`V Year Project`
Martensitic phase transition of bacteriophage T4's tail sheath

Mahesh Gandikota
Y1011025
V year
Integrated MSc
NISER, Bhubaneswar

December 4, 2014

Under the guidance of
Prof. Somendra Bhattacharjee
IOP, Bhubaneswar

Official guide for the project
Dr. Sumedha
NISER, Bhubaneswar

Contents

List of Figures

1 Introduction

T4 bacteriophage[1], one of the largest of the phages infects the bacteria - E.coli. The main parts of the T4 phage is an elongated icosahedron head which contains the DNA of the virus, a rigid tail which is composed of an inner tail tube surrounded by a tail sheath, baseplate, tail fibers and the neck which connects the head to the tail.

`Mechanism of infection`: The tail-fibers contact a bacterium, attachment of the tail fibers to the bacterial cell wall transforms the baseplate from the hexagonal to a star-shaped form. This in turn triggers a transformation in the tail sheath[2] where it rotates and contracts into a wider and shorter form. This process is simultaneous with an almost one full rotation[2] of the inner tail tube which bores it's way into the cell-wall and finally the DNA of the virus is injected through the tube into the bacterium[1].

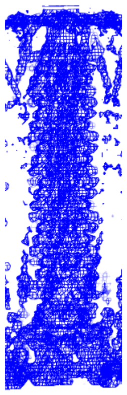


Figure 1: Electron density map of extended tail-sheath phase [11]

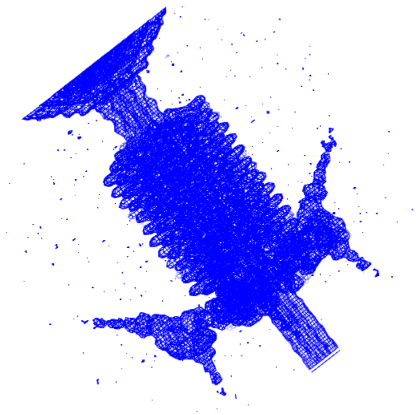


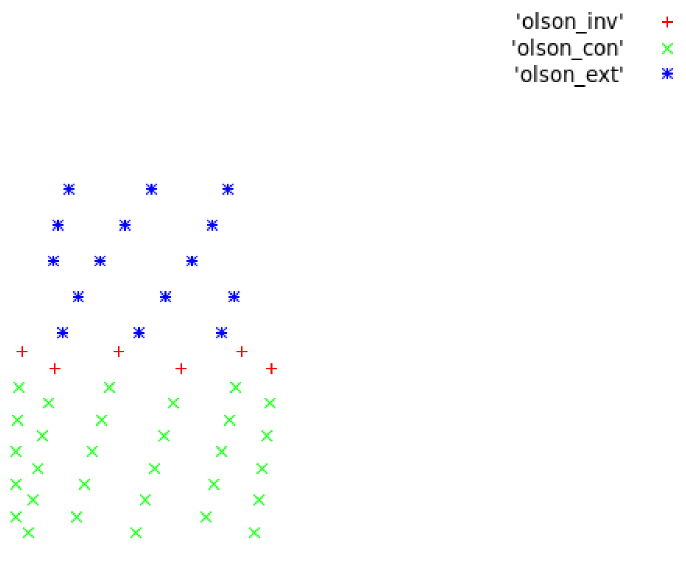
Figure 2: Electron density map of contracted tail-sheath phase [12]

Figures (1) and (2) were produced using CCP4.

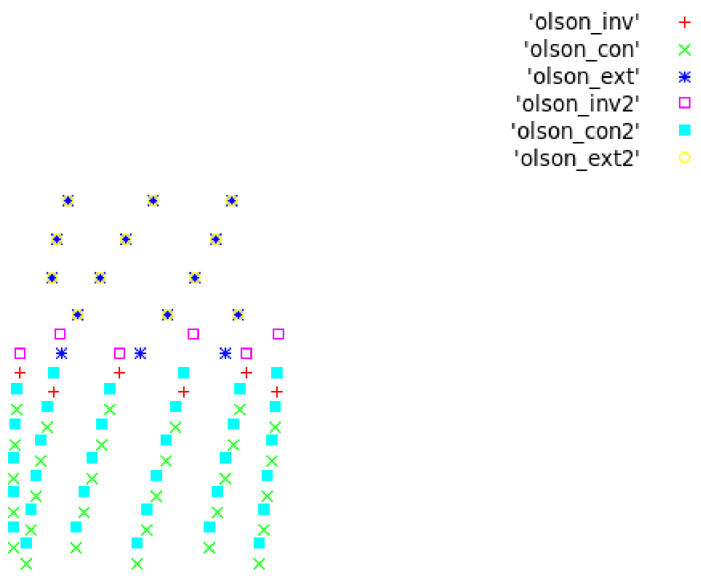
Olson and Hartman in their 1982 paper[2] have studied this transformation by considering the tail-sheath as a plane-lattice unwrapped from a cylindrical lattice. Each lattice vector corresponds to a gp-18 protein, 138 copies of which form the tail-sheath[1]. They concluded that the transition was a martensitic phase transition. We reproduced their work pertaining to T4 phage[9].

[1]Figures and text in this section are largely taken from [9].

[2]Inner tube rotates by 345.4^o[1].



(a) Interface. Red points: interface, green points: contracted phase, blue points: extended phase



(b) One step

Figure 3: Movement of interface.

Moody's experiments suggest that the interface moves from the baseplate to the head as the contracted phase of the tail-sheath grows in the same direction. Adding to Olson and Hartman's work, we tried to model the movement of the interface step-by-step[9]. One *step* of the interface upwards means one more row of proteins added and subtracted to the contracted and extended tail-sheath phase respectively.

Legend: inv - invariant line, con - contracted phase, ext - extended phase. A conclusion regarding the movement of the interface was reached: *The bottom row of the lattice points of the new interface is directly above the top row of the lattice points of the old interface.*

In this project, our objective is to study the movement of this interface further by using Monte Carlo methods and modeling the transition using Landau's theory of phase transitions. To get acquainted with Monte Carlo methods, the first half of this project was spent in applying MC methods to 2D Ising Model.

2 Brief Theory

Statistical mechanics is the microscopic framework which can explain the macroscopic properties of systems. The partition function is central to this framework and all the thermodynamic quantities can be derived from this function.

$$Z = \sum_i e^{-\beta E_i} \tag{1}$$

The summation is over all states i for a system kept in a heat bath at temperature T. This heat bath introduces fluctuations in the system. $\beta \equiv \frac{1}{kT}$ where k is the Boltzmann constant.

$$U \equiv \langle E \rangle = \frac{\sum_E E\,\Omega(E)\,e^{-\beta E}}{Z} \tag{2}$$

U is the average internal energy. The specific heat $C \equiv \frac{dU}{dT}$ can be derived using Eq. (2) and is found to be

$$C = k\beta^2 \left(\langle E^2 \rangle - \langle E \rangle^2\right) \tag{3}$$

It can be seen that specific heat is directly proportional to the variance of energy. The energy fluctuations, $\Delta E = \sqrt{\langle E^2 \rangle - \langle E \rangle^2}$ can now be written in the form,

$$\Delta E = T\sqrt{kC} \tag{4}$$

The magnitude of energy fluctuations was found entirely in terms of the thermodynamic quantities (Eq. 4). However, this result itself is impossible to derive under the ambit of thermodynamics because the derivation involved the microscopic details which is not accessed by thermodynamics at all.

To estimate the magnitude of ΔE, lets consider a liter of air contained in a box. The typical specific heat of such a system is $1JK^{-1}$ giving RMS fluctuations $\Delta E = 10^{-18}J$. The internal energy for the system is around 10^2J. Thus, the ratio $\Delta E/E \approx 10^{-20}$ shows that the fluctuation of internal energy is very small compared to the average internal energy[2]. It can also be observed that specific heat and average internal energy being extensive quantities increase linearly with volume V. Thus $\Delta E/E \sim 1/\sqrt{V}$. Thus bigger the system, smaller the fluctuations.

Ising Model

Ising model, a simple model for ferromagnetism. It is one of the simplest models to show phase transition and also one of the very few cases where an analytical solution for the thermodynamic quantities have been calculated.

The Ising model exhibits two phases: *Ferromagnetic phase* below the critical point T_c and the *paramagnetic phase* above it.

The Ising model on a square lattice $(L \times L)$ is a set of interacting spins. Each spin (s_i) is fixed to a lattice point (i) and can either point up or down $(s_i = \pm 1)$. The number of lattice sites in the system is $N \equiv L \times L$. The Hamiltonian for the system is,

$$H = -J \sum_{i \neq j} s_i s_j - B \sum_i s_i \tag{5}$$

The first summation is only over all the pairs of nearest neighbours. J is the coupling constant and took to be 1 in this work. This means, parallel spins lower the energy of the system. The second summation is over all the spins in the lattice. B is a uniform magnetic field over the lattice. B is taken to be 0 for the present purpose. Onsager solved the Ising model in 1944[1]. He found that the critical point of the system occurs at

$$T_c = \frac{2J}{k\ log(1+\sqrt{2})} = 2.269185.. \tag{6}$$

The number got is after putting $k = 1$. We keep $J = k = 1$ throughout this report.

The energy E of a state can be in the range $((-2JN, 2JN))$. When all the spins point in the same direction and one of the spins change, then $\Delta E = 8J$. After this event, the change in energy can occur in steps of $4J$ or $8J$.

The total magnetization of the system is M.

$$M = \sum_{i}^{N} s_i \tag{7}$$

The magnetization of a state can be in the range $(-N, N)$ in steps of 1.

The zero field magnetic susceptibility $\chi = \frac{\partial M}{\partial B}|_{B=0}$ is,

$$\chi = \beta\left(\langle M^2\rangle - \langle M\rangle^2\right) \tag{8}$$

The number of configurations possible for a lattice of size $L \times L$ is 2^N.

$$2^N = e^{log(2^N)} = e^{N\ log 2}$$

. Thus we see that the number of configurations increases exponentially with N.

An Ising model on a 10x10 lattice has $\approx 10^{30}$ microstates. If a computer assigned to calculate the average of a thermodynamic quantity takes $1ns$ to consider each microstate. The computation then would last for around thirty billion years![3]

In the present work, Monte Carlo methods are applied to Ising model on 2D square lattice to estimate the thermodynamic quantities.

3 Monte Carlo Methods

Monte Carlo (MC) method is any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties[4].

This method uses random numbers to calculate non-random quantities. As a simple example, the area of the circle can be estimated by using random numbers. Note that area is not a random but a fixed quantity. A computer generating two sets of uniformly distributed random numbers in the range $(0, 2r)$ gives randomly chosen co-ordinates in a square of side $2r$. The area of the circle is estimated by the ratio of the randomly chosen co-ordinates inside the circle to the total random co-ordinates generated.

The history of this method is older than it's name which was given by Nicolas Metropolis in 1949. This approach is taken even today. The method itself originated in the 19th century, earlier to the advent of computers and was known by the name "statistical sampling". Due to the innate randomness of this method, Metropolis named it after a relative who had a propensity to gamble. Monte Carlo in the province of Monaco, south of France is famous for gambling.

The earliest purpose of MC method was to estimate integrals of tough functions. Now, it is also used for estimating many quantities that occur in statistical mechanics.

The partition function (Eq. (1)) involves a summation over all the microstates of the system. Though straightforward, the summation is usually over a very large number of states. Monte Carlo methods provide a way to estimate the thermodynamical quantities by averaging over only selected 'important' states instead of all accessible states of the system. This is much faster than the exact calculation using the partition function.

4 Metropolis Algorithm

As it is too time consuming to sample all states of the system, we would like to sample only the 'important states': the states where the system spends the most time. This reduction of the total accessible states to the important states is called *importance sampling*. We know that at thermal equilibrium, the probability of the system being in a state is given by the Boltzmann distribution. For the purpose of importance sampling, we cannot sample all states and keep only those states which have high probability according to the Boltzmann distribution. This would be as worse as the problem we are trying to solve (avoiding to sample all states).

We should also make sure that there are fluctuations in the simulated system even after equilibration. However, these fluctuations should be such that the Boltzmann distribution is not disturbed.

The above two problems may be solved using the approach of *Markov process*. This process works as the generating engine to get the set of important states we need. Markov process is a stochastic process which starts from a chosen state and evolves to a different state every time-step. The probability of moving from state μ to ν is given by $P(\mu \to \nu)$ which would form a $n \times n$ matrix where n is the total number of states of the system. $P(\mu \to \nu)$ is called the *transition probability* and the matrix it forms is called *transition probability matrix*. The matrix uniquely defines the Markov process. Markov process have two defining properties:

1. The transition from current state to another should depend only on the current state and not it's history.
2. The transition probability matrix is independent of time[3].

Note that $\sum_\nu P(\mu \to \nu) = 1$ because the process must evolve to *some* state from μ. The chain of states (starting from the initial chosen state) as we evolve the initial state over time is called the *Markov chain*. The objective is to have the last elements of this chain to take states with Boltzmann distribution. Note that we are evolving the system over time and not considering the ensemble. We calculate average thermodynamic quantities by averaging over time which equals the ensemble average by the virtue of the *ergodic theorem*.

For getting the end-product states of Markov chain to occur with Boltzmann distribution, we have to impose two additional conditions.

1. Ergodicity

   The *condition of ergodicity* means that the Markov chain should be allowed to reach any state (ν) from any chosen state (μ). $P(\mu \to \nu)$ can be zero but there should exist at-least one path through which the Markov process can evolve to reach ν. The non-compliance to this condition would not give us the desired Boltzmann distribution which assigns non-zero probabilities to all states.

2. Detailed Balance
   For obtaining a steady state, the following condition should be satisfied,

$$\sum_\nu p_\mu P(\mu \to \nu) = \sum_\nu p_\nu P(\nu \to \mu)$$

   Though this condition gives a steady state, it won't ensure the distribution to be at equilibrium because the above condition allows multiple stationary distributions[4]. This is avoided by choosing a stricter condition called the *condition of detailed balance*:

$$p_\mu P(\mu \to \nu) = p_\nu P(\nu \to \mu) \tag{9}$$

[3]Strictly, these processes are called a *homogeneous* Markov process.

[4]Distribution of the states in the limit $t \to \infty$

A theorem of Markov processes states that an irreducible[5] Markov process with finite state space always achieves a unique stationary distribution . We know that our Markov process is irreducible (by condition of ergodicity) and that it has a finite state space (we are not considering thermodynamic systems), we are thus ensured a stationary distribution.

Now, we have to choose $P(\mu \to \nu)$ and $P(\nu \to \mu)$ and define a Markov process such that

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{p_\nu}{p_\mu} = e^{-\beta(E_\nu - E_\mu)} \tag{10}$$

We have now reduced the problem of importance sampling to that of finding a Markov chain such that it's transition probabilities obey (Eq.(10)).

Instead of choosing transition probabilities in trial and error to satisfy Eq.(10), a neater procedure can be used. The transition probability is broken into two parts:

$$P(\mu \to \nu) = g(\mu \to \nu)\ A(\mu \to \nu) \tag{11}$$

$g(\mu \to \nu)$ is called the *selection probability* for choosing a ν state being in state μ. $A(\mu \to \nu)$ is called the *acceptance probability* for the process to accept evolving to state ν. As only the ratio of transition probabilities is fixed and not the transition, selection, acceptance probabilities, we are free to choose them such that the algorithm can attain equilibration faster.

