Submission Phase

1. Do assignment ☑ (/smac-001/human_grading/view/courses/971628/assessments/12/submissions)

Evaluation Phase

2. Evaluate peers ☑ (/smac-001/human_grading/view/courses/971628/assessments/12/peerGradingSets)

Results Phase

3. See results ☑ (/smac-001/human_grading/view/courses/971628/assessments/12/results/mine)

Your effective grade is 18

Your unadjusted grade is 18, which is simply the grade you received from your peers.

See below for details.

In this homework session 8 of **Statistical Mechanics: Algorithms and Computations**, you will study the **local Metropolis**, the **heatbath** and the **Cluster Monte Carlo algorithms** for the two-dimensional Ising model on a square lattice with periodic boundary conditions.

At the end of the homework session, you will have validated all three programs against known analytic results. furthermore, you will have compared the results to a classic result of Ferdinand and Fisher, from 1969, analyzed the time-behavior of the cluster algorithm of Wolff (1989), and started to understand Propp and Wilson's coupling from the past approach... So -let's get started with **Homework 8 of Statistical Mechanics: Algorithms and Computations**.

A Here, you implement the local Metropolis algorithm, and validate it through the comparison with an exactly known result. In a real-life application, this validation would run for many days on a computer, to establish agreement with the exact result to 5 or 6 significant digits.

A1 Download (cut-and-paste) the **local Metropolis algorithm** for the Ising model, as shown below.

```
import random, math, os

def energy(S, N, nbr):
    E = 0.0
    for k in range(N):
        E -=  S[k] * sum(S[nn] for nn in nbr[k])
    return 0.5 * E

L = 6
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
                                    for i in range(N)}
T = 2.0
filename = 'local_'+ str(L) + '_' + str(T) + '.txt'
S = [random.choice([1, -1]) for k in range(N)]
nsteps = N * 100
beta = 1.0 / T
Energy = energy(S, N, nbr)
E = []
for step in range(nsteps):
    k = random.randint(0, N - 1)
    delta_E = 2.0 * S[k] * sum(S[nn] for nn in nbr[k])
    if random.uniform(0.0, 1.0) < math.exp(-beta * delta_E):
        S[k] *= -1
        Energy += delta_E
    E.append(Energy)
E_mean = sum(E)/ len(E)
print sum(E) / len(E) / N, 'mean energy'
```

Greatly increase the number of iterations of the algorithm. **Run it** for L=6 and for T = 2.0. Check that you recover the exact value E/N =-1.747 (mean energy per spin) (known from exact enumeration). **Communicate the results obtained** in **four independent runs** of the algorithms (no calculation of error bars required). **Indicate your value of nsteps**.

3 600 nsteps, mean energy = -1.72194444444

36 000 nsteps, mean energy = -1.74187037037

360 000 nsteps, mean energy = -1.74417561728

3 600 000 nsteps, mean energy = -1.74959552469

36 000 000 nsteps, mean energy = -**1.74723912037**

So, we do recover the exact value of mean energy per spin (E/N) = -1.747 when we simulate for 36 million nsteps.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

In this section, you should evaluate whether your fellow student can perform a rough validation of a Monte Carlo program.
There are two issues:
1/ the value of nsteps should be given (nsteps = N*1000 or N*10000 or N*100000 or N*1000000 are OK)
2/ Four results of runs should be given (but it is also OK, if the mean and the standard deviation or mean and error are given)

The following answer would have obtained full score:

I tried nsteps = N * 100000. in four runs, I obtained
-1.74861148148
-1.74520611111
-1.74742802469
-1.74742802469
All these values are very close to -1.747, the exact result. I suppose therefore that the program is OK.

POINTS - DESCRIPTION
0 points - question not treated
1 point - question treated but results not conclusive or nsteps not indicated
2 points - program was run four times, conclusions clear, value of nsteps given

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

A2 Modify your program so that it allows you to read in and write out configurations and also to print configurations on the screen. **For simplicity, we provide this program**, you are free to download (cut-and-paste), run and modify it.

```
import random, math, os, pylab

def x_y(k, L):
    y = k // L
    x = k - y * L
    return x, y

L = 128
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
                                    for i in range(N)}

T = 1.0
filename = 'local_'+ str(L) + '_' + str(T) + '.txt'
if os.path.isfile(filename):
    f = open(filename, 'r')
    S = []
    for line in f:
        S.append(int(line))
    f.close()
    print 'starting from file', filename
else:
    S = [random.choice([1, -1]) for k in range(N)]
    print 'starting from scratch'
nsteps = N * 100
beta = 1.0 / T
for step in range(nsteps):
    k = random.randint(0, N - 1)
    delta_E = 2.0 * S[k] * sum(S[nn] for nn in nbr[k])
    if random.uniform(0.0, 1.0) < math.exp(-beta * delta_E):
        S[k] *= -1

conf = [[0 for x in range(L)] for y in range(L)]
for k in range(N):
    x, y = x_y(k, L)
    conf[x][y] = S[k]
pylab.imshow(conf, extent = [0, L, 0, L], interpolation='nearest')
pylab.set_cmap('hot')
pylab.colorbar()
pylab.title('Local_'+ str(T) + '_' + str(L))
pylab.savefig('Local_'+ str(T) + '_' + str(L)+ '.png')
pylab.show()
f = open(filename, 'w')
for a in S:
   f.write(str(a) + '\n')
f.close()
```

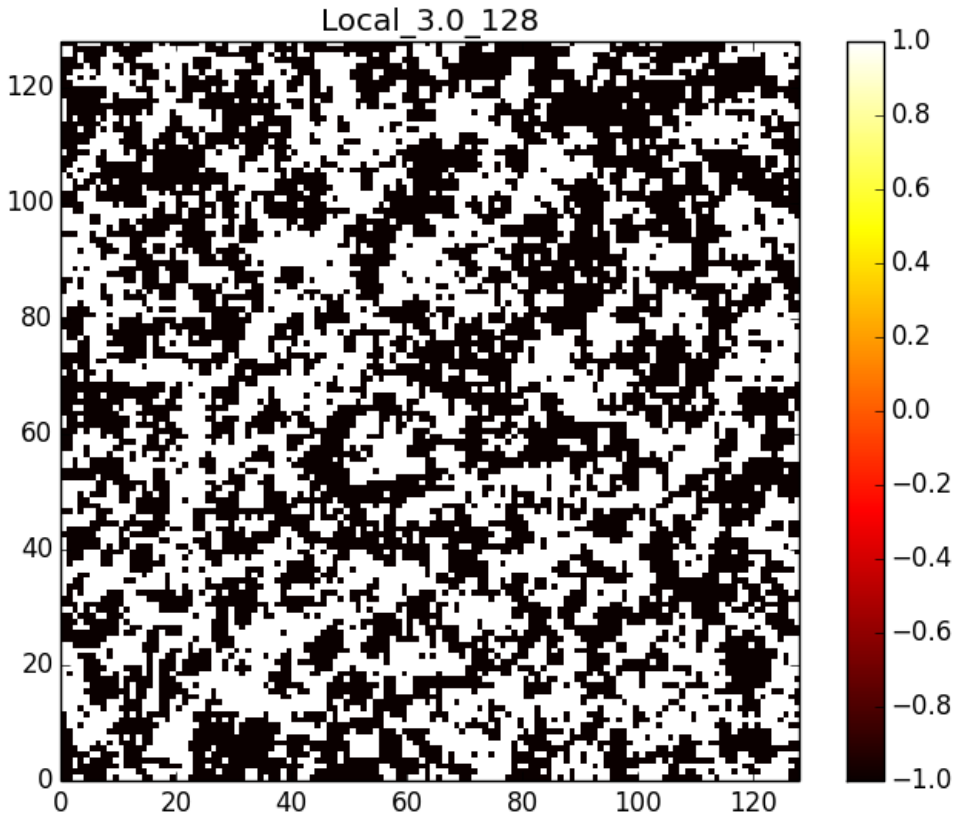
From a random initial configuration, run this program at high temperature **T = 3.0 (L=128)** and at the critical temperature **T_crit = 2.27 (L=128)**. Run sufficiently long so that you have reached what you would consider as the t-> \infty limit. (At T_crit, you should run your code for at least a few minutes). **Upload**

