212: Final Project
Cluster Monte Carlo on a square lattice in 2, 3, & 5 dimensions

Logan Bishop-Van Horn

March 13, 2018

1 Introduction

1.1 Ising Model

The Ising Model, invented by Lenz in 1920 and solved in one dimension by his student Ising in 1924, is a model for ferromagnetism consisting of discrete variables known as spins lying on a lattice, each of which can take on the value $+1$ ('up') or -1 ('down'). The model can be parameterized by the interactions or couplings amongst spins, and the coupling between individuals spins and an external magnetic field. In the trivial case where the spins do not interact, the physics is identical to that of a single spin in a magnetic field. Explicitly, the energy of a single spin with magnetic dipole moment μ and spin $s = \pm 1$ in a magnetic field B is given by the Hamiltonian $H_{1\text{ spin}} = -\mu B s$, with $H_{1\text{ spin}}|s\rangle = E_s|s\rangle$, where the minus sign comes in because the energy is minimized when s and B have the same sign. The partition function for this simple system at temperature T is given by

$$Z_{1\text{ spin}} = \sum_{s=\pm 1} e^{-\frac{E_s}{k_{\text{B}}T}} \equiv \sum_{s=\pm 1} e^{-\beta E_s} = \sum_{s=\pm 1} e^{\beta\mu B s} = 2\cosh h,$$

where $h \equiv \beta\mu B$ is a dimensionless measure of the strength of the magnetic interaction relative to the thermal energy of the system. The magnetization, or average value of the spin, for a given h is

$$m_h \equiv \langle s\rangle_h = \frac{1}{Z_{1\text{ spin}}}\sum_{s=\pm 1} s e^{-\beta E_s} = \frac{1}{Z_{1\text{ spin}}}\sum_{s=\pm 1} s e^{hs} = \frac{e^{+h}-e^{-h}}{2\cosh h} = \tanh h.$$

In the case of a system of N noninteracting spins, the partition function $Z_{N\text{ spins}}$ is simply $(Z_{1\text{ spin}})^N$, and the magnetization per spin is precisely what we've written above.

In the next simplest case, each spin interacts only with its nearest neighbors on the lattice, and there is a single coupling constant J that characterizes the strength of these interactions. In this case, for a given spin configuration $\{s_i\}$, the total energy of the system is

$$E(\{s_i\}) = -J \sum_{\langle i,j \rangle} s_i s_j - \mu B \sum_i s_i,$$

where the notation $\langle i, j\rangle$ indicates that indices i and j correspond to lattice nearest neighbors. The Ising partition function is then

$$Z = \sum_{\{s_i\}} e^{-\beta E(\{s_i\})} = \sum_{\{s_i\}} \exp\left[\beta J \sum_{\langle i,j \rangle} s_i s_j + \beta \mu B \sum_i s_i\right] \equiv \sum_{\{s_i\}} \exp\left[K \sum_{\langle i,j \rangle} s_i s_j + h \sum_i s_i\right],$$

where K is dimensionless and characterizes the energy scale of the spin-spin interactions relative to the thermal energy of the system. Qualitatively, when $K \ll 1$ (that is, when the interactions are much weaker than thermal fluctuations) one would expect the system to be disordered, and when $K \gg 1$ (i.e. when the interactions are much stronger than thermal fluctuations) one would expect the system to be ordered.

In one dimension, as Ising himself showed, the model hosts no phase transitions. However, the two-dimensional Ising model is one of the simplest models which does exhibit a phase transition. There are various approximation and expansion schemes dedicated to finding the critical coupling K_{c} (or critical temperature $T_{\mathrm{c}} \equiv 1/K_c$, given in units of J/k_{B}) at which the system transitions from a disordered to an ordered state, as well as for finding the scaling of physical observables near criticality, which is dictated by the universality class of the model.

While at first glance the Ising model seems very amenable to numerical simulation, naive calculation of observables via direct enumeration of all possible spin configurations very quickly becomes computationally intractable beyond very small lattices in a very small number of dimensions. For numerical simulations of the Ising model to be feasible, we need an efficient way of generating spin configurations $\{s_i\}$ in such a way that the configurations have the correct Boltzmann weights $e^{-\beta E(\{s_i\})}$.

1.2 Monte Carlo Methods

Suppose the spin configurations are Boltzmann-distributed. That is, each spin configuration $\{s_i\}$ occurs with probability

$$P(\{s_i\}) = \frac{e^{-\beta E(\{s_i\})}}{\sum_{\{s_i\}} e^{-\beta E(\{s_i\})}} = \frac{1}{Z} e^{-\beta E(\{s_i\})}.$$

If there are N spins in the system, then there are 2^N possible spin configurations, and the expected value of an observable $\mathcal{O}$ is given by

$$\langle \mathcal{O} \rangle = \sum_{\{s_i\}} \mathcal{O}(\{s_i\}) P(\{s_i\}).$$

The idea behind all Monte Carlo methods is to generate some sample of $M \ll 2^N$ configurations $\{s_m\}$ such that for $N \gg 1$ the expected value of the observable $\mathcal{O}$ can be approximated by the sample average of $\mathcal{O}$:

$$\langle \mathcal{O} \rangle \approx \frac{1}{M} \sum_m \mathcal{O}(\{s_m\}).$$

The most famous method for generating sample configurations is known as the Metropolis algorithm. Very briefly, the Metropolis algorithm for an Ising system goes as follows:

1. Choose some initial spin configuration (e.g., set each spin to $+1$ or -1 with equal probability), with fixed system temperature β.
2. Make a trial change to the spin configuration (i.e., flip the sign of a randomly-chosen spin).
3. Compute the change in energy ΔE between the initial spin configuration and the configuration after the trial spin flip. If $\Delta E < 0$, accept the trial flip with probability 1. If $\Delta E > 0$, accept the trial flip with probability $w = e^{-\beta \Delta E}$.
4. Repeat step 3 many times until the system reaches thermal equilibrium.
5. Calculate averages of physical observables.

Unfortunately, near the critical coupling K_c, algorithms such as Metropolis, which are based on flipping a single spin at a time and are therefore inherently local, suffer from so-called "critical slowing down" resulting from the scale-invariance of fluctuations near a critical point. This means that as one approaches the phase transition, such algorithms struggle to efficiently generate relevant spin configurations.

1.3 Cluster Monte Carlo

To avoid the problems created by critical slowing down, so-called Cluster Monte Carlo were developed, in which instead of flipping a single spin at a time, large clusters of spins are flipped simultaneously with a high probability.

1.3.1 Swendsen-Wang Algorithm

One of the original cluster Monte Carlo algorithms was developed by Swendsen and Wang in 1987 [Phys. Rev. Lett., 58 (2): 86 (1987).]. To construct the clusters that will later be flipped, we introduce a bond index b connecting two interacting spins $s_{i(b)}s_{j(b)}$, where $b = 1, 2, \ldots, dN$ for a d dimensional cubic lattice with N sites. The total energy of a given spin configuration $\{s_i\}$ is then

$$E(\{s_i\}) = -|J| \sum_{b=1}^{dN} [1 + s_{i(b)}s_{j(b)}] = \sum_{b=1}^{dN} E_b.$$

The gives us the following partition function:

$$Z = \sum_{\{s_i\}} e^{-\beta E(s_i\})} = \sum_{\{s_i\}} e^{\beta E_b} = \sum_{\{s_i\}} \prod_{b=1}^{dN} [1 + (e^{\beta E_b} - 1)] \equiv \sum_{\{s_i\}} \prod_{b=1}^{dN} [F_b(0) + F_b(1)],$$

for a bond function F_b defined on $\{0, 1\}$ as

$$F_b(s) = \begin{cases} 1 & \text{for } s = 0 \\ e^{\beta E_b} - 1 & \text{for } s = 1 \end{cases}.$$