For the present work, *single-spin dynamics* has been chosen. i.e. only one spin is flipped at every time step.

The choice of the acceptance ratio characterizes the *Metropolis algorithm*[2]. The algorithm is:

1. $g(\mu \to \nu) = \frac{1}{N}$ where ν is a state accessible to μ by flipping a single spin in that state. The selection probability is set to zero otherwise. Thus this algorithm considers going to other accessible states with uniform probability.

2. $$A(\mu \to \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)} & \text{if } E_\nu - E_\mu > 0 \\ 1 & \text{otherwise} \end{cases}$$

This choice for acceptance probabilities speeds up the dynamics by promoting transitions and equilibration can be achieved sooner.

[5]If every state is connected to the other state with some non-zero probability by the Markov Process.

5 MC simulations of 2D Ising model on a square lattice

5.1 Thermodynamic Quantities

- The initial configuration of spins is chosen to be completely random. Thus, the lattice is in a $T(\infty)$ state[6] initially and equilibrates to the finite T as the Monte Carlo simulation runs.
- Number of MC loops/spin $= 10^5$
- Number of MC loops/spin allowed for equilibration $= 10^3$
- Lattice sizes: $L = 8, 16, 32$
- Periodic boundary conditions imposed in both dimensions of the lattice to avoid boundary effects.

All the plots in this report are thermodynamic quantities calculated/spin. For example, U/N is plotted in Fig.(4) instead of U. This allows us to compare how internal energy differs when we change the lattice size.

Internal energy

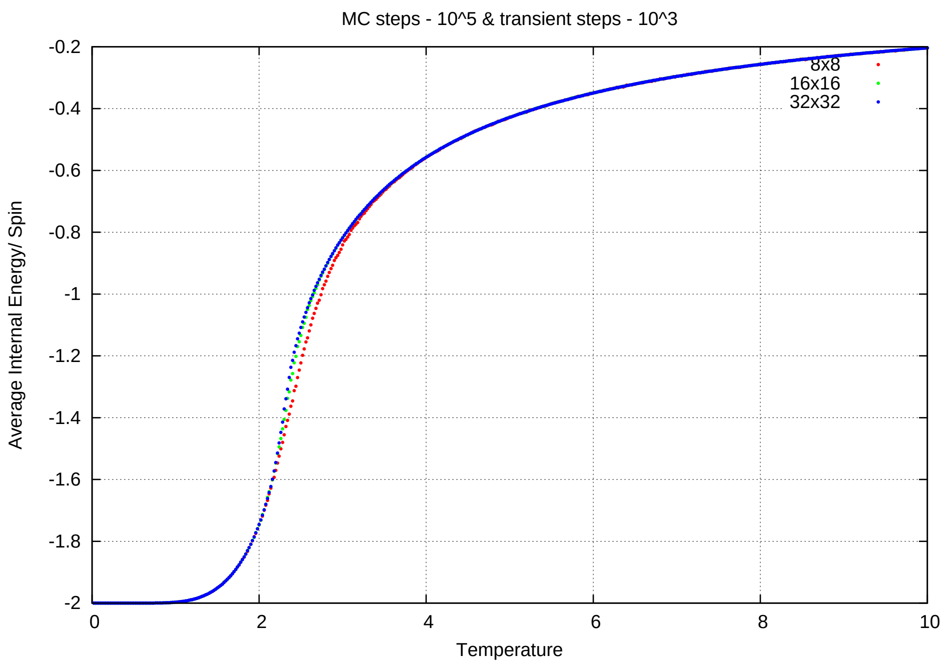


Figure 4: Average Internal energy

[6]Each spin can be seen as being decided by a coin toss.

Specific heat

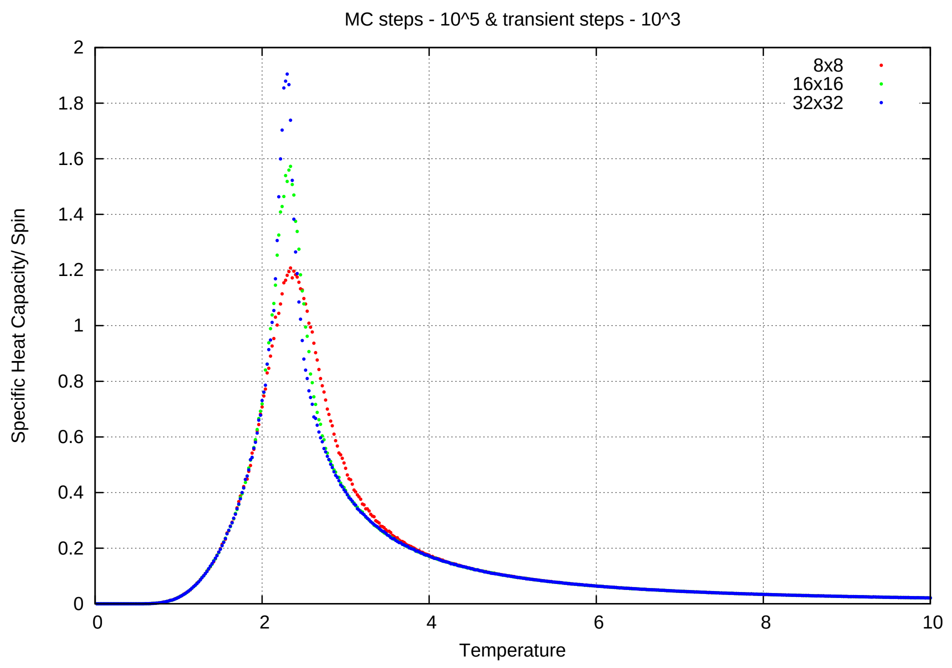


Figure 5: Specific heat

In Fig.(5), the peaks are seen to shift towards left with increase in lattice size. The peak is moving towards the thermodynamic limit ($L = \infty$) of $T_c = 2.269$. We can also observe that the peaks become sharper as L increases indicating that in the thermodynamic limit, there will be a singularity in the specific heat.

Absolute Magnetization

For 2D Ising model on square lattice with interaction coefficients J and J' in x and y axis, Onsager's[1] result in the thermodynamic limit is:

$$m = [1 - \{sinh(2\beta J)\ sinh(2\beta J')\}^{-2}]^{\frac{1}{8}} \tag{12}$$

where $m \equiv M/N$ It can be observed from Fig.(6) that our algorithm is overestimating the magnetization for $T > T_c$. Theoretically $M = 0$ for $T > T_c$. The reason for the over-estimation is because we are plotting $|m|$ and not m. As we take bigger lattices, even $|m|$ will die to zero. However, for any finite lattice, as $|m|$ is an average over non-negative numbers, we will never be ending up with zero as we want it to be.

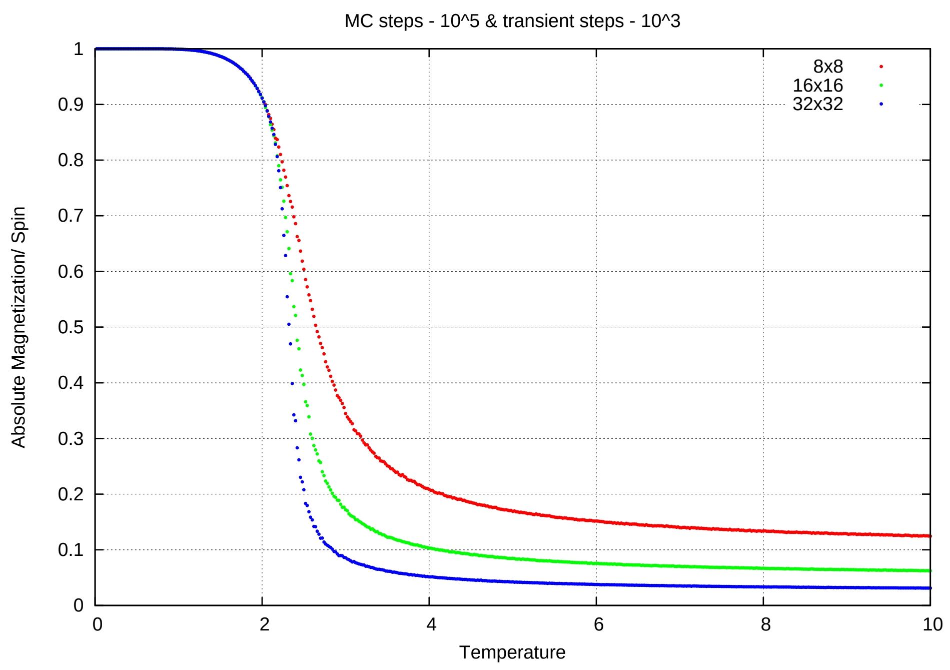


Figure 6: Absolute magnetization

Zero field susceptibility

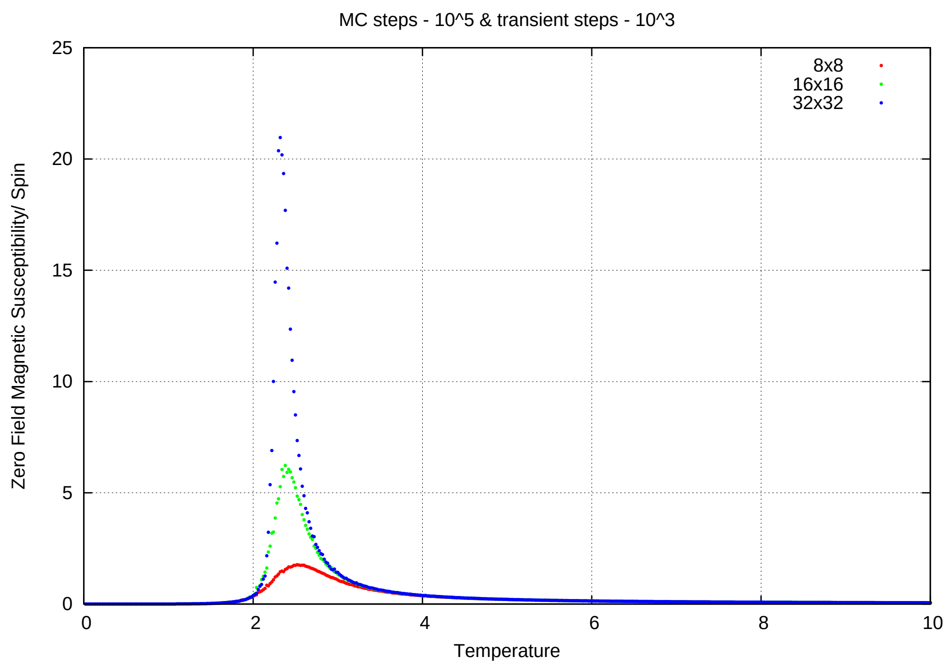
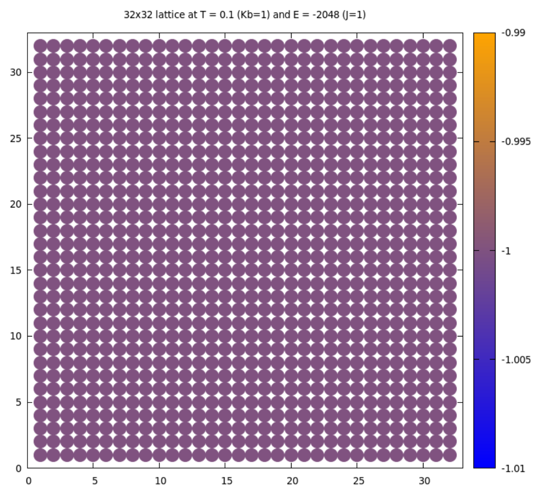


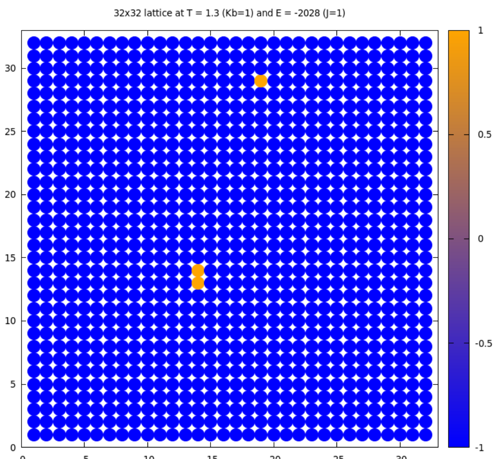
Figure 7: Susceptibility

5.2 Typical states at different temperatures

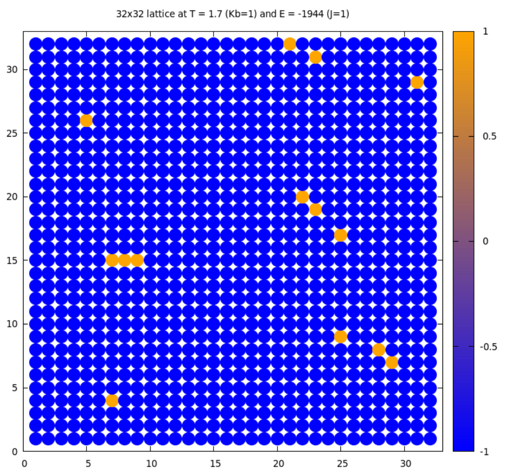
Pictures of typical states at different temperatures are shown in this section. Each state is chosen such that it's energy is near to the average internal energy at that temperature.