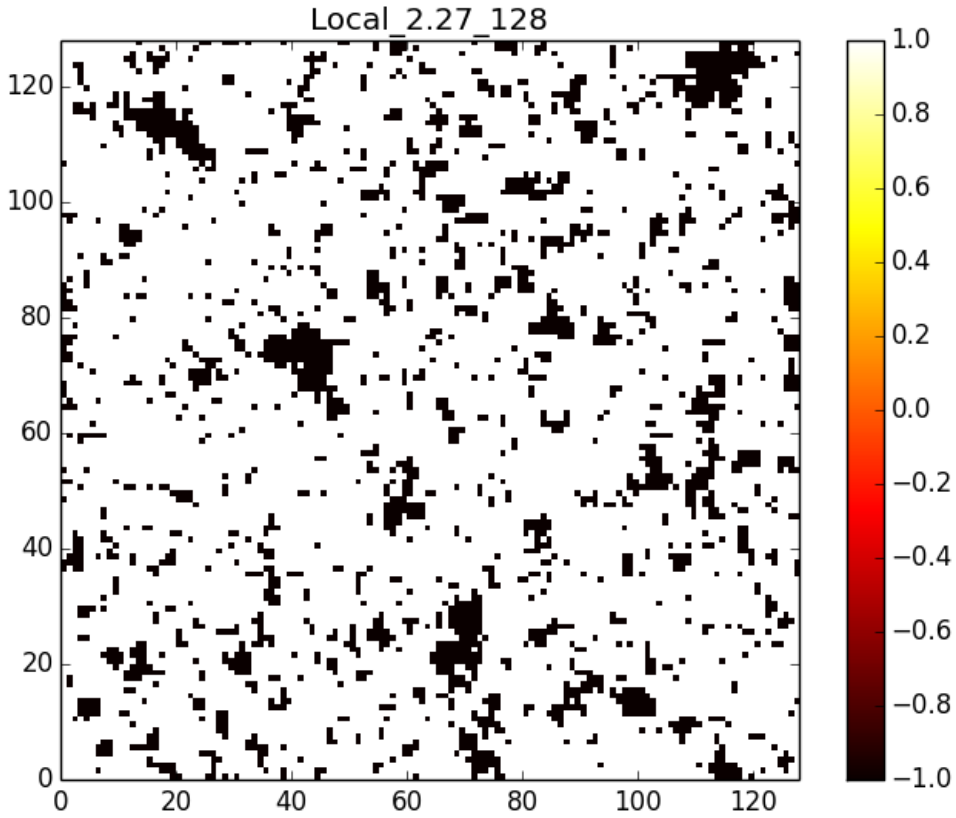
typical configurations that you obtained and comment on what you see. Notice that subsequent runs of this program realize a Markov chain (three runs with 100 + 100 + 100 iterations = 1 run with 300 iterations).

At low temperature, **T = 1.0**, run your code for **L=32,** first for **10 N** iterations, then again, **several times for 10 N** iterations, then several times **for 100 N** iterations (if you see a stripe, run your program longer, up to several ties 1000 N or several times 10000 N iterations). You should observe that, after a quite long time, your simulation goes from a state with domain walls into a purely ferromagnetic state (essentially all spins +1 or essentially all spins -1). **Upload two or three graphics files at different times. Do you observe** (running for several times 10000 N iterations) that the local Metropolis algorithm flips between configurations of negative overall magnetization (mostly black) and configurations of positive overall magnetization (mostly white)?

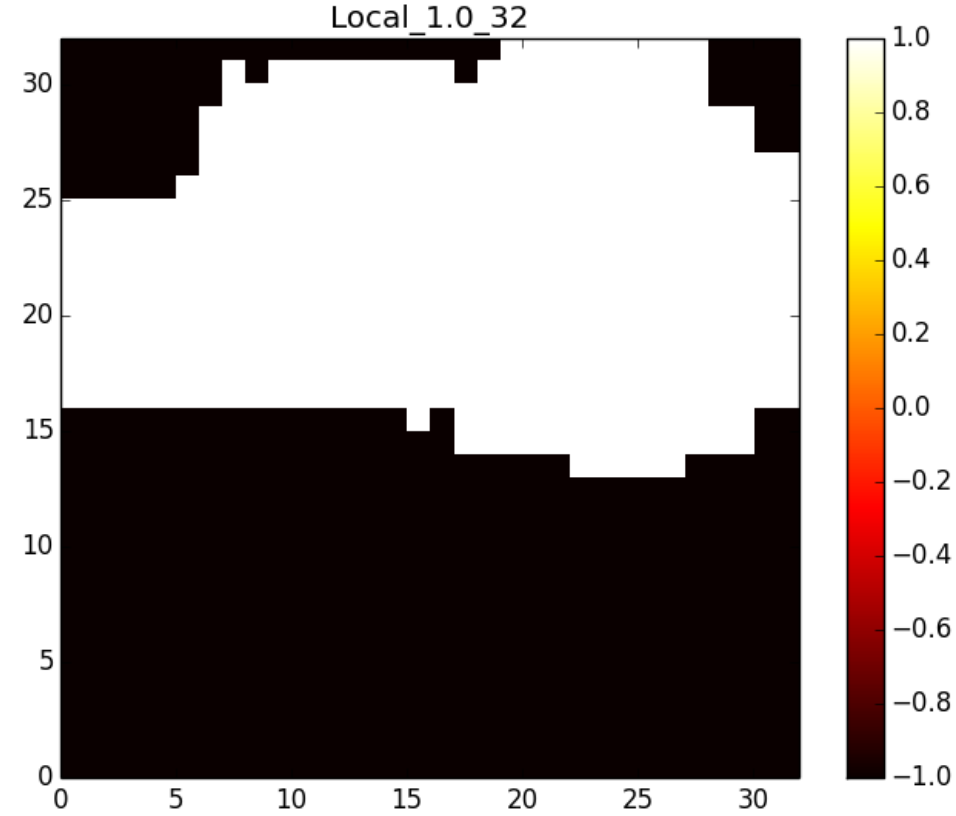
At low temperature, **T = 1.0** (as before), run your code for **L=128** and **upload a graphics file**. **Explain** what you observe. In particular, do you observe that the system becomes homogeneous (mostly white or mostly black) on the available time scales?