Note that the bond function F_b depends on the value of the spins $s_{i(b)}$, $s_{j(b)}$ connected by bond b:

$$\begin{aligned} F_b(0) =& 1 \, \forall s_{i(b)}, s_{j(b)} \\ F_b(1) =& \begin{cases} e^{2\beta|J|} - 1 & \text{for } s_{i(b)} = s_{j(b)} \\ 0 & \text{for } s_{i(b)} \neq s_{j(b)} \end{cases} \end{aligned}$$

For each bond b, we can define a bond variable $\tau_b = \pm 1$, allowing us to rewrite the partition function as a sum over spin configurations $\{s_i\}$ and bond configurations $\{\tau_b\}$:

$$Z = \sum_{\{s_i\}} \sum_{\{\tau_b\}} \prod_{b=1}^{dN} F_b(\tau_b).$$

A bond with $\tau_b = 1$ corresponds to a bond between parallel spins, and is known as a filled bond. We can see that the probability weight for a given spin and bond configuration is

$$W(\{s_i\}, \{\tau_b\}) = \prod_{b=1}^{dN} F_b(\tau_b) = (e^{2\beta|J|} - 1)^{N_f},$$

where N_f is the number of filled bonds on the lattice. For a given spin configuration, the probability of bond configuration $\{\tau_b\}$ is

$$P(\{\tau_b\}) = \prod_{b=1}^{dN} P_b(\tau_b), \text{ where } P_b(\tau_b) = \frac{F_b(\tau_b)}{F_b(0)+F_b(1)}.$$

Therefore the probability associated with a filled bond $\tau_b = 1$ is

$$P_b(\tau_b = 1) = \begin{cases} 1 - e^{2\beta|J|} & \text{for } s_{i(b)} = s_{j(b)} \\ 0 & \text{for } s_{i(b)} \neq s_{j(b)} \end{cases}$$

Note that for a given spin and bond configuration, if we simultaneously flip all spins in a cluster connected by filled bonds, the weight $W(\{s_i\},\{\tau_b\}) = (e^{2\beta|J|} - 1)^{N_f}$ is unchanged. With this observation, we can define a single Monte Carlo step of the Swendsen-Wang algorithm as follows:

1. Start with a random configuration of spins
2. Populate filled bonds between spins $s_{i(b)}$, $s_{j(b)}$ according to the probabilities

$$P_b(\tau_b = 1) = \begin{cases} 1 - e^{2\beta|J|} & \text{for } s_{i(b)} = s_{j(b)} \\ 0 & \text{for } s_{i(b)} \neq s_{j(b)} \end{cases}$$

3. Flip all resulting clusters of spins with probability 1/2

1.3.2 Wolff Algorithm

A somewhat simpler cluster Monte Carlo algorithm was published in 1989 by Wolff [Phys. Rev. Lett., 62 (4): 361 (1989).]. In the Wolff algorithm, a single spin cluster is constructed at each Monte Carlo step and flipped with probability 1. A single Monte Carlo step of the Wolff algorithm goes as follows:

1. Start with a random spin configuration
2. Select a lattice site i at random. This site will be the "seed" of the cluster
3. For each of i's nearest neighbors j, if i and j are parallel and the bond between i and j has not already been counted, add j to the cluster with probability $1 - e^{-2\beta J}$
4. Place each spin j that gets added to the cluster onto the stack. After all of i's neighbors have been considered, repeat step 2 with the site j that is on the top of the stack. Repeat until the stack is empty

5. Finally, flip all spins in the cluster.

2 Implementation

I've written a small Python module that performs temperature sweeps of a zero-field Ising system with nearest neighbor interactions on a cubic lattice in any integer number of dimensions using the Wolff algorithm, and calculates physical observables such as magnetization, susceptibility, specific heat, and spin-spin correlation at each temperature. The program consists of three classes: `Lattice`, `Ising`, and `IsingTempSeries`.

- `Lattice` class: Defines a cubic lattice with a given linear size, in a given dimension, and at a given temperature (in units of J/k_B). Includes methods for generating a random spin configuration, selecting a spin at random, finding nearest neighbors (with periodic boundary conditions), building clusters according to the Wolff algorithm, etc.

- `Ising` class: performs N_eq "equilibration" iterations of the Wolff algorithm ($N_\mathrm{eq} = 10^4$ by default, but this is probably overkill) on a `Lattice` at a given temperature, followed by $N_\mathrm{eq}/2$ "measurement" iterations. Calculates magnetization per spin, susceptibility, and specific heat, Binder ratios, and spin-spin correlation averaged over all measurement iterations.

- `IsingTempSeries` class: Simulates an `Ising` object for a given set of temperatures. Records and plots observables and diagnostics as a function of temperature.

See Appendix A for a listing of the code, which is intended to be run inside a Jupyter Notebook but could be easily modified to run from a terminal. To simulate, for example, a 20×20 lattice at 50 temperatures from J/k_B to $4J/k_\mathrm{B}$, one could run the following commands in a Jupyter Notebook (assuming the module is defined in `ising.py` and is in the Python path):

```python
from ising import *
%matplotlib inline

ising_20_2_50 = IsingTempSeries(20, 2, 1, 4, 50)
ising_20_2_50.do_series()
ising_20_2_50.save('ising_20_2_50')
```

(Note: The cluster-building portion of the code is written in a recursive fashion. For large lattices or in many dimensions, you may hit the system recursion depth limit. There are ways around this issue, but for this project I was limited by computation time, not recursion depth. Systems up to about $N = 10^3$ total lattice sites can simulated on a reasonable timescale on an old MacBook Pro.)

3 Results

I will report the following physical observables/diagnostics, using $\langle x \rangle$ to denote the average of quantity x over the $N_{\rm eq}/2$ measurement iterations:

- Magnetization: $\langle m \rangle \equiv \langle \frac{1}{N} \sum_i s_i \rangle$.
- Susceptibility: $\chi \equiv \frac{N}{T}(\langle m^2 \rangle - \langle m \rangle^2)$.
- Specific heat: $C \equiv \frac{1}{T^2}(\langle \epsilon^2 \rangle - \langle \epsilon \rangle^2)$, where ϵ is the energy per spin for a given spin configuration
- Spin-spin correlation function: $\langle s_0 s_r \rangle - \langle m \rangle^2$
- First Binder ratio $Q \equiv Q_2 = \frac{\langle M^2 \rangle}{\langle |M| \rangle^2}$, where M is the total magnetization of a given configuration. At low temperatures, $Q \to 0$, at high temperatures, $Q \to \pi/2$, and the point at which Q ratios from different lattices sizes are equal provides an accurate estimate for $T_{\rm c}$.

The critical exponents for two, three, and five dimensions are given in Table 1.

Table 1: Critical exponents for the cubic Ising model.

	$d=2$	$d=3$	$d=5$
α	0	0.11008(1)	-1
β	1/8	0.326419(3)	1/3
γ	7/4	1.237075(10)	1
η	1/4	0.036298(2)	-1
ν	1	0.629971(4)	1/3

3.1 Two Dimensions

In two dimensions, I simulated lattices with linear size $L = 4, 6, 8, 12, \& \, 20$ at 50 temperatures from 1 to 4 (in units of $J/k_{\rm B}$), and lattices with linear size $L = 32, \& \, 40$ at 15 temperatures from 2 to 3. Note that the two dimensional Ising model on a square lattice is exactly solvable, and the critical temperature is $T_{\rm c,\,2D} = 2/\log(1+\sqrt{2}) \approx 2.2691853$. Figure 1 shows observables and diagnostics resulting from $N_{\rm eq} = 10^4$ thermal equilibration Monte Carlo steps and $N_{\rm eq}/2$ measurement steps. As expected, we see peaks in the susceptibility and specific heat near $T_{\rm c,\,2D}$, and the Binder ratios for different lattice sizes all intersect very close to $T_{\rm c,\,2D}$.

