Author: Laurin Pannullo
Laurin Pannullo: pannullo@th.physik.uni-frankfurt.de

Final Project: The 2D Ising Model

Study of the phase transition via Monte Carlo simulations

1 Introduction

The subject of this project will be the two dimensional Ising model. This model will be used to describe the interaction of spins (either spin up or spin down) on a two-dimensional lattice e.g. in a ferromagnet. At low temperatures, the system is ordered (almost all spins are aligned), but if we raise the temperature above a critical temperature T_c, the temperature fluctuations destroy the order and there is no preferred spin alignment any more (compare Fig. 1). The task will be to study this phase transition numerically using Monte Carlo simulations. Many of the techniques you will use are theoretically rather complex in their theory. Therefore they will only be discussed on a level such that you will be able to implement and use them. The interested reader is therefore encouraged to inform oneself in detail about these concepts during or after the project.

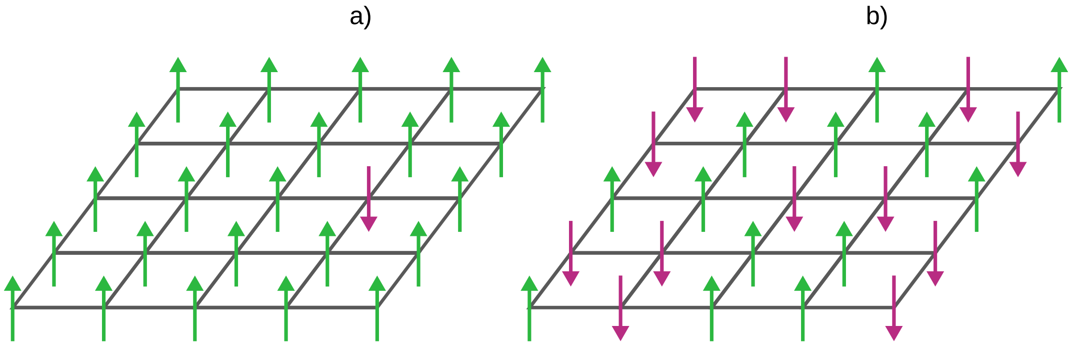

Figure 1: Schematic representation of the spins in the two dimensional Ising model. The arrows represent spin up or down. a): The system is below T_c and ordered b): The system is above T_c and unordered.

2 Theory

2.1 2D Ising Model

We will assume a square lattice Λ of 'volume' $L \times L = V$ and denote the spins at positions i, j as $s_{i,j}$ with $s_{i,j} = \pm 1$. The Hamiltonian $\mathcal{H}$ of the system is

$$\mathcal{H}(S) = -\frac{J}{2} \sum_{i,j \in \Lambda} s_{i,j} \left(s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} \right), \tag{1}$$

where J is the interaction strength of spins, the 1/2 prevents double counting, $S = \{s_{i,j}\}$ is a configuration of spins and we assume periodic boundary conditions in both directions: $s_{i+L,j} = s_{i,j}$, $s_{i,j+L} = s_{i,j}$.

The canonical partition function of the system is then given by

$$\mathcal{Z}(\beta) = \sum_{S} \exp\left[-\beta \mathcal{H}(S)\right], \tag{2}$$

where $\beta = 1/(k_B T)$ with T the temperature, k_B Boltzmann's constant and the sum runs over all possible spin configurations S of the system. Later, it will be useful to express everything in dimensionless quantities. Therefore let us already define $\hat{\beta} := \beta J$ and $\hat{\hat{\mathcal{H}}} := {}^{\hat{\mathcal{H}}}/_{J}$. Then the the Boltzmann weight is $\exp[-\hat{\beta}\,\hat{\mathcal{H}}]$. It is easy to see, that both the dimensionless and the original weight have the same value. We will use these dimensionless quantities when we talk about the implementation of the code.

Observables are then calculated as

$$\langle O \rangle = \frac{1}{\mathcal{Z}(\beta)} \sum_{S} O(S) \exp\left[-\beta \mathcal{H}(S)\right]. \tag{3}$$

The number of possible states S is $N_{\text{total}} = 2^{L^2}$. This renders a direct calculation of Eq. (3) already for quite small volumes impossible. Therefore, in the following section we will briefly introduce a statistical method to approximate sums like Eq. (3).

