UMEÅ UNIVERSITY
Department of Physics
Peter Olsson

April 7, 2022
Monte Carlo, 7.5hp

Two Monte Carlo methods for the Ising model

1 Monte Carlo programs for the 2D Ising model

The task is to write two programs to do Monte Carlo on the 2D Ising model. You should implement both the simpler single spin Metropolis algorithm and the more complicated Wolff cluster algorithm.

1.1 Get source files

Use the command below to extract some files for the Ising lab in three different directories.

```
tar xzf /home/peol0002/MonteCarlo/ising.tgz
```

If you are at not at the Linux system at the physics department you will first have to download `ising.tgz` from the link at the web page `http://www.tp.umu.se/mc`. Then extract its content in much the same way:

```
tar xzf ising.tgz
```

If everything is correct you will now have three new directories: `src`, `Metro`, and `Cluster` which each contains some files. Some source files are in `src` and the `Makefiles` in `Cluster` and `Metro` are set up to compile the cluster version and the Metropolis version, respectively.

1.2 Metropolis Monte Carlo

Several essential parts are missing in `src/ising.c`. You will therefore have to go through the following steps to get a working program:

1. Start by looking at `ising.h` and the `main` function in `ising.c`.

   The program is meant to be run with several temperatures in a sequence, e.g. with `./ising L=16 nblock=16 T=2.2 run T=2.22 run T=2.24 run`. Try to understand how how `main` and `read_args` work. Also examine the functions `initialize_mc` and `mc`. Check the effect of `ntherm`, `nblock`, and `nsamp` in the code.

   The comments like "Fix this (2)" should be taken care of in the suggested order. The numbers 2–5 refer to the steps in this list.

2. Write a routine `update` that performs a Monte Carlo sweep over the system and returns the number of accepted changes.

   Note that Chapter 7 in the lecture notes, "Technical considerations", have suggestions regarding the implementation of both 2D arrays and periodic boundary conditions.

   - For 2D arrays I prefer "Indexing yourself" and that is also consistent with the functions in `config.c`. You are however free to implement the 2D arrays in a different way.
   - Boundary conditions: If you choose the second alternative in Sec. 7.2, you also have to include a test in the program that the given system size is a valid one.

   When the `update` routine is completed it should be possible to run the program, even though it doesn't produce any measured results. Make a test run with $L = 8$ and $T = 2.2$:

   ```
   ./ising L=8 T=2.2 run
   ```

   If everything is correct you should get acceptance ratio $\approx 14\%$.

3. Write a function `measure` to measure magnetization and energy. Also put in some code in `mc` to accumulate $|M|$, E, and E^2 (which is needed for the heat capacity.) Then complete the function `result` which should print out $|m|$, e, and c (i.e. magnetization, energy and heat capacity per spin). It is good to print out both averages for each block and the total values. The output could e.g. look like

   ```
     energy     c          magn
   -1.563313  1.056386  0.819306
   -1.560219  1.103098  0.817187
   -1.574837  1.035064  0.826266
   -1.565069  1.046963  0.819550
    --------  --------  --------
   -1.565859  1.060773  0.820577
   ```

   If everything is correct the energy per spin for $L = 8$ and $T = 2.2$ should be ≈ -1.568. For a longer run and higher precision, you can try

```
./ising L=8 T=2.2 nblock=64 run
```

4. In the simplest implementation the exponential function is called many times. This is a vaste of time since there will only be a few different values of ΔE. Change the program by first filling an array with values for $\exp(-\Delta E/T)$ in `init_tables` and then using this array in the update routine. Use the Unix command `time` (see below) with the old and new versions to check that the new version really is considerably faster.

   ```
   mv ising slow-ising
   make
   time ./slow-ising L=64 T=2.2 seed=1 run
   time ./ising L=64 T=2.2 seed=1 run
   ```

   Note that the commands above set the seed to the random number generator by hand `seed=1` and this is convenient to check that two versions of the program do the same thing. If everything is correct the two versions should therefore produce identical results. (With the default value seed=0 in the call to `init_ran` a new seed is instead taken from the clock and the sequence of random numbers is therefore different each time, see `ran.c`.)

5. We also need to be able to store configurations in files and read configurations from file into memory. Note the `sprintf(fname,...)` statement in `initialize_mc`. Uncomment the `read_config` and `write_config` calls in function `mc`. Also create a directory `conf` where the configuration files will be stored (see `config.c`).

Plotting

To get the data into the plotting program in a convenient way it might be good to write the parameters (temperature, T, and linear size, L) and results (energy, heat capacity, and magnetization) as five columns to one or several files directly from the simulation program. Note that we want different symbols for different system sizes in the figures.

