The magnetization in the 2D Ising model: average values and large deviations

Francesca Pietracaprina

March 24, 2012

Introduction

Let us consider the 2D Ising model, whose Hamiltonian is

$$H = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_i s_i \quad \text{with } s_i = \pm 1.$$

From now on we will consider the magnetic field $H = 0$ and consider periodic boundary conditions. The idea is to simulate this system using Monte Carlo tecniques and, in particular, to use the heat bath algorithm to obtain the properties of the system at equilibrium and the population dynamics algorithm to obtain the large deviations.

In the first section I will comment the heat bath algorithm and show the result for the magnetization of a 2D Ising system as a function of the temperature. In the second section I will discuss the population dynamics algorithm and its applications; I will show its result for the large deviations of the magnetization.

1 Average values: the heat bath algorithm

The heat bath algorithm is able to generate the equilibrium state of the system by doing a series of Markovian steps in the configuration space. The algorithm works in the following way:

1. the system is initialized with random initial conditions;
2. for each particle in the system we perform the evolution of the configuration: the particle is singled out and its new state is determined (with a certain probability) while the state of all the other particles is considered fixed;
3. the evolution step is repeated the desired number of times.

In practice, the evolution of the configuration is splitted in two series; the first one, in which the system thermalizes, is trown away, while only the second part of the evolution is considered when computing the average quantities. This is due to the fact that we have to wait until the system, which is initialized with random spins, reaches the equilibrium state.

In the 2D Ising system we can easily compute the probability of the state[1] $s_i|S-s_i$: since $H(s_i|S - s_i) = -J(n_+ - n_-) \equiv -Jn$, where n_+ and n_- are the number of spin

[1] With the notation $s_i|S - s_i$ I indicate the state in which the i-th spin has value s_i (which can be equal or different from the value at the previous step) while all the spins in the system S, except s_i, are kept fixed.

1 and -1 respectively, the new configuration will have

$$s_i = \begin{cases} 1 & \text{with probability } p_+ = \frac{e^{\beta J n}}{2\cosh(\beta J n)} \\ -1 & \text{with probability } p_- = 1 - p_+ = \frac{e^{-\beta J n}}{2\cosh(\beta J n)} \end{cases}$$

In the code I implement the lattice as a grid of spins of dimension `npart` $\times$ `npart`, called `walker`. Each of them is identified with its row and column position. The periodic boundary condition is then implemented by imposing that:

```
walker[npart][i]=walker[0][i]
walker[i][npart]=walker[i][0]
```

The evolution step is then performed by the following function:

```
for(i=0;i<npart;i++) {
  for(j=0;j<npart;j++) {
    jplus1=j+1;
    jminus1=j-1;
    iplus1=i+1;
    iminus1=i-1;
    if (j==0) jminus1=npart-1; else if (j==npart-1) jplus1=0;
    if (i==0) iminus1=npart-1; else if (i==npart-1) iplus1=0;
    n=walker[iplus1][j]+walker[iminus1][j]+ \
      walker[i][jplus1]+walker[i][jminus1];
    p_up=exp(beta * J * n)/(exp(beta * J * n)+exp(-beta * J * n));
    x=(double)rand()/(double)RAND_MAX;
    if (x<=p_up) walker[i][j]=1; else walker[i][j]=-1;
  }
}
```

At the end of each evolution step the magnetization is computed and stored in an accumulator. After performing the required number of steps, the average value is computed:

$$\langle M \rangle = \frac{M_{\text{accumulator}}}{\#\text{steps}}$$

Moreover, the susceptibility can be computed through the result of the fluctuation–dissipation theorem:

$$\chi = \frac{\langle M^2 \rangle - \langle M \rangle^2}{k_b T}.$$

To estimate the correlation between the data points, due to the markovian nature of the walk in the configuration space, we can compute the autocorrelation function, which is given by

$$c(t) = \frac{\langle M(i+t)M(i) \rangle - \langle M \rangle^2}{\langle M^2 \rangle - \langle M \rangle^2}$$

and which typically decays exponentially: $c(t) \sim \mathrm{e}^{-\frac{t}{\tau}}$. In the program this is implemented via the following function:

```
void autocorrelations(double m[MAXITER],double maverage, \
          double maverage2,double T,int iter)
{
  double c[MAXTCORR];
  FILE * corrout;
```