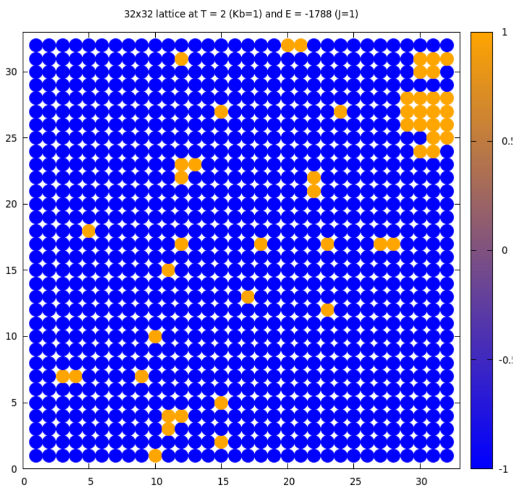
(a) T = 0.1



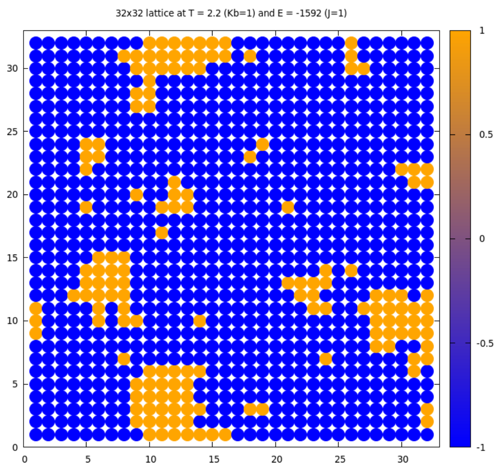
(b) T = 1.3



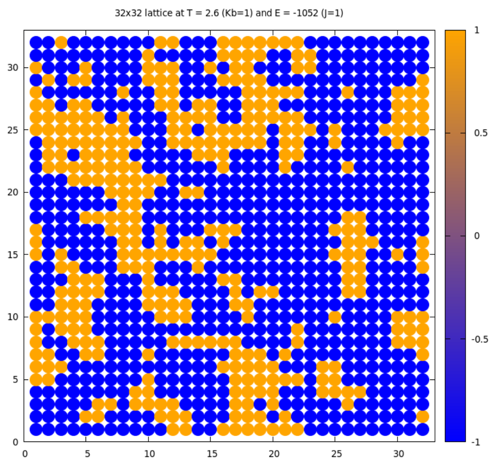
(a) T = 1.7



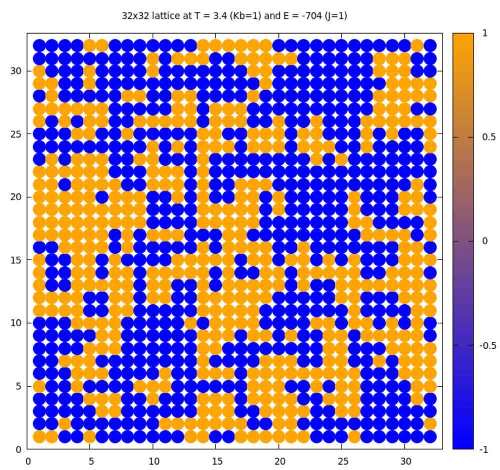
(b) T= 2



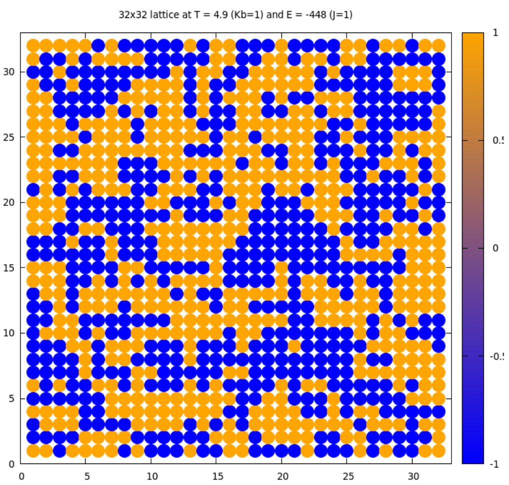
(a) T = 2.2



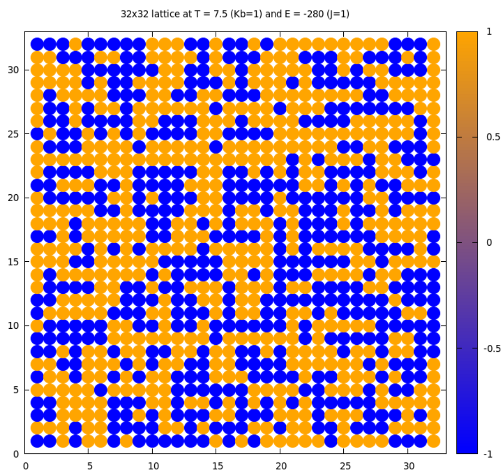
(b) T = 2.6



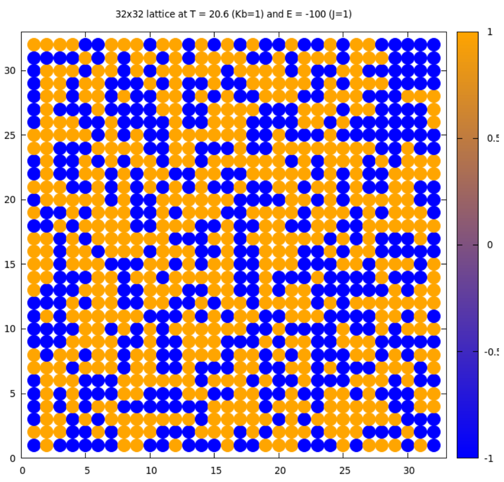
(a) T = 3.4



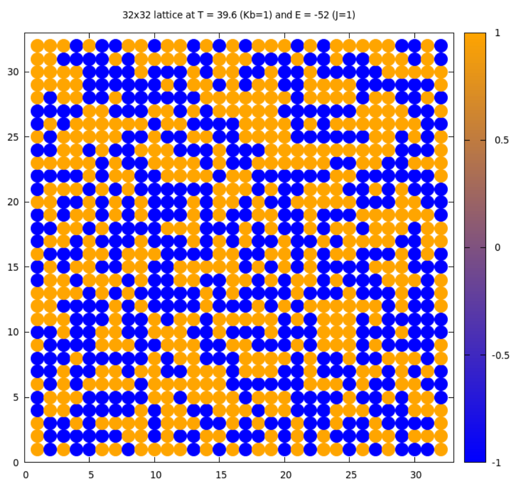
(b) T = 4.9



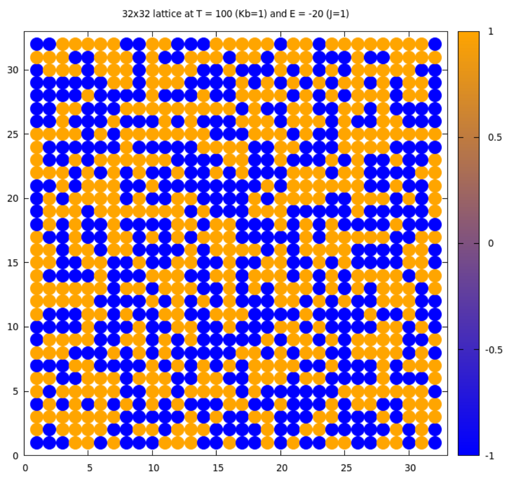
(a) T = 7.5



(b) T = 20.6



(a) T = 39.6



(b) T = 100

5.3 Histogram method

The MC simulation code (sec.(8.1)) calculates average thermodynamic quantities at every temperature. Histogram method offers a technique where the probability distribution of the system at temperature T_1 can be used to generate the probability distributions at temperature T_2. Once the probability distribution at T_2 is known, all the thermodynamic quantities like $U, \langle |M| \rangle, c, \chi$ can be calculated using it. This removes the necessity of running the MC simulation for each temperature.

- Finding $U(T_2)$

  If one is interested only in finding average internal energy at other temperatures, then the technique is simpler. The probability distribution of energies at T_1 and T_2 are $P_1(E)$ and $P_2(E)$. The former distribution is known and the latter is to be found in terms of it.

$$P_1(E) = \frac{\Omega(E)\ e^{-\beta_1 E}}{\sum_{E'} e^{-\beta_1 E'}} \tag{13}$$

$$P_2(E) = \frac{\Omega(E)\ e^{-\beta_2 E}}{\sum_{E'} e^{-\beta_2 E'}} \tag{14}$$

  $P_2(E)$ can be rewritten as,

$$P_2(E) = \frac{\Omega(E)\ e^{-(\beta_2-\beta_1)E}\ e^{-\beta_1 E}}{\sum_{E'} \Omega(E')\ e^{-(\beta_2-\beta_1)E'}\ e^{-\beta_1 E'}}$$

  Dividing numerator and denominator by the partition function at T_1, $Z_1 = \sum_{E'} \Omega(E') e^{-\beta_1 E'}$,

$$P_2(E) = \frac{e^{-(\beta_2-\beta_1)E}\ (\Omega(E)\ e^{-\beta_1 E}/Z_1)}{\sum_{E'} \Omega(E')\ e^{-(\beta_2-\beta_1)E'}\ (\Omega(E') e^{-\beta_1 E'}/Z_1)}$$

  Using Eq.(13),

$$P_2(E) = \frac{e^{-(\beta_2-\beta_1)E}\ P_1(E)}{\sum_{E'} e^{-(\beta_2-\beta_1)E'}\ P_1(E')} \tag{15}$$

  Now, average internal energy $U(T_2)$ specific heat $c(T_2)$ can be found.

- Finding $\langle E(T_2) \rangle$ and $\langle |M(T_2)| \rangle$

  For calculating $\langle |M(T_2)| \rangle$, we need to generate the distribution $P_1(M, E)$ at T_1.

$$P_1(M, E) = \frac{\Omega_M(E)\ e^{-\beta_1 E}}{\sum_{E'} \Omega(E')\ e^{-\beta_1 E'}} \tag{16}$$

  Here $\Omega_M(E)$ is the number of degenerate states with *total* magnetization M and energy E. Note that we are not using a probability distribution in terms of the absolute magnetization.

$$P_2(M, E) = \frac{\Omega_M(E)\ e^{-\beta_2 E}}{\sum_{E'} \Omega(E')\ e^{-\beta_2 E'}} \tag{17}$$

  This can be rewritten as,

$$P_2(M, E) = \frac{\Omega_M(E)\ e^{-(\beta_2-\beta_1)E}\ e^{-\beta_1 E}}{\sum_{E'} \Omega(E')\ e^{-(\beta_2-\beta_1)E'}\ e^{-\beta_1 E'}}$$

  Dividing by Z_1,

$$P_2(M, E) = \frac{e^{-(\beta_2-\beta_1)E}\ (\Omega_M(E)\ e^{-\beta_1 E}/Z_1)}{\sum_{E'} e^{-(\beta_2-\beta_1)E'}\ (\Omega(E')\ e^{-\beta_1 E'}/Z_1)}$$

Thus,

$$P_2(M,E) = \frac{e^{-(\beta_2-\beta_1)E}\ P_1(M,E)}{\sum_{E'} e^{-(\beta_2-\beta_1)E'}\ P_1(E')} \tag{18}$$

Once, we have the joint probability distribution $P_2(M,E)$, we can find the probability distribution for magnetization $P_2(M)$ by

$$P_2(M) = \sum_{E=-2NJ}^{2NJ} P_2(M,E) \tag{19}$$

Now, we can find $\langle |M(T_2)| \rangle$ and $\chi(T_2)$.
Similarly, the probability distribution for energy is found by

$$P_2(E) = \sum_{M=-N}^{N} P_2(M,E) \tag{20}$$

Now, we can find U and χ.

The advantages of histogram method is that by generating one good distribution, we can find thermodynamic quantities at other temperatures too and thus saving computation time. However, the technique does not work well for estimating distributions at far off temperatures. The reason being that, if a particular energy E is not sampled at T_1 when generating the initial distribution, then $P_1(E) = 0$. This implies that $P_2(E) = 0$ too (from Eq.(15)), even though if at temperature T_2, energy E has non-negligible probability of occurrence.

A good choice for generating the initial probability distribution is at $T_1 = T_c$. This is because a wide range of energies[7] is sampled at the critical temperature and we can avoid $P_1(E)$ being zero.

$P_1(2.4)$ was generated as the initial distribution and the histogram method was used to generate probability distributions at $T = 2.8,\ 5$. Fig.(8) compares the distribution thus generated with the distributions got using MC simulation at same temperatures.

[7]This follows from the fact that c diverges at T_c which means that the variance of E is very large ((Eq.(3)).

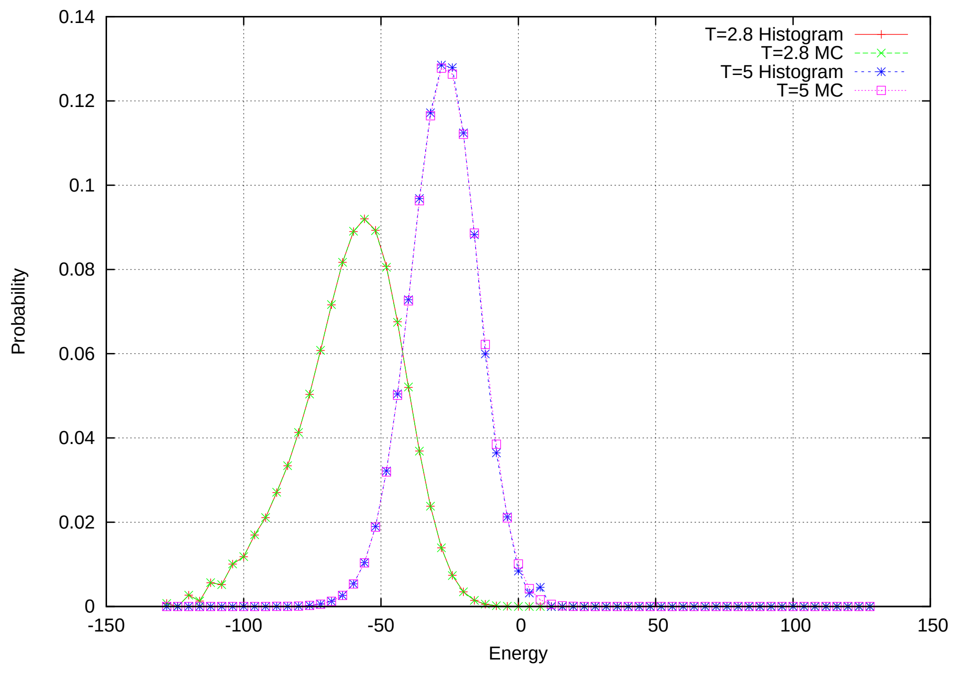


Figure 8: Histogram method for 8x8 lattice

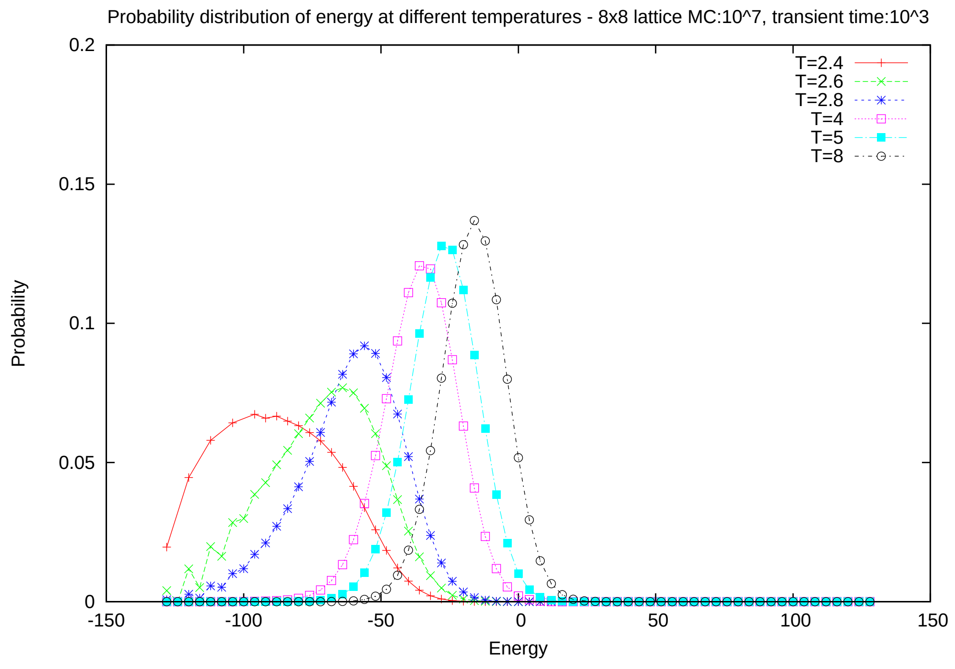


Figure 9: Evolution of energy probability distribution with temperature

Fig.(9) was generated using the histogram method too. It shows how the shape of the energy probability distribution is changing with temperature.