Figure 2 shows the spin-spin correlation function $\langle s_0 s_r \rangle - \langle m \rangle^2$ for various temperatures and lattice sizes, as well as a semi-log plot of the correlation length ξ derived from a fit to $\langle s_0 s_r \rangle - \langle m \rangle^2 = e^{-r/\xi}$. As expected, there is peak in the correlation length near the critical temperature. Near $T_{\rm c}$,

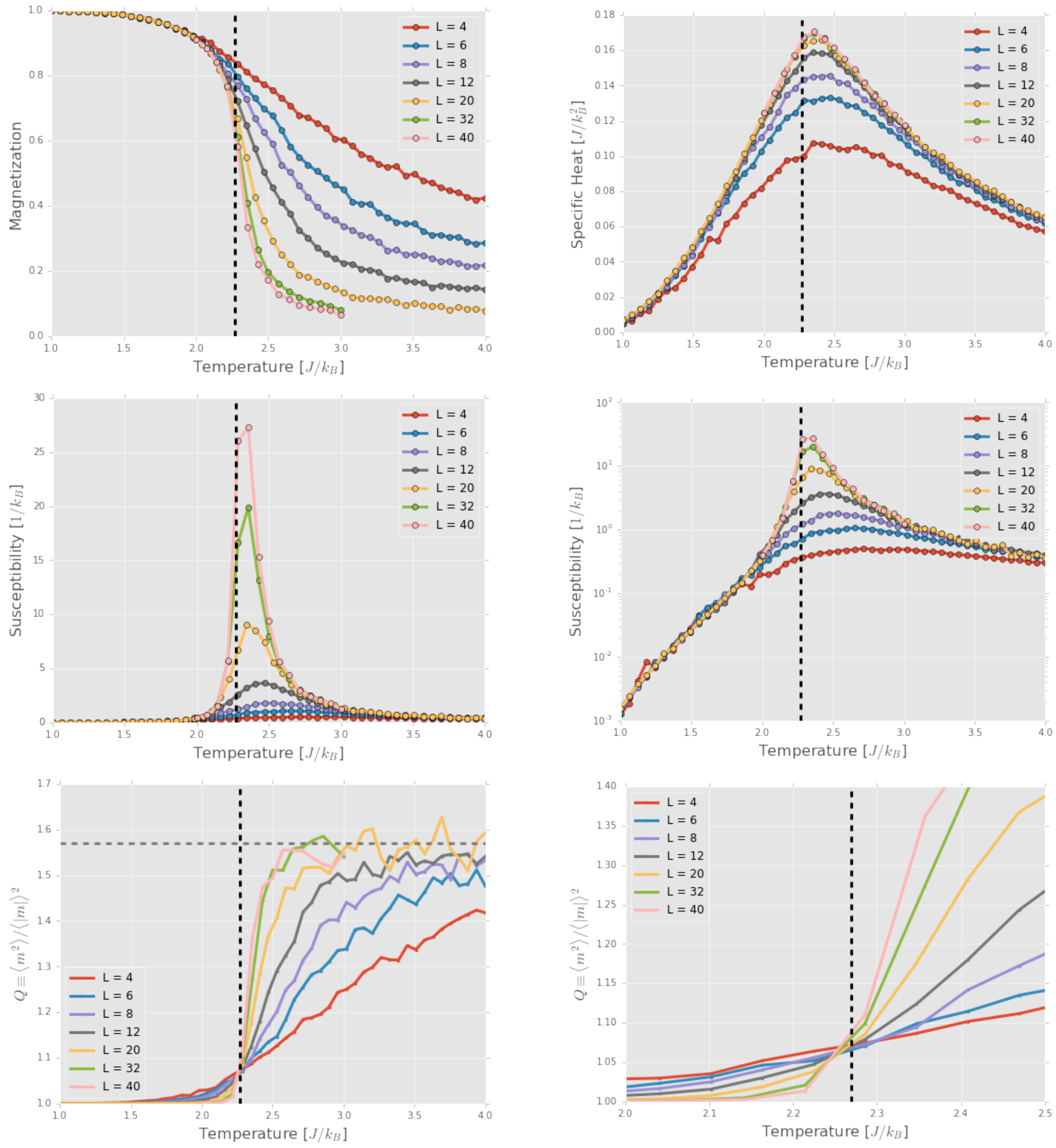

Figure 1: Two dimensions. Top row: Magnetization per spin (left) and specific heat (right). Middle row: susceptibility on linear (left) and semi-log (right) scales. Bottom row: Binder ratio Q for the whole temperature sweep (left) and near T_c (right). Dashed vertical line indicates exact critical temperature $T_{c,\,2D} = 2/\log(1+\sqrt{2})$. Dashed horizontal line in the Binder ratio figure indicates theoretical high-temperature limit of $Q = \pi/2$.