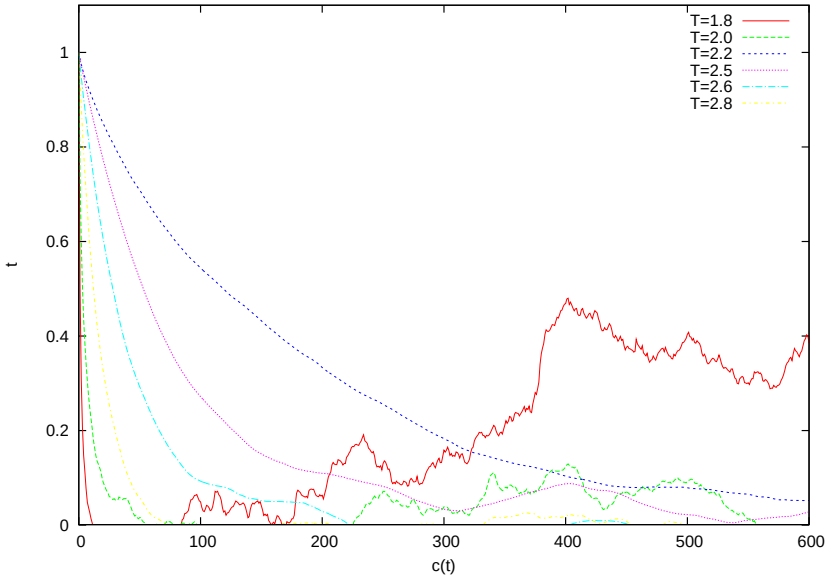

Figure 1: Autocorrelation function for the simulations at various different temperatures. Note that the function becomes noisy after it decays to very small values.

```
  char corrout_name[LNAME];
  int t,i;

  sprintf(corrout_name,"autocorr_%.1f.out",T);
  corrout = fopen(corrout_name,"w");
  for(t=0;t<MAXTCORR;t++){
    for(i=0;i<iter-t;i++) {
      c[t]+=m[t+i]*m[i];
    }
    c[t]=((c[t]/(double) (iter-t)) - pow(maverage,2)) / \
          (maverage2-pow(maverage,2));
    fprintf(corrout,"%i\t%lf\n",t,c[t]);
  }
  fclose(corrout);
}
```

The autocorrelation time τ is found to be or order 10 away from the transition and of order 100 reasonably near the critical point. Some representative examples of the autocorrelation function are shown in figure 1. The error on the Monte Carlo values is estimated as $\Delta M = \sigma_M \sqrt{\tau}$, where σ_M is the standard deviation "naively" calculated as $\sigma_M = \sqrt{\frac{\langle M^2 \rangle - \langle M \rangle^2}{\#\text{steps}}}$.

By running the program at different temperatures the graph of the magnetization, shown in figure 2, and the susceptibility, shown in figure 3, can be constructed. The simulation data was collected with a lattice of size 100×100 and with 50000 simulation steps. A certain number of equilibration steps have been considered: for values of the temperature sufficiently far from the transition equilibrium is reached afer $5 \div 10 \times 10^3$

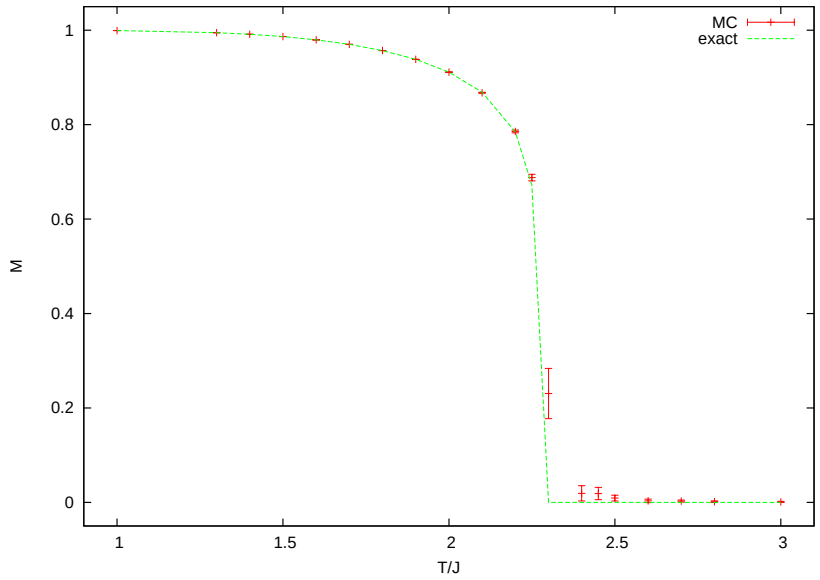

Figure 2: Magnetization in a 2D 100×100 Ising lattice and comparison with the exact result.