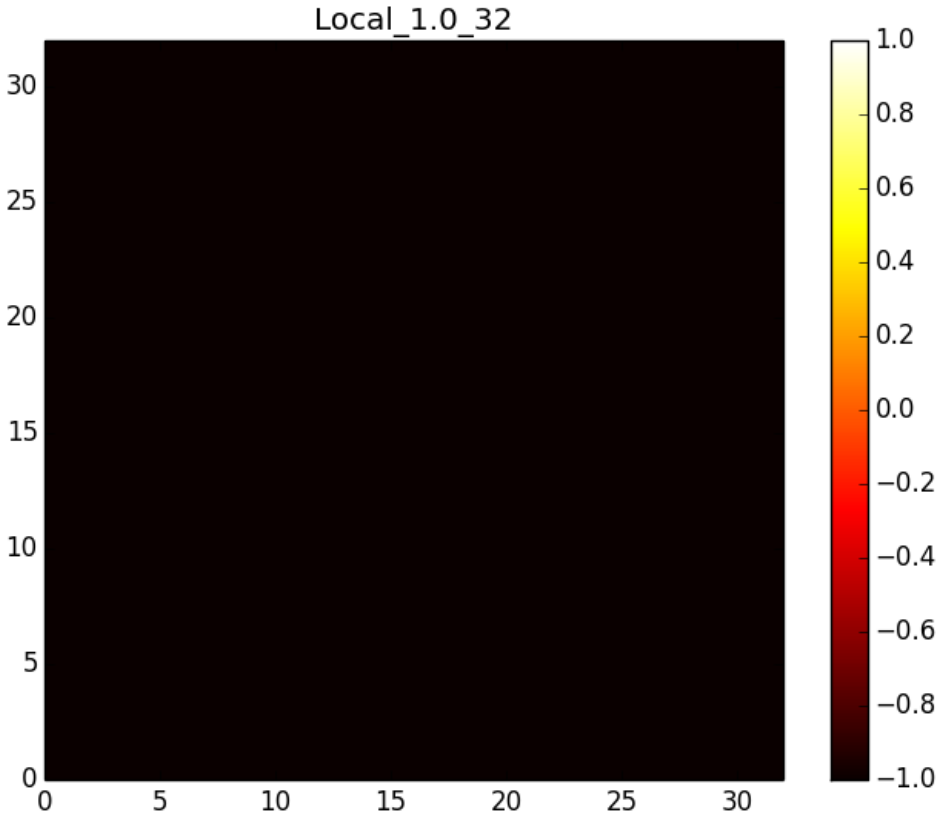
From a random initial configuration, the program was run at high temperature **T = 3.0 (L=128)** and at the critical temperature **T_crit = 2.27 (L=128)**.

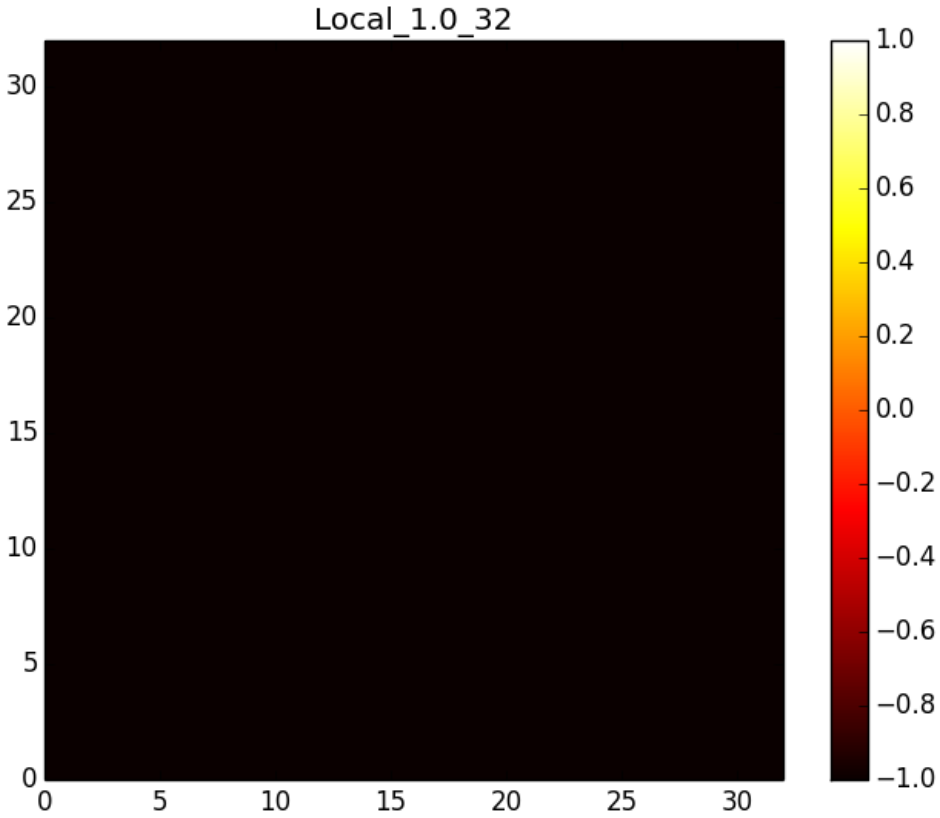




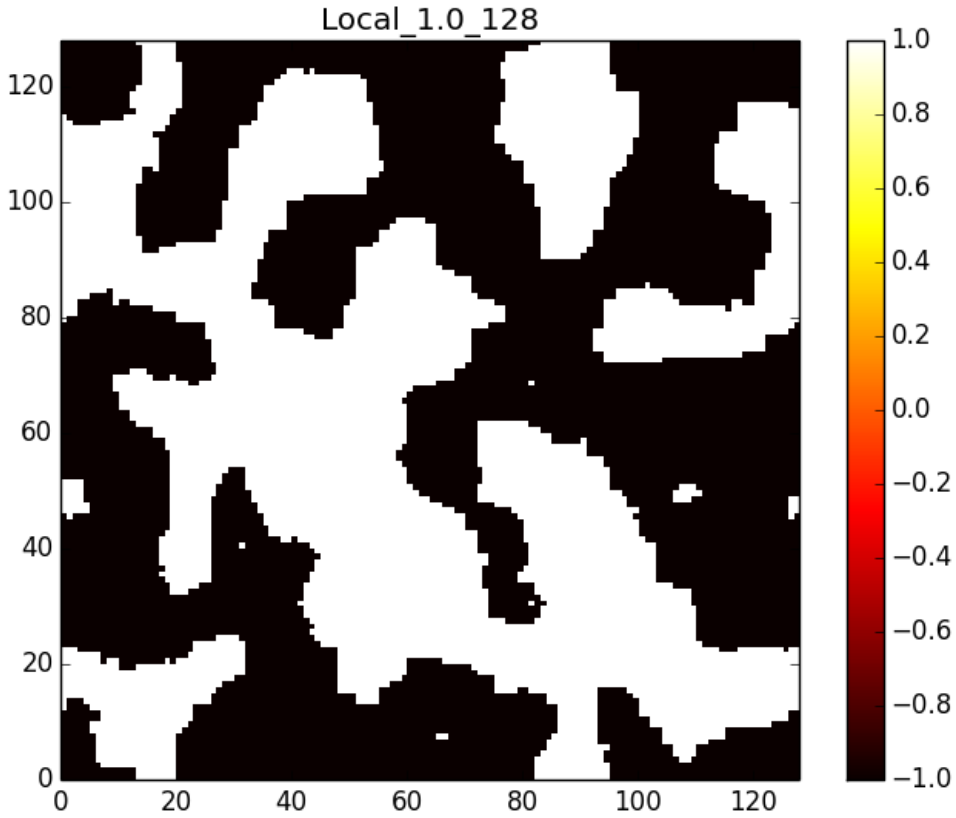
At low temperature, **T = 1.0**, the code was run for **L=32, 5 times for 10 N, 100 N**, **1000 N** and **10000 N** iterations each. Unfortunately, despite running 5 times with 10000N steps, no flip from overall negative (black) magnetization to overall positive (white) magnetization was seen.

