473112679923
2.06	1.27109757	0.999385229503799
2.08	1.22334697	0.9942430121193441
2.1	1.36696199	0.9990635048502007
2.12	1.29589457	0.9939874491753194
2.13	1.33756044	0.9965473753647345
2.15	1.28944768	0.9944482548502082

Table 4: Regression result in Figure 23

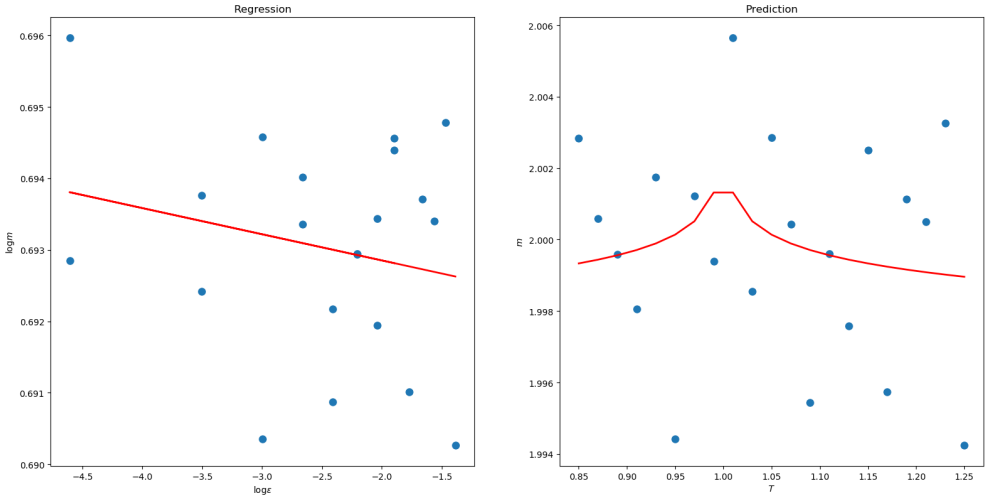


Figure 24: Coefficient: -0.00036644, R^2= 0.049446125340963265

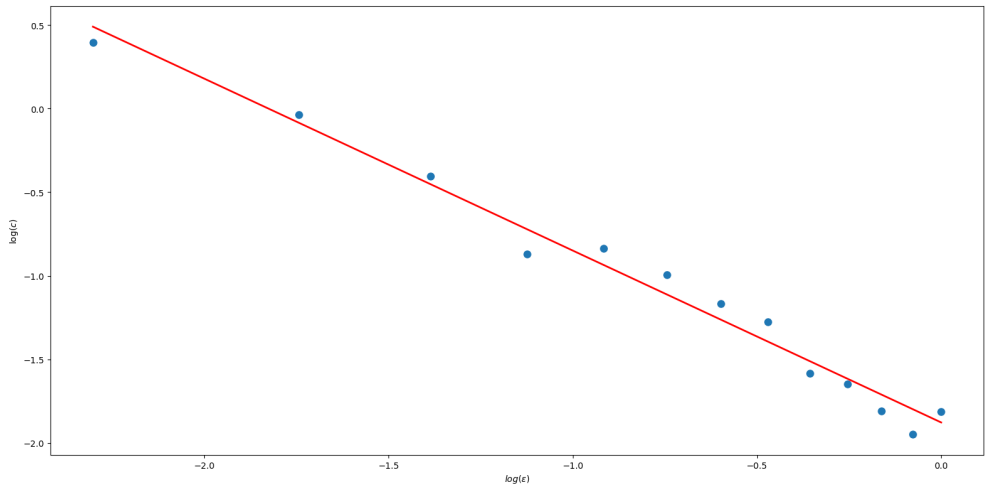


Figure 25: Coefficient: -1.02818623, R^2= 0.9798311050315772

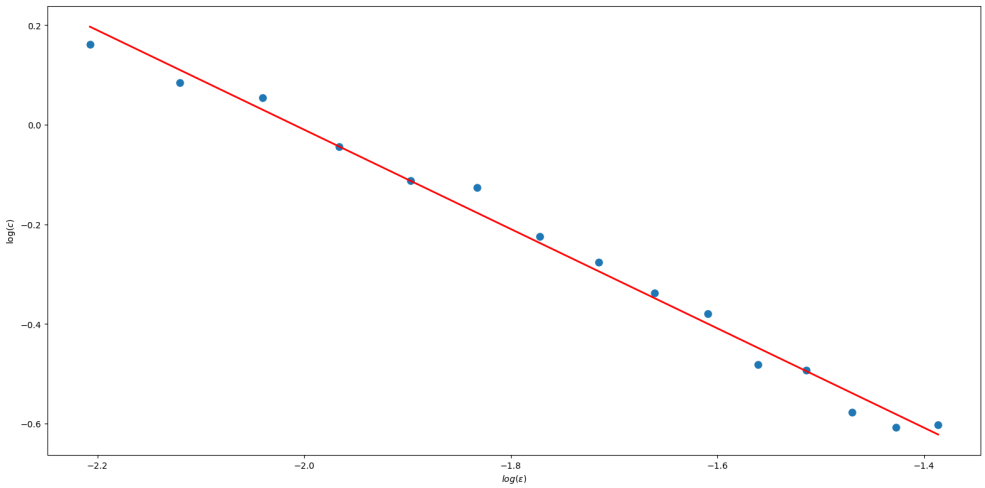


Figure 26: Coefficient: -0.99738486, R^2= 0.9897278743168862