2.2 Monte Carlo simulation

A Monte Carlo method, in general, is a computational algorithm that relies on repeated random samples to obtain numerical results. For example, integrals can be performed using Monte Carlo techniques. The basic idea is that we create N samples in our domain of integration A and evaluate our function at these samples. For an infinite number of samples the true value of the integral is the average of the function samples. For a 1D integral this results in

$$I = \int_A \mathrm{d}x f(x) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i), \tag{4}$$

where x_i are uniformly distributed randomly chosen samples in A. In practice, as an approximation of the integral, we can choose N to be large but finite. This method can also be applied to sums as in Eq. (3). For our problem, however, we would need too many samples for a good approximation of the sum, if we would create uniformly distributed random samples (a sample of our domain of integration in the context of the ising model is just one possible spin configuration S).

Although there is a large number of possible states of our system, the Boltzmann weight $\exp\left[-\beta H(S)\right]$ of the majority of spin configurations S is very small and therefore their contribution to the partition function is close to zero. Thus it is a good approximation to use mainly those spin configurations with a significant contribution for the calculation of the partition function. The technique to generate these *important* configurations is called importance sampling.

We will use the Metropolis-Hastings algorithm to generate such a set of configurations. Given a start configuration the next configuration is generated by doing the following steps:

1. Flip one spin of your lattice: $s_{i,j} \to s'_{i,j} = -s_{i,j}$[1].
   Thus obtaining a new set of spins $S = \{s_{1,1}, s_{1,2}, \dots, s_{i,j}, \dots, s_{L,L}\} \to S' = \{s_{1,1}, s_{1,2}, \dots, -s_{i,j}, \dots, s_{L,L}\}$.
2. Calculate the resulting change in energy $\delta\mathcal{H} = \mathcal{H}(S') - \mathcal{H}(S)$.
3. Accept S' as the new configuration with the probability

$$W\left(s_{i,j} \to s'_{i,j}\right) = \begin{cases} 1 & \text{if } \delta\mathcal{H} < 0 \\ e^{-\beta\,\delta\mathcal{H}} & \text{otherwise} \end{cases}. \tag{5}$$

   When S' is rejected keep S as the new configuration.

One full update corresponds to repeating these steps L^2 times. The next configuration in the set of configurations to be used in the Monte-Carlo method is obtained after one full update. The previous configuration is used as start configuration of the next update. The configurations created with this algorithm are obviously not independent, since new configurations are always created from previous ones. This is called *auto-correlation* and we will discuss this further in Sec. 6. Therefore, in practice one full update corresponds to N_{skip}-times L^2 Metropolis-Hastings

[1]There are different ways to choose which spin $s_{i,j}$ to flip. We will discuss the different approaches in Sec. 4.2

steps, where N_{skip} is chosen such that the obtained configuration is essentially independent of the previous one.

We estimate the expectation values of an observable O as in Eq. (3) by

$$\langle O \rangle \approx \frac{1}{N} \sum_{n=0}^{N} O_n, \tag{6}$$

where $O_n = O(S_n)$ is the observable measured on the n-th generated configuration S_n and N is number of generated configurations.

2.3 Observables

Moments

Given an observable O_n we can define moments m_k of this observable as

$$m_k = \frac{1}{N} \sum_{n=0}^{N} O_n^k, \tag{7}$$

and the central moments μ_k

$$\mu_k = \frac{1}{N} \sum_{n=0}^{N} (O_n - m_1)^k. \tag{8}$$

It can be shown that the central moments can also be calculated directly from the moments as

$$\begin{aligned} \mu_1 &= 0\ , & \mu_2 &= m_2 - m_1^2, \\ \mu_3 &= m_3 - 3m_2m_1 + 2m_1^3\ , & \mu_4 &= m_4 - 4m_3m_1 - 3m_2^2 + 12m_2m_1^2 - 6m_1^4 + 3\mu_2^2. \end{aligned} \tag{9}$$

As a last definition we introduce the standardized moment

$$B_n := \frac{\mu_n}{(\mu_2)^{\frac{n}{2}}}. \tag{10}$$

Order Parameter

An order parameter is an observable O, that is a measure of the phase of a system. It normally ranges between zero in one phase and non-zero in the other. Moments of this order parameter characterize the phase transition and dynamics of the system:

- **Mean** m_1: We denote this also with $\langle O \rangle$. This quantity reflects in which phase the system is. When crossing $\hat{\beta}_c$ the mean will jump immediately from zero to constant non zero for a first order phase transition in the infinite volume.
- **Susceptibility** $\chi = \mu_2 V$: This quantity will peak (diverge in an infinite volume) at the phase transition.
- **Skewness** B_3: This quantity gives information about the asymmetry of the probability distribution of our observable.

$$B_3 \begin{cases} > 0 & \text{if the distribution has a tail to right} \\ = 0 & \text{if the distribution is symmetric} \\ < 0 & \text{if the distribution has a tail to left} \end{cases} \tag{11}$$

  Zeros of the skewness might signal a phase transition.
- **Kurtosis** B_4: When calculated at $\hat{\beta}_c$ the kurtosis assumes distinct values depending on the order of the phase transition and the symmetries of the system (Analyzing the phase transition with this quantity goes beyond the scope of this project.).

The magnetization

$$M_n = \frac{1}{L^2} \sum_{i,j \in \Lambda} s_{i,j} \tag{12}$$

serves as an order parameter for this system.

3 Tasks

The project consists of a **mandatory** part and *optional* tasks.

The **mandatory** part is:

1. Perform a Monte Carlo simulation of the two dimensional Ising model.
2. Determine $\hat{\beta}_c$ via the mean Magnetization $\langle M \rangle$ and the susceptibility χ of M.
3. Plot B_3 of M as a function of $\hat{\beta}$. Would you try to locate $\hat{\beta}_c$ with this quantity?

If you finish early with this task and are motivated two *optional* tasks are proposed

1. Simulate at different volumes and approach the infinite volume limit to understand the order of the phase transition.
2. Visualize the Monte Carlo history of the spin configurations with an animation.

4 Implementation

This section is meant to help you to write your code and give you hints about its structure. A kind of flowchart of the code can be seen in Fig. 5, where it is shown how the different parts of your program are connected and work together. Unless you have good reasons not to do so, follow the suggested implementation!

4.1 Structure of the C program

In the design of your code you should separate the generation of configurations and the measurement of observables. This prevents you from having to regenerate configurations if you need to measure other observables. Therefore the source code will be split into three files. So the main function of your program should only parse a command line argument, which decides if your program generates configurations or measures observables.

```c
int main(int argc, char const *argv[])
{
  if (strcmp(argv[1], "--generate") == 0)
    generate(argv[2]);
  else if (strcmp(argv[1], "--measure") == 0)
    measure(argv[2]);
  else
  {
    printf("Mode %s not recognized\n", argv[1]);
    exit(1);
  }
  exit(0);
}
```

The second command line argument should be the name of a configuration file `config_file`[2] where you can set parameters for your simulation and measurement, that are parsed by your program. The easiest way to do this is

[2]Note that certain output or input files will be referred to with a variable like name e.g. `config_file`, but this is only meant as an identifier within this text and Fig. 5. For your program the files should have meaningful names by which you and your program can distinguish files e.g. outputfiles of runs with different parameters.

a file, where the different lines correspond to the value assigned to their corresponding variable. Therefore, it is important that the order of the parameters is defined and fixed. Possible parameters given via this file are shown in Tab. 1. The meaning of some of them will only become clear after reading the following sections.

Parameter	function	used by
`N`	sets total number of lattice updates	`generate.c`
`NSkip`	sets amount of lattice updates between two measurements	`generate.c`
`L`	sets the lattice size	`generate.c, measure.c`
`betaLower`	sets lower bound of the $\hat{\beta}$ interval	`generate.c, measure.c`
`betaUpper`	sets upper bound of the $\hat{\beta}$ interval	`generate.c, measure.c`
`betaStep`	sets step size of the $\hat{\beta}$ interval	`generate.c, measure.c`
`startParameter`	sets the initial configuration	`generate.c`
`id`	gives your result files a unique identifier	`generate.c, measure.c`
`outputConfig`	controls whether the spin configurations are saved	`generate.c`
`NThermal`	sets the thermalization time after which observables are calculated	`measure.c`

Table 1: Proposed parameters for `config_file`.