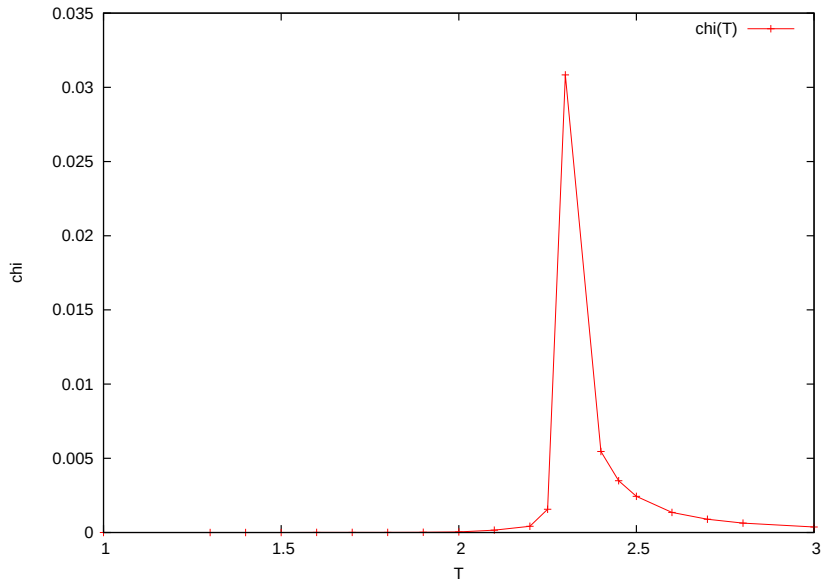

Figure 3: Susceptibility in the simulated Ising lattice.

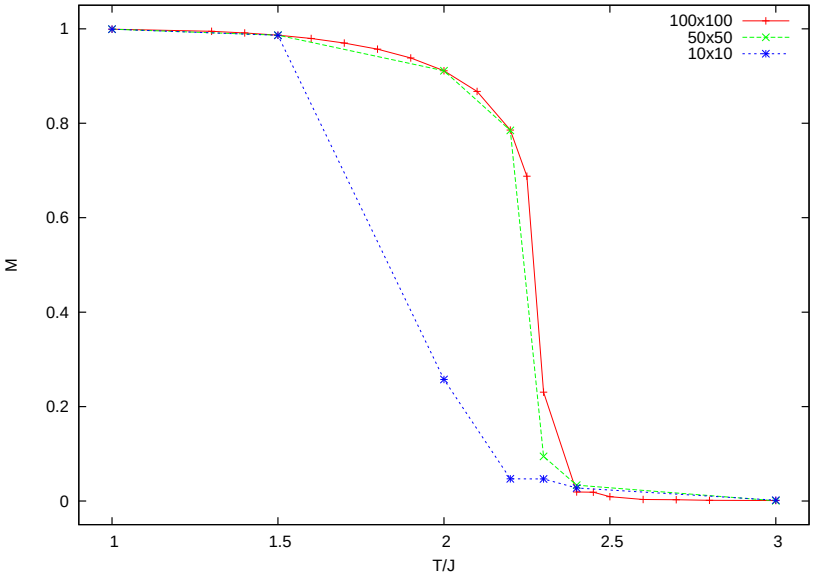

Figure 4: Comparison of the magnetization for simulations run on different lattice sizes.

steps, while near the transition the number of steps required is greater due to the critical slowing down and is of the order of the ones of the simulation itself ($30 \div 50 \times 10^3$ steps). The number of equilibration steps has been chosen high enough to let the system thermalize for all the values of the temperature considered in the simulations.