5.4 Binder Cumulant

As we see in the plots of sec.(5.1), the peaks of the 'general susceptibilities' occur at different temperatures. The T_c of the thermodynamic system is the limit of these temperatures as $L \to \infty$ would give T_c. Binder cumulant offers a method in which the T_c can be calculated by using the MC data of few different sized lattices. Binder cumulant is,

$$U_L = 1 - \frac{\langle M^4 \rangle}{3 \ \langle M^2 \rangle^2} \tag{21}$$

For $T < T_c$, $\lim_{L\to\infty} U_L = \frac{2}{3}$. For $T > T_c$, U_L goes to zero. The point of intersection of U_L for different L gives an estimate for T_c.

Using the MC simulations for $L = 4, 8, 16, 32, 64$, T_c could be estimated as 2.26. The precise value of T_c is 2.269...

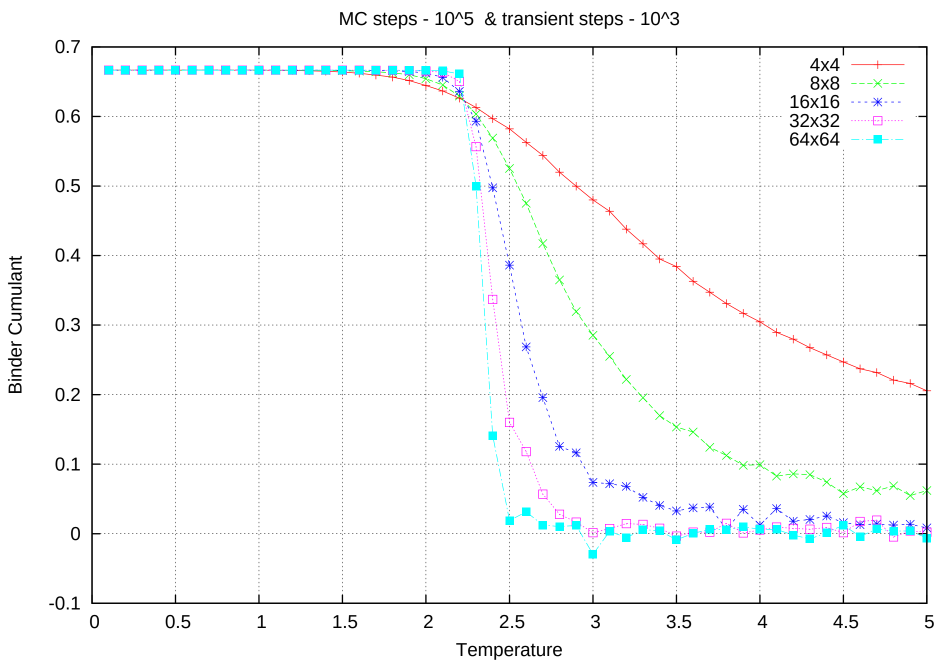


Figure 10: Binder Cumulant

Fig.(10) has data points at intervals of $dT = 0.1$. To get T_c, data points were produced at intervals of $dT = 0.02$ in the neighborhood of the intersection found in Fig.(10). T_c was determined using this denser data. The corresponding figure is not shown in this report.

T_c can also be determined by using the method of finite-size scaling, ref: sec(6.2)

5.5 Correlation

Correlation is a purely statistical mechanics quantity and does not occur in thermodynamics. Simply because correlations here are between the microscopic constituents of the system and nowhere in building the theory of thermodynamics, the microscopic constituents is considered. It follows that this theory cannot explain the microscopic phenomenon but only predict the relations between thermodynamic quantities irrespective of their microscopic structure. Just like evaluating every other statistical mechanics quantity, we can use Monte Carlo methods for getting estimates of correlation.
Here, correlation was calculated as,

$$G(i) = \langle s_0 \, s_i \rangle - \langle s_o \rangle \langle s_i \rangle \tag{22}$$

- Number of MC loops/spin $= 10^7$
- Number of MC loops/spin for equilibration $= 10^2$

The exact form of the correlation function is,

$$G(x) = x^{-\eta} \, e^{-\frac{x}{\xi}} \tag{23}$$

Using the value of $\eta = \frac{1}{4}$ and taking the log of this equation and rewriting,

$$\log\left[G(x)\, x^{\frac{1}{4}}\right] = -\frac{x}{\xi} \tag{24}$$

Thus by finding the correlation and plotting it as in eqn. (24), we get the correlation length for that particular temperatures.

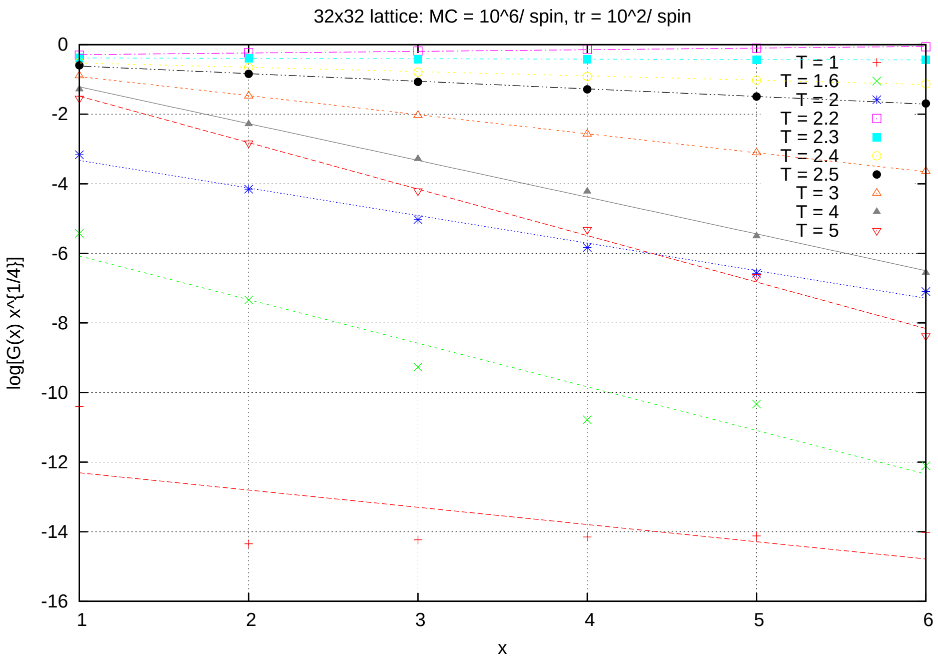


Figure 11: Correlation for 32x32 lattice

We see from Fig. (11), that the slope of the lines as $T \to T_c$ is tending to zero implying that ξ is becoming large[8]

[8] ξ cannot become ∞ because we are working on a finite lattice. Thus maximum of ξ is limited to L.

5.6 Entropy

We know the thermodynamic relation

$$dQ = TdS \tag{25}$$

As $dQ = CdT$, thus

$$CdT = TdS$$

From this relation, we can find the change in entropy in terms of temperature and specific heat.

$$S(T) = \int_0^T \frac{C}{T} dT$$

Dividing by N to get entropy/spin,

$$s(T) = \int_0^T \frac{c}{T} dT \tag{26}$$

We find $s(T)$ by integrating the measured specific heat values got by MC simulation. Trapezoidal integration was used.

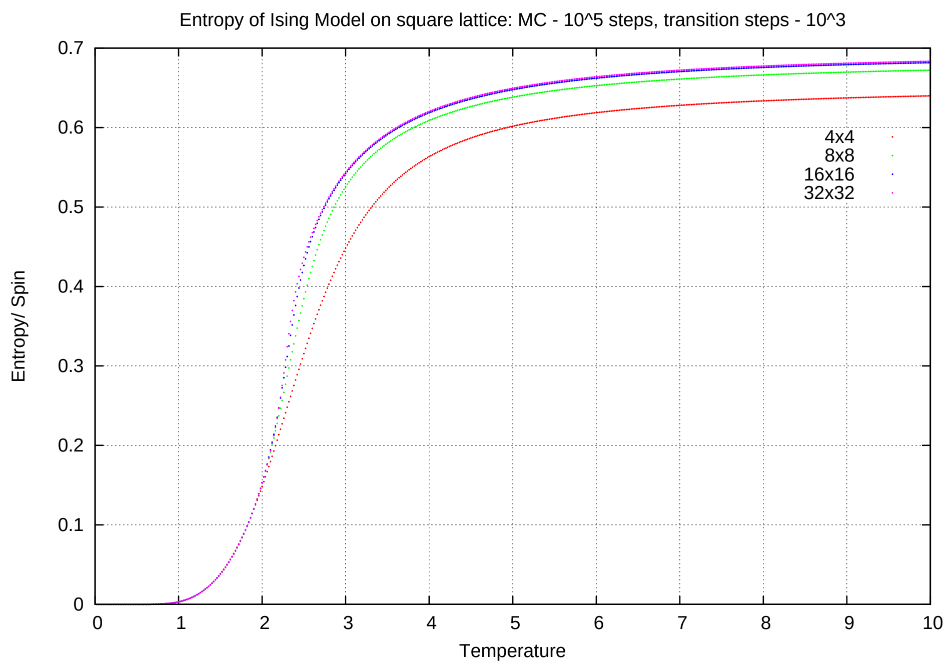


Figure 12: Entropy

We see from Fig.(12) that as expected entropy increases when the ferromagnetic (ordered) material goes to a paramagnetic material (disordered). The saturation of the tails (high T) indicate that the disorder of the system has (almost) attained it's maximum and can no more be increased by an increase in temperature.

5.7 Error

MC simulation is a stochastic method and there will be statistical error associated with it. There are many methods to estimate error. Errors for the data used in sec.(5.1) is reported in this section.

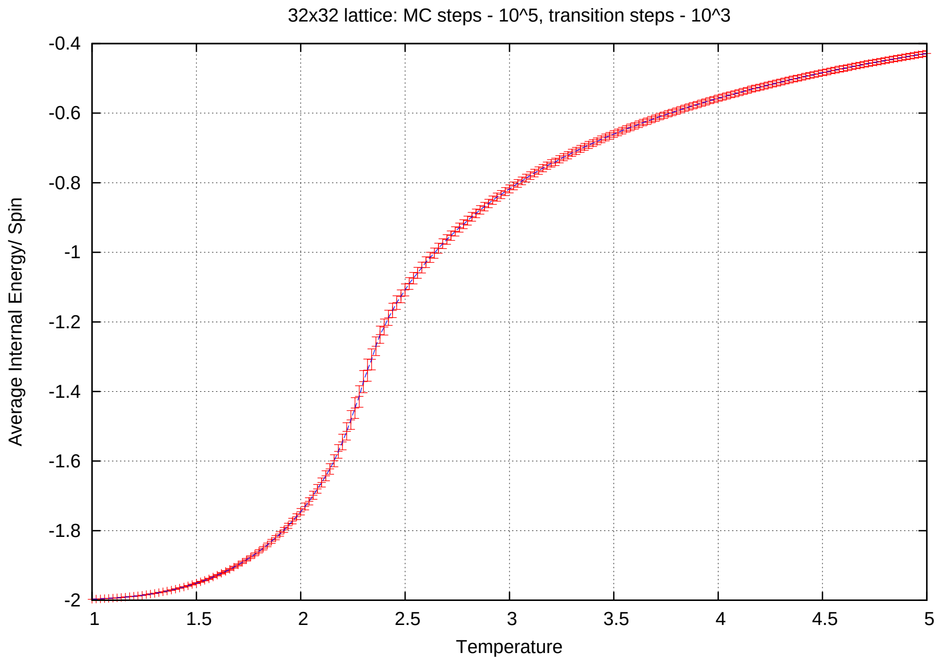


Figure 13: Error in internal energy

The error bars in Fig.(13) were calculated by using the variance of energy. In fact, no separate simulation is needed to calculate this error. One can just use the specific heat measurements which are nothing but variance in energy multiplied with a constant factor. This procedure was followed here.

Error bars can be seen to become bigger as T_c is approached in Fig.(13).

Another method of estimating error is the *bootstrap method.* This is a re-sampling method. For estimating the error bars in specific heat, the procedure is:

1. Select a set of n data points from the list of n MC simulation data points of energies at a particular temperature.
2. Calculate specific heat using the set of n data points.
3. Store c and c^2
4. Go to step 1 and repeat.
5. $\sigma = \sqrt{\langle c^2 \rangle - \langle c \rangle^2}$ gives the error bar for the specific heat at that temperature.

We apply this method for estimating error bars for specific heat and magnetic susceptibility.

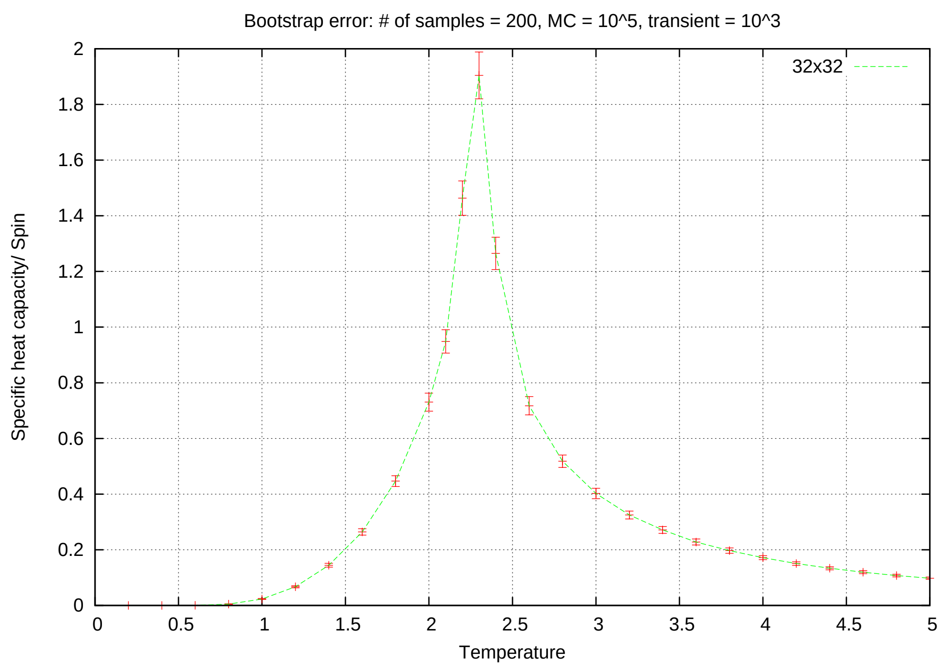


Figure 14: Boot strap - specific heat. Error multiplied by 10 for visibility

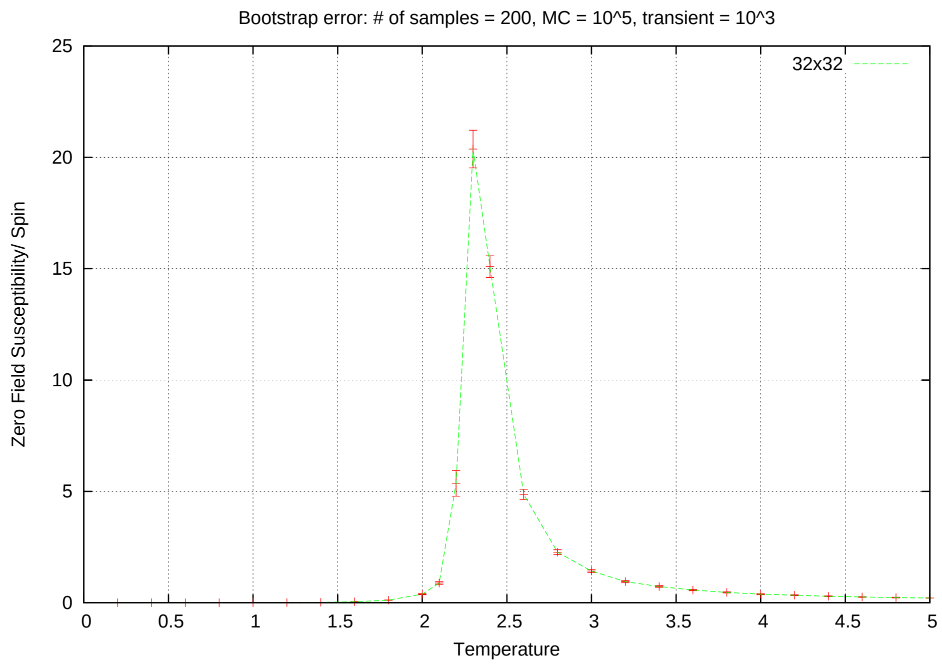


Figure 15: Boot strap - susceptibility. Error multiplied by 10 for visibility

Again in Fig.(14) and (15), it can be seen that error bars increase as T_c is approached.

