Computational Physics Project 4
Studies of phase transitions in magnetic systems

Marius B. Pran,[1] Espen Hodne,[1] and Marc K. Pestana[1]
[1] *Institute of Theoretical Astrophysics, University of Oslo*

The goal of my project is to study Ising model in two dimensions to simulate phase transitions. In this program, the values represented two spin states as pointing up or down as the model for our system, a so-called binary system where the objects at each lattice site can only take two values which were 0 and 1. To this end, I implemented the model by developing a C++ program. I found that at a given critical temperature, my model exhibited a phase transition from a magnetic phase (a system with a finite magnetic moment) to a phase with zero magnetization.

INTRODUCTION.

The Ising model has been extremely popular, with applications spanning from studies of phase transitions to simulations in statistics. In one and two dimensions its has analytical solutions to several expectation values and it gives a qualitatively good underatanding of several types of phase transitions.

In its simplest form the energy of the Ising model is expressed as, without an externally applied magnetic field,

$$E = -J \sum_{<kl>}^{N} s_k s_l$$

with $s_k = \pm 1$. The quantity N represents the total number of spins and J is a coupling constant expressing the strength of the interaction between neighboring spins. The symbol $< kl >$ indicates that we sum over nearest neighbors only. We assumed that we had a ferromagnetic ordering, viz $J > 0$. We used periodic boundary conditions and the Metropolis algorithm only.

THEORETICAL MODELS

The material on the Ising model is summerized as follows The model I employed in my studies of phase transitions at finite temperature for magnetic systems is the so-called Ising model. In its simplest form the energy is expressed as

$$E = -J \sum_{<kl>}^{N} s_l s_k - B \sum_{k}^{N} s_k$$

with $s_k = 1$, N is the total number of spins, J is a coupling constant expressing the strength of the interaction between neighboring spins and B is an external magnetic field interacting with the magnetic moment set up by the spins. The symbol $< kl >$ indicates that we sum over nearest neighbors only. Notice that for $J > 0$ 0 it is energetically favorable for neighboring spins to be aligned. This is consistent the tendency for canonical systems is to "seek" the lowest energy state, it is energetically favorable for neighboring spins to be aligned, since alignment means that $s_k s_l > 0$. This plus $J > 0 \Rightarrow E < 0$ and thus a lower energy state. At low enough temperatures, this feature leads to a cooperative phenomenon called spontaneous magnetization. Thus, a given magnetic moment can influence the alignment of spins that are separated from the given spin by a macroscopic distance by propagation through interactions between nearest neighbors. These long range correlations between spins are associated with a long-range order in which the lattice has a net magnetization in the absence of a magnetic field.

Boltzmann distribution In order to calculate expectation values such as the mean energy hE_i or magnetization hM_i in statistical physics at a given temperature, I used a probability distribution or Boltzmann distribution

$$P_i(\beta) = \frac{e^{-\beta E_i}}{Z}$$

with $\beta = 1/kT$ being the inverse temperature, k the Boltzmann constant, E_i is the energy of a state i while Z is the partition function for the canonical ensemble defined as

$$Z = \sum_{i=1}^{M} e^{-\beta E_i}$$

where the sum extends over all microstates M. Pi expresses the probability of finding the system in a given configuration i. Energy for a specific configuration The energy for a specific configuration i is given by Ei = ?J X N ¡kl¿ sksl . Configurations To better understand what is meant with a configuration, consider first the case of the one-dimensional Ising model with B = 0. In general, a given configuration of N spins in one dimension may look like ? ? ? . . . ? ? ? . . . ? ? 1 2 3 . . . i ? 1 i i + 1 . . . N ? 1 N In order to illustrate these features let us further specialize to just two spins. With two spins, since each spin takes two values only, we have 2 2 = 4 possible arrangements of the two spins. These four possibilities are 5 1 =?? 2 =?? 3 =?? 4 =?? Boundary conditions, free ends What is the energy of each of these configurations? For small systems, the way we treat the ends matters.

Two cases are often used. In the first case we employ what is called free ends. This means that there is no contribution from points to the right or left of the endpoints. For the one-dimensional case, the energy is then written as a sum over a single index Ei = ?J N X?1 j=1 sj sj+1, Free ends and the energy If we label the first spin as s1 and the second as s2 we obtain the following expression for the energy E = ?Js1s2. The calculation of the energy for the one-dimensional lattice with free ends for one specific spin-configuration can easily be implemented in the following lines for (j=1; j ¡ N; j++) energy += spin[j]*spin[j+1]; where the vector spin[] contains the spin value sk = 1. Free ends and energy For the specific state E1, we have chosen all spins up. The energy of this configuration becomes then E1 = E?? = ?J. The other configurations give E2 = E?? = +J, E3 = E?? = +J, and E4 = E?? = ?J. 6 Periodic boundary conditions We can also choose so-called periodic boundary conditions. This means that the neighbour to the right of sN is assumed to take the value of s1. Similarly, the neighbour to the left of s1 takes the value sN . In this case the energy for the one-dimensional lattice reads Ei = ?J X N j=1 sj sj+1, and we obtain the following expression for the two-spin case E = ?J(s1s2 + s2s1). Energy with PBC In this case the energy for E1 is different, we obtain namely E1 = E?? = ?2J. The other cases do also differ and we have E2 = E?? = +2J, E3 = E?? = +2J, and E4 = E?? = ?2J