To check how the values scale with the lattice size, for some values of the temperature the magnetizations for a 100×100, 50×50 and 10×10 lattice were compared. The result is shown in figure 4. Note that for small enough systems there is a high enough probability that the magnetization is reversed (that is, the system goes into the other minimum) and this can cause anomalies in the measure of the magnetization. This kind of event actually happens in the data point at temperature 2.0 for the 10×10 lattice; taking a look at the (MC) time series of the evolution of the magnetization (figure 5) we indeed see that the system has switched the magnetization at a certain point.

The data reproduces well the values of the magnetization given by the exact result in the termodynamic limit. Note that around the critical temperature of the system in the termodynamic limit, which is $T_c \approx 2.269J$, the algorithm requires a very long time to relax to the equilibrium state, even in this relatively small finite size system, due to the critical slowing down.

2 Large deviations

An algorithm to compute directly the large deviations of quantities or, in other words, the atypical trajectories of a system in configuration space is presented by Giardina et al. in ref.s [1, 2]. Given a certain global observable Q, we are interested in its

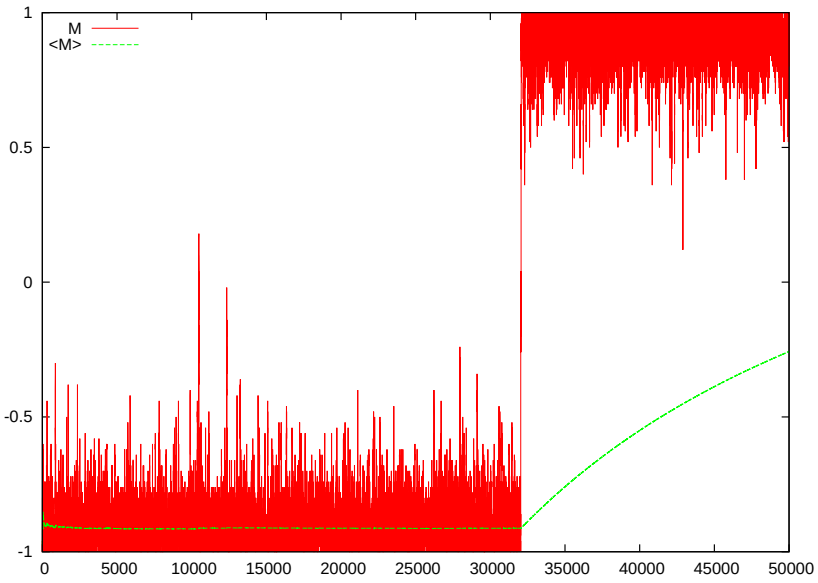

Figure 5: Magnetization in the 10×10 lattice as a function of the Monte Carlo time at $T = 2.0J$.

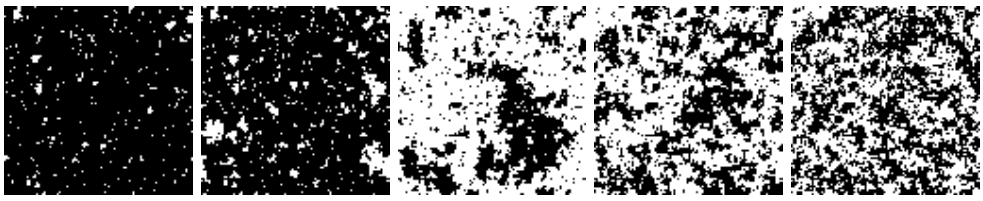

Figure 6: Snapshot of the Ising system at various temperatures: from left to right $\frac{T}{J} = 2, 2.2, 2.3, 2.5, 3$.

large deviations to better understand its probability distribution; moreover, the corresponding atypical trajectories of the system can give insight into the nature of the configuration which give rise to such large deviations, which, for many systems and observables, could be nontrivial.

Examples of nontrivial atypical trajectories can be easily found in chaotic dynamical systems. This method can be applied almost without variations (it is necessary to add a small stochastic perturbation to let the system explore the whole configuration space, due to the deterministic nature of the trajectories) to find with preference the integrable and the chaotic trajectories and, between these, "special" modes can be identified, such as, for example, the breathers modes and the solitons in the Fermi-Pasta-Ulam system.