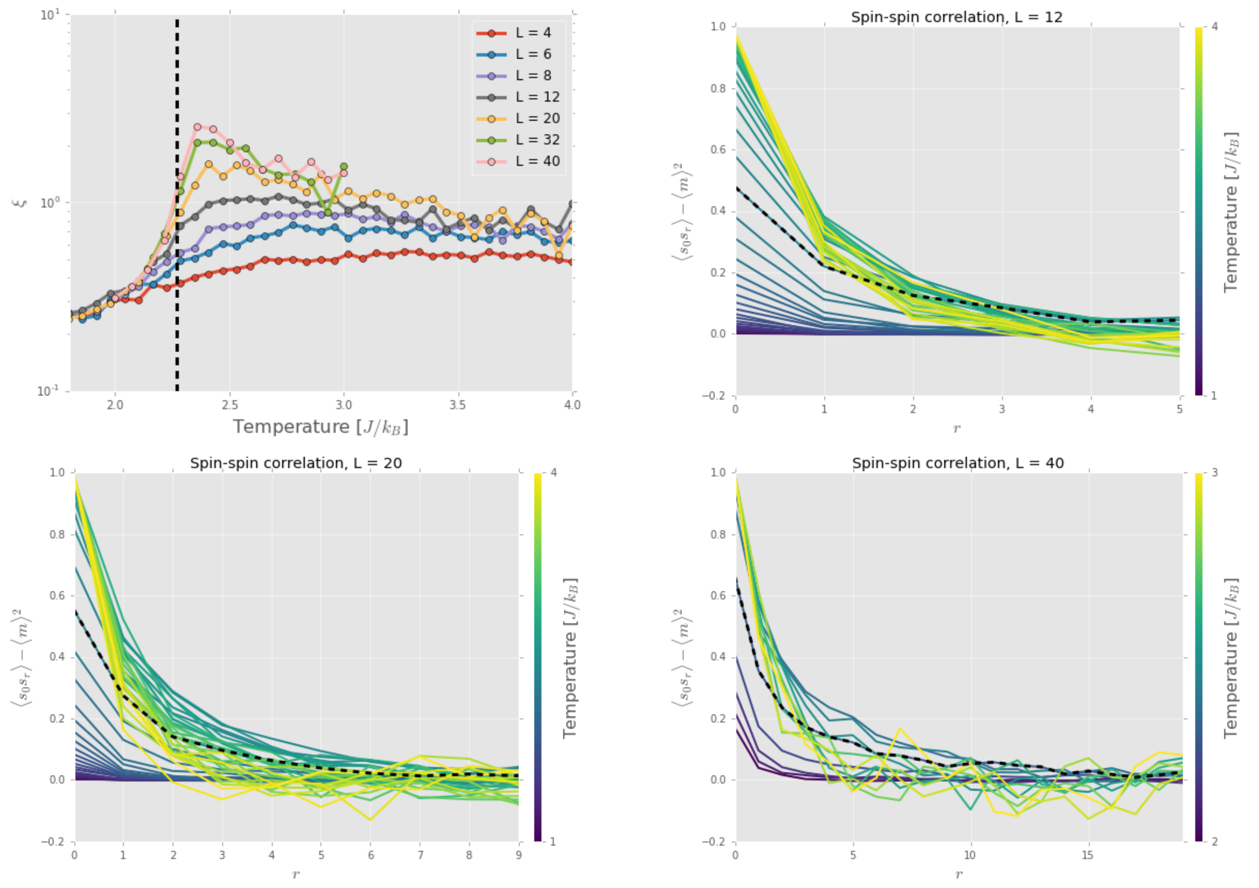

Figure 2: Upper left: Semi-log plot of correlation length ξ (extracted from a fit to an exponential $\propto e^{-r/\xi}$) as a function of temperature for various lattice sizes. Other panels: spin-spin correlation as a function of distance r and temperature for $L = 12$, 20, & 40. Dotted line indicates temperature closest to T_c.

the model has the following critical scaling behavior (with $t \equiv |T_c - T|/T_c$).

$$\begin{aligned}\langle m\rangle &\sim t^{\beta}\\ \chi &\sim t^{-\gamma}\\ C &\sim t^{-\alpha}\\ \xi &\sim t^{-\nu}\\ \langle s_0 s_r\rangle - \langle m\rangle^2 &\sim \frac{1}{r^{d-2+\eta}} \text{ at } T_c\end{aligned}$$

where χ is the susceptibility, C is the specific heat, ξ is the correlation length, $\langle s_0 s_r\rangle$ is the spin-spin correlation function, and d is the dimensionality of the system. Below T_c we can fit the magnetization as a function of $T_c - T$ to a function of the form $A(T_c - T)^{\beta}$ with A and β free parameters. For the lattice sizes I considered, the magnetization clearly does not follow a power law, let alone one with the correct exponent (see Figure 3). A power law fit to the magnetization of the largest two-dimensional lattice ($L = 40$) yields $\beta = 0.10774231$, roughly 14% smaller than the exact value of $\beta = 1/8$. For smaller lattices, the observed β is even smaller.

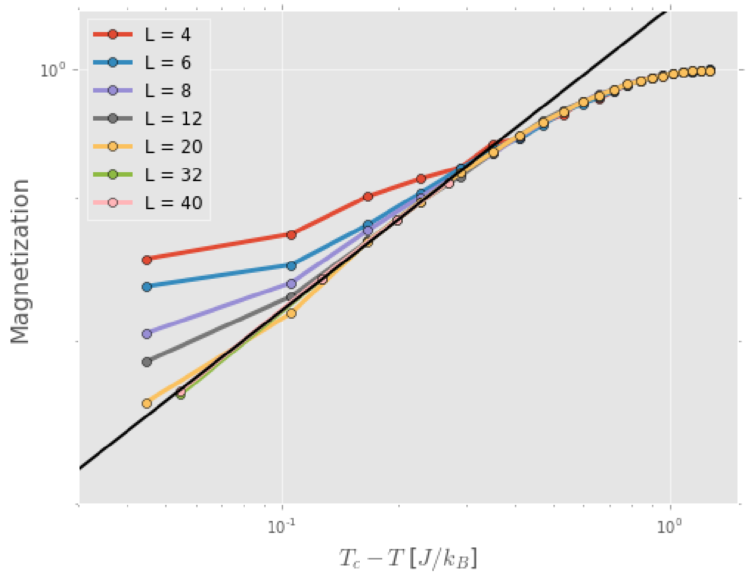

Figure 3: Log-log plot of the magnetization as a function of $T_c - T$ in two dimensions. Black line is a power law fit to the $L = 40$ curve, yielding $\beta = 0.10774231$.

We can apply finite size scaling techniques to estimate other critical exponents such as γ and ν. In particular, the maximum value of the susceptibility on a lattice of linear size L should go as $\chi_{\max} \sim L^{\gamma/\nu}$, and the reduced temperature at which the maximum occurs should go as $t_{\max} \sim L^{-1/\nu}$. So we can estimate γ and ν performing a power law fit on $\chi_{\max}(L)$ and $t_{\max}(L)$. In two dimensions, this procedure yields estimates of $\gamma/\nu \approx 1.622$, $\nu \approx 1.231$, $\gamma \approx 1.997$. The exact values are $\nu = 1$ and $\gamma = \gamma/\nu = 7/4 = 1.75$. Figure 4 shows $\chi_{\max}(L)$ and $t_{\max}(L)$ on a log-log scale. We can see that $\chi_{\max}(L)$ appears to follow a power law, but it seems that the point density near T_c was not large enough to get an accurate $t_{\max}$.

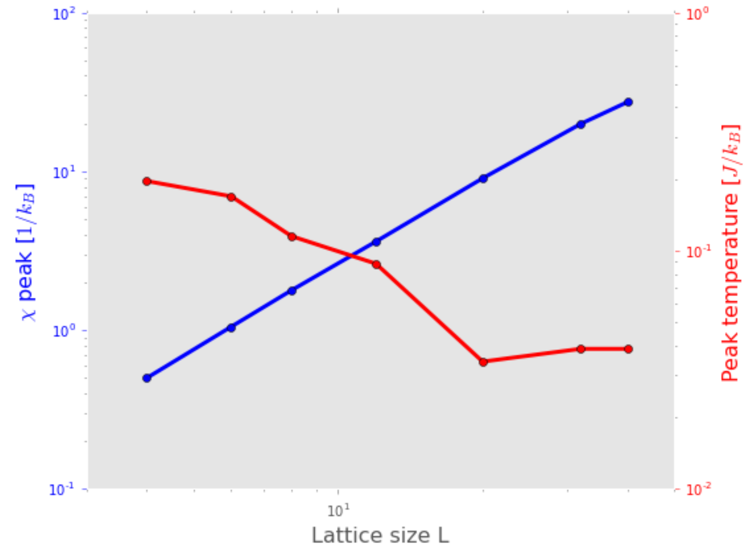

Figure 4: Log-log plot of $\chi_{\max}(L)$ and $t_{\max}(L)$, yielding exponents of $\gamma/\nu \approx 1.622$ and $\nu \approx 1.231$.

3.2 Three Dimensions

In three dimensions, I simulated lattices with linear size $L = 3,\ 4,\ 6,\ \&\ 8$ at 50 temperatures from 2 to 8, and lattices with linear size $L = 10\,\&\,12$ at 15 temperatures from 3.5 to 5.5. I will be comparing to the published critical temperature of $T_{\text{c, 3D}} \approx 1/0.2216544 \approx 4.511528$ [J. Phys. A 29: 5727-5734 (1996)]. Once again, we see peaks in the susceptibility and specific heat near T_c, and the Binder ratios meet very near to T_c.