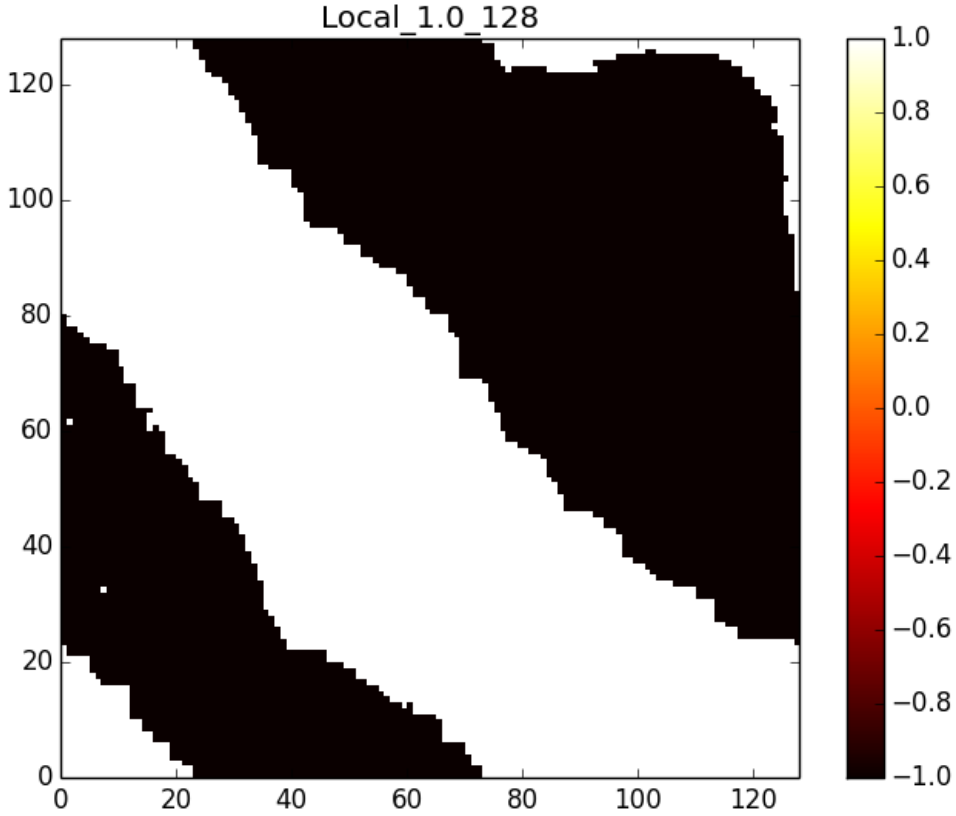


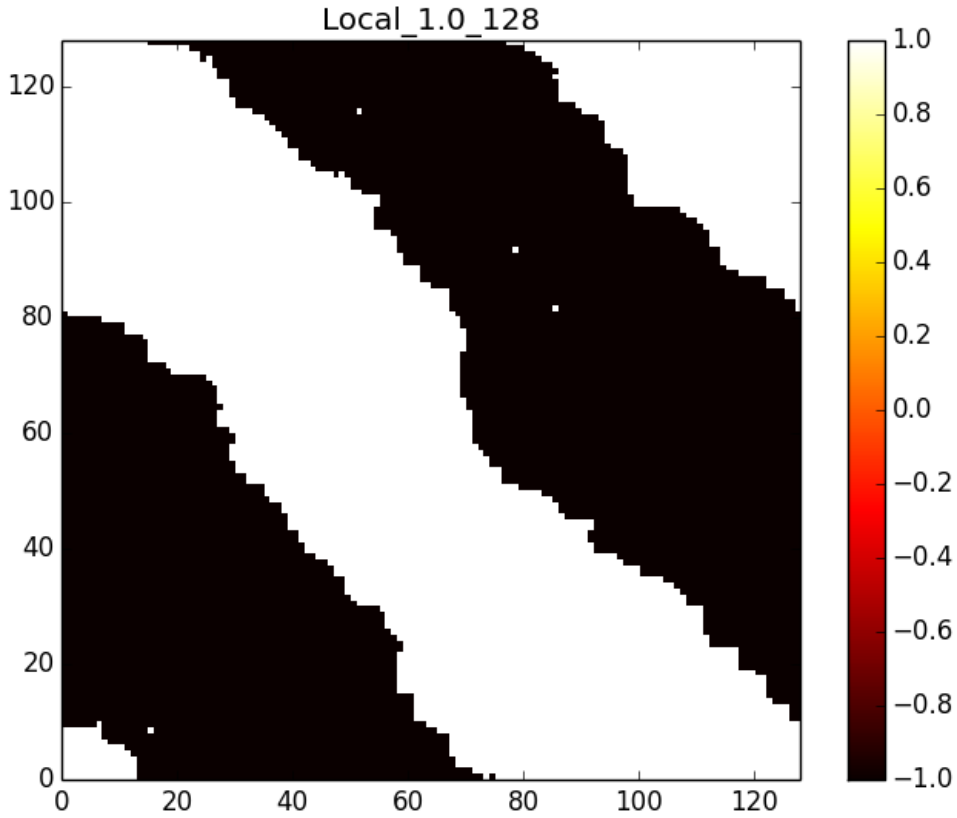




At low temperature, **T = 1.0**,the code was run for **L=128**. Again, 5 times 10N, 100N and 1000N. We do observe that the system becomes homogeneous (mostly white or mostly black) on the available time scales, though running for longer times might make it more obvious.


Local_1.0_128
0
20
40
60
80
100
120
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0


Local_1.0_128
0
20
40
60
80
100
120
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0



Evaluation/feedback on the above work

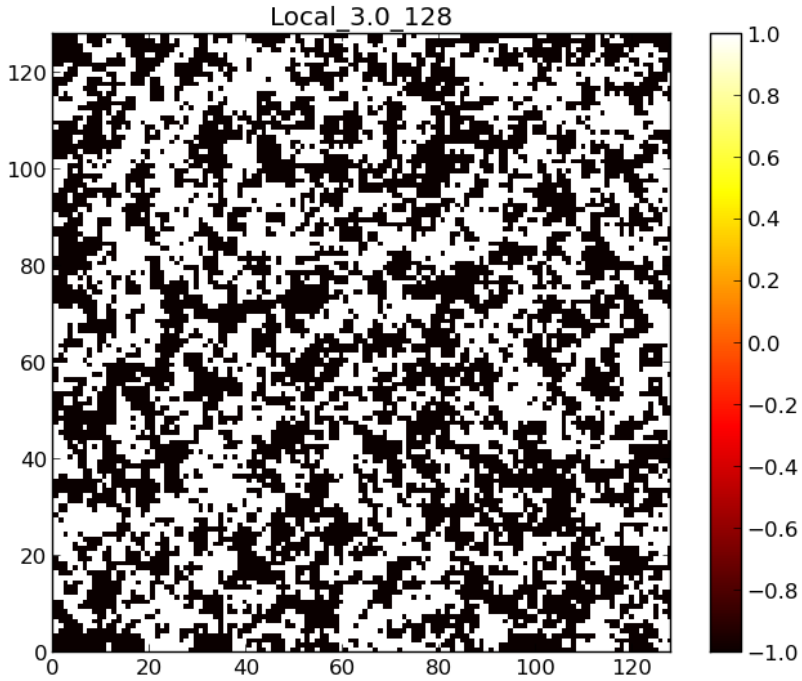
Note: this section can only be filled out during the evaluation phase.