1.3 Check data for the heat capacity

We will here check that the two different ways to calculate the heat capacity give similar results. Running the program with

```
./ising  L=16 nblock=64 T=2.0 run T=2.1 run T=2.2
```

will give you values for both E and C (from the flucutation formula) at the specified temperatures. Use

$$C\left(\frac{T_1+T_2}{2}\right)=\frac{E(T_2)-E(T_1)}{T_2-T_1},$$

to calculate $C(2.05)$ and $C(2.15)$. Plot the two determinations of the heat capacity per spin in the same diagram with different symbols. **report**

1.4 Phase transition and size dependency

The phase transition in the Ising model is seen clearly in both the magnetization and the specific heat. The transition is perfectly sharp only in an infinite system and the quantities that ideally have a sharp and abrupt behavior become smoothened in smaller systems. You should do the simulations with `nblock=64` and four different system sizes, $L = 8$, 16, 32, and 64. At $L = 8$ it is sufficient to take temperatures $T = 2.0, 2.1, \ldots 2.8$ but for the other sizes you should take a larger number of temperatures, $T = 2.00$, 2.10, 2.16, 2.18, 2.20, 2.22, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29, 2.30, 2.32, 2.34, 2.36, 2.38, 2.40, 2.44, 2.48, 2.52, 2.56, 2.60, 2.70.

Note that the simulations for each size may conveniently be taken care of by a single run of the program since the program then continues from a configuration from a nearby temperature, which means that we can do with rather short times for thermalization (which are chosen automatically). Note also that the configurations are stored automatically in the `conf` directory and that it is possible to read in a configuration with e.g. `read=064_2.250` specified on the command line (as seen in `read_args`).

Plot $\langle |m| \rangle$ versus T for the four different sizes in a single figure. Connect the symbols with lines and remember to label the axes properly. Make another figure with C versus T. **report**

1.5 Cluster update

Write a routine that performs a Wolff cluster update step as described in Sec. 4.3.3. Note that a call to the routine only should give a single cluster. Again test your program by running for $L = 8$ and $T = 2.2$.

1. Do simulations with $L = 256$ and several temperatures close to T_c: $T = 2.20$, 2.22, 2.24, 2.25, 2.255, 2.26, 2.262, 2.264, 2.266, 2.268 and 2.269. (To save time you can here use `nblock=1`.)

2. Also run with $L = 1024$ and temperatures $T = 2.265$, 2.266, 2.267, 2.268, and 2.269. (To get good precision you now have to run longer; each run should take at least a few minutes. Good data will make the analysis in the next step easier.)

3. Plot your data as m versus $T_c - T$ close to T_c on a log-log scale for $L =$ 64, 256, and 1024. (Use $T_c = 2.26919$ rather than the less precise 2.269.) If everything is correct it should be possible to identify a straight line through the data, but there should also be clear deviations from this behavior for smaller L. Make use of the seemingly reliable data points close to T_c to determine the slope, and thereby the exponent β in $m \sim (T_c - T)^\beta$. Draw a line in the same figure to show the fitted line. **report**

2 For extra points

2.1 The correlation time (2p)

The drawback of Markov chains is that the produced configurations are correlated to one another. To study this effect we will examine the time correlation function for the energy

$$C_E(t) = \langle \delta E(t') \delta E(t' + t) \rangle.$$

Here $\delta E = E - \langle E \rangle$ and the average is over a large number of reference times t'. Instead of using the above formula directly (which presumes that $\langle E \rangle$ is known) one can calculate the same thing through

$$C_E(t) = \langle E(t') E(t' + t) \rangle - \langle E \rangle^2 .$$

One expects the correlations to decay exponentially with time,

$$C_E(t) \sim e^{-t/\tau},$$

which is usually true for all but the smallest t. $t > \tau$ is usually safe.

- Add some code to measure $C_E(t)$. Determine the energy correlations for the Metropolis algorithm with $L = 32$ for times up to $t = 200$ at temperature $T = 1.8$, 2.0, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, and 2.8. Around T_c where the correlation time is longer you will need longer runs to get good precision in the data. Plot $C_E(t)$ versus t in a single figure with a log scale on the y axis. Skip data which is only noise. **report**
- Determine the correlation time and plot τ versus T.
- Also make a separate figure with $C_E(t)$ obtained with the Wolff cluster algorithm at $T = 2.3$.

2.2 Visualization of the simulation process (2p)

Use the `g2` library to be able to see the update moves on the screen. Some information about the `g2` library is available at a link from the `mc` web page. If the `g2` library isn't installed (check if `/usr/lib/libg2.so.0` exists on the computer you are using) contact Peter.