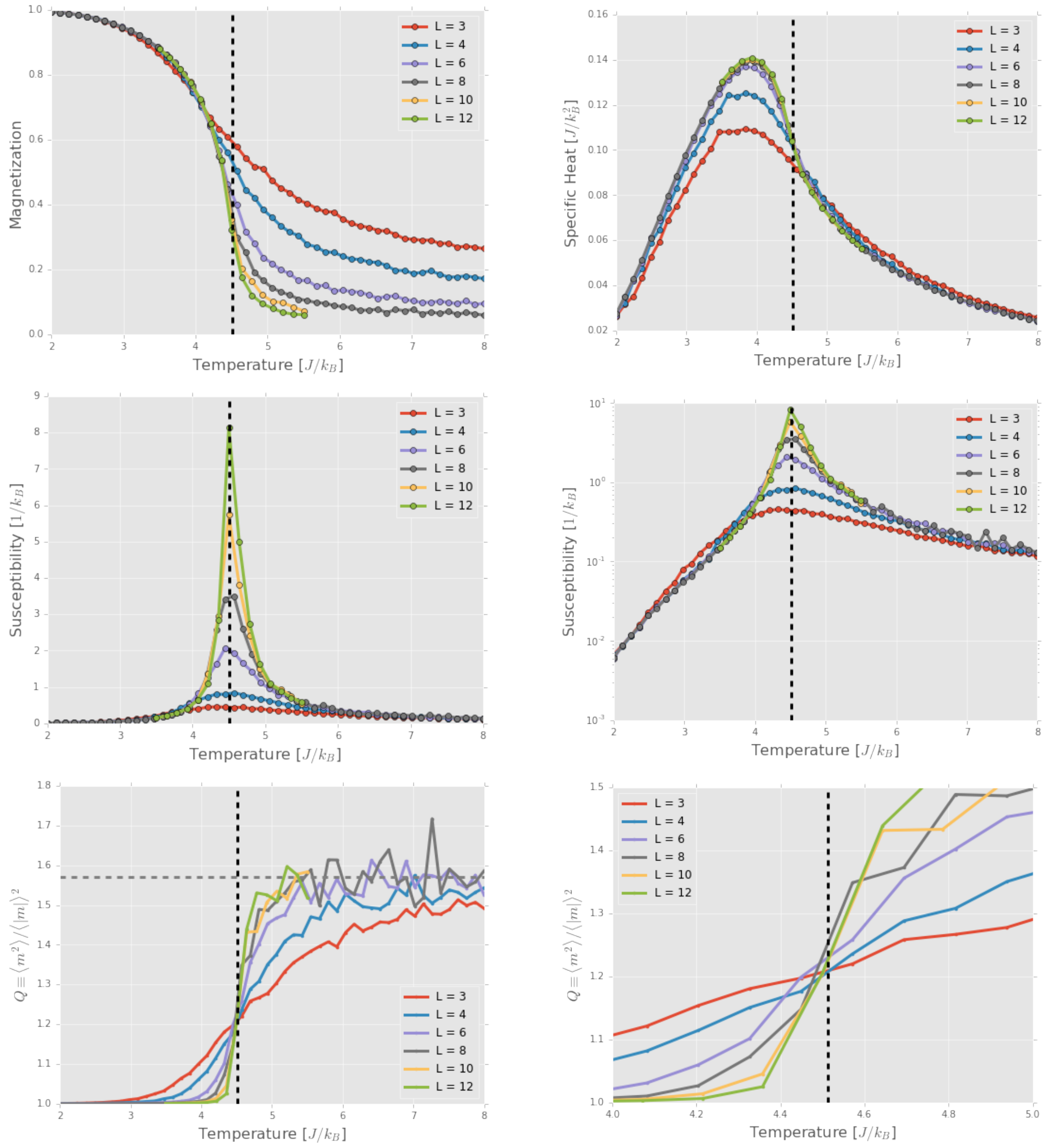

Figure 5: Three dimensions. Top row: Magnetization per spin (left) and specific heat (right). Middle row: susceptibility on linear (left) and semi-log (right) scales. Bottom row: Binder ratio Q for the whole temperature sweep (left) and near T_c (right). Dashed vertical line indicates critical temperature $T_{c,\ 3D} \approx 1/0.2216544 \approx 4.511528$ [J. Phys. A 29: 5727-5734 (1996)]. Dashed horizontal line in the Binder ratio figure indicates theoretical high-temperature limit of $Q = \pi/2$.

Figure 6 shows the spin-spin correlation function $\langle s_0 s_r \rangle - \langle m \rangle^2$ for various temperatures and lattice sizes, as well as a semi-log plot of the correlation length ξ derived from a fit to $\langle s_0 s_r \rangle - \langle m \rangle^2 = e^{-r/\xi}$. As expected, there is peak in the correlation length near the critical temperature.

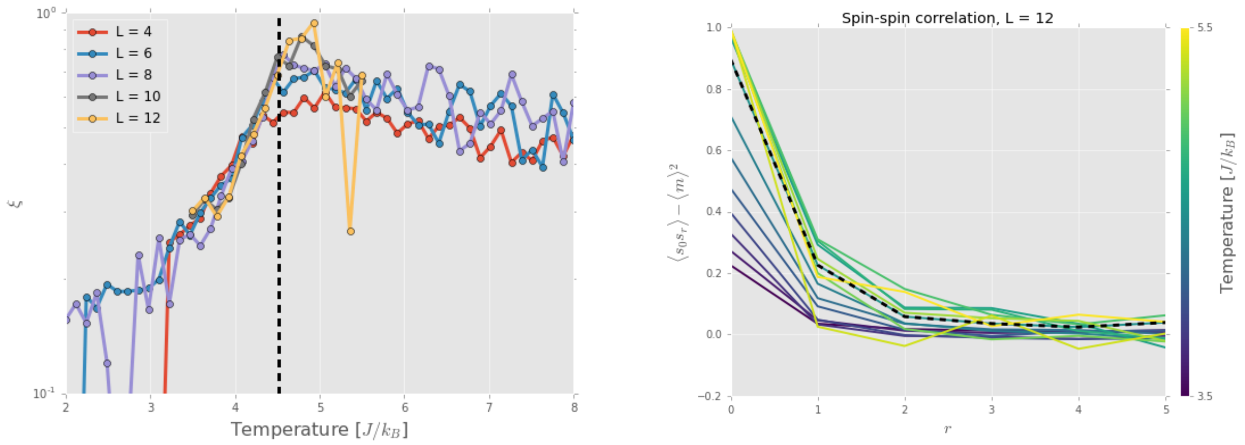

Figure 6: Left: Semi-log plot of correlation length ξ (extracted from a fit to an exponential $\sim e^{-r/\xi}$) as a function of temperature for various lattice sizes. Right: spin-spin correlation as a function of distance r and temperature for $L = 12$. Dotted line indicates temperature closest to T_c.

Figure 7 shows the magnetization as a function of temperature on a log-log scale, along with a power law fit to the $L = 12$ data, yielding a best-fit of $\beta = 0.23964831$, roughly 27% below the literature value of $\beta = 0.326419$ [J. Phys. A 29: 5727-5734 (1996)].

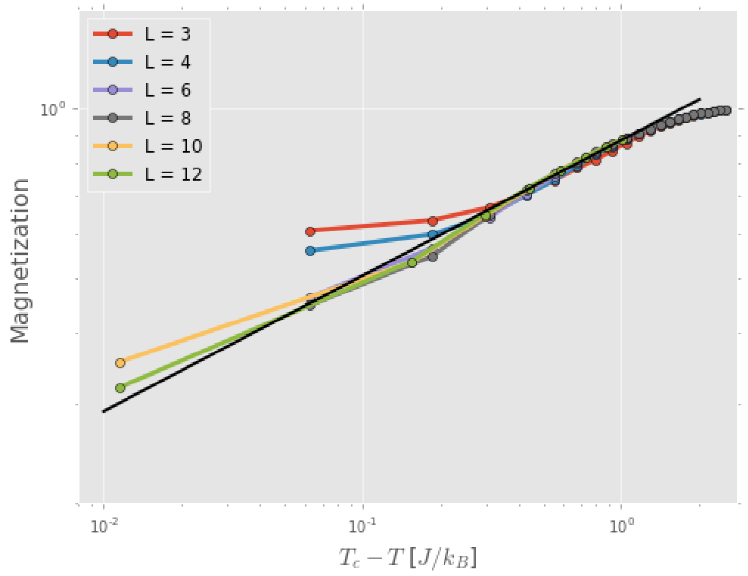

Figure 7: Log-log plot of the magnetization as a function of $T_\mathrm{c} - T$ in three dimensions. Black line is a power law fit to the $L = 12$ curve, yielding $\beta = 0.23964831$.