6 Critical exponents

Till now, the concern was to simulate finite systems and get results pertaining to finite systems. This data can prove it's utility by predicting the behaviour of the thermodynamic system ($\infty \times \infty$ lattice) too. This section pertains to this aspect.

A true phase transition cannot occur on a finite system. We always get smoothed out curves instead of singularities. However, finite-size simulations show precursor signs of the phase transition as we saw in Sec.(5.1) where the *generalized susceptibilities* like the heat capacity and magnetic susceptibility have peaks[9]. The MC simulation data got for finite size lattices when combined with finite-size scaling, the critical exponents of the true phase transition occurring in the thermodynamical limit of the finite system can be found.

The singular behaviour of thermodynamic quantities in the neighbourhood of the critical point can be characterized by the *critical exponents*.

A dimensionless quantity named the *reduced temperature* is defined which acts as a measure of the distance from the critical temperature T_c.

$$t = \frac{T - T_c}{T_c} \tag{27}$$

Thermodynamic quantities follow these power laws:

$$\xi \sim |t|^{-\nu} \tag{28}$$

$$\chi \sim |t|^{-\gamma} \tag{29}$$

$$c \sim |t|^{-\alpha} \tag{30}$$

$$\begin{aligned} m &\sim |t|^{\beta} \qquad \text{for } T < T_c \\ &= 0 \qquad\quad \text{for } T > T_c \end{aligned} \tag{31}$$

The Rushbrooke's scaling law is

$$\alpha + 2\beta + \gamma = 2 \tag{32}$$

6.1 Direct measurement of critical exponents

These exponents can be directly extracted from the Monte Carlo data. For example, the slope of the log-log graph of m and t would give the critical exponent β. However, there are several serious problems in regard to this straight-forward approach:

1. Equation (31) holds true only near T_c. How to estimate till what t the MC data can be used to fit a straight line? Every choice gives a different fit and slope (critical exponent) value changes. Error estimation becomes difficult.

2. There are models such as the random-field Ising model have two different values of critical exponents depending on how far you are from T_c. Such *cross over* effects are difficult to account for if we do not know already where the cross over is happening.

[9]The free energy curve becomes flat at the critical point and since there is no first order restoring force, it allows for bigger fluctuations in the system [10]. As the generalized susceptibilities are nothing but variances, they diverge.

3. This method assumes the knowledge of T_c which can be precisely determined only for very few cases (like the 2D Ising model). For all other cases, the exact value of T_c is not known to start with and has to be found out.

   This method was applied to find β/ν and γ/ν. The lattice sizes took were $L = 2, 4, 8, 16, 32$. The results were: $\beta/\nu = 0.17$ against the expected value of 0.125 and $\gamma/\nu = 1.84$ against the expected value of 1.75. The log-log plots are put:

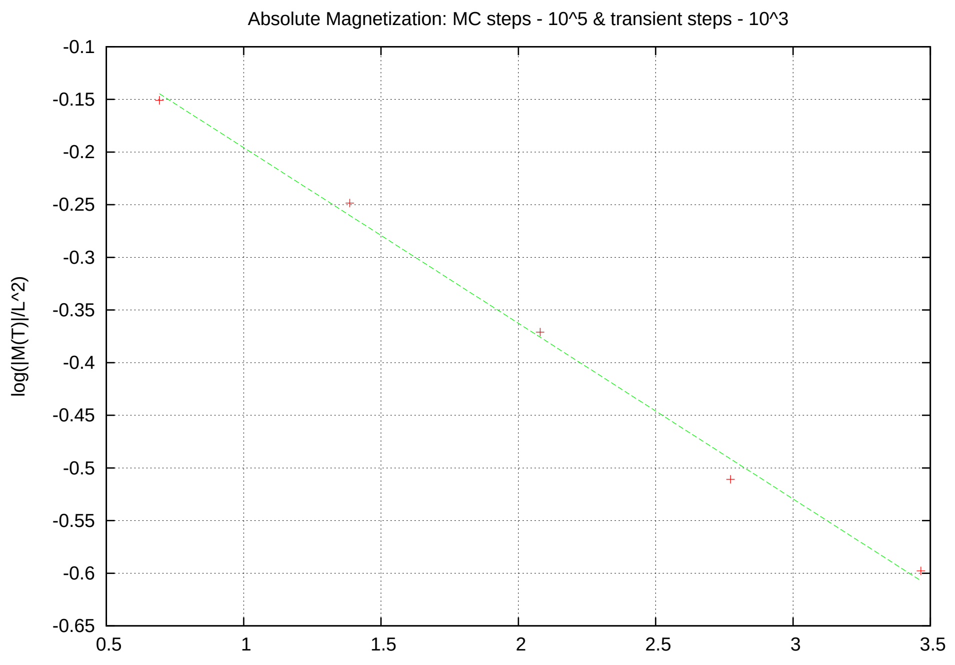


Figure 16: Calculation of β/ν by direct measurement

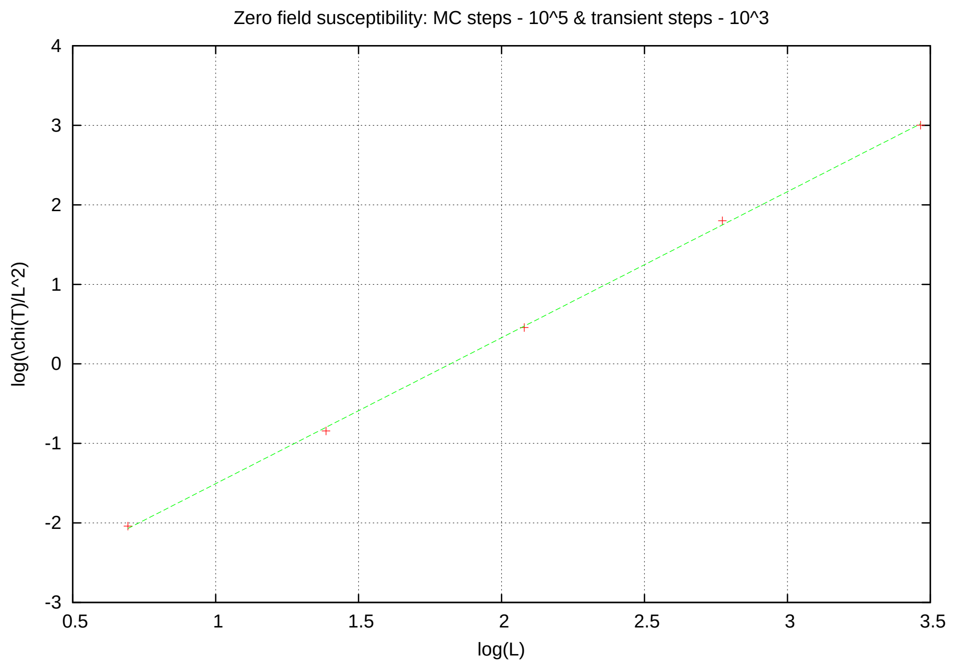


Figure 17: Calculation of γ/ν by direct measurement

6.2 Finite size scaling

Finite sized scaling is the most popular method for finding these exponents from the data generated using Monte Carlo techniques. It circumvents all the problems posed by the direct measurement method. This method is illustrated for the case of magnetization.

From equations (31) and (29), we see that

$$m \sim \xi^{-\frac{\beta}{\nu}} \tag{33}$$

This equation shows the behaviour of m near T_c for thermodynamic system ($\infty \times \infty$ lattice). For finite lattices, we can expect thermodynamic behaviour as above when $\xi << L$. When $\xi > L$, the[10] effect of the finiteness of the lattice is felt. Unlike for the infinite lattice which remains in $m = 0$ state when[11] $T \to T_c^+$, the absolute magnetization is non-zero for $T > T_c$ in a finite lattice. This paragraph can be translated into mathematics by the following equation.

$$m = \xi^{-\frac{\beta}{\nu}}\, m_o\left(\frac{L}{\xi}\right) \tag{34}$$

where $m_o(x)$ is a function of the dimensionless variable $x = \frac{L}{\xi}$. $m_o(x)$ should have the following properties:

$$m \sim \begin{cases} \left(\frac{L}{\xi}\right)^{-\frac{\beta}{\nu}} & \text{for } x >> 1 \\ \text{constant} & \text{for } x \to 0 \end{cases}$$

These equations have ξ which is the correlation length of the thermodynamic system of which we would not have any knowledge and has to be replaced. The following rearrangements are made towards this.

$$\begin{aligned} m &= \frac{1}{\xi^{\frac{\beta}{\nu}}}\, m_o\left(\frac{L}{\xi}\right) \\ &= L^{-\frac{\beta}{\nu}} \left(\frac{L}{\xi}\right)^{\frac{\beta}{\nu}} m_o\left(\frac{L}{\xi}\right) \end{aligned}$$

Defining $x \equiv \left(\frac{L}{\xi}\right)^{\frac{1}{\nu}}$

$$m = L^{-\frac{\beta}{\nu}}\, x_o^{\beta}\, m_o(x^{\nu})$$

Defining a new function $\tilde{m} = x^{\beta}\, m_o(x^{\nu})$,

$$m = L^{-\frac{\beta}{\nu}}\, \tilde{m}(x)$$

We have pushed all ξ dependence of m neatly into the argument of the function $\tilde{m}$ but we have still not got rid of it. For that notice,

$$\begin{aligned} x &= \left(\frac{L}{\xi}\right)^{\frac{1}{\nu}} \\ &= L^{\frac{1}{\nu}}\, |t| \end{aligned}$$

[10]Note: All the while ξ represents the correlation length of the thermodynamic system at the same temperature as the finite lattice. ξ is not the correlation length for the finite lattice.

[11]Approaching T_c from the right.

where we used Eq. (29). Substituting in the above equation, we get the basic equation for the finite size behaviour of absolute magnetization[12]:

$$m = L^{-\frac{\beta}{\nu}}\ \tilde{m}(L^{\frac{1}{\nu}}\ t) \tag{35}$$

Similar derivations would give,

$$\chi = L^{\frac{\gamma}{\nu}}\ \tilde{\chi}(L^{\frac{1}{\nu}}\ t) \tag{36}$$

$$c = L^{\frac{\alpha}{\nu}}\ \tilde{c}(L^{\frac{1}{\nu}}\ t) \tag{37}$$

The function $\tilde{m}$ is yet unknown. Rewriting Eq. (35),

$$\tilde{m}(L^{\frac{1}{\nu}}\ t) = L^{\frac{\beta}{\nu}}\ m_L(t)$$

Here $m_L(t)$ denotes the MC data got for a particular lattice. Interpolation of the data gives the function $\tilde{m}$ and this *scaling function* is the same for all lattices. So, for every L, we find $\tilde{m}$ and have to tweak the exponents such that all of the $\tilde{m}$ s match. Then, the data for all lattices fall on the same curve. The data collapse would fetch us not only the values of the critical exponents but also the critical temperature T_c.

Thus, it can be observed that the MC simulation data for finite lattices can be viewed in two perspectives.

1. As a study of properties of finite systems.

2. As a study of critical phenomenon of thermodynamic systems.

6.3 A measure of data-collapse for scaling

The finite-size scaling method removes the serious problems which plagues the direct measurement of critical exponents. In the former method, the critical exponents should be chosen such that all the data points for all lattice sizes fall on the same curve. Instead of finding these exponents by trial and error, a method of searching has been devised by Bhattacharjee, Seno [5]. A measure is proposed to quantify the 'goodness' of the data collapse. The critical exponents are found by a minimization procedure. This method removes the subjectiveness of searching for the exponents by trial and error.

The operational definition of scaling is this: A quantity $m(t, L)$ depending on two variables, $t = (T - T_c)$ and L, is considered to have scaling if it can be expressed as[5]

$$m(t, L) = L^d f(t/L^c) \tag{38}$$

Eq. (38) is just Eq. (35) in different notations. f instead of $\tilde{m}$ and

$$\begin{aligned} d &= -\frac{\beta}{\nu} \\ c &= -\frac{1}{\nu} \end{aligned} \tag{39}$$

Assuming the knowledge of the scaling function $f(x)$, a collapse can be called *perfect* if by choosing the right c, d if

$$R = \frac{1}{N}\sum |L^{-d}\ m - f(t/L^c)| \tag{40}$$

[12]Note that the behaviour of m on either side of T_c is not symmetric and we should have two functions: $\tilde{m}$ for each side of T_c. However, we can combine these two functions to make a single new function and can thus change $|T|$ to t.

is zero.

However, we do not know the scaling function f beforehand as we search for fitting parameters c, d. We have in hand sets of data points (T, m) for lattices of different size. We take the scaling function to be defined by one of the sets - set p. Some values for c, d is assumed. The procedure for finding the right c, d without the knowledge of f is given below.

1. The *measure* P_b is defined.

$$P_b = \left[\frac{1}{N_{\text{over}}} \sum_p \sum_{j \neq p} \sum_{i,\text{over}} |L_j^{-d} m_{i,j} - \mathcal{E}_p(L_j^{-c} t_{i,j})| \right]^{\frac{1}{q}} \tag{41}$$

   All the remaining data sets $(j \neq p)$[13] are considered and given their values for $x = L_j^{-c} t_{ij}$, the $y = m_{i,j}$ value is decided by interpolating. i, j indicates the i^{th} value of t in j^{th} set. Here $\mathcal{E}$ is the interpolating function which uses set p as the scaling function and interpolates x to find corresponding y for all data points which lie in the range of the function defined by set p. This is indicated by 'over' in the summation. No extrapolation is done.

2. Now, the set p which defines the extrapolating function is changed to a different set and whole procedure is repeated. Similarly all sets are used to define $\mathcal{E}$ and P_b is calculated. N_{over} is the number of points which overlapped with the extrapolating function. This gives a value for P_b.