This method allows to explore selectively trajectories which have values of the observable higher or lower than the average value. For sufficiently large deviations such a procedure is the only viable way to do so, since, provided that the observable, which is global with respect to the t iterations, has a sufficiently rapidly decaying probability distribution function, the large deviation theorem tells us that

$$P(Q(t) = qt) \sim \mathrm{e}^{ts(q)}$$

where $s(q)$ is the rate function; therefore the probability of observing deviations from the value q^*, which maximizes $s(q)$, is exponentially small in the number of iterations t.

In the next section I will describe in more detail how the algorithm works and how it is implemented in the source code; then I will show some results that can be obtained in the Ising model by biasing the trajectories towards higher or lower values of the magnetization and how this simple case can be interpreted as the action of an external magnetic field.

2.1 The population dynamics algorithm

In the specific case of the Ising system biased towards higher or lower magnetizations, we can define a partition function on the trajectories of the Markov chain (or, in mathematical terms, the moment generating function):

$$Z_t(\alpha) = \langle \mathrm{e}^{\alpha M} \rangle$$

where $M = M(\{x_n\})$ is the sum of the magnetization of all configurations of the trajectory of the chain, and the average is defined on the path of the Markov process. Considering the Markovian process which simulates the time evolution of the system and using the master equation, we can write the partition function explicitly as

$$\begin{aligned} Z_t(\alpha) &= \langle \mathrm{e}^{\alpha M} \rangle \\ &= \sum_{x_0, x_1, \ldots, x_t} \mathrm{e}^{\alpha M} P(x_0) p(x_0, x_1) p(x_1, x_2) \ldots p(x_{t-1}, x_t) \end{aligned}$$

where $p(x, y)$ are the transition probabilities from configuration x to y and $P(x_0)$ is the initial time probability distribution. The magnetization can be written in the more convenient way as a sum over transitions $M(\{x\}) = \sum_{n=1}^{t} M(x_n)[n - (n-1)]$; we can then put the partition function in the form

$$Z_t(\alpha) = \sum_{x_0, x_1, \ldots, x_t} P(x_0)\, \mathrm{e}^{\alpha M(x_1)} p(x_0, x_1)\, \mathrm{e}^{\alpha M(x_2)} p(x_1, x_2) \ldots \mathrm{e}^{\alpha M(x_t)} p(x_{t-1}, x_t)$$

that is, the chain evolves with its usual rates $p(x, y)$ and the extra factor $k(x_n) = \mathrm{e}^{\alpha M(x_n)}$ can be interpreted as a cloning step.

This form of the partition function allows the application of a population dynamics framework for the implementation of the computation. We start from an initial population of $N(0)$ clones of the system. We then repeat the following steps for each iteration:

1. each one of the clones evolves with $p_(x_n, x_{n-1})$ which, in our case, is equivalent to performing an heat bath sweep of the lattice;
2. each clone is replaced in average, with a uniform distribution, by a number $k(x_n)$ (`cloningrate` in the code) clones;
3. we compute the total number of clones at time n and store the ratio $\mathrm{n}(n) = \frac{N(n)}{N(n-1)}$ (in the code the array `nnewclones` is used);
4. we keep population constant to prevent explosion or extinction by uniformly removing or replicating clones with the rescaling factor $\mathrm{n}(t)$.

At the end, since $N(n) = \sum_{x,y} p(x, y)k(x)N(n-1, x)$, the partition function is given by

$$Z_t(\alpha) = \prod_{n=1}^{t} \mathrm{n}(n)$$

which gives all information about the moments of the distribution of the magnetization.