Fitting $\chi_{\max}(L)$ and $t_{\max}(L)$ yields critical exponent estimates of $\gamma/\nu \approx 2.033$, $\nu \approx 0.547$, $\gamma \approx 1.113$. The published values are $\nu = 0.629971$ and $\gamma = 1.237075$. This amounts to a 13% error in ν and a 10% error in γ. Once again the temperature point density near T_c appears to be an issue in calculating ν.

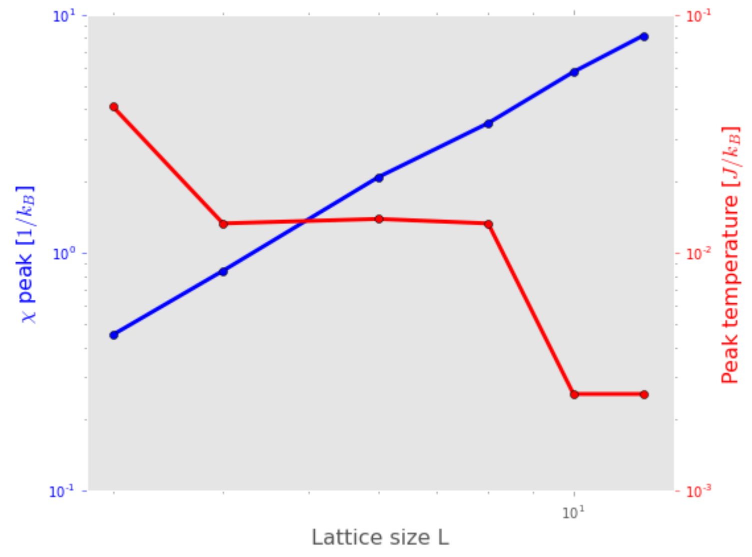

Figure 8: Log-log plot of $\chi_{\max}(L)$ and $t_{\max}(L)$, yielding exponents of $\gamma/\nu \approx 2.033$ and $\nu \approx 0.547$.

3.3 Five Dimensions

In five dimensions, I simulated lattices with linear size $L = 2\,\&\,3$ at 50 temperatures from 2 to 12, and lattices with linear size $L = 4$ (5) at 15 (10) temperatures from 8 to 9.5. I will be comparing to the published critical temperature of $T_{\text{c, 5D}} \approx 1/0.11391498 \approx 8.7784767$ [Nuclear Physics B 895, 305-318 (2015)].

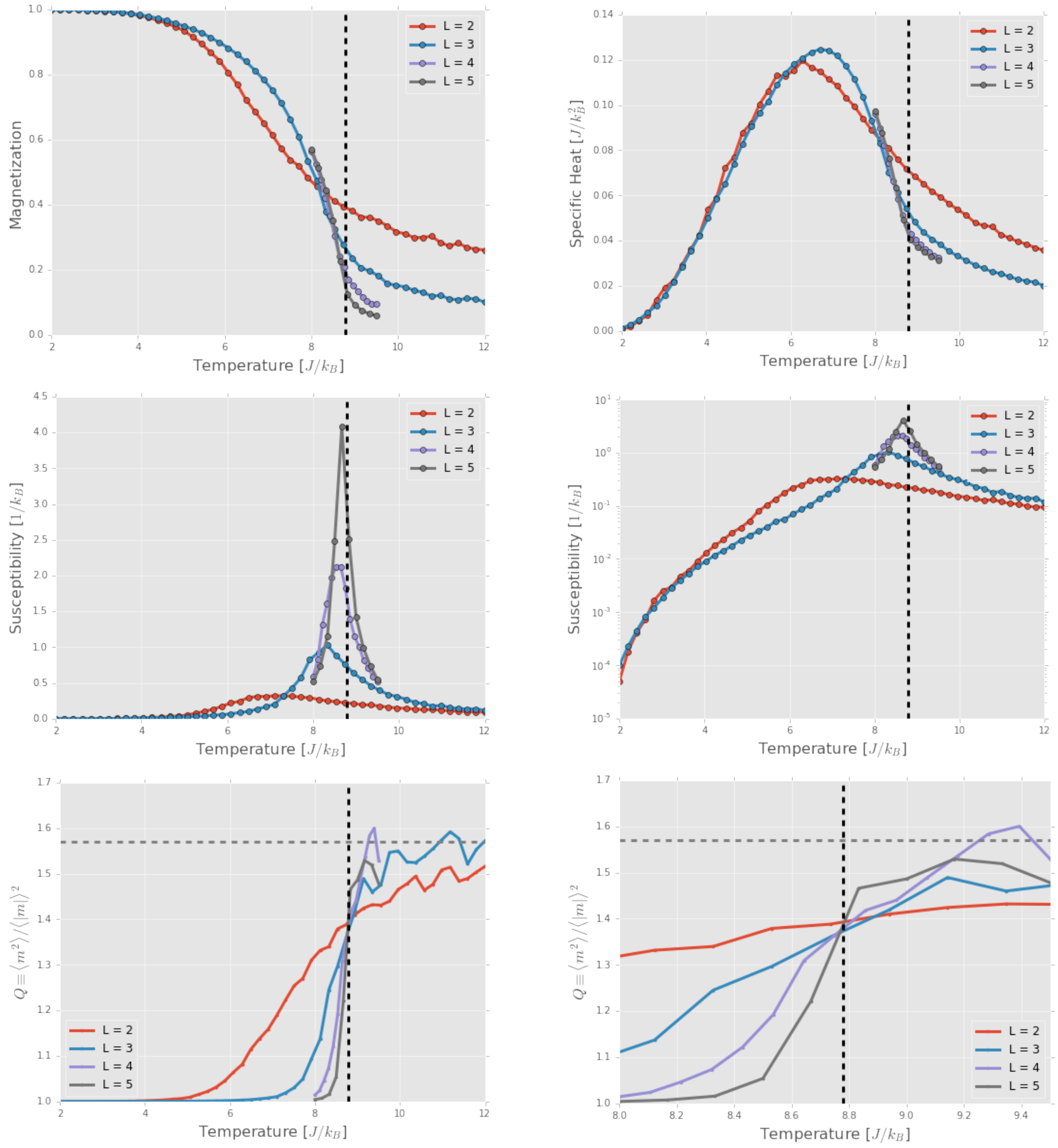

Figure 9: Five dimensions. Top row: Magnetization per spin (left) and specific heat (right). Middle row: susceptibility on linear (left) and semi-log (right) scales. Bottom row: Binder ratio Q for the whole temperature sweep (left) and near T_c (right). Dashed vertical line indicates critical temperature $T_{c,\,5D} \approx 1/0.11391498 \approx 8.7784767$ [Nuclear Physics B 895, 305-318 (2015)]. Dashed horizontal line in the Binder ratio figures indicates theoretical high-temperature limit of $Q = \pi/2$.

Figure 10 shows the magnetization as a function of temperature on a log-log scale, along with a power law fit to the $L = 4$ data, yielding a best-fit of $\beta = 0.39612343$. The mean field value for β, which is exact in five dimensions is $\beta = 1/(5-2) = 1/3$, an error of 19%. Figure 11 shows a log-log plot of correlation length ξ as a function of temperature for $L = 4$ and $L = 5$. One could convince oneself that there is a peak in ξ at T_c.