3. P_b is minimized by changing the values of c, d and iterating the whole procedure.

$q = 1$ is chosen in [5] and good results are reported:

- For the one-dimensional six-vertex model having $d = 1,\ c = -1$, the program returned values of $d = 0.007 \pm 0.004,\ c = -0.98 \pm 0.06$. P_b was minimized to 0.056881.
- For the two-dimensional Kasteleyn dimer model having $d = 0.5,\ c = -1$, the program returned values of $d = 0.5 \pm 0.03,\ c = -0.945 \pm 0.2$. P_b was minimized to 0.012424.

The authors had implemented this program in Fortran using numerical recipes[6]. The same program is implemented here using C++ by using numerical recipes[7]. The minimization is done using `amoeba` and 4-point extrapolation using `interp_1d.h` and sorting (necessary for using `interp_1d.h`) is done using `sort`. `amoeba` needs four guessed co-ordinates: (c, d, T_c) which form a tetrahedron in the three dimensional space of T_c, c, d around the correct values of (T_c, c, d).

The code was used for lattice sizes $L = 8, 10, 12, 14$. The input data was created by histogram method. The histograms for all lattices were generated at $T = 2.269$ using 10^7 MC loops/spin and allowing 10^3 MC loops/spin for equilibration. The input data of 10 points centered around $T = 2.26$ was fed to the code. The following results for ν, β, T_c were obtained:

- $\nu = 0.92$
- $\beta = 0.10$
- $T_c = 2.26$
- Minimum of the measure P_b reached is 3.8×10^{-5}.

[13]The $j = p$ set is not considered as it would not contribute.

The precise exponents got from theory are:

- $\nu = 1$
- $\beta = 0.125$
- $T_c = 2.269..$

Algorithm as flow-chart and the C++ code is enclosed in Appendix (8.5).

7 MC simulation of 3D Ising model on a square lattice

The code (sec.(8.1)) for the 2D Ising model can be modified in a straight forward manner to make MC simulations for the 3D Ising model on a square lattice with periodic boundary conditions.

- All spins aligned initially ($T = 0$ state) unlike the initial condition for 2D Ising Model ($T = \infty$) .
- MC loops/spin = 10^4
- MC loops/spin to allow for equilibration = 10^3

Internal energy

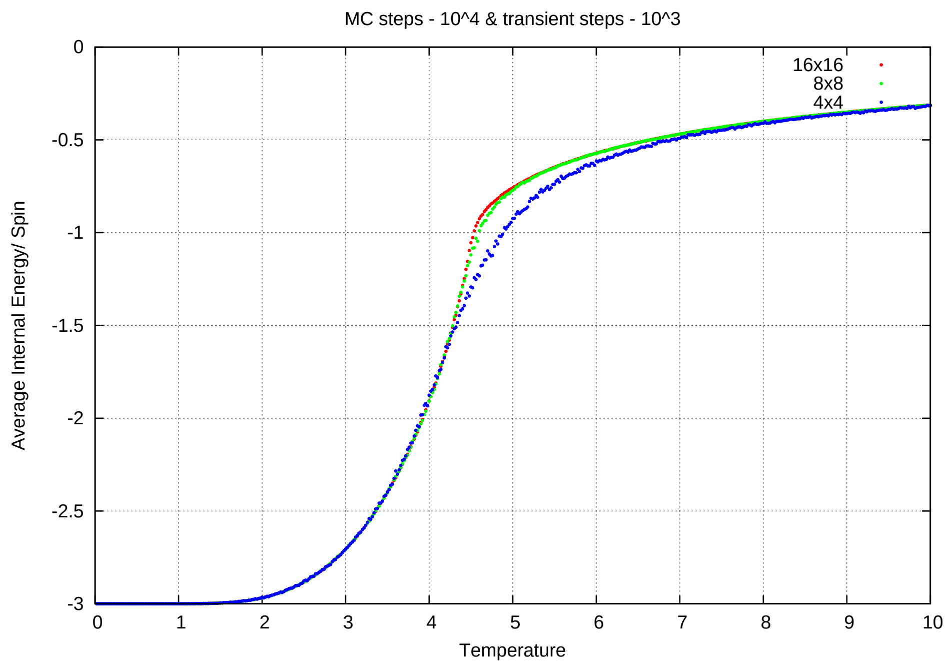


Figure 18: Average Internal energy for 3D Ising model

Specific heat

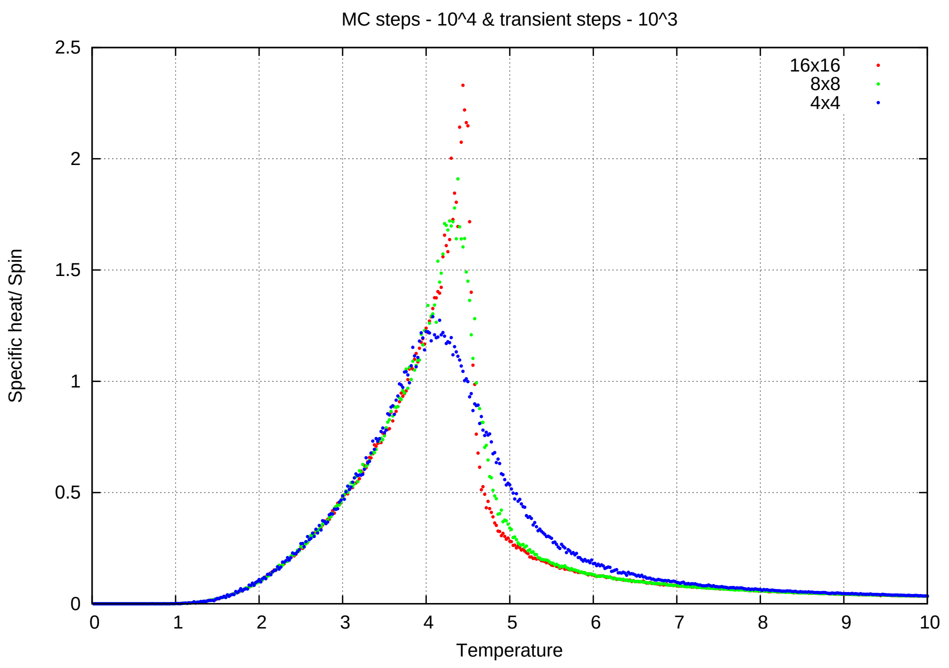


Figure 19: Specific heat for 3D Ising model

In comparison with fig.(5) where the peaks approach the limit T_c by moving left as L increases, the peaks in fig.(19) approach the limit T_c by moving right. The direction of approaching the limit, it is expected that

$$T_L = Tc + \frac{A}{L^{\nu}} \tag{42}$$

The sign of A is not necessarily universal.

Absolute Magnetization

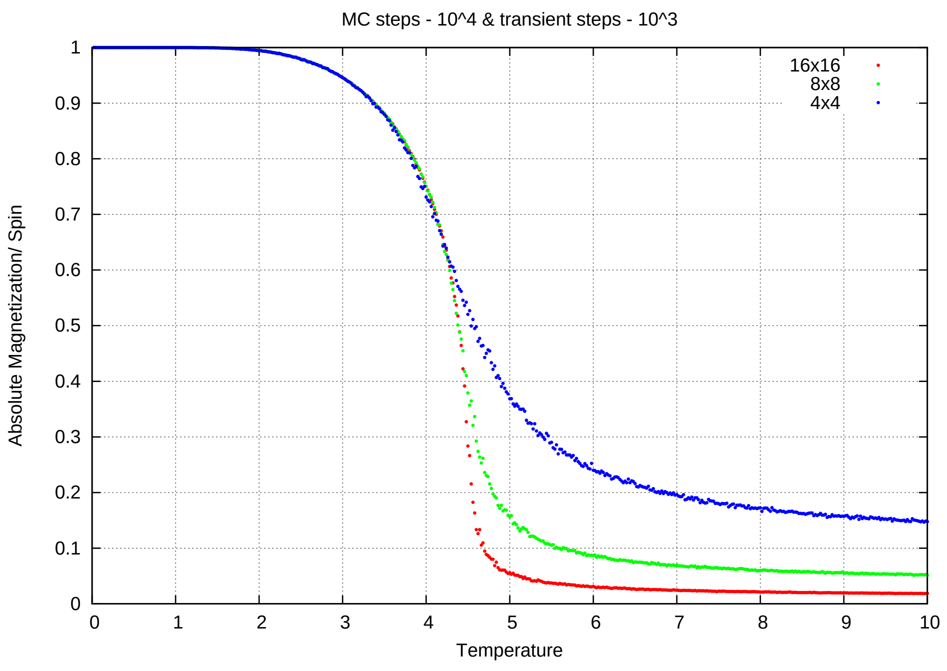


Figure 20: Absolute magnetization for 3D Ising model

Magnetic Susceptibility

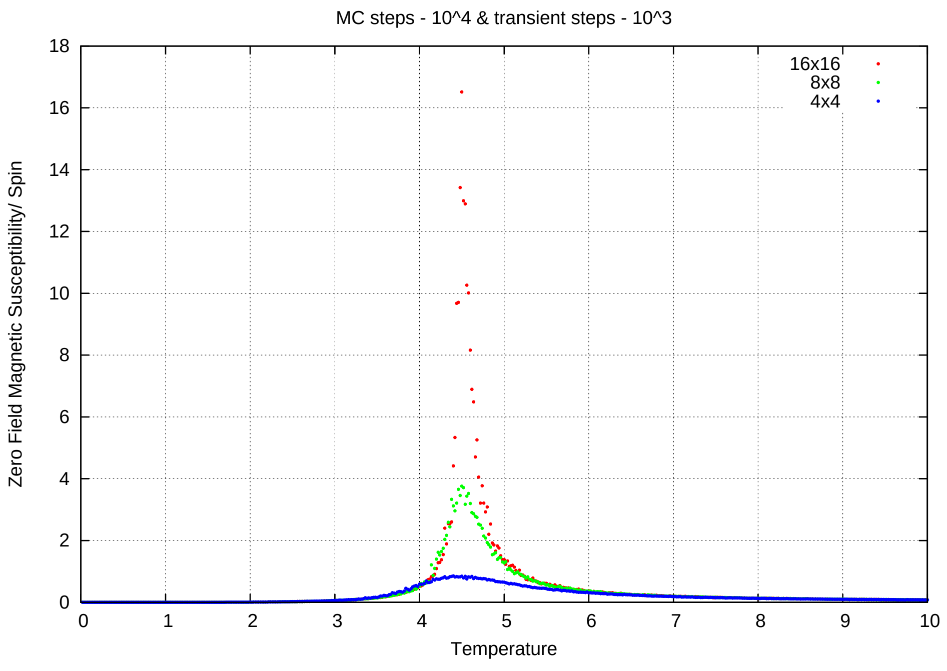


Figure 21: Zero field magnetic susceptibility for 3D Ising model

Binder Cumulant

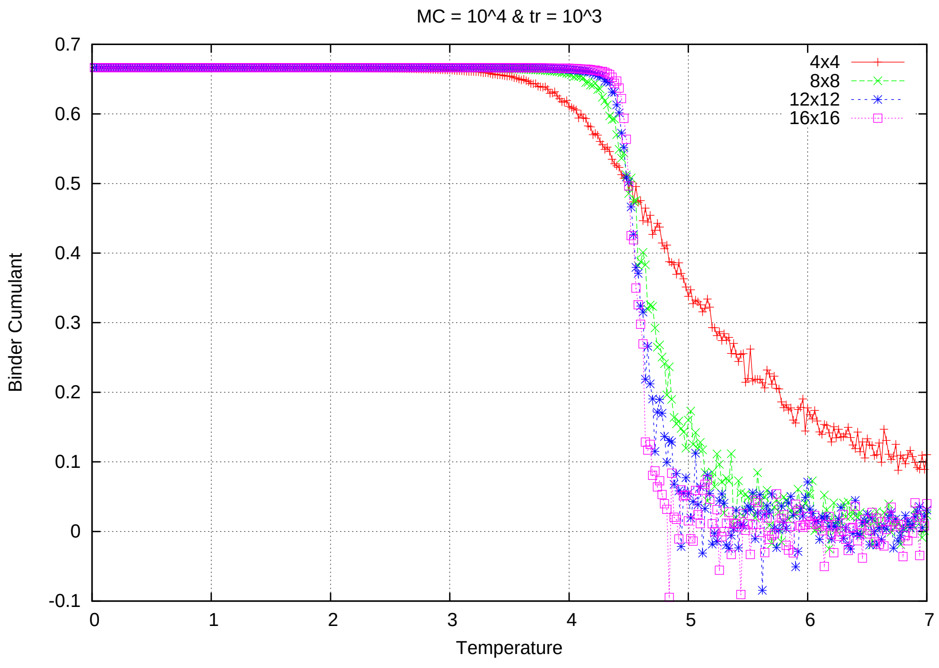


Figure 22: Binder cumulant for 3D Ising model

T_c was estimated to be 4.5.

The curves in Fig.(22) can be smoothened by using the histogram method. This would give a better estimate for T_c of 3D.