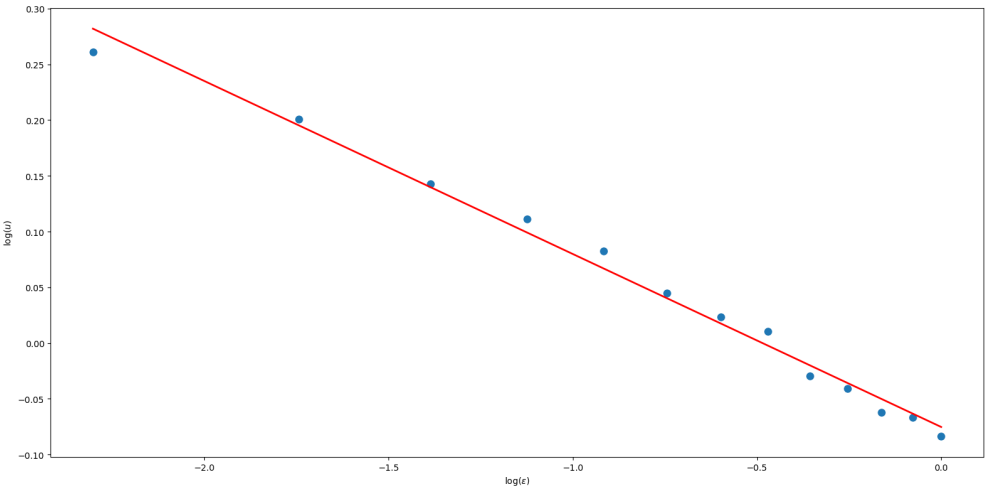


Figure 27: Coefficient: -0.1551193, R^2 = 0.9898122420770055

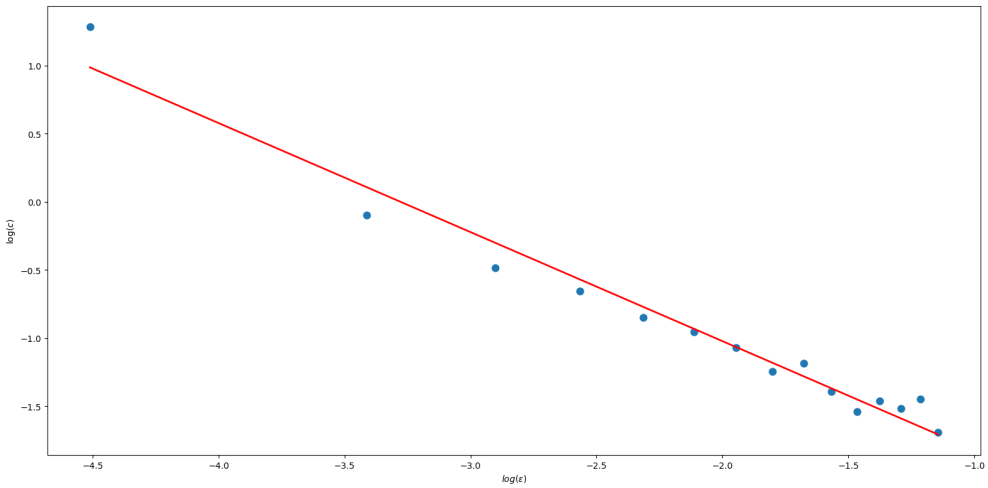


Figure 28: Coefficient: -0.80061366, R^2= 0.9691818682978928

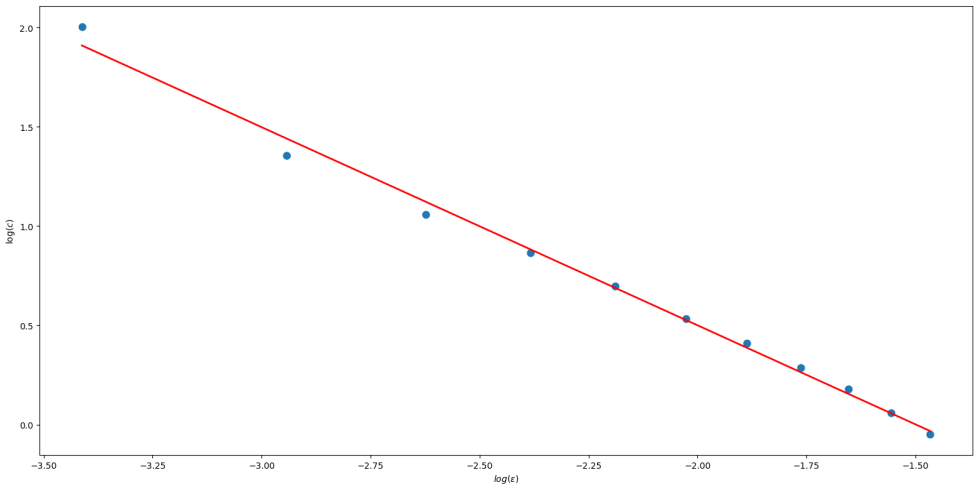


Figure 29: Coefficient: -0.99834636, R^2= 0.993995433909582

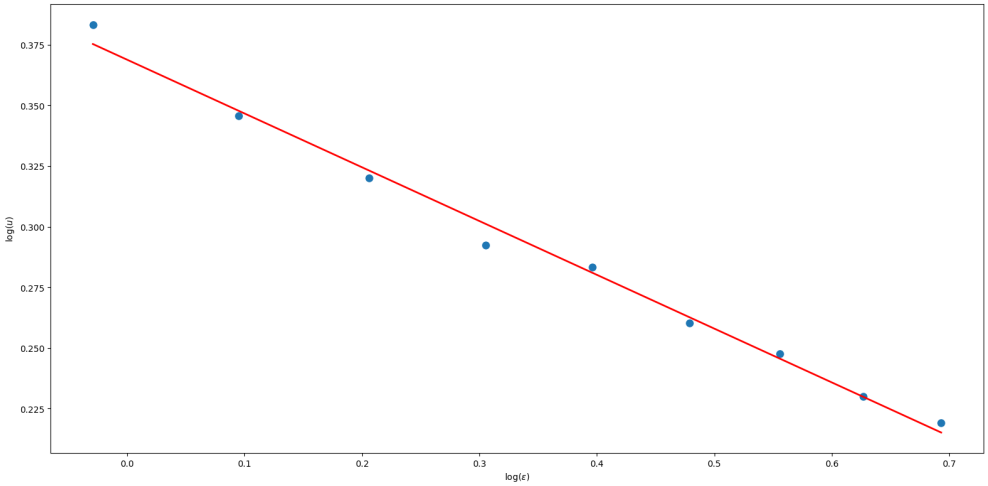


Figure 30: Coefficient: -0.18174866, R^2 = 0.9922116970486973

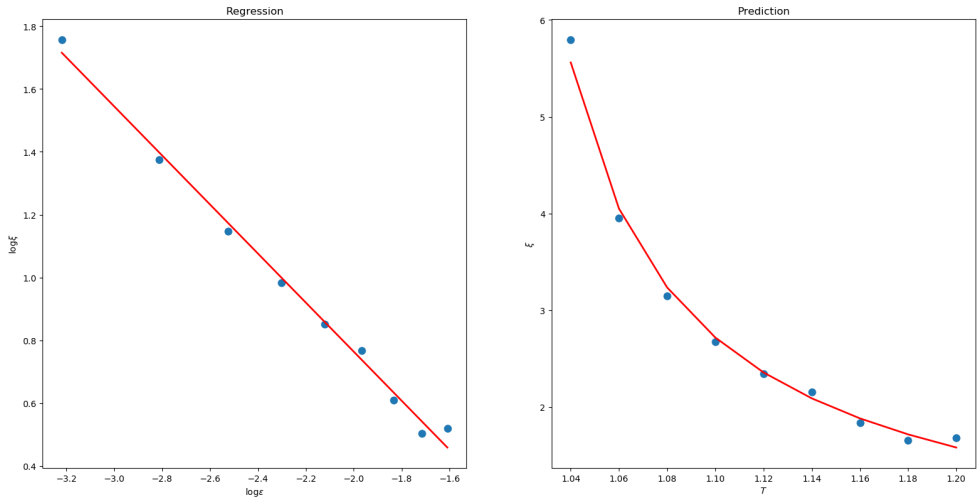


Figure 31: Coefficient: -0.78130386, R^2= 0.9970796277584468

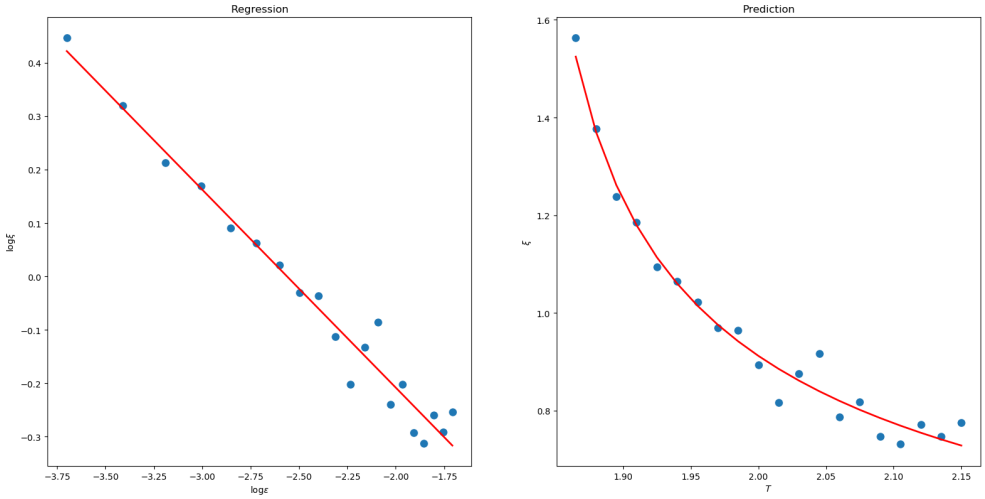


Figure 32: Coefficient: -0.37062525, R^2= 0.9684188990866911

