Jakub Stawik
218715

Ising model

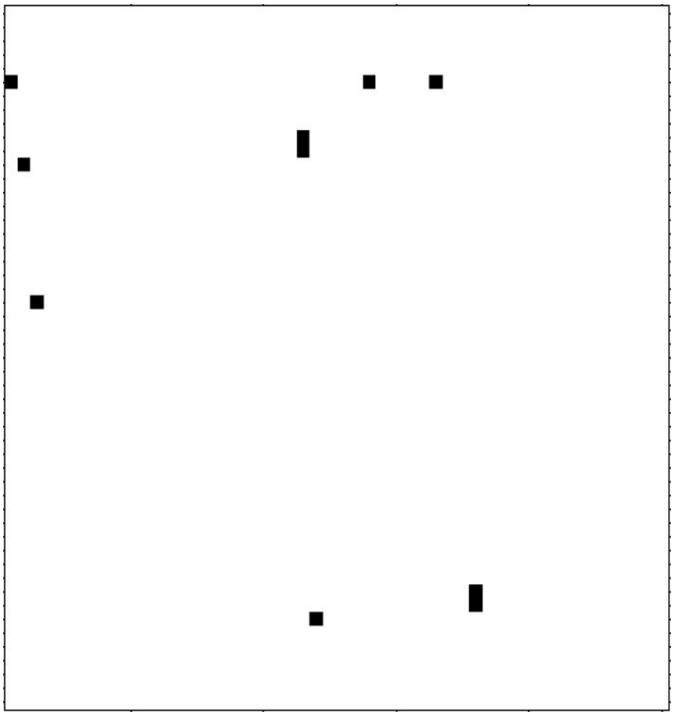
The code consists of two major parts: an engine written in c++ allowing for an increase in speed and multiple python scripts acting as plotters and drivers running multiple instances of the engine at once. Source code is attached at the end of the document but can be found @ github.com/jstawik/WUST-BDA/Statistical_physics/Ising2 as well. Keep in mind that directory Ising is my previous, failed attempt left there for historical accuracy.

1. Behaviour at different temperatures

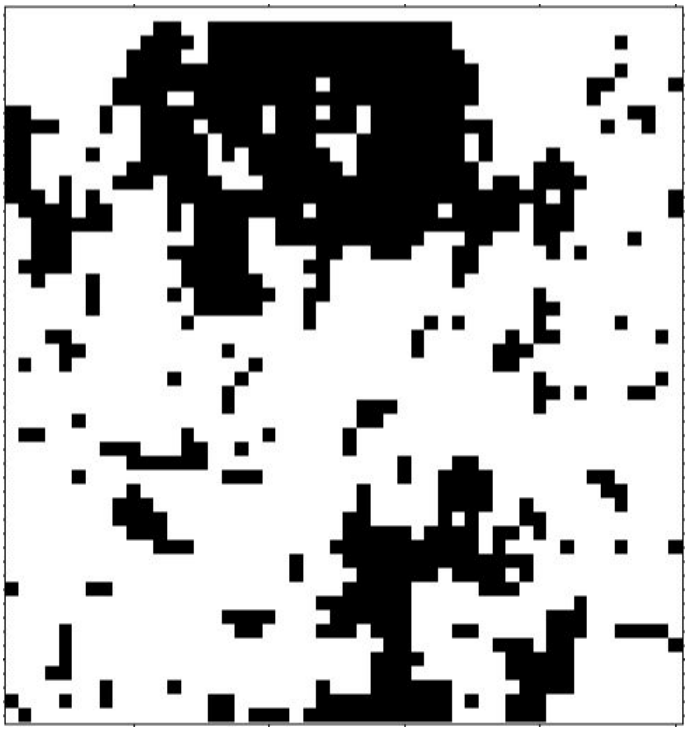
Below is a visualisation of equilibrium state for different temperatures T*. Parameters of the simulation:

Parameters	
Lattice size	50x50
Temperatures	1.5, 2.27, 3
MCSes skipped towards equilibrium	30 000