8 Appendix of C++ codes

8.1 MC for 2D Ising Model on a square lattice with periodic boundary conditions

File: /home/mahesh/msc/physics/year...oject/ising_model/codes/mc.cpp Page 1 of 2

```
using namespace std;
#include <iostream>
#include <math.h>
#include <fstream>
#include <stdlib.h>

int main()
{
const int L=14,N=L*L;
int s[L+1][L+1],h[L+1][L+1][3],i,k,u,v,up,vp;
const float TE=2.48,TB=2.48,dT=0.02,J=1,Kb=1;
const long int r=(TE-TB)/dT+1,M=pow(10,7)*N,tr=pow(10,3)*N; //(M/N) is the number of Monte Carlo
loops;
//tr is to allow for transition iterations for system to reach thermal equilibrium
double H=0,hold,U[r+1],U2[r+1],Mg[r+1],Mg2[r+1],m=0,T,beta,expo4,expo8,expo;


srand48(98374329);
ofstream out ("14b",ios::out);

        for(u=1;u<=L;u++){     //Initialization of spins in lattice using random numbers
                for(v=1;v<=L;v++){
                        if (drand48()>0.5){s[u][v]=1; m++;}  else {s[u][v]=-1; m--;}
                }
        }

        //d=1: Down bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(u==L){up=1;}  else{up=u+1;}
                h[u][v][1]=s[u][v]*s[up][v];
                H +=h[u][v][1];
                }
        }

        //d=2: Right bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(v==L){vp=1;}  else{vp=v+1;}
                        h[u][v][2]=s[u][v]*s[u][vp];
                        H +=h[u][v][2];
                }
        }
H *=-J; //This is the energy of the lattice for chosen configuration

        for(k=1;k<=r;k++){     //FOR loop for changing temperature
        T=(k-1)*dT+TB;
        beta=1/(Kb*T);
        expo4=exp(-4*J*beta);
        expo8=exp(-8*J*beta);
        U[k]=0;  U2[k]=0;
        Mg[k]=0; Mg2[k]=0;

                for(i=1;i<=M+tr;i++){
                u=ceil(drand48()*L);
                v=ceil(drand48()*L);

                        if(u==1){up=L;} else{up=u-1;}
                        if(v==1){vp=L;} else{vp=v-1;}
                        hold=-J*(h[u][v][1]+h[u][v][2]+h[up][v][1]+h[u][vp][2]);

                        if(hold>=0){
                        s[u][v]=-s[u][v]; //spin flipped
                        H +=-2*hold;

                        h[u][v][1]=-h[u][v][1];
                        h[u][v][2]=-h[u][v][2];
                        h[up][v][1]=-h[up][v][1];
                        h[u][vp][2]=-h[u][vp][2];

                        m +=2*s[u][v];
                        }
                        else if(hold<0){
                                if(hold==-J*2) {expo=expo4;} else{expo=expo8;}  //Avoids recalculating
exponential in every
                                                                                iteration
                                if(drand48()<expo){
                                s[u][v]=-s[u][v]; //spin flipped
                                H +=-2*hold;

                                h[u][v][1]=-h[u][v][1];
                                h[u][v][2]=-h[u][v][2];
                                h[up][v][1]=-h[up][v][1];
                                h[u][vp][2]=-h[u][vp][2];

                                m +=2*s[u][v];
                                }
                        }

                        if(i>tr and i%N==0){      //Greater than tr loops to remove transient
behaviour. Every N loops,                                                   lattice energy recorded

                        U[k] +=H;
                        U2[k] +=H*H;
                        Mg[k] +=fabs(m);
                        Mg2[k] +=m*m;
                        }
                }

        U[k]=U[k]/(M/N);
        U2[k]=U2[k]/(M/N);
        Mg[k]=Mg[k]/(M/N);
        Mg2[k]=Mg2[k]/(M/N);
        out<<T<<" "<<U[k]/N<<" "<<(Kb/N)*beta*beta*(U2[k]-pow(U[k],2))<<" "<<
        Mg[k]/N<<" "<<(beta/N)*(Mg2[k]-pow(Mg   [k],2))<<endl;
        }
}
```

8.2 Histogram method

File: /home/mahesh/delete/hibinew.cpp Page 1 of 3

```
//By putting Q=1, one spin is biased due to the extra magnetic field.
//Keep J=Q=1. Also should change program if you want to bias a different or more number of spins by
putting magnetic field.

using namespace std;
#include <iostream>
#include <math.h>
#include <fstream>
#include <stdlib.h>

int main()
{
const int L=4,N=L*L,J=1,Q=0;
int s[L+1][L+1],h[L+1][L+1][4],i,k,u,v,up,vp,m=0,H=0,hold;
const float T=2.5,Kb=1;
const int M=pow(10,4)*N,tr=pow(10,3)*N;
double beta=1/T,expo1,expo2,expo3,expo5,expo6,expo,MEpp[N+1][2*N*J+5],MEpn[N+1][2*N*J+5],MEnp[N+1]
[2*N*J+5],MEnn[N+1][2*N*J+5],pl,Ep[2*N*J+5],En[2*N*J+5],Mp[N+5],Mn[N+5],pe;

srand48(98374329);
ofstream out ("hibinew16",ios::out);
ofstream out2("hibienpro16",ios::out);
ofstream out3("hibimgpro4",ios::out);

        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if (drand48()>0.5){s[u][v]=1; m=m+1;}  else {s[u][v]=1; m=m+1;}
                }
        }

        //d=1: Down bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(u==L){up=1;}  else{up=u+1;}
                h[u][v][1]=s[u][v]*s[up][v];
                H=H+h[u][v][1];
                }
        }

        //d=2: Right bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(v==L){vp=1;}  else{vp=v+1;}
                h[u][v][2]=s[u][v]*s[u][vp];
                H=H+h[u][v][2];
                }
        }
H=-J*H;


        //d=3: Energy due to magnetic field
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                h[u][v][3]=0;
                }
        }
h[1][1][3]=s[1][1];  //The magnetic field acts only on the first spin
H=H-Q*h[1][1][3]; //This is the energy of the lattice for chosen configuration

        for(u=0;u<=N;u=u+1){
                for(v=0;v<=2*N*J+4;v=v+1){
                MEpp[u][v]=0; MEnp[u][v]=0; MEpn[u][v]=0; MEnn[u][v]=0;
                }
        }

expo1=exp(-(4*J+2*Q)*beta);
expo2=exp(-(4*J-2*Q)*beta);
expo3=exp(-(8*J+2*Q)*beta);
expo5=exp(-4*J*beta);
expo6=exp(-8*J*beta);
pl=0;
        for(i=1;i<=M+tr;i++){
```

```
        u=ceil(drand48()*L);
        v=ceil(drand48()*L);

                if(u==1){up=L;} else{up=u-1;}
                if(v==1){vp=L;} else{vp=v-1;}
        hold=-J*(h[u][v][1]+h[u][v][2]+h[up][v][1]+h[u][vp][2])-Q*h[u][v][3];

                if(hold>=0){
                s[u][v]=-s[u][v]; //spin flipped
                H=H-2*hold;
                h[u][v][1]=-h[u][v][1];
                h[u][v][2]=-h[u][v][2];
                h[up][v][1]=-h[up][v][1];
                h[u][vp][2]=-h[u][vp][2];
                h[u][v][3]=-h[u][v][3];
                m=m+2*s[u][v];
                }
                else if(hold<0){
                        if(hold==-J*2-Q) {expo=expo1;} else if(hold==-J*2+Q) {expo=expo2;} else if
(hold==-J*4-Q)                          {expo=expo3;}  else if(hold==-J*2) {expo=expo5;} else if
(hold==-J*4) {expo=expo6;}  else {}
                        if(drand48()<expo){
                        s[u][v]=-s[u][v]; //spin flipped
                        H=H-2*hold;
                        h[u][v][1]=-h[u][v][1];
                        h[u][v][2]=-h[u][v][2];
                        h[up][v][1]=-h[up][v][1];
                        h[u][vp][2]=-h[u][vp][2];
                        h[u][v][3]=-h[u][v][3];
                        m=m+2*s[u][v];
                        }
                }

        if(i>tr and i%N==0){
                if(m<0 and H<0) {MEnn[-m][-H]=MEnn[-m][-H]+1;}  else if(m<0 and H>=0) {MEnp[-m][H]=MEnp
[-m][H]+1;}
                else if(m>=0 and H<0) {MEpn[m][-H]=MEpn[m][-H]+1;} else {MEpp[m][H]=MEpp[m][H]+1;}
        } //Greater than tr loops to remove transient behaviour. Every N loops, lattice energy
recorded
        }

        for(u=1;u<=N;u++){                                                              //Output of joint
probability P(M,E)

                for(v=1;v<=2*N*J+4;v++){
                out<<u<<" "<<v<<"  "<<MEpp[u][v]*N/M<<endl;
                out<<-u<<" "<<v<<"  "<<MEnp[u][v]*N/M<<endl;
                out<<u<<" "<<-v<<"  "<<MEpn[u][v]*N/M<<endl;
                out<<-u<<" "<<-v<<"  "<<MEnn[u][v]*N/M<<endl;
                pl=pl+MEpp[u][v]+MEpn[u][v]+MEnp[u][v]+MEnn[u][v];
                }
        }
u=0;
        for(v=1;v<=2*N*J+4;v++){
        out<<u<<" "<<-v<<" "<<MEpn[u][v]*N/M<<endl;
        pl=pl+MEpn[u][v];
        }
        for(v=0;v<=2*N*J+4;v++){
        out<<u<<" "<<v<<" "<<MEpp[u][v]*N/M<<endl;
        pl=pl+MEpp[u][v];
        }
        v=0;
        for(u=1;u<=N;u++){
        out<<-u<<" "<<v<<" "<<MEnp[u][v]*N/M<<endl;
        pl=pl+MEnp[u][v];
        }
        for(u=1;u<=N;u++){  //u=0,v=0 considered already in above loops

        out<<u<<" "<<v<<" "<<MEpp[u][v]*N/M<<endl;
        pl=pl+MEpp[u][v];
        }
        cout<<pl*N/M<<endl;
```

```
pe=0;                                                                                //Output of P(E)
        for(v=0;v<=2*N*J+4;v++){
        Ep[v]=0; En[v]=0;
                for(u=0;u<=N;u++){
                Ep[v]=Ep[v]+MEpp[u][v]+MEnp[u][v]; //Ep[0]>=0 and En[0]=0
                En[v]=En[v]+MEpn[u][v]+MEnn[u][v];
                }
        Ep[v]=Ep[v]*N/M;
        En[v]=En[v]*N/M;
        out2<<v<<" "<<Ep[v]<<endl;;
                if(v!=0){out2<<-v<<" "<<En[v]<<endl;}
        pe=pe+Ep[v]+En[v];
        }
cout<<pe<<endl;

pe=0;                                                                                 //Output of P(M)
        for(v=0;v<=N;v++){
        Mp[v]=0; Mn[v]=0;
                for(u=0;u<=2*N*J+4;u++){
                Mp[v]=Mp[v]+MEpp[v][u]+MEpn[v][u];
                Mn[v]=Mn[v]+MEnp[v][u]+MEnn[v][u];
                }
        Mp[v]=Mp[v]*N/M;
        Mn[v]=Mn[v]*N/M;
                if(v%2==0){
                out3<<v<<" "<<Mp[v]<<endl;
                        if(v!=0){out3<<-v<<" "<<Mn[v]<<endl;}
                }
        pe=pe+Mp[v]+Mn[v];
        }
cout<<pe<<endl;
}
```

This program generates the initial probability distribution. To get thermodynamic quantities at other temperatures, this data should be fed into another program given below.

```
using namespace std;
#include <iostream>
#include <math.h>
#include <fstream>
#include <stdlib.h>

int main()
{
const float Kb=1,T1=2.40,beta1=1/(Kb*T1),TE=10,TB=0.1,dT=0.02; //T1 is the critical temperature at
which initial data is generated.
float T2,beta2;
const int L=14,N=L*L,J=1,r=(TE-TB)/dT+1;
int u,v,i,j;
double d,MEpp1[N+1][2*N*J+5],MEpn1[N+1][2*N*J+5],MEnp1[N+1][2*N*J+5],MEnn1[N+1][2*N*J+5],MEpp2[N+1]
[2*N*J+5],MEpn2[N+1][2*N*J+5],MEnp2[N+1][2*N*J+5],MEnn2[N+1][2*N*J+5],Ep1[2*N*J+5],En1[2*N*J+5],exmi
[2*N*J+5],expl[2*N*J+5],Ep2[2*N*J+5],En2[2*N*J+5],Mp2[N+1],Mn2[N+1],U,U2,M,M2,pl,pe,pm; //all the +5 s
defined in matrices become useless for Q=0 case. More lines are printed in output but they all will be
0s, nothing would be affected

ifstream ifl ("gen_newTc14",ios::in);
ifstream ifl2 ("gen_newTcen14",ios::in);
ofstream out ("out_new14", ios::out);

        for(u=1;u<=N;u++){                              //Input of the joint probability distribution P
(M,E) at T_C
                for(v=1;v<=2*N*J+4;v++){
                ifl>>MEpp1[u][v]>>MEpp1[u][v]>>MEpp1[u][v];
                ifl>>MEnp1[u][v]>>MEnp1[u][v]>>MEnp1[u][v];
                ifl>>MEpn1[u][v]>>MEpn1[u][v]>>MEpn1[u][v];
                ifl>>MEnn1[u][v]>>MEnn1[u][v]>>MEnn1[u][v];
                }
        }
u=0;
        for(v=1;v<=2*N*J+4;v++){
        ifl>>MEpn1[u][v]>>MEpn1[u][v]>>MEpn1[u][v];
        }
        for(v=0;v<=2*N*J+4;v++){
        ifl>>MEpp1[u][v]>>MEpp1[u][v]>>MEpp1[u][v];
        }
v=0;
        for(u=1;u<=N;u++){
        ifl>>MEnp1[u][v]>>MEnp1[u][v]>>MEnp1[u][v];
        }
        for(u=1;u<=N;u++){  //u=0,v=0 considered already in above loops
        ifl>>MEpp1[u][v]>>MEpp1[u][v]>>MEpp1[u][v];
        }

        for(v=0;v<=2*N*J+4;v++){                                        //Input of P(E) at T_c
        ifl2>>Ep1[v]>>Ep1[v];
                if(v!=0){ifl2>>En1[v]>>En1[v];}
        }

        for(j=1;j<=r;j++){                                              //Temperature loop
        d=0; U=0; U2=0; M=0; M2=0;   //Do not confuse U to be associated with T_c. U and U2 are both
associated with T2
        T2=(j-1)*dT+TB;
        beta2=1/(Kb*T2);
                for(i=-2*J*N-4;i<=-1;i++){
                exmi[-i]=exp(-(beta2-beta1)*(i));
                d=d+En1[-i]*exmi[-i];
                }

                for(i=0;i<=2*J*N+4;i++){
                expl[i]=exp(-(beta2-beta1)*(i));
                d=d+Ep1[i]*expl[i];                     //denominator calculated
                }
        pl=0;
                for(u=0;u<=N;u++){              //Input of the joint probability distribution P(M,E)
at T_C
                        for(v=0;v<=2*N*J+4;v++){
                        MEpp2[u][v]=expl[v]*MEpp1[u][v]/d;
                        MEnp2[u][v]=expl[v]*MEnp1[u][v]/d;
                        MEpn2[u][v]=exmi[v]*MEpn1[u][v]/d;
                        MEnn2[u][v]=exmi[v]*MEnn1[u][v]/d;
                        pl=pl+MEpp2[u][v]+MEnp2[u][v]+MEpn2[u][v]+MEnn2[u][v];
                        }
                }
        pe=0;
                for(v=0;v<=2*N*J+4;v++){                                          //Getting average
internal energy at T2
                Ep2[v]=0; En2[v]=0;
                        for(u=0;u<=N;u++){
                        Ep2[v]=Ep2[v]+MEpp2[u][v]+MEnp2[u][v];
                        En2[v]=En2[v]+MEpn2[u][v]+MEnn2[u][v];
                        }
                U=U+v*Ep2[v]-v*En2[v];
                U2=U2+v*v*(Ep2[v]+En2[v]);
                pe=pe+Ep2[v]+En2[v];
                }
        pm=0;
                for(v=0;v<=N;v++){                                                //Getting average
magnetization at T2
                Mp2[v]=0; Mn2[v]=0;
                        for(u=0;u<=2*N*J+4;u++){
                        Mp2[v]=Mp2[v]+MEpp2[v][u]+MEpn2[v][u];
                        Mn2[v]=Mn2[v]+MEnp2[v][u]+MEnn2[v][u];
                        }
                M=M+v*Mp2[v]+v*Mn2[v];  //Absolute magnetization
                M2=M2+v*v*(Mp2[v]+Mn2[v]);
                pm=pm+Mp2[v]+Mn2[v];
                }
        cout<<T2<<" "<<pl<<" "<<pe<<" "<<pm<<endl;
        out<<T2<<" "<<U/N<<" "<<(Kb/N)*pow(beta2,2)*(U2-pow(U,2))<<" "<<M/N<<" "<<(beta2/N)*(M2-pow
(M,2))<<endl;
        }
}
```