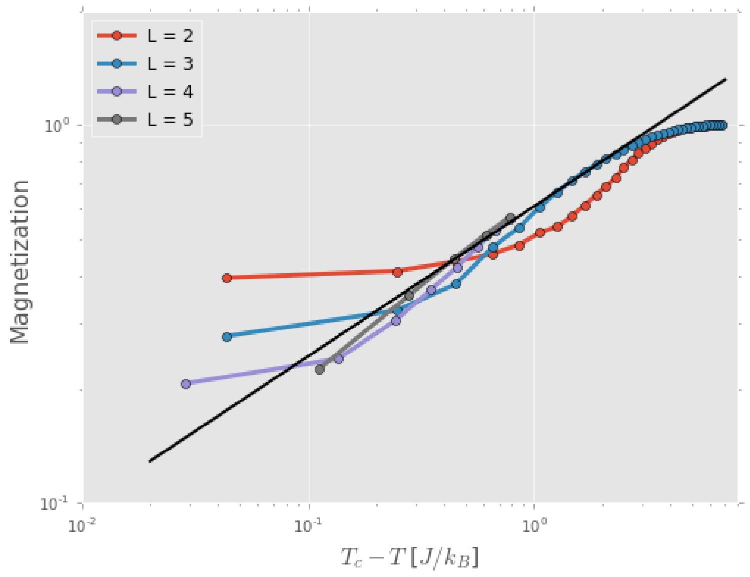

Figure 10: Log-log plot of the magnetization as a function of $T_c - T$ in five dimensions. Black line is a power law fit to the $L = 4$ curve, yielding $\beta = 0.39612343$.

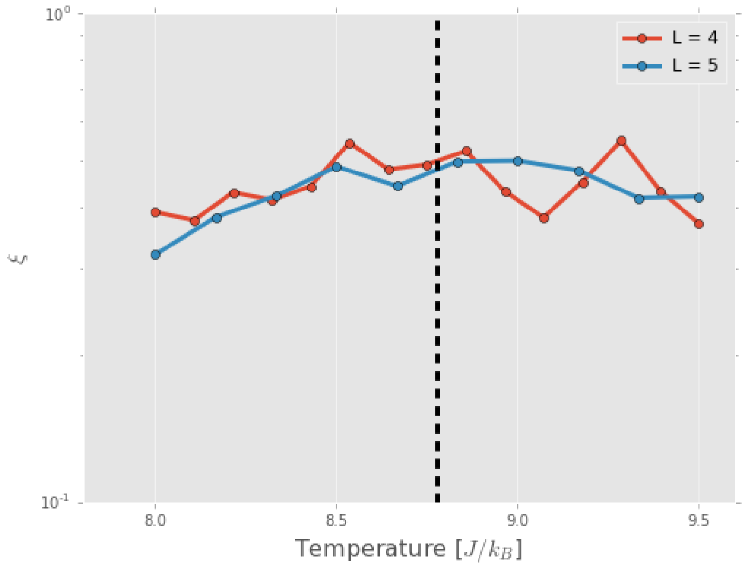

Figure 11: Semi-log plot of correlation length ξ (extracted from a fit to an exponential $\sim e^{-r/\xi}$) as a function of temperature for $L = 4$, & 5.

Fitting $\chi_{\max}(L)$ and $t_{\max}(L)$ yields critical exponent estimates of $\gamma/\nu \approx 2.789$, $\nu \approx 0.328$, $\gamma \approx 0.915$. The exact (mean field) values are $\nu = 1/3$ and $\gamma = 1$. This amounts to a 1.6% error in ν and an 8.5% error in γ. It appears we get a hint of mean field theory on small lattices in five dimensions.

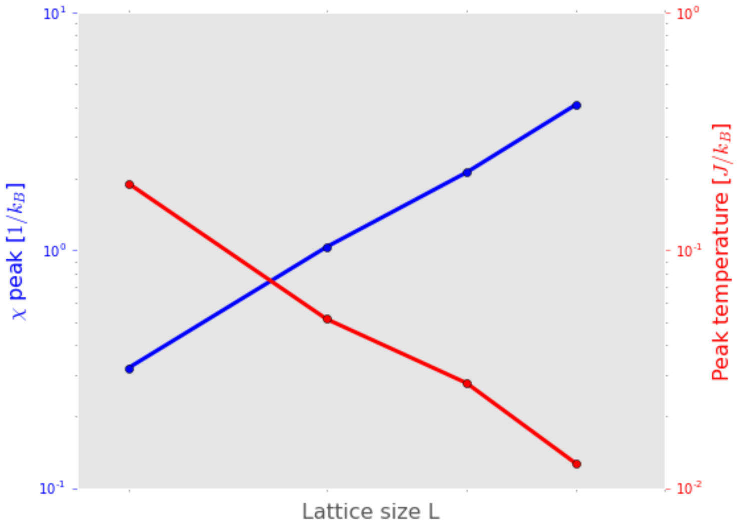

Figure 12: Log-log plot of $\chi_{\max}(L)$ and $t_{\max}(L)$, yielding exponents of $\gamma/\nu \approx 2.789$ and $\nu \approx 0.328$.

Given more time, I would simulate larger lattices with more temperatures near T_c to try to get more reasonable values for the critical exponents.