T* = 1.5



T* = 2.27

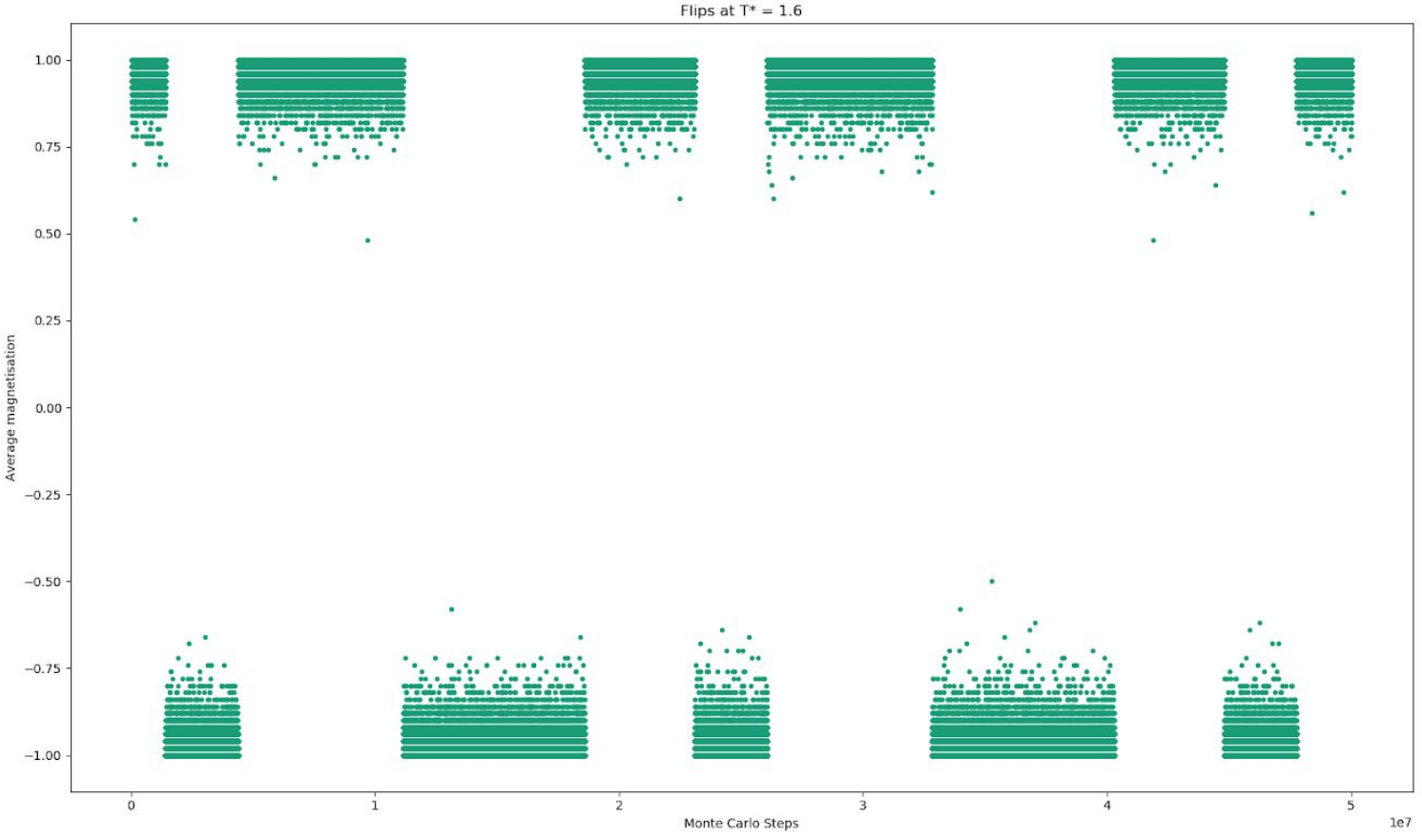


T* = 4.5

2. Flips between states

Flips between states have been observed for the following parameters:

Parameters	
Lattice size	10x10
Temperature	1.6
MCSes skipped towards equilibrium	30 000
Configurations	500 000
Steps between configurations	100