8.3 Error analysis

Bootstrap method

File: /home/mahesh/msc/physics/year...error/boot-strap/bootstrap.cpp Page 1 of 1

```
using namespace std;
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>

int main()
{
ifstream ifl ("32_7", ios::in);
ofstream out ("boot32_7",ios::out);

int i,j,k,a;
const int jack=200,L=32,N=L*L;
const float TE=1.8,TB=1.8,dT=0.2;
const long int MC=pow(10,5)+1;
float T;
double mag,en,mag2,en2,m[MC+1],e[MC+1],C,C2,chi,chi2,b;

srand48(9873980);

        for(T=TB;T<=TE;T=T+dT){
        C=0; C2=0; chi=0; chi2=0;

                for(i=1;i<=MC;i++){
                ifl>>e[i]>>m[i];
                }

                for(j=1;j<=jack;j++){
                mag=0; mag2=0; en=0; en2=0;
                        for(i=1;i<=MC;i++){
                        a=ceil(drand48()*pow(10,5));
                        en=en+e[a];
                        en2=en2+e[a]*e[a];

                        mag=mag+fabs(m[a]);
                        mag2=mag2+m[a]*m[a];
                        }
                en=en/MC;
                en2=en2/MC;

                mag=mag/MC;
                mag2=mag2/MC;

                b=(N/(pow(T,2)))*(en2-pow(en,2));
                C=C+b;
                C2=C2+pow(b,2);

                b=(N/T)*(mag2-pow(mag,2));
                chi=chi+b;
                chi2=chi2+pow(b,2);
                }

        C=C/jack;
        C2=C2/jack;

        chi=chi/jack;
        chi2=chi2/jack;

        out<<T<<" "<<sqrt(C2-pow(C,2))<<" "<<sqrt(chi2-pow(chi,2))<<endl;

        }
}
```

8.4 Correlation

File: /home/mahesh/msc/physics/year...orrelation/new/vers2.0/cor.cpp Page 1 of 2

```
using namespace std;
#include <iostream>
#include <math.h>
#include <fstream>
#include <stdlib.h>

int main(){
const int L=32,N=L*L; //Put even L
int s[L+1][L+1],h[L+1][L+1][3],i,j,k,u,v,up,vp;
const float TE=5,TB=0.1,dT=0.1,J=1,Kb=1;
const long int r=(TE-TB)/dT+1,M=pow(10,6)*N,tr=pow(10,2)*N;
double H=0,hold,m=0,T,beta,expo4,expo8,expo,cors[L],a,b[L];

srand48(98374329);
ofstream out ("cor32",ios::out);

        for(u=1;u<=L;u++){      //Initialization of spins
                for(v=1;v<=L;v++) {
                        if (drand48()>0.5){s[u][v]=1; m=m+1;}  else {s[u][v]=-1; m=m-1;}
                        }
        }

        //d=1: Down bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(u==L){up=1;}  else{up=u+1;}
                h[u][v][1]=s[u][v]*s[up][v];
                H=H+h[u][v][1];
                }
        }

        //d=2: Right bond's energy
        for(u=1;u<=L;u++){
                for(v=1;v<=L;v++){
                        if(v==L){vp=1;}  else{vp=v+1;}
                h[u][v][2]=s[u][v]*s[u][vp];
                H=H+h[u][v][2];
                }
        }
H=-J*H; //This is the energy of the lattice for chosen configuration

        for(k=1;k<=r;k++){      //FOR loop for changing temperature
        T=(k-1)*dT+TB;
        beta=1/(Kb*T);
        expo4=exp(-4*J*beta);
        expo8=exp(-8*J*beta);

        for(j=1;j<=L/2+1;j++){
        cors[j]=0;      b[j]=0;
        }
        a=0;

                for(i=1;i<=M+tr;i++){
                u=ceil(drand48()*L);
                v=ceil(drand48()*L);

                        if(u==1){up=L;} else{up=u-1;}
                        if(v==1){vp=L;} else{vp=v-1;}
                hold=-J*(h[u][v][1]+h[u][v][2]+h[up][v][1]+h[u][vp][2]);

                        if(hold>=0){
                        s[u][v]=-s[u][v]; //spin flipped
                        H=H-2*hold;

                        h[u][v][1]=-h[u][v][1];
                        h[u][v][2]=-h[u][v][2];
                        h[up][v][1]=-h[up][v][1];
                        h[u][vp][2]=-h[u][vp][2];

                        m=m+2*s[u][v];
                        }
                        else if(hold<0){
                                if(hold==-J*2) {expo=expo4;} else{expo=expo8;}  //Avoids recalculating
exponential in every
                                                                                iteration
                                if(drand48()<expo){
                                s[u][v]=-s[u][v]; //spin flipped
                                H=H-2*hold;

                                h[u][v][1]=-h[u][v][1];
                                h[u][v][2]=-h[u][v][2];
                                h[up][v][1]=-h[up][v][1];
                                h[u][vp][2]=-h[u][vp][2];

                                m=m+2*s[u][v];
                                }
                        }

                        if(i>tr and i%N==0){      //Greater than tr loops to remove transient
behaviour. Every N loops,                                                   lattice energy recorded
                        a=a+s[1][1];
                                for(j=2;j<=L/2+1;j++){
                                b[j]+=s[1][j]+s[1][L-(j-2)]+s[j][1]+s[L-(j-2)]
[1];

                                cors[j]+=s[1][1]*(s[1][j]+s[1][L-(j-2)]+s[j][1]+s[L-(j-2)][1]);
                                }
                        }
                }

        out<<endl<<endl<<endl<<"T = "<<T<<endl<<endl;
        a=a/(M/N);
                for(j=2;j<=L/2+1;j++){
                b[j]=b[j]/(4*M/N);
                cors[j]=cors[j]/(4*M/N);
                out<<j-1<<" "<<cors[j]-a*b[j]<<endl;   //j indicates lattice point, j-1 the distance
                                                                                  from (1,1)
                }
        }
}
```

8.5 Data collapse

Algorithm

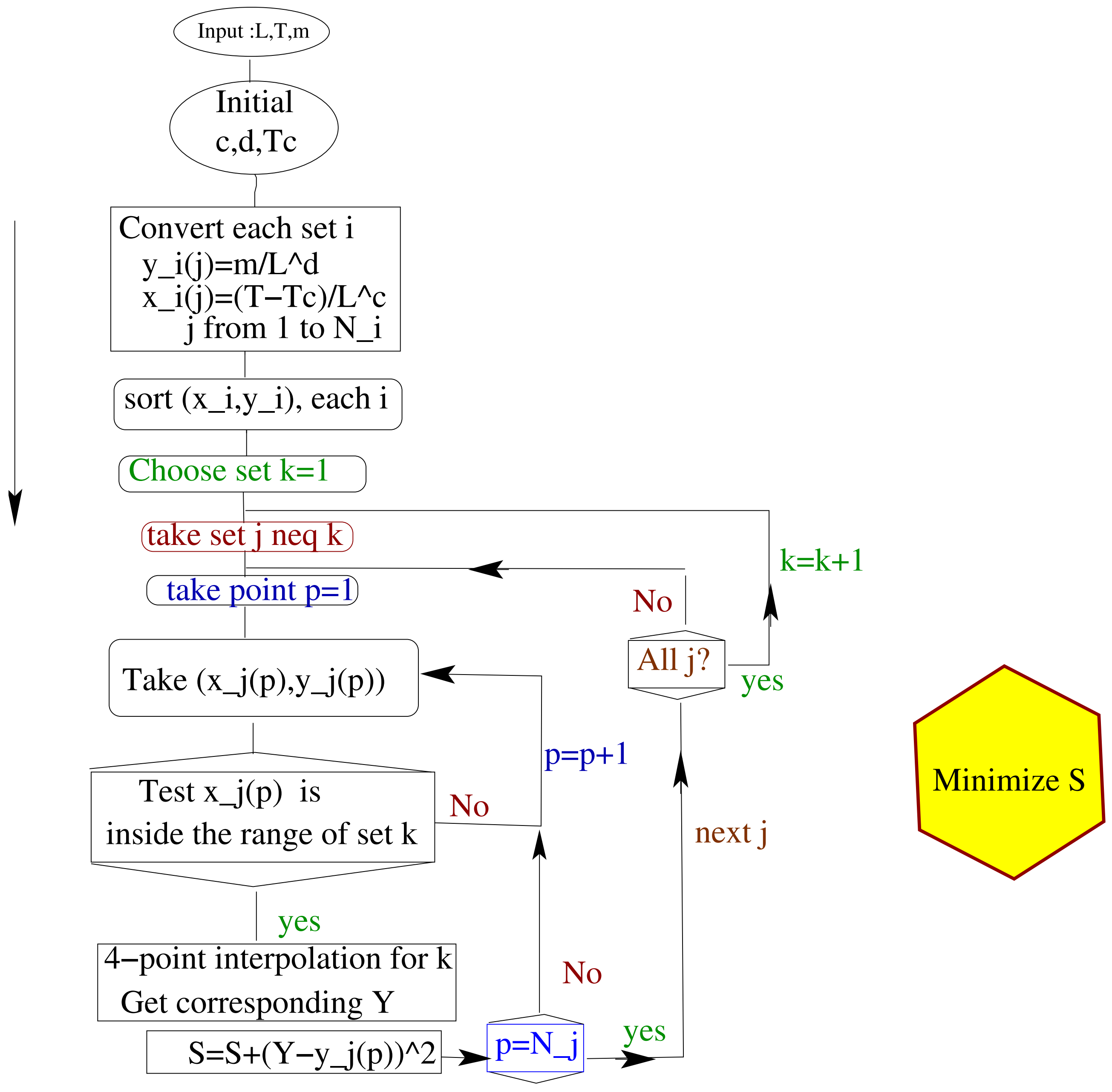


Figure 23: Flow-chart created by S.M. Bhattacharjee. Picture used with permission.

Code

File: /home/mahesh/msc/physics/year...es/data_collapse/collapse2.cpp Page 1 of 2

```
#include "nr3.h"
#include "amoeba.h"
#include "interp_1d.h"
#include "sort.h"

Doub use_pol(VecDoub a,VecDoub b,Doub c);
Doub work(VecDoub & par);


        int main(){
        //INITIATION
        VecDoub par(3);
        const Doub ftol=pow(10,-6);
        int i,j;

        Amoeba am(ftol);

        const int m=4,n=3;
        Doub p_array[m*n] = {                   //Defining initial simplex for amoeba
        -0.6, -0.1, 2.1,
        -1.4, -0.1, 2.1,
        -0.8, -0.166666667, 2.1,
        -1.1, -0.135, 2.4
        };

        cout << fixed << setprecision(6);
        MatDoub p(m,n,p_array);
        am.minimize(p,work);
        cout<<"Minimum of function "<<am.fmin<<endl;
        cout << "Number of function evaluations: " << am.nfunc << endl << endl;

        for (Int i = 0; i < m; i++) {
            cout<< i;
            for (Int j = 0; j < n; j++) {
                cout <<" " << am.p[i][j];
            }
          cout <<" "<< am.y[i] << endl;
           }
        }


Doub use_pol(VecDoub a,VecDoub b,Doub c){
Poly_interp myfunc(a,b,4);
return myfunc.interp(c);
}


        Doub work(VecDoub & par){
        ifstream ifl8 ("mc8x8",ios::in);     //all files have data points only around T_c (of the
respective lattices)
        ifstream ifl10 ("mc10x10",ios::in);
        ifstream ifl12 ("mc12x12",ios::in);
        ifstream ifl14 ("mc14x14",ios::in);

        const int l=10; //# of data points in input files
        const int lat=4; //# of lattices

        VecDoub L8x(l),L8y(l),L10x(l),L10y(l),L12x(l),L12y(l),L14x(l),L14y(l);
        VecDoub ax(l),ay(l),bx(l),by(l);
        int i,j,k;
        Doub y;
        Doub S=0;

        //INPUT
                for(i=0;i<l;i++){
                ifl8>>L8x[i]>>L8y[i];
                ifl10>>L10x[i]>>L10y[i];
                ifl12>>L12x[i]>>L12y[i];
                ifl14>>L14x[i]>>L14y[i];
                }
        //GETTING x_i(j),y_i(j)
                for(i=0;i<l;i++){
                L8x[i]=(L8x[i]-par[2])/pow(8,par[0]);                   L8y[i]=L8y[i]/pow(8,par[1]);
                L10x[i]=(L10x[i]-par[2])/pow(10,par[0]);                L10y[i]=L10y[i]/pow(10,par[1]);
                L12x[i]=(L12x[i]-par[2])/pow(12,par[0]);                L12y[i]=L12y[i]/pow(12,par[1]);
                L14x[i]=(L14x[i]-par[2])/pow(14,par[0]);                L14y[i]=L14y[i]/pow(14,par[1]);
                }

        //SORT
        sort2(L8x,L8y);
        sort2(L10x,L10y);
        sort2(L12x,L12y);
        sort2(L14x,L14y);

                for(i=0;i<lat;i++){
                        if(i==0){ax=L8x; ay=L8y;}   //These should be MANUALLY put
                        else if(i==1){ax=L10x; ay=L10y;}
                        else if(i==2){ax=L12x; ay=L12y;}
                        else if(i==3){ax=L14x; ay=L14y;}

                        for(j=0;j<lat;j++){
                                if(j!=i){       //To avoid same lattices
                                if(j==0){bx=L8x; by=L8y;}
                                else if(j==1){bx=L10x; by=L10y;}
                                else if(j==2){bx=L12x; by=L12y;}        //These should be MANUALLY put
                                else if(j==3){bx=L14x; by=L14y;}

                                        for(k=0;k<l;k++){
                                                if(bx[k]>=ax[0] and bx[k]<=ax[l-1]){//bx[k] should lie
in range of ax[k]

                                                //INTERPOLATION
                                                y=use_pol(ax,ay,bx[k]);
                                                S+=pow(y-by[k],2);
                                                }
                                        }
                                }
                        }
                }
        return S;
        }
```

9 References

1. *Crystal Statistics. I. A Two Dimensional Model with an Order-Disorder Transition* - Lars Onsager - Physical Review, Vol. 65 (1944)

2. *Monte Carlo Methods in Statistical Physics* - M.E.J. Newman & G.T. Barkema, Oxford University Press

3. *Monte Carlo Methods in Statistical Physics* - K.P.N. Murthy, University Press

4. Weisstein, Eric W. "*Monte Carlo Method.*" From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MonteCarloMethod.html`

5. *A Measure of Data Collapse for Scaling* - Somendra M Bhattacharjee and Flavio Seno 2001 J. Phys. A: Math. Gen. 34 6375

6. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical recipes in FORTRAN 77*, Cambridge University Press 1992.

7. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press 2007.

8. *Statistical Mechanics: Entropy, Order Parameters, and Complexity* - James P. Sethna, Oxford University Press

9. *Martensitic phase transition of bacteriophage T4's tail sheath* - Summer Project '14, Mahesh Gandikota under the instruction of S. M. Bhattacharjee

10. *Thermodynamics and an introduction to thermostatics* - Callen

11. The tail structure of bacteriophage T4 and its mechanism of contraction - Kostyuchenko et.al - nature: structural and molecular biology (2005)

12. Three-Dimensional Rearrangement of Proteins in the Tail of Bacteriophage T4 on Infection of Its Host - Leiman et.al. - Cell (2004)