A2 Here, you should check your fellow student's correct analysis of the Ising model at high and at low temperature, and at the critical point. In essence, it should come out that

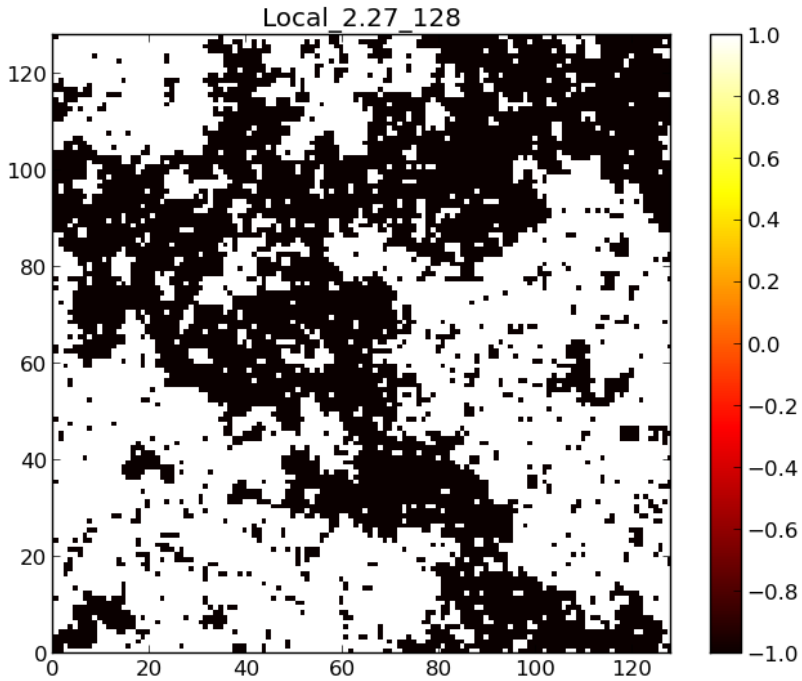
1. At T=3, there is some fine-graned structure (in physical terms, the systems correlation length is smaller that the system size).
2. At T = 2.27, there is large-scale structure, as discussed in the lecture (correlation length \sim system size).
3. At T = 1, for L = 32, the system naturally relaxes into a state with domain walls, that disappear very slowly
4. At T = 1, L = 32, your fellow student may have observed flips from the all-up to the all-down configuration, but this is not required (as it depends on chance).
5. In contrast, at T = 1, L= 128, it is normally not possible to see the transition to a state without domain walls (again it is possible that an all up or an all down configuration was observed by accident).

The following answer would have obtained full score:

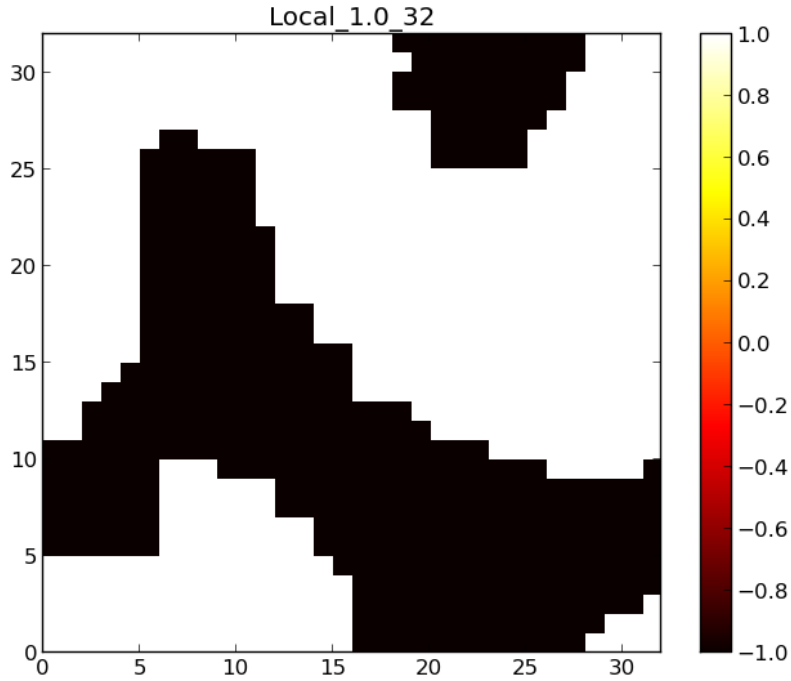
At T =3 L = 128, I observe a configuration with small-scale structure (short correlation length), as shown here (after 2000 N iterations)



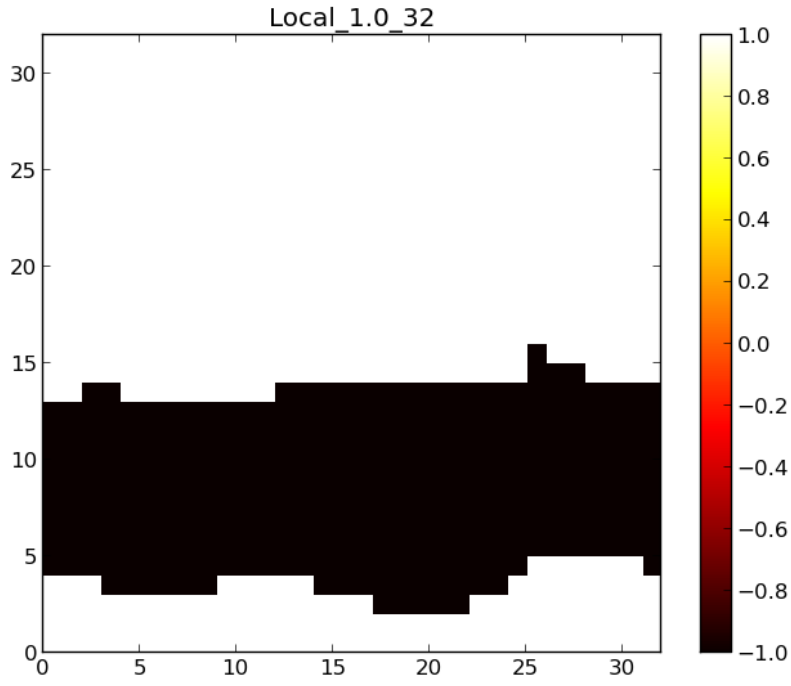
At T = 2.27 L = 128, I observe observe large scale structures, as shown here (after 120000 N iterations)



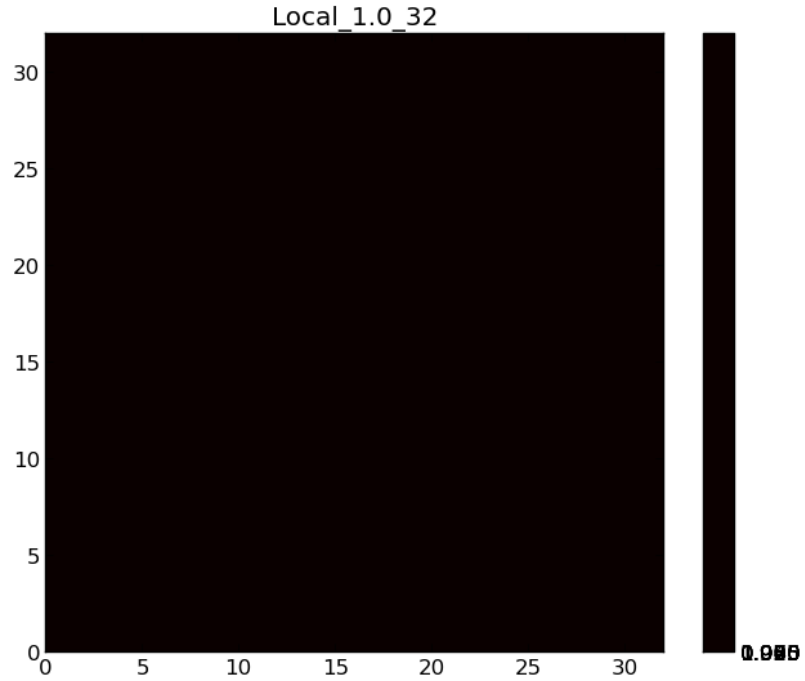
Here , at T = 1 L = 32, I observe domain walls after short time (30 iterations)