The content of `config_file` could look like

```
10000
100
128
0.1
0.9
0.001
1
run1
0
200
```

where one line corresponds to the value of one variable.

Alternatively you could use a two column file, where the first column specifies the parameter and the second the given value. Your program then has to parse the entry of the first column to assign the value of the second column to the right variable inside your program. Your file `config_file` could then look like this:

```
N 10000
NSkip 100
L 128
...
```

4.2 `generate.c`

The code in this file is responsible for

- initializing the lattice,
- do the following for every $\hat{\beta}$ specified by `config_file`:
  - generating configurations accordingly to the Metropolis-Hastings algorithm,
  - measuring the magnetization per configuration M_n accordingly to Eq. (12),
  - writing all calculated M_n to `output_file`
  - optional: writing the generated spin configurations to `spinConfig_file`

To guide you a bit further, we will now discuss the vital aspects of this code.

The generate function

In the provided example of the main function there is a function called generate that handles all tasks for the generation of the configurations. It could be structured like

```
void generate(char const *fileName)
{
  /*
    - declare variables
    - parse config_file
    - call init function
  */

  for(double b = betaLower; b <= betaUpper; b+=betaStep)
  {

    /*
      - set initial spin configuration
    */

    for(int n = 0; n <= N; ++n)
    {
      /*
        - update your Lattice
        - store magnetization every NSkip steps
      */

    }

    /*
      - save results in file
    */

  }
}
```

Again, the meaning or necessity of some structures will only become clear when reading the following section.

Initialization

Once you've parsed your configuration file, you know crucial values like the size of the lattice or the number of configurations to generate. This will affect the allocation of memory for different arrays among other things. To handle such initialization tasks create a function

```
void init(...);
```

Lattice setup

You will need an array latticeSpin to store the spin values $s_{i,j}$. Instead of a two dimensional array, a one dimensional array is usually used, which is indexed with a so-called superindex n. This is done by a function $n : \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$, $n(i,j) := j \cdot L + i$, whose signature could read

```
unsigned int getSuperIndex(const int i, const int j);
```

where i and j are the x and y coordinates of the site. This is a suitable location to implement the periodic boundary conditions.

Since the interaction takes place only between nearest neighbours, you will often have to retrieve the neighbouring spin values, for which you would have to calculate the superindeces of these neighbours. However, since these neighbour indices are constant during your simulation, it is recommended to create a two dimensional array neighbourArray of dimensions $L^2 \times 4$ during initialization that contains the superindices of all nearest neighbours to a grid point. Filling neighbourArray with the correct indeces should be part of init.

Random numbers

In the implementation of some functions you will need to generate uniformly distributed random numbers. Therefore implement a function

```
double drawRandomNumber(void);
```

which returns a random number $x \in [0, 1]$ drawn from a uniform distribution. Regarding our problem the basic C generator is sufficient.

Initial spin configuration

The Metropolis-Hastings algorithm always needs a configuration to be updated. This means that we need to choose an initial spin configuration at the beginning of our simulation. There are two common options

- cold start: Set all spins to the same value ($\rightarrow +1$ or -1)
- hot start: Randomly set spins to ± 1

The cold start is recommended. In case you do the optional task of visualization try to answer the following: What happens with a hot start at large $\hat{\beta}$? This should answer why a cold start is recommended.

Implement a function

```
void setInitialConfig(int* latticeSpin, int mode);
```

to handle the task of setting the initial configuration.

Metropolis-Hastings algorithm

The next task is to implement the Metropolis-Hastings algorithm. Implement a function that performs the Metropolis-Hastings algorithm , whose signature could read

```
void updateLattice(int* latticeSpin, const int** neighbourArray, const double beta);
```

where one call of this function should perform L^2 iterations of the Metropolis-Hastings algorithm. To do this, we look at the computational requirement of every step of the algorithm:

1. Step **- Flipping of a spin:** By now you will have already wondered how to choose the spins to flip. There are three answers that might come to mind
   - Sequential update: Just go through the whole lattice in order $\rightarrow$ *Not recommended* as the generated configurations will be highly correlated. But it is encouraged to explore this option in combination with the visualization task.
   - Random update with doubling: Choose a spin of your lattice randomly $\rightarrow$ *Recommended* option as it has the right balance between cost and auto-correlation.
   - Random update without doubling: Choose the order of the spins randomly, but ensure that updateLattice updates every site exactly once $\rightarrow$ *Not recommended* as it is very expensive and is not needed.
2. Step **- Calculating $\delta\hat{\mathcal{H}}$:** The easiest way to do it, is to calculate $\hat{\mathcal{H}}(S')$ and $\hat{\mathcal{H}}(S)$ and take the difference. However, this calculates more than you need to. Since the interaction of the spins are only nearest neighbours, the energy will only change locally. Therefore, you will have to simplify $\delta\hat{\mathcal{H}}$ analytically.
3. Step **- The acceptance step:** Use drawRandomNumber to implement the acceptance step.

Calculate M_n

Calculate the magnetization M_n every N_{Skip} configurations and store them in an array.

Output files

As proposed in Fig. 5 this part of your program should be able to output a file `output_file` where every line corresponds to the magnetization M_n of a generated configuration in your Monte Carlo simulation. To keep it simple, generate one file per value of $\hat{\beta}$. Generate unique names for you files from the run parameters. It should look like

```
...
0.912109
0.908203
0.913574
0.898926
0.917480
0.912109
...
```

If you intend to do the optional task of the visualization, your code should also be able to output files of the spin configuration `spinConfig_file`. Generate matrix blocks with the spin values and separate them by a double blank line. Their content should look like

```
...
-1 -1 +1 -1 -1 -1
-1 -1 +1 +1 +1 -1
-1 +1 +1 -1 -1 -1
+1 +1 -1 -1 -1 +1
-1 +1 +1 +1 +1 -1
-1 +1 -1 +1 +1 +1


-1 -1 +1 -1 -1 +1
...
```

for a 6×6 lattice. Make the output of such files optional and only output them when needed, as their size can quickly increase for larger lattices and statistics (having a few GB file is easily possible).

4.3 `measure.c`

The code in this file is responsible for

- reading in the previously generated `output_file` corresponding to the $\hat{\beta}$ specified by `config_file`
- calculate $|\langle M \rangle|$ and χ, B_3, B_4 of M as defined in Sec. 2.3 for all configurations after N_{Therm}.
- write them to a central file `results_file`, where the first column is $\hat{\beta}$ and the next columns are the corresponding quantities calculated in the previous step.

The content of `results_file` should look like

```
...
0.014000 0.029142 0.001359 -0.085496 3.066083
0.015000 0.024791 0.001016 0.091971 3.347088
0.016000 0.026640 0.001137 0.124383 2.998146
0.017000 0.026147 0.001075 0.034916 2.857480
...
```

Thermalization

When doing Monte Carlo simulations the system needs time to *thermalize*. This means that the system is not yet in equilibrium (with respect to the Monte Carlo simulation) and the generated configurations are not representative. This is best seen when looking at the Monte Carlo history from `output_file`. The Monte Carlo history of M_n of a simulation for $\hat{\beta} < \hat{\beta}_c$ initialized with a cold start is shown in Fig. 2. At the beginning the value starts at

1 (due to the cold start) and then slowly evolves towards zero until it only fluctuates around this value, where it seems to be in equilibrium. The amount of configurations it takes until the simulation is in equilibrium is called thermalization time. One should only start to calculate expectation values of observables with configurations *after* this thermalization time. Keep in mind that this thermalization time depends for one on your system's size, but also on the value $\hat{\beta}$. Therefore is strongly recommended to have a quick look into some of the Monte Carlo histories before calculating observables to have a rough estimate of N_{Therm}. When in doubt it is better to overestimate N_{Therm}.