ALGORITHMS

The Metropolis algorithm is summarized as follows. BLAH BLAH BLAH

CASE STUDIES AND THEIR METHODS

Case 1, project 4a): For case 1, I studied a simple 2×2 *lattice, analytical expressions.* I assumed that we had only two spins in each dimension, that is $L = 2$. I found the analytical expression for the partition function and the corresponding expectations values for the energy E, the mean absolute value of the magnetic moment $|M|$ (in this report refered to this as the mean magnetization), the specific heat C_V and the susceptibility χ as functions of T using periodic boundary conditions. I used these results results as benchmark calculations for my next steps.

Case 2, project 4b): For case 2, a C++ program implementing the Ising model. I wrote a C++ program for the Ising model which computes the mean energy E, mean magnetization $|M|$, the specific heat C_V and the susceptibility χ as functions of T using periodic boundary conditions for $L = 2$ in the x and y directions. I compared my results with the expressions from a) for a temperature $T = 1.0$ (in units of kT/J).

I determined the number of Monte Carlo cycles do you need in order to achieve a good agreement.

Case 3, project 4c): For case 3, determine when the most likely state reached? I choose now a square lattice with $L = 20$ spins in the x and y directions.

In the previous exercise I did not study carefully how many Monte Carlo cycles were needed in order to reach the most likely state. Here I performed a study of the time (here it corresponds to the number of Monte Carlo sweeps of the lattice) needed before my model reached an equilibrium condition that allowed me to start computing various expectations values. My first iteration was a rough and plain graphical one, where I plotted various expectations values as functions of the number of Monte Carlo cycles.

I choose the first temperature of $T = 1.0$ (in units of kT/J) and studied the mean energy and magnetization (absolute value) as functions of the number of Monte Carlo cycles. I let the number of Monte Carlo cycles (sweeps per lattice) represent time. I used both an ordered (all spins pointing in one direction) and a random spin orientation as starting configuration. My program reported the number of Monte Carlo cycles do you need before you reach an equilibrium situation? I repeated this analysis for $T = 2.4$. I estimated, based on these values, an equilibration time, and made a plot of the total number of accepted configurations as function of the total number of Monte Carlo cycles that showed how the number of accepted configurations behaved as function of temperature T?

Project 4d): For case 4, I analyzed the probability distribution. I computed the probability $P(E)$ for the previous system with $L = 20$ and the same temperatures, that is at $T = 1.0$ and $T = 2.4$. I computed this probability by simply counting the number of times a given energy appears in my computation. I started the computation after the steady state situation has been reached. Then, I compared my results with the computed variance in energy σ_E^2 and compared (below in the results section) the behavior I observed.

Studies of phase transitions. Near T_C we can characterize the behavior of many physical quantities by a power law behavior. As an example, for the Ising class of models, the mean magnetization is given by

$$\langle M(T) \rangle \sim (T - T_C)^{\beta} ,$$

where $\beta = 1/8$ is a so-called critical exponent. A similar relation applies to the heat capacity

$$C_V(T) \sim |T_C - T|^{\alpha} ,$$

and the susceptibility

$$\chi(T) \sim |T_C - T|^{\gamma} , \tag{1}$$

with $\alpha = 0$ and $\gamma = 7/4$. Another important quantity is the correlation length, which is expected to be of the

order of the lattice spacing for $T >> T_C$. Because the spins become more and more correlated as T approaches T_C, the correlation length increases as we get closer to the critical temperature. The divergent behavior of ξ near T_C is

$$\xi(T) \sim |T_C - T|^{-\nu} . \tag{2}$$

A second-order phase transition is characterized by a correlation length which spans the whole system. Since we are always limited to a finite lattice, ξ will be proportional with the size of the lattice. Through so-called finite size scaling relations it is possible to relate the behavior at finite lattices with the results for an infinitely large lattice. The critical temperature scales then as

$$T_C(L) - T_C(L = \infty) = aL^{-1/\nu}, \tag{3}$$

with a a constant and ν defined in Eq. (**??**). We set $T = T_C$ and obtain a mean magnetisation

$$\langle \mathcal{M}(T) \rangle \sim (T - T_C)^{\beta} \rightarrow L^{-\beta/\nu}, \tag{4}$$

a heat capacity

$$C_V(T) \sim |T_C - T|^{-\gamma} \rightarrow L^{\alpha/\nu}, \tag{5}$$

and susceptibility

$$\chi(T) \sim |T_C - T|^{-\alpha} \rightarrow L^{\gamma/\nu}. \tag{6}$$

Project 4e): Case 5, Numerical studies of phase transitions. I studied the behavior of the Ising model in two dimensions close to the critical temperature as a function of the lattice size $L \times L$. I calculated the expectation values for $\langle E \rangle$ and $\langle |M| \rangle$, the specific heat C_V and the susceptibility χ as functions of T for $L = 40$, $L = 60$, $L = 80$ and $L = 100$ for $T \in [2.0, 2.3]$ with a step in temperature $\Delta T = 0.05$ or smaller. I found it convenient to narrow the domain for T to blah blah blah.