This is a configuration observed after 330 iterations (domains get straight):

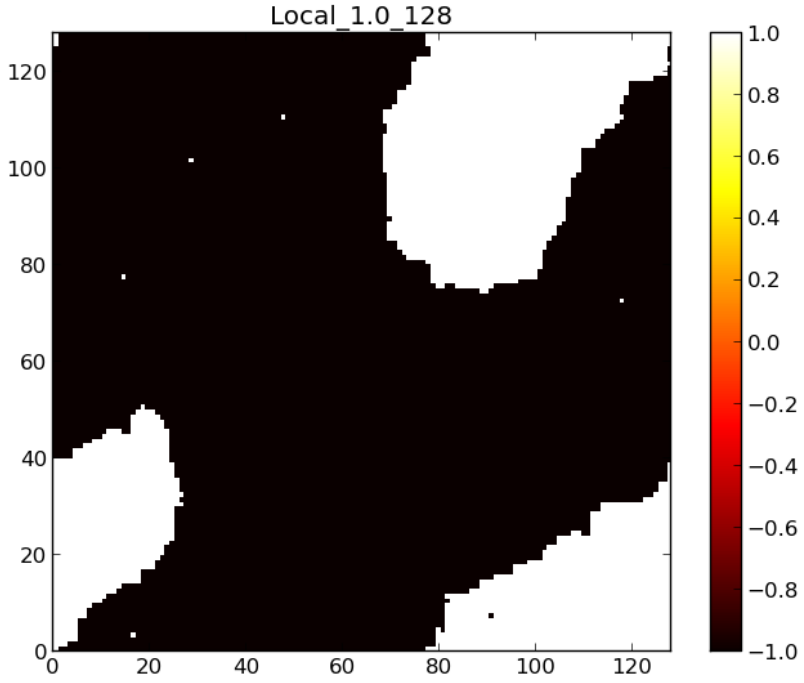


much later at 20,000 iterations, I obtained the following configuration that is homogeneous



I have observed the flipping of the entire spin configuration back to the all + (white) after 40,000 iterations

Here is a configuration obtained after a long simulation at T=1, L= 128. The domains do not go away on the timescale of the simulation. I would have to run even longer.



POINTS - DESCRIPTION

Give between 0 and 5 points according to how many of the below points were treated adequately.

1. At T=3, L= 128: fine-graned structure (in physical terms, the systems correlation length is smaller that the system size)
2. At T = 2.27,L= 2.27: large-scale structure, as discussed in the lecture
3. At T = 1, for L = 32: domain walls that disappear very slowly
4. At T = 1, L = 32, Observation of flips from the all-up to the all-down configuration normally not observed. Any answer on this question is acceptable.

5. T = 1 I = 128, it is normally not possible to see the transition to a state without domain walls (again it is possible that an all up or an all down configuration was observed by accident)

Score from your peers: **5**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → There wasn't enough commentary to get full marks on this.

peer 3 → *[This area was left blank by the evaluator.]*

B In this section, you run the Wolff cluster algorithm, one of the great (but little understood) algorithms of statistical physics.

B1 Download (cut-and-paste) the Wolff cluster Monte Carlo algorithm for the Ising model shown below. Familiarize yourself with how it works before going on. **Explain in a few words** what the **Pocket** stands for.

```
import random, math

def energy(S, N, nbr):
    E = 0.0
    for k in range(N):
        E -=  S[k] * sum(S[nn] for nn in nbr[k])
    return 0.5 * E

L = 6
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N)
                                    for i in range(N)}
T = 2.0
p  = 1.0 - math.exp(-2.0 / T)
nsteps = 10000
S = [random.choice([1, -1]) for k in range(N)]
E = [energy(S, N, nbr)]
for step in range(nsteps):
    k = random.randint(0, N - 1)
    Pocket, Cluster = [k], [k]
    while Pocket != []:
        j = random.choice(Pocket)
        for l in nbr[j]:
            if S[l] == S[j] and l not in Cluster \
                   and random.uniform(0.0, 1.0) < p:
                Pocket.append(l)
                Cluster.append(l)
        Pocket.remove(j)
    for j in Cluster:
        S[j] *= -1
    E.append(energy(S, N, nbr))
print sum(E)/ len(E) / N
```

Greatly increase the number of iterations of the algorithm. Run it for **L=6 and T = 2.0**. Check that you recover the exact value for the mean energy **E/N(T=2, L=6) = -1.747** (known from exact enumeration). **Communicate the results obtained** in four independent runs of the algorithms. Do not forget to **indicate your value of nsteps**.
Next, answer the following question: is it necessary to **pick** a **random element j** of the pocket (element that we eliminate from the pocket through **Pocket.remove(j)**), or can we simply pop the last element of Pocket (j = Pocket.pop())?
Modify then run the modified program (note that it has one less line, the Pocket.remove() line is no longer needed), and **explain your observations**.

Pocket comprises of those sites of the clusters whose neighbors have not already been scrutinized.

Below are the results of the run for several nsteps:

nsteps	**E/N**
100	-1.66336633663

1000	-1.74991674992
10000	-1.74723638747
100000	-1.74778807767
1000000	-1.74781902996

We recover the exact value for the mean energy **E/N(T=2, L=6) = -1.747** known from exact enumeration.

It's not important to pick a random element of the pocket to remove. We can just pop the last element. Below are results with the modification in code.

nsteps	**E/N**
100	-1.77667766777
1000	-1.75946275946
10000	-1.75154706752
100000	-1.74790918757
1000000	-1.7467214755

For this modified code, we also recover the exact value for the mean energy **E/N(T=2, L=6) = -1.747** known from exact enumeration.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you should check that your fellow student has some understanding of the cluster algorithm, and that he/she can validate it carefully against known results.

The following answer would have obtained full score:

Pocket is the set (or list) of active sites of the cluster. [Not required): It avoids checking several times an edge from a cluster site to a site outside the cluster]
For 1,000,000 iterations, I obtained:
-1.74856958749
-1.73630414736
-1.74561432746
-1.74388116744
All these values are very close to the exact value -1.747, so I suppose the program is OK.
It is not necessary to pick a random element from the pocket. The last one will do, as it is just necessary to check all of them them modified program is statistically indistinguishable from the one provided..