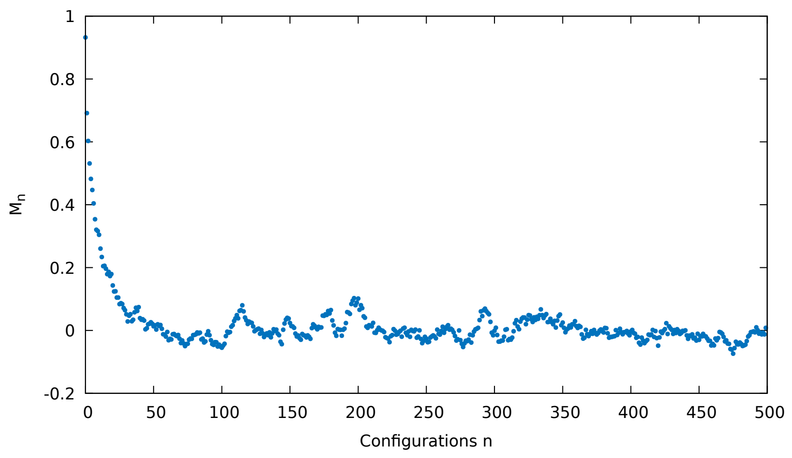

Figure 2: Monte Carlo history of M_n for $L = 256$, $N_{\text{Skip}} = 10$ and cold start.

4.4 Visualizing the Monte Carlo simulation

As an optional task you could try to visualize the evolution of configurations saved in `spinConfig_file` in an animation e.g. using `gnuplot` (or any other tool of your choice). To do so you might need some functionalities, that you might not have used so far. These include `gif terminal`, `do for` loops and the `index` keyword for the splot command. Your script could contain the following lines (note that comments are indicated by `#` in `gnuplot`):

```
...
set terminal gif animate size 800,600  # How does this terminal work?
...
stats datafile nooutput   # Gathers informations about the the file: line number etc.
...
# set up output
# set the different ranges
# set a colorpalette
...
set view map
...
do for [i=1: ... ] { # Which value has to be set in the second part of the square
splot ...            # bracket? Think of the stats command that we used. You will need
                     # the index keyword in splot.
}
```

A frame of your finished animation should then look like Fig. 3. Of course you are free to generate the animation in any other way e.g. as a video file.

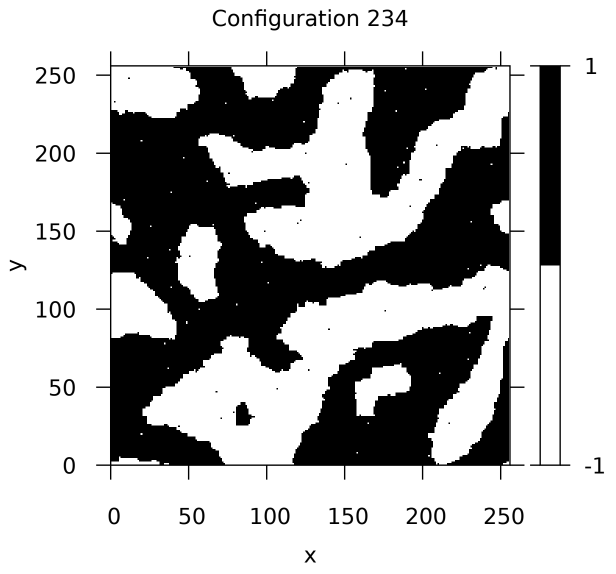

Figure 3: Frame of the animation visualizing the Monte Carlo history of the spin configurations.

5 Running your Code

To do the main task - determining $\hat{\beta}_c$ - a run with $L = 30$, $\hat{\beta} \in \{0.000, 0.001, \ldots, 0.999, 1.000\}$ with $N = 10000$ and $N_{\text{Skip}} = 100$ should be appropriate[3]. Fortunately, there is an analytic result for infinite volume $\hat{\beta}_c = \frac{1}{2}\ln(1+\sqrt{2}) \approx 0.4407$ you can compare the precision of your simulations to. In Fig. 4 $|\langle M\rangle|$ and χ from a run with the suggested parameters are plotted against $\hat{\beta}$. Why is a better idea to plot $|\langle M\rangle|$ instead of $\langle M\rangle$?