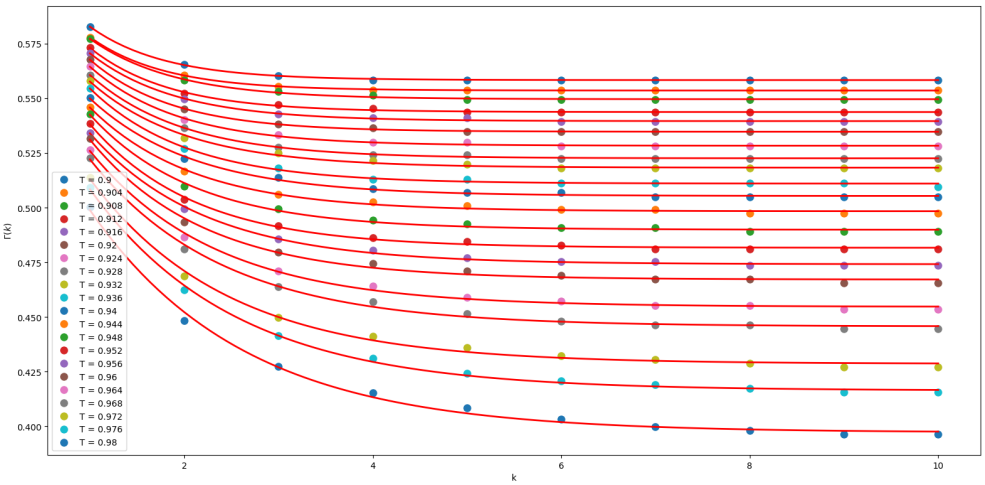


Figure 33: Correlation length estimate below the critical temperature, $d = 2$

T	ξ	$Std(\xi)$
0.9	0.7894295187317633	0.018051027426050206
0.904	0.7894295012814484	0.03249188809217059
0.908	0.8984587840742883	0.038921204652690854
0.912	0.855186249007688	0.03589226038845141
0.916	0.911345516067944	0.03491352359350346
0.92	0.8813066688591262	0.05571881852196759
0.924	0.9460116429416412	0.027446181345848843
0.928	0.9936954866012577	0.02748667312910578
0.932	1.0313572430381424	0.05696003250317435
0.936	1.0384136581872696	0.04438254479570324
0.94	1.1096121043972278	0.02582364621914657
0.944	1.1052404354674223	0.05941405180926211
0.948	1.1132620507808864	0.05769323997031782
0.952	1.1239440992883385	0.05227383360101045
0.956	1.220035942359717	0.041928632236221064
0.96	1.2136784414019566	0.05807645464879385
0.964	1.339838370992224	0.05469798101759823
0.968	1.4019293355667757	0.07876030433143291
0.972	1.5144294310541708	0.07736198552421113
0.976	1.540253221112417	0.08223964500110074
0.98	1.6305943435425092	0.07928883644490212

Table 5: Regression results of $\Gamma(r) = a\exp(-r/\xi) + C$ under different T in Figure 33