In the source code the previous steps are implemented in the main cycle of the program:

```
/*  Main iteration cycle*/

for(i=0; i<nstep1; i++) {
  nnewclones[i]=0;
  for(iclone=0;iclone<nclones;iclone++) {

    hbstep(walker, npart, beta, J, iclone);

    mcurrent[iclone] = magnetization(walker,npart,iclone);
    msum[iclone]  = msum[iclone]  + mcurrent[iclone] \
          * exp(-alpha*mcurrent[iclone]);

    if (i%1000==0) printf("Step: %6i     Clone:%6i     \
      Magnetization:  %10.8f   Mean Magnetization:  %10.8f\
      ", i+1,iclone+1,mcurrent[iclone],msum[iclone]/(double) (i+1));

    cloningrate=exp(alpha * mcurrent[iclone]);
    x=(double)rand()/(double)RAND_MAX;
    if (x<cloningrate-floor(cloningrate))
      cloningnumber=floor(cloningrate);
      else
      cloningnumber=floor(cloningrate)+1;
    for(j=0;j<cloningnumber;j++) {
      clone_walker(walker,newwalker,nnewclones[i]+j,npart,iclone);
    }
    nnewclones[i]+=cloningnumber;
    if (i%1000==0) printf("     Cloned %6i times\n",cloningnumber);
  }
```

```
  if (nnewclones[i]>=MAXCLONES) {printf("Failure: Population too small \
    for deviation %lf\n",alpha); return 1;}
  rescale_walker_population(walker,nclones,newwalker,nnewclones[i], \
    npart);
  if (i>0) {Zalpha*=(double)nnewclones[i]/(double)nnewclones[i-1];}
    else {Zalpha*=nnewclones[0]/(double)nclones;}
  if (i%10==0) printf("At step %i Population rescaled with ratio \
    %i/%i\n",i+1,nnewclones[i],nclones);
}
```

The function which performs the cloning is simply implemented in the following way:

```
void clone_walker(int walker[MAXCLONES][LATTICE][LATTICE], \
  int newwalker[MAXCLONES][LATTICE][LATTICE],int pos, \
  int npart, int iclone)
{
  int i,j;

  for(i=0;i<npart;i++) {
    for(j=0;j<npart;j++) {
      newwalker[pos][i][j]=walker[iclone][i][j];
    }
  }
}
```

The rescaling of the population requires more attention, since we have to randomly generate $|N(n) - N(0)|$ unique indeces which select the clones to remove or replicate. The operation is performed by the following function:

```
void rescale_walker_population(int walker[MAXCLONES][LATTICE][LATTICE],\
  int nclones,int newwalker[MAXCLONES][LATTICE][LATTICE], \
  int nnewclones, int npart)
{
  int k,t,indeces[MAXCLONES],indecescount;
  int x;
  int try_again;

    indecescount=0;
    for(;indecescount<abs(nnewclones-nclones);) {
      try_again=0;
      x=round((double)rand()/(double)RAND_MAX * nnewclones);
      for(t=0;t<indecescount;t++) {if (indeces[t]==x) try_again=1;}
      if (try_again==0) {
        indeces[indecescount]=x;
        indecescount++;
      }
    }
    for(k=0;k<indecescount;k++) {
      clone_walker(newwalker,walker,k,npart,indeces[k]);
    }

}
```

At the end of the iterations the value of the partition function and average magnetization between the surviving clones, along with its error, is computed. Moreover a sample of final comfigurations is saved on image files.

```
maverage=0;
for(iclone=0;iclone<nclones;iclone++) {
  maux=fabs(magnetization(walker,npart,iclone));
  maverage+=maux;
  maverage2+=maux*maux;
  mequilibr+=msum[iclone];
```

```
}

maverage=maverage/nclones;
maverage2=maverage2/nclones;
mequilibr=mequilibr/(nclones*nstep1);

printf("Magnetization (averaged over clones: large deviation): \
  %10.8f +-  %10.8f\n",maverage,merror);
printf("Magnetization squared (averaged over clones: \
   large deviation): %10.8f\n",maverage2);
printf("Magnetization (accumulated value: equilibrium): \
  %10.8f\n",mequilibr);
printf("Z(alpha): %10.8lf\n",Zalpha);
```

Note that when computing the large deviations of the magnetization (averaged over the population at the end of the iterations) it is not necessary to first do a certain number of thermalization step, since the final values are not averaged over the whole trajectory. It is even more efficient to start from a non thermalized state, since it helps the trajectory to go far from the typical ones faster than starting from equilibrium configurations.