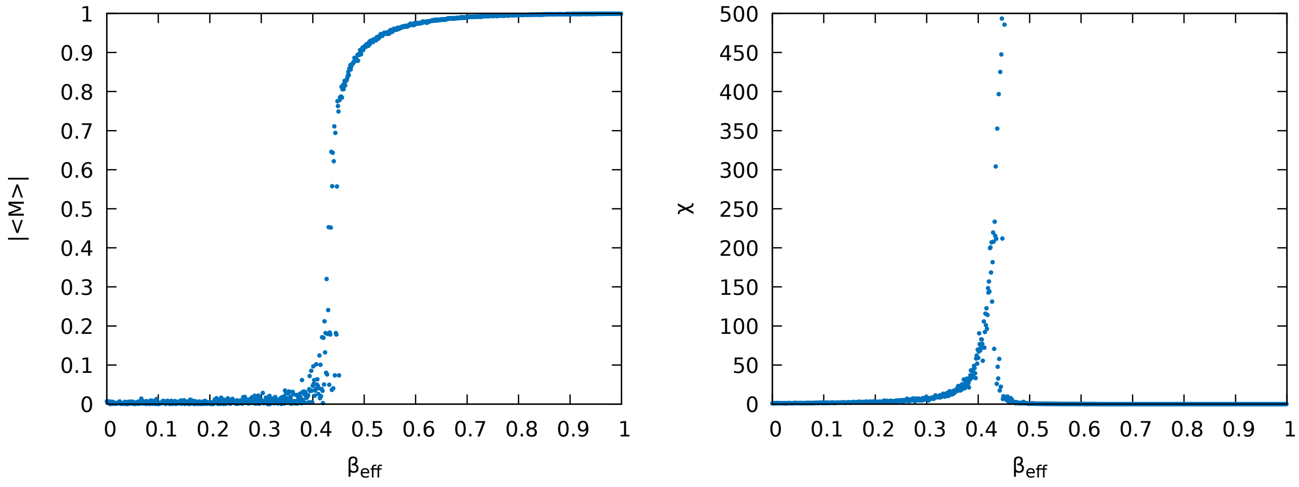

Figure 4: Results from a run with the suggested parameters.

5.1 Infinite volume limit

Most of the time we are interested in studying physics in an infinite volume. In contrast to this, our simulations will always be in a finite volume. However, we can simulate with increasing volumes and see if e.g. $\hat{\beta}_c$ converges

[3]On the tutor's office pc this took about 6 minutes.

to a fix value. To investigate this behaviour determine $\hat{\beta}_c$ for $L \in \{4, 8, 16, 32, \ldots, 256\}$ (or up to any volume that your computer is able to simulate in a reasonable time[4]). Do the small volumes have any relevant information? Do you think that you reached the infinite volume limit? Are you able to determine the order of the phase transition? Does B_3 improve? Keep in mind that the thermalization time and auto-correlation will drastically increase with the volume. So an increase in N_{Therm} and N_{Skip} will be necessary.

6 Closing remarks

Errors

So far, we did not speak about errors of observables nor instructed you to calculate these. However, of course, any numerical result is not free from errors and one has to be very careful. The standard deviation alone is not suitable, since the auto-correlation has to be taken into account in the estimation of the errors. Understanding and doing this can be a very complex topic and therefore it is left out of the project.

Further literature

If you want to learn more in detail about Monte Carlo simulation we recommend the following books *Markov Chain Monte Carlo Simulations and Their Statistical Analysis* by Bernd A. Berg and *Monte Carlo Methods in Statistical Physics* by M. E. J. Newman and G. T. Barkema. An important application of Monte Carlo simulation in high-energy physics is lattice field theory in general and lattice QCD. *Quantum Chromodynamics on the Lattice* by C. Gattringer and C. B. Lang is an excellent introduction to this topic for readers familiar with the path integral formalism of Quantum field theory

[4]One hour of run time is still reasonable for such a simulation.