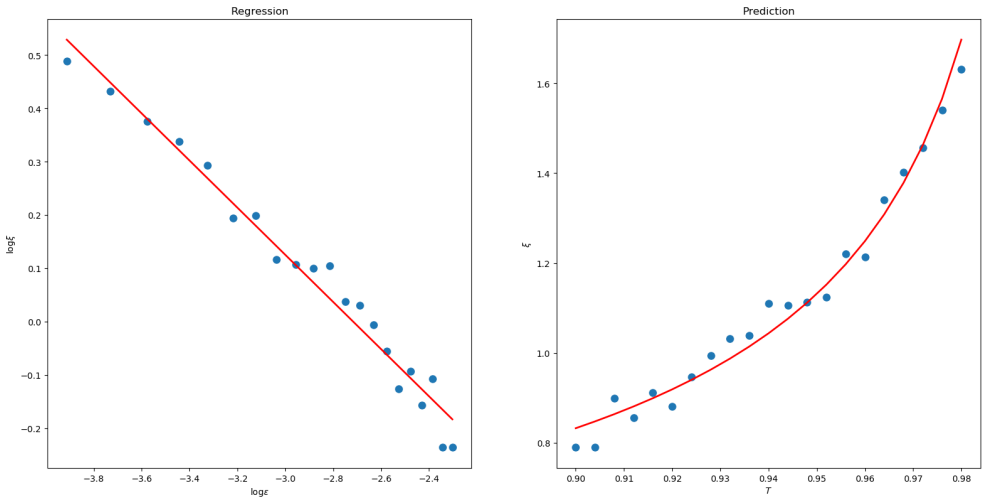


Figure 34: Coefficient: -0.44256058, R^2= 0.9729370258539385

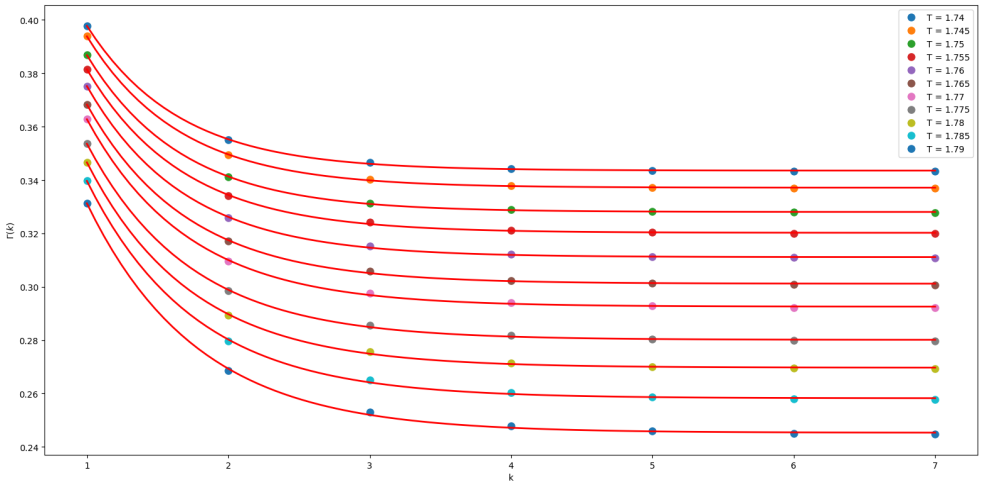


Figure 35: Correlation length estimate below the critical temperature, $d = 3$

T	ξ	$Std(\xi)$
1.74	0.6536944319852122	0.012428278928246366
1.745	0.6596580658506238	0.008933644624566955
1.75	0.673258285418814	0.01049307103504379
1.755	0.684433471016979	0.014616748805003363
1.76	0.6910221233098542	0.013077675080758338
1.765	0.7067323746459651	0.01717593057400444
1.77	0.7188185118093298	0.01668921120997964
1.775	0.7323812474082978	0.016640474984547154
1.78	0.7436391294288337	0.016084679721158363
1.785	0.7647093779820858	0.019509769847641686
1.79	0.7840927357082804	0.022589482769588033

Table 6: Regression results of $\Gamma(r) = a\exp(-r/\xi) + C$ under different T in Figure 35

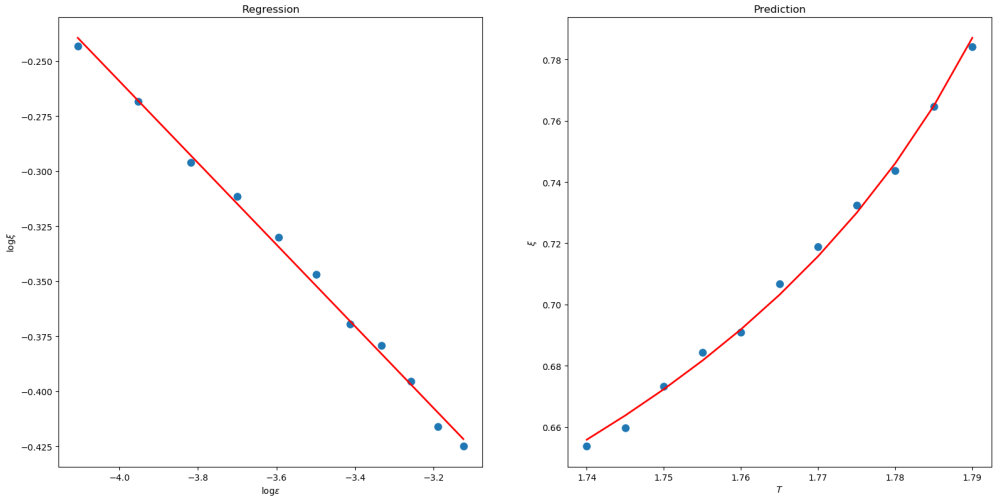


Figure 36: Coefficient: -0.18588607, R^2= 0.9958192699152079

6 Appendix B: Codes Explanations

1. Throughout the program, I use a dictionary type variable "args" to share parameters in different situations. Apart from basic parameters, we use the key "n" to represent the numeber of workers, use the key "k_lst" to store all different choice of k during one simualtion, and use "critical" to store the estimated critical temperature, which will only be used in estimating the critical exponents.