I plotted $\langle E \rangle$, $\langle |M| \rangle$, C_V and χ as functions of T. blah blah blah an indication of a phase transition? I used the absolute value $\langle |M| \rangle$ when you evaluate χ. For these production runs I parallelized the code using OpenMP can be used. I use optimization flags, blah blah blah, when compiling. I performed a timing analysis of some selected runs in order to see that you optimized the speedup using the parallelized code.

Project 4f): Case 6, Extracting the critical temperature. I used Eq. (**??**) and the exact result $\nu = 1$ in order to estimate T_C in the thermodynamic limit $L \rightarrow \infty$ using my simulations with $L = 40$, $L = 60$, $L = 100$ and $L = 140$ The exact result for the critical temperature (after Lars Onsager) is $kT_C/J = 2/ln(1 + \sqrt{2}) \approx 2.269$ with $\nu = 1$.

RESULTS

Case 1, project 4a): For case 1, benchmark calculations for my next steps.

Case 2, project 4b): For case 2, comparing the C++ code implementing the Ising mode with the analytical results for case 1. The number cycles to reach good agreement is blah blah blah.

Case 4, project 4c): For case 4, expectations values as functions of the number of Monte Carlo cycles. blah blah blah PLOT blah blah blah The number of Monte Carlo cycles needed before an equilibrium was reached blah blah blah equilibrium time blah blah blah PLOT blah blah blah figure X showed how the number of accepted configurations behaved as function of temperature T

Project 4d): For case 4, probability distribution.

Background literature

If you wish to read more about the Ising model and statistical physics here are three suggestions.

- M. Plischke and B. Bergersen, *Equilibrium Statistical Physics*, World Scientific, see chapters 5 and 6.
- D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge, see chapters 2,3 and 4.
- M. E. J. Newman and T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford, see chapters 3 and 4.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.

- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Use Devilry to hand in your projects, log in at http://devilry.ifi.uio.no with your normal UiO username and password and choose either 'fys3150' or 'fys4150'. There you can load up the files within the deadline.
- Upload **only** the report file! For the source code file(s) you have developed please provide us with your link to your github domain. The report file should include all of your discussions and a list of the codes you have developed. Do not include library files which are available at the course homepage, unless you have made specific changes to them.
- In your git repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parametxers.
- In this and all later projects, you should include tests (for example unit tests) of your code(s).
- Comments from us on your projects, approval or not, corrections to be made etc can be found under your Devilry domain and are only visible to you and the teachers of the course.

Finally, we encourage you to work two and two together. Optimal working groups consist of 2-3 students. For this specific report you need to hand in an individual report.

How to install openmpi and/or OpenMP on your PC/laptop

If you use your own laptop, for linux/ubuntu users, you need to install two packages (alternatively use the synaptic package manager)

```
sudo apt-get install libopenmpi-dev
sudo apt-get install openmpi-bin
```

For OS X users, install brew (after having installed xcode and gcc, needed for the gfortran compiler of openmpi) and then run

```
brew install open-mpi
```

When compiling from the command line, depending on your choice of programming language you need to compile and link as for example

```
mpic++ -O3 -o <executable> <programname.cpp>
```

if you use c++ (alternatively mpicxx) and

```
mpif90 -O3 -o <executable> <programname.f90>
```

if you use Fortran90.

When running an executable, run as

```
mpirun -n 10 ./<executable>
```

where -n indicates the number of processes, 10 here.

With openmpi installed, when using Qt, add to your .pro file the instructions at Svenn-Arne Dragly's site

You may need to tell Qt where opempi is stored.

For the machines at the computer lab, openmpi is located at /usr/lib64/openmpi/bin Add to your .bashrc file the following

```
export PATH=/usr/lib64/openmpi/bin:$PATH
```

For Windows users we recommend to follow the instructions at the Open MPI site.

If you use OpenMP, for linux users, compile and link with for example

```
c++ -O3 -fopenmp -o <executable>  <programname.cpp>
```

For OS X users, you need to install clang-omp using brew, that is

```
brew install clang-omp
```

and then compile and link with for example

```
clang-omp++ -O3 -fopenmp -o <executable>  <programname.cpp>
```

If you program in Fortran and use **gfortran**, compile as for example

```
gfortran -O3 -fopenmp -o <executable>  <programname.f90>
```

If you have access to Intel's **ifort** compiler, compile as

```
ifort -O3 -fopenmp -o <executable>  <programname.f90>
```