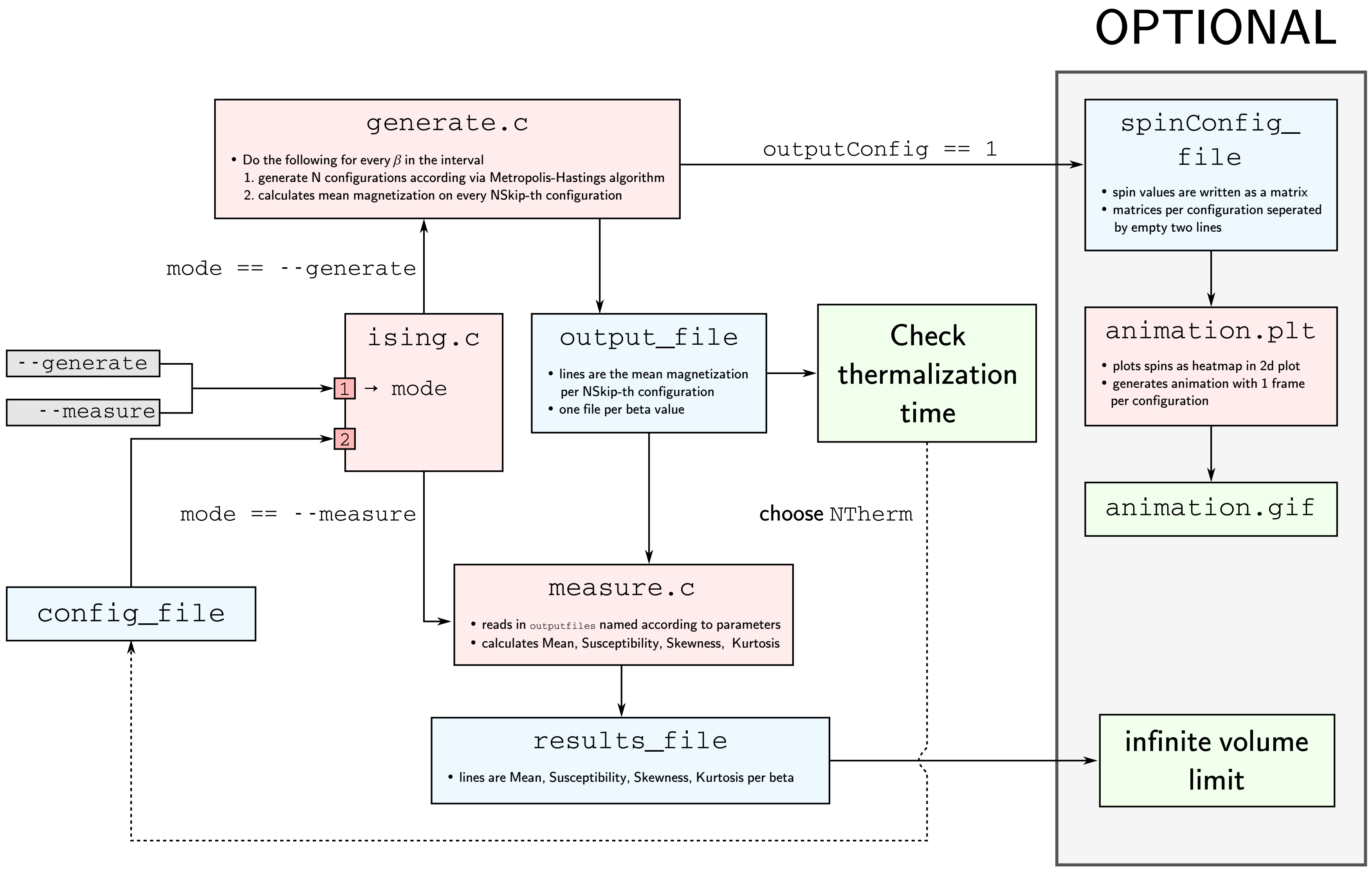
OPTIONAL
generate.c
• Do the following for every β in the interval
1. generate N configurations according via Metropolis-Hastings algorithm
2. calculates mean magnetization on every NSkip-th configuration
outputConfig == 1
spinConfig_ file
• spin values are written as a matrix
• matrices per configuration seperated by empty two lines
mode == --generate
ising.c
1 → mode
2
--generate
--measure
output_file
• lines are the mean magnetization per NSkip-th configuration
• one file per beta value
Check thermalization time
animation.plt
• plots spins as heatmap in 2d plot
• generates animation with 1 frame per configuration
animation.gif
mode == --measure
choose NTherm
config_file
measure.c
• reads in outputfiles named according to parameters
• calculates Mean, Susceptibility, Skewness, Kurtosis
results_file
• lines are Mean, Susceptibility, Skewness, Kurtosis per beta
infinite volume limit

Figure 5: Proposed structure of the task and program