The equilibrium ("non-deviated") result, called `mequilibr` in the code, is recovered by considering the average of the magnetization along the iterations and among the clones, reweighted with a factor $\mathrm{e}^{-\alpha M}$ (see the source code extract of the main iteration) to account for the weight $\mathrm{e}^{\alpha M}$ that we put in the partition function (and which we interpreted as cloning); since here the thermalization is not done, however, the number of iterations should be high enough so that the initial state becomes influent. The equilibrium value found, however, is the one of the high temperature state (that is, zero magnetization); this is most probably because the algorithm is insensitive to the sign of the magnetization (for a given sign of α) and there is no preference towards one of the two minima. In other words, since the algorithm is trying to find deviations from the two equilirium states (in the low temperature regime), it frequently jumps from one minima to the other and the end result is a mean magnetization close to zero. At this stage then this method can not be used to compute a useful equilibrium value.

2.2 Large deviations of the magnetization

Before commenting the results of simulation of the large deviations, let us make a remark. The procedure shown is completely general and allows to compute the cumulant generating function Ψ of the probability distribution of the quantity M as a function of α, the variable conjugated to M with respect to the Legendre transform of Ψ.[2] In the specific case of the (global) magnetization in the Ising system however there is a very simple interpretation for the extra term in the partition function:

$$Z_t(\alpha) = \langle \mathrm{e}^{\alpha M} \rangle = \langle \mathrm{e}^{\alpha \sum_i s_i} \rangle = \sum_{\{s_i\}} \mathrm{e}^{\beta J \sum_{\langle i,j \rangle} s_i s_j + \alpha \sum_i s_i}$$

[2] In more detail, we are considering the cumulant generating function (dynamical free energy):

$$\Psi = \lim_{t\to\infty} \frac{Z_t(\alpha)}{t};$$

the rate function (dynamical entropy) is then given by the large deviation theorem as the Legendre transform:

$$s(q) = \sup_{\alpha} \left(q\alpha - \Psi(\alpha)\right).$$

Figure 7: Snapshots of the Ising system at $T = 1$ with large deviation parameter $\alpha = 2$.

Figure 8: Snapshots of the Ising system at $T = 1.5$ with large deviation parameter $\alpha = 2$.

that is, what we are doing is considering a system with a nonzero external field $H = \frac{\alpha}{\beta}$. The large deviations (atypical values) of M can then be seen as the typical effect of an external magnetic field.

The program has been run for an intermediate value of α (with respect to the temperature and the coupling), which I chose equal to 2. The effect is that of a magnetic field of opposite sign regardless of the sign of α in the ordered phase due to the freedom of the selection of the ground state of the Monte Carlo algorithm[3]. I have also set the dimension of the lattice to 100^2, the initial population number to $N = 100$ and I have performed 10^3 iterations[4] for four different values of the temperature: $T = 1$, 1.5, 2.2, $3J$. The snapshots of some representative clones of the system are shown in figures 7, 8, 9, and 10. Supposing no previous knowledge of the nature of the large deviations configurations, it can be shown from this analysis that the unusually low magnetization in the ordered phase is due to the appearance of domains of inverted spins, while in the disordered phase domains are not formed even though the magnetization is different (higher in absolute value) than the typical (zero field) value.

Finally, I considered the system at the temperature $T = 2.0J$, fixed, and ran simulations with different values of the parameter α. The results for $\langle M \rangle$ e $\langle M^2 \rangle$

[3]This algorithm finds only (or preferentially, depending on the value of α) dampening large deviations due to the fact that the phase space is explored with the constraint given by the allowed values of the total magnetization.

[4]The results should be however checked with a greater number of iterations, especially for the temperature $T = 2.2$ due to the critical slowing down effect.

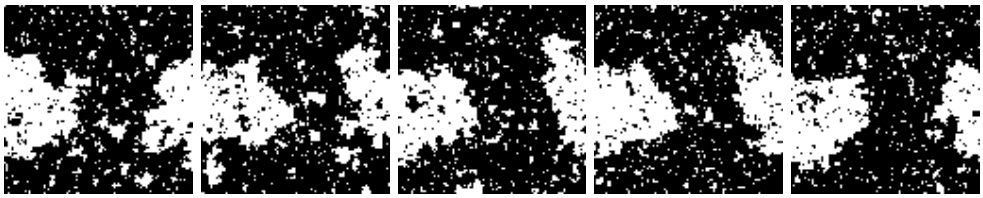

Figure 9: Snapshots of the Ising system at $T = 2.2$ with large deviation parameter $\alpha = 2$.

Figure 10: Snapshots of the Ising system at $T = 3$ with large deviation parameter $\alpha = 2$.

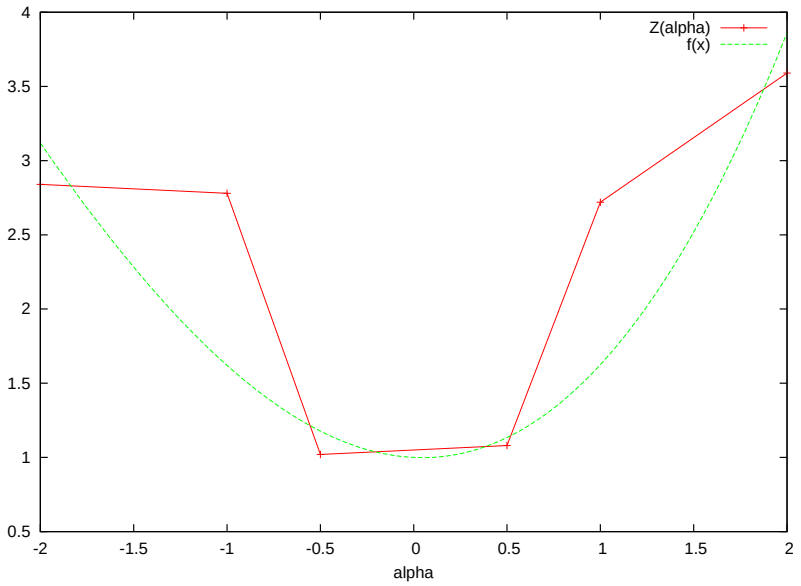

Figure 11: Moment generating function Z for the distribution of the magnetization at temperature $T = 2.0J$ as a function of the large deviation parameter α.

are shown in figure 12 and those for $Z(\alpha)$ are in figure 11. The coefficients of the fit with a polynomial gives a very rough estimate of the moments of the distribution; approximating to the third order we find

$$\langle M \rangle = -0.06 \pm 0.77 \quad \langle M^2 \rangle = 1.25 \pm 0.33 \quad \langle M^3 \rangle = 0.4 \pm 1.3.$$

The averages are those of the symmetric phase due to the fact that they are computed on the whole partition function.

References

[1] Giardina, Kurchan, Lecomte, Tailleur, *Simulating rare events in dynamical processes*, arXiv 1106.4929

[2] Tailleur, Lecomte, *Simulation of large deviation functions using population dynamics*, arXiv 0811.1041

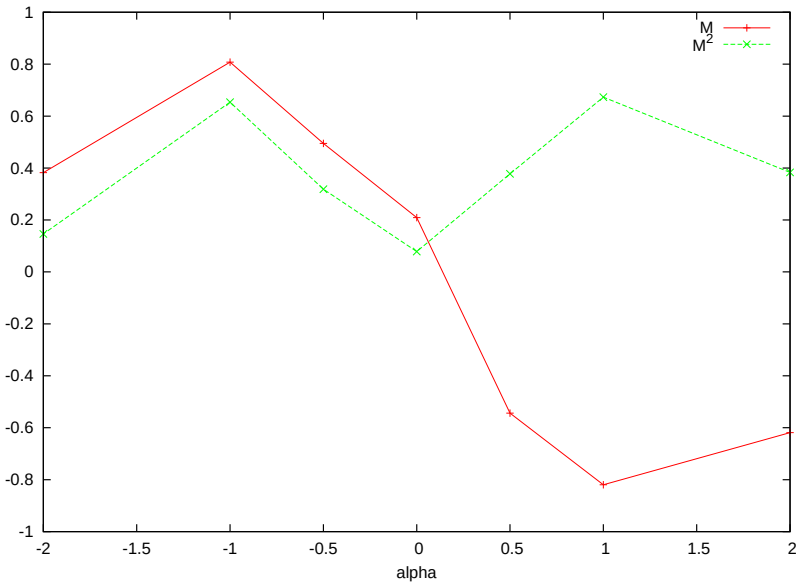

Figure 12: Mean magnetization $\langle M \rangle$ of the system and $\langle M^2 \rangle$ at temperature $T = 2.0J$ as a function of the large deviation parameter α.

[3] Gardiner, *Stochastic Methods*, Springer

[4] *Numerical Recipes*