A Appendix

```python
import pickle
from tqdm import tnrange
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')

class Lattice:
    """ Defines a square lattice of linear size L in dimension dim
        at temperautre T (in units of J/kB).
        Methods:
            - init_lattice(): populates lattice with +/- 1 at random
            - random_site(): returns the position of a randomly-chosen
                site on the lattice
            - get_neighbors(pos):
                + input: pos, a list of length dim,
                    which defines a position on the lattice
                + returns: a list of the positions of the 2 * dim
                    nearest neighbors of pos
            - get_spin(pos): returns the value (-1 or 1)
                of the spin at position pos
            - flip_spin(pos): flips the spin at position pos
            - build_cluster(pos, cluster, visited):
                + inputs:
                    - pos, a site on the lattice
                    - cluster, a list of lattice sites
                        making up the current cluster
                    - visited, a list of lattice sites
                        that have already been visited
                + returns: cluster, the final list of
                    lattice sites in the cluster
            - get_config(): returns the current
                configuration of the lattice (array of +/-1)
            - display(): dislays the current spin configuration
                in image form if dim==2
    """
    def __init__(self, L, dim, T):
        self.L = L
        self.dim = dim
        self.T = T
        self.init_lattice()

    def init_lattice(self):
        self.config = np.random.choice(
            [-1, 1],
            size=tuple([self.L] * self.dim))

    def random_site(self):
        return list(np.random.randint(0, self.L, size=self.dim))

    def get_neighbors(self, pos):
        dim = len(pos)
        neighbors = []
        for i in range(dim):
            neighbors.append(pos[:i] + [(pos[i]-1)%self.L] + pos[(i+1):])
            neighbors.append(pos[:i] + [(pos[i]+1)%self.L] + pos[(i+1):])
        return neighbors

    def get_spin(self, pos):
        return self.config[tuple(pos)]

    def flip_spin(self, pos):
        self.config[tuple(pos)] = -self.config[tuple(pos)]

    def build_cluster(self, pos, cluster, visited):
        spin = self.get_spin(pos)
        neighbors = self.get_neighbors(pos)
        cluster.append(pos)
        visited.append(pos)
        prob = 1 - np.exp(-2 / self.T)
        for n in neighbors:
            if self.get_spin(n) == spin:
                if n not in visited and np.random.random() < prob:
                    cluster = self.build_cluster(n, cluster, visited)
        return cluster

    def display(self):
        if self.dim == 2:
            self.image = plt.imshow(self.config)
            plt.show()

class Ising:
    """ Simulates the Ising model on a lattice of linear size
        L in dimension dim at temperature T (in units of J/kB).
        Methods:
            - flip_cluster(cluster): flips the spin of
                every spin in cluster
            - run_wolff(): performs a single iteration of
                the Wolff algorithm
            - simulate(Neq, progbar=False): performs Neq equilibration
                iterations of run_wolff(),
                then Neq//2 measurement iterations. Calculates average
                magnetization, susceptibility, and specific heat per spin
                Displays a tqdm progress bar if progbar==True.
            - get_mag_per_spin(config): returns the magnetization per spin
                for single config
            - get_specific_heat(config): returns the specific heat
            per spin for single config
            - get_energy_per_spin(config, rs): returns energy per spin
                and energy**2 per spin for single config
                (used for calculating specific heat)
            - get_spin_spin_corr(config, rs): for a sigle config,
                returns array containing the product of the spin at
                the origin and the spin r lattice sites away for r=1:L-1
    """
    def __init__(self, L, dim, T):
        self.lattice = Lattice(L, dim, T)
        self.L = L
        self.dim = dim
        self.T = T
        self.N = L**dim
        self.rs = np.arange(0, self.L // 2)

    def flip_cluster(self, cluster):
        for s in cluster:
            self.lattice.flip_spin(s)

    def run_wolff(self):
        start = self.lattice.random_site()
        cluster = self.lattice.build_cluster(start, [], [])
        self.flip_cluster(cluster)

    def simulate(self, Neq, progbar=False):
        if progbar:
            print('Running {} eqilibration iterations.'.format(Neq))
            for _ in tnrange(Neq):
                self.run_wolff()
        else:
            for _ in range(Neq):
                self.run_wolff()
        Nruns = Neq // 2
        mag_tot = 0
        mag2_tot = 0
        C_tot = 0
        s0sr_tot = np.zeros(len(self.rs))
        if progbar:
            print('Done with equilibration.')
            print('Running {} measurement iterations.'.format(Nruns))
            for _ in tnrange(Nruns):
                self.run_wolff()
                config = self.lattice.config
                m = self.get_mag_per_spin(config)
                mag_tot += m
                mag2_tot += m**2
                C_tot += self.get_specific_heat(config)
                s0sr_tot += self.get_spin_spin_corr(config, self.rs)
        else:
            for _ in range(Nruns):
                self.run_wolff()
                config = self.lattice.config
                m = self.get_mag_per_spin(config)
                mag_tot += m
                mag2_tot += m**2
                C_tot += self.get_specific_heat(config)
                s0sr_tot += self.get_spin_spin_corr(config, self.rs)
        self.Neq = Neq
        self.Nruns = Nruns
        self.mag_avg = mag_tot / Nruns
        self.mag2_avg = mag2_tot / Nruns
        self.susc = (mag2_tot / Nruns - (mag_tot / Nruns)**2) / self.T
        self.C = C_tot / Nruns
        self.binderQ = (mag2_tot / Nruns) / (mag_tot / Nruns)**2
        self.s0sr = s0sr_tot / Nruns

    def get_mag_per_spin(self, config):
        return abs(config.mean())

    def get_specific_heat(self, config):
        E, Esq = self.get_energy_per_spin(config)
        return (Esq - E**2) / self.T**2

    def get_energy_per_spin(self, config):
        E = 0
        Esq = 0
        for pos, _ in np.ndenumerate(config):
            Enn = (-0.5 * sum([config[pos] *
                    config[tuple(n)] for n in
                    self.lattice.get_neighbors(list(pos))]))
            E += Enn
            Esq += Enn**2
        E /= self.N
        Esq /= self.N
        return E, Esq

    def get_spin_spin_corr(self, config, rs):
        s0sr = np.zeros(len(rs))
        pos0 = [0] * self.dim
        s0sr[0] = 1
        for i in range(1,len(rs)):
            posr = []
            for j in range(self.dim):
                posr.append(pos0[:j] + [(pos0[j]-rs[i])] + pos0[(j+1):])
                posr.append(pos0[:j] + [(pos0[j]+rs[i])] + pos0[(j+1):])
            s0sr[i] = (sum(config[tuple(pos0)] * config[tuple(p)]
                for p in posr) / (2 * self.dim))
        return s0sr

class IsingTempSeries:
    """ Simulates the Ising model on a square lattice of linear size
```

```
        L in dimension dim at Nts temperatures from Tmin to Tmax
        (in units of J/kB), using the Wolff algorithm with Neq
        equilibration iterations and Neq//2 measurement iterations
        for each temperature.
        Calculates magnetization, susceptibility, specific heat,
        and spin-spin correlation function at each temperature.
        Methods:
            - do_series(progbar=True, plot=True):
                performs the temperature series.
                Displays a tqdm progress bar if progbar==True;
                plots magnetization, susceptibility, and specific heat
                if plot==True.
            - save(filename): saves IsingTempSeries object to filename
    """
    def __init__(self, L, dim, Tmin, Tmax, Nts, Neq=10**4):
        self.L = L
        self. dim = dim
        self.Ts = np.linspace(Tmin, Tmax, Nts)
        self.Neq = Neq
        self.Nruns = Neq // 2
        self.mag = np.zeros(Nts)
        self.susc = np.zeros(Nts)
        self.C = np.zeros(Nts)
        self.binderQ = np.zeros(Nts)
        self.rs = np.arange(0, self.L // 2)
        self.s0sr = []

    def do_series(self, progbar=True, plot=True):
        if progbar:
            for i in tnrange(len(self.Ts)):
                ising = Ising(self.L, self.dim, self.Ts[i])
                ising.simulate(self.Neq, progbar=False)
                self.mag[i] = ising.mag_avg
                self.susc[i] = ising.susc
                self.C[i] = ising.C
                self.binderQ[i] = ising.binderQ
                self.s0sr.append(ising.s0sr)
        else:
            for i in range(len(self.Ts)):
                ising = Ising(self.L, self.dim, self.Ts[i])
                ising.simulate(self.Neq, progbar=False)
                self.mag[i] = ising.mag_avg
                self.susc[i] = ising.susc
                self.C[i] = ising.C
                self.binderQ[i] = ising.binderQ
                self.s0sr.append(ising.s0sr)
        if plot:
            fig, ax1 = plt.subplots()
            ax1.plot(self.Ts, self.mag, 'bo-')
            ax1.set_ylabel('Magnetization', color='b', fontsize=16)
            ax1.set_xlabel(r'Temperature [$J/k_B$]', fontsize=16)
            ax1.tick_params('y', colors='b')
            ax1.grid(b='off')

            ax2 = ax1.twinx()
            ax2.plot(self.Ts, self.susc, 'ro-')
            ax2.set_ylabel(r'Susceptibility [$1/k_B$]', color='r',
                fontsize=16)
            ax2.tick_params('y', colors='r')
            ax2.grid(b='off')
            fig.tight_layout()
            plt.show()

            plt.plot(self.Ts, self.C, 'bo-')
            plt.ylabel(r'Specific Heat [$J/k_B^2$]', fontsize=16)
            plt.xlabel(r'Temperature [$J/k_B$]', fontsize=16)
            plt.show()

            plt.plot(self.Ts, self.binderQ, 'bo-')
            plt.ylabel(r'$\langle{m^2}\rangle/\langle|m|\rangle^2$',
                fontsize=16)
            plt.xlabel(r'Temperature [$J/k_B$]', fontsize=16)
            plt.show()

            plt.plot(self.rs[:], self.s0sr[0][:], 'bo-')
            plt.xlabel(r'$r$', fontsize=16)
            plt.ylabel(r'$<s_0s_r>$', fontsize=16)
            plt.title('Spin-spin correlation at T = {}'
                .format(self.Ts[0]))
            plt.show()

    def save(self, filename):
        pickle.dump(self, open(filename, 'wb'))
```