2. We construct a class "Potts" to encompass all the method in the simulation. Since the code is designed for general d, we need extra code to specify one spin's neighbours. Other methods include calculating the Hamiltonian, the magnetization, the correlation, and the difference Hamiltonian and correlation when one spin is flipped.
3. The function parameter_specify is used to generate the size and all the coordinates of the model, which is also designed to accommodate arbitrary d.
4. The function "Metropolis" and "Wolff" takes "args" as input to perform MCMC simulation. We calculate all the required quantities in one function to shorten the length of the codes.
5. Function "main_simulation" takes "args" and "method" as inputs, where "args" is the dictionary type variable and "method" is either "MH" or "Wolff" to determine the simulation method. Some parameters needs to be fine-tuned according to the main texts. In the function, we use sklearn.linear_model to perform linear regression. Numerical results are printed during the execution, which is collected in the Table 1, 2, 3 and 4.
6. If readers want to reproduce these results, please be sure to have one day's free CPU time, or readers can use Weiming Teaching.

To use the Weiming Teaching, we first upload file "run.slurm" in the following format.

```
#!/bin/bash

#SBATCH -o output%j.txt
#SBATCH --partition=compute
#SBATCH -J potts
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH -t 24:00:00

module add anaconda

python potts.py
```

and then run

```
sbatch run.slurm
```