POINTS - DESCRIPTION
1 point for adequate answer about the pocket
1 point if agreement between MC and exact results treated successfully
1 point if it is understood that the random pocket element can be replaced by pop()

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

B2 As discussed in lecture 8, the **Ising model was solved exactly by Onsager**, in 1944, for the infinite lattice. The exact solution for the finite lattice was obtained by **Kaufman in 1949** . Based on her (sic!) magnificent paper, **Ferdinand and Fisher, in 1969**, computed the specific heat for finite lattices with periodic boundary conditions, exactly the system that we consider, and the result is shown here:

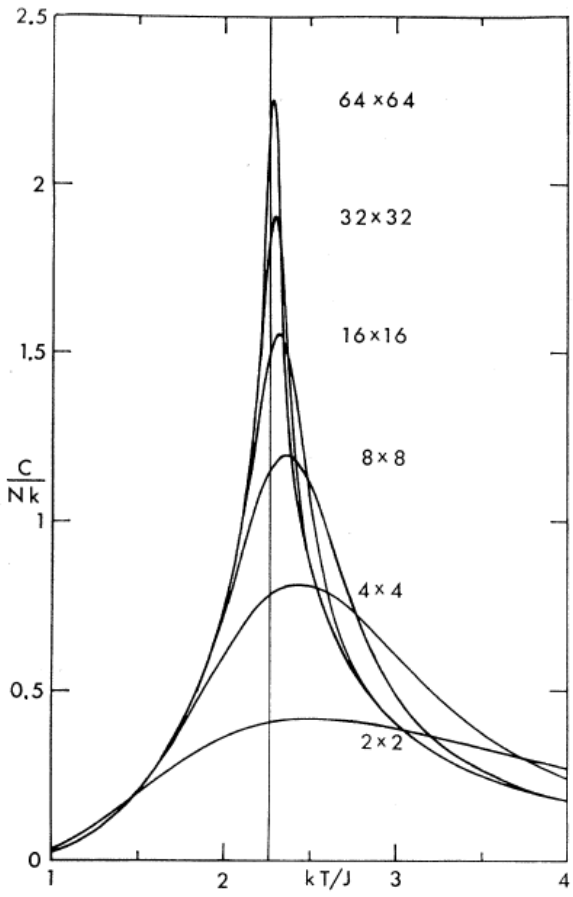


FIG. 1. The specific heat per spin for small Ising lattices; exact results for the $m \times n$ square lattice with periodic boundary conditions are displayed for $m=n=2, 4, 8, 16, 32$, and 64 ($N=mn$). The limiting critical point is marked by a vertical line.

Now, incorporate into the cluster algorithm the read-in, write-out part of section **A2**, which allows you to do a 'warm start' (that is, discard some initial data). Run this modified cluster algorithm for lattices of size L=2, 4, 8, 16, 32 and check Ferdinand and Fisher's analytical results. To do so, **implement a few lines analogous to:**

```
E_mean = sum(E)/ len(E)
E2_mean = sum(a ** 2 for a in E) / len(E)
cv = (E2_mean - E_mean ** 2 ) / N / T ** 2
```

(**Remember the definition** of the specific heat from lecture 8). **Communicate your results** (without error bars, just quote the results of one or two runs, after a warm start, for each value of the parameters) at the critical temperature T = 2.27. NB: we just need rough agreement.

Can you confirm that, as indicated in the conclusion of lecture 8, observables as the specific heat for the

Ising model in the transition region strongly depend on lattice size?

nsteps = L * L * 1000 (Ran 5 times for each lattice size). For L=32, the simulations took really long!

L	Cv
2	0.400705319138
4	0.782992941952
8	1.20310676891
16	1.62097477168
32	1.83010912881

Clearly, the specific heat for the ising model in the transition region (Tc = 2.27) depends strongly on the lattice size. We also see a rough agreement with the figure shown in the problem description.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you check your fellow student's ability to check literature data, without getting lost in detail.

The following answer would have obtained full score:

At T = 2.27, I ran the following lattice sizes:
L = 2: cv \sim 0.39
L = 4: cv \sim 0.78
L = 8: cv \sim 1.16
L =16: cv \sim 1.55
L = 32 cv \sim 1.80

Yes, I can confirm that there is a lot of variation of cv with N around the critical temperature, whereas for small or large T, dependence on system size is less noticeable.

POINTS - DESCRIPTION
1 point if numerical comparison ok, but incomplete
1 additional point if complete comparison
1 additional point if the evident observation is made that the specific heat per spin is strongly lattice-size dependent.

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

C Here, you run the heatbath algorithm.

C1 Download (cut-and-paste) the heatbath algorithm for the Ising model shown below. Familiarize yourself with how it works before going on.

```
import random, math

L = 6
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
                                    for i in range(N)}
nsteps = 10 * N
T = 2
beta = 1.0 / T
S = [random.choice([-1, 1]) for site in range(N)]
E = -0.5 * sum(S[k] * sum(S[nn] for nn in nbr[k]) \
                                for k in range(N))
Energies = []
for step in range(nsteps):
    k = random.randint(0, N - 1)
    Upsilon = random.uniform(0.0, 1.0)
    h = sum(S[nn] for nn in nbr[k])
    Sk_old = S[k]
    S[k] = -1
    if Upsilon < 1.0 / (1.0 + math.exp(-2.0 * beta * h)):
        S[k] = 1
    if S[k] != Sk_old:
        E -= 2.0 * h * S[k]
    Energies.append(E)
print sum(Energies) / len(Energies) / N
```

Greatly increase the number of iterations of the algorithm. Run it for L=6 and for T = 2.0. Check that you recover the exact value for the mean energy E/N(T=2, L=6) = -1.747 (known from exact enumeration). **Communicate the results obtained** in four independent runs of the algorithm. **Indicate your value of nsteps**.

nsteps = L * L * X

X	E/N
100	-1.6587345679
1000	-1.75270987654
10000	-1.74858055556
100000	-1.74817240741
1000000	-1.74672147556

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

In this section, you should evaluate whether your fellow student can perform a rough

validation of the heatbath program.
There are two issues:
1/ the value of nsteps should be given (nsteps = N*1000 or N*10000 or N*100000 or N*1000000 are OK)
2/ Four results of runs should be given (but it is also OK, if the mean and the standard deviation or mean and error are given)

The following answer would have obtained full score:

For 1000000 iterations:. in four runs, I obtained
-1.75073522222
-1.74398488889
-1.742884
-1.739473
All these values are very close to -1.747, the exact result. I suppose therefore that the program is OK.

POINTS - DESCRIPTION
0 points - question not treated
1 point - question treated but results not conclusive or nsteps not indicated
2 points - program was run four times, conclusions clear, value of nsteps given

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

C2 Modify the heatbath algorithm so that it does TWO computations in parallel, as discussed in Tutorial 8: one starting from the all plus spin configuration, and one from the all minus configuration. **For simplicity, we provide this program**, you are free to download (cut-and-paste), run and modify it, but familiarize yourself thoroughly with this program.

```
import random, math

L = 32
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
                                    for i in range(N)}
T = 2.3
beta = 1.0 / T

t_coup = []
for iter in range(10):
    print iter
    S0 = [1] * N
    S1 = [-1] * N
    step = 0
    while True:
        step += 1
        k = random.randint(0, N - 1)
        Upsilon = random.uniform(0.0, 1.0)
        h = sum(S0[nn] for nn in nbr[k])
        S0[k] = -1
        if Upsilon < 1.0 / (1.0 + math.exp(-2.0 * beta * h)):
            S0[k] = 1
        h = sum(S1[nn] for nn in nbr[k])
        S1[k] = -1
        if Upsilon < 1.0 / (1.0 + math.exp(-2.0 * beta * h)):
            S1[k] = 1
        if step % N == 0:
            n_diff = sum(abs(S0[i] - S1[i]) for i in range(N))
            if n_diff == 0:
                t_coup.append(step / N)
                break
print t_coup
print sum(t_coup) / len(t_coup)
```

Compute the time (in "**sweeps**", that is in number of steps / N) at which the algorithm couples, that is, at which the difference between the configurations generated falls to zero. Plot this coupling time (averaged over 10 runs with different random numbers) for L = 32 at temperature T= 5.0, 4.0, 3.0, 2.5, 2.4, 2.3,**Attention**: run T=2.3 only if you have a lot of CPU time available. **Describe** what you see. In the Ising model, the coupling time is, up to a logarithmic factor in N, equal to the correlation time, so this coupling time informs you of the order of magnitude of the correlation time (as discussed in Tutorial 1).

Ferdinand and Fisher, in a footnote of their 1969 paper that we discussed earlier, compared with their exact calculations with the earliest Monte Carlo calculations on the Ising model. Here is the footnote:

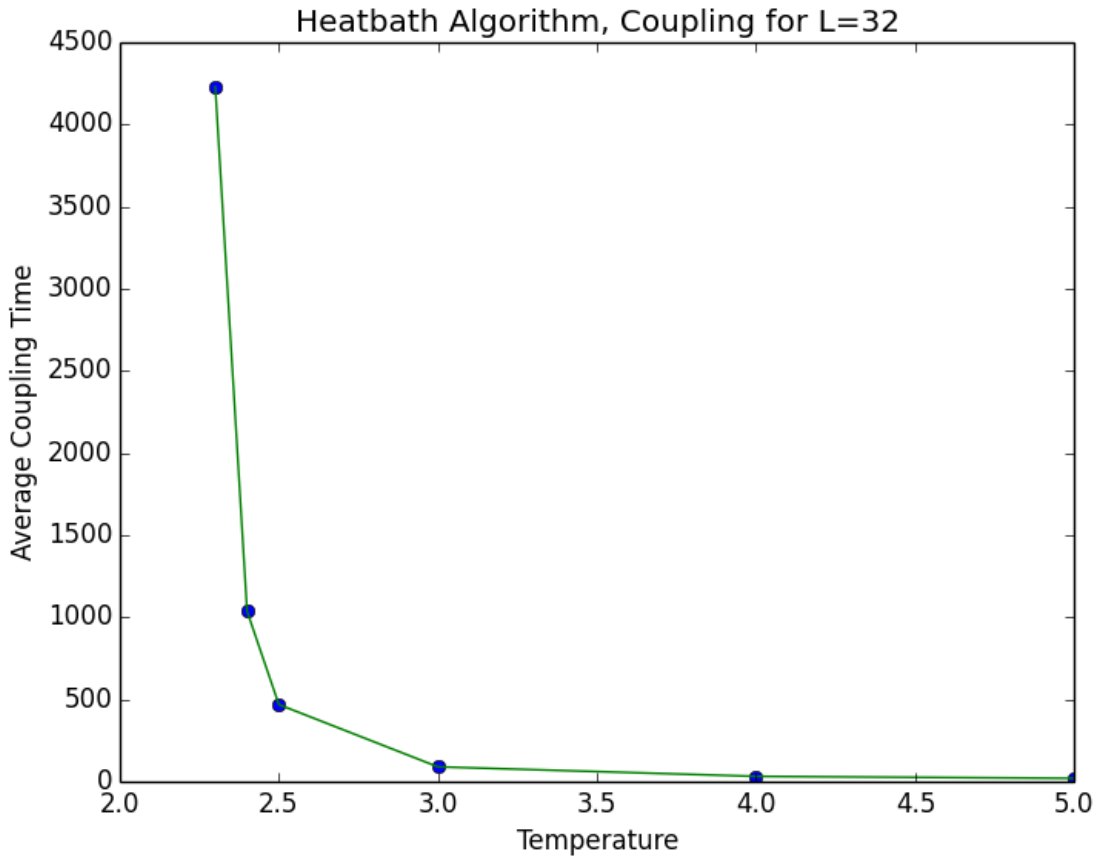
[20] We have also compared the exact results for the specific heats of finite tori of sizes 4×4, 8×8, and 12×12 with the interesting Monte Carlo calculations of C. P. Yang, Proc. Symp. Appl. Math. **15**, 351 (1963) [American Mathematical Society]. Away from the maximum the Monte Carlo results are accurate to within their standard deviation of 2 or 3%. In the vicinity of the maximum, however, errors or order 10–15% occur in all three cases. These errors are about 7–12 times the standard deviations and have a somewhat systematic appearance. Yang makes some pertinent comments on the difficulties of the Monte Carlo calculations in the critical region but probably one would still not have anticipated errors as large as our comparisons revealed. [Graphs showing the exact and Monte Carlo results are given by A. E. Ferdinand, Ph.D. thesis, published by the University of London (1967).]

NB: "Torus" is the same as "lattice with periodic boundary conditions".

In the light of your exact bounds on the correlation time, can you **explain the insufficient agreement between theory and numerics reported by Ferdinand and Fisher**? (Remember the lesson of homework 1: underestimating the correlation time makes one underestimate the error).

T	Coupling Time
5.0	21
4.0	32
3.0	90
2.5	469
2.4	1037
2.3	4230

The plot is below:



The coupling time grows exponentially when the temperature becomes close to the critical temperature. As explained in the problem description, this also implies that the correlation time increases exponentially close to the critical temperature. In the Ferdinand and Fischer study, the correlation times are underestimated, which leads to underestimation of the error causing the insufficient agreement between theory and numerics.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you simply check your fellow students ability to understand the coupling that was treated in Tutorial 8.

The following answer would have obtained full score:
T | t_coup in number of sweeps
5 | 21
4 | 30
3 |104
2.5| 431
2.4| 969
2.3| 5047 (not required)

The coupling time, and therefore the correlation time increase **very rapidly** on the approach of T_critical.
The error calculation in C. P. Young's paper was certainly wrong, as the very long correlation times were certainly not taken into account. But, then again, it deserves credit for being the very first ISING-MODEL simulation paper.

POINTS - DESCRIPTION

1 point - incomplete numerical calculation of t_coup, or results presented confusingly
2 points - numerical results convincing. Clear understanding that coupling time increases rapidly
3 points - all of the above + understanding that in the early paper, long correlation times were not taken into account.

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

The algorithm in C2 couples, and its coupling time provides a rigorous limit on the convergence time. Nevertheless, the configuration obtained is not really a random configuration, simply because the coupling time itself has fluctuations.

For your information and enjoyment, we provide here a rudimentary yet exact algorithm following the "coupling from the past principle" that, rather than simulating from time t=0 to a coupling time t_coup:

```
import random, math

L = 16
N = L * L
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) % N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
                                    for i in range(N)}
nsteps = 10
T = 3.0
beta = 1.0 / T
S0 = [1] * N
S1 = [-1] * N
k = {}
Upsilon = {}
n_diff = 10
while n_diff != 0:
    nsteps *= 2
    for step in range(-nsteps, 0):
        if step not in k:
            k[step] = random.randint(0, N - 1)
            Upsilon[step] = random.uniform(0.0, 1.0)
        h = sum(S0[nn] for nn in nbr[k[step]])
        S0[k[step]] = -1
        if Upsilon[step] < 1.0 / (1.0 + math.exp(-2.0 * beta * h)):
            S0[k[step]] = 1
        h = sum(S1[nn] for nn in nbr[k[step]])
        S1[k[step]] = -1
        if Upsilon[step] < 1.0 / (1.0 + math.exp(-2.0 * beta * h)):
            S1[k[step]] = 1
    n_diff = 0
    for i in range(N):
        if S0[i] != S1[i]: n_diff += 1
    print nsteps, n_diff
```

The configuration at t=0 is a perfect sample, following the coupling from the past idea of Propp and Wilson (1996). You can see that an infinite simulation, from t = -infty up to t = 0, would output the sample at t=0.
NB: This part is for your personal enchantment, no points given, no questions asked.

Nothing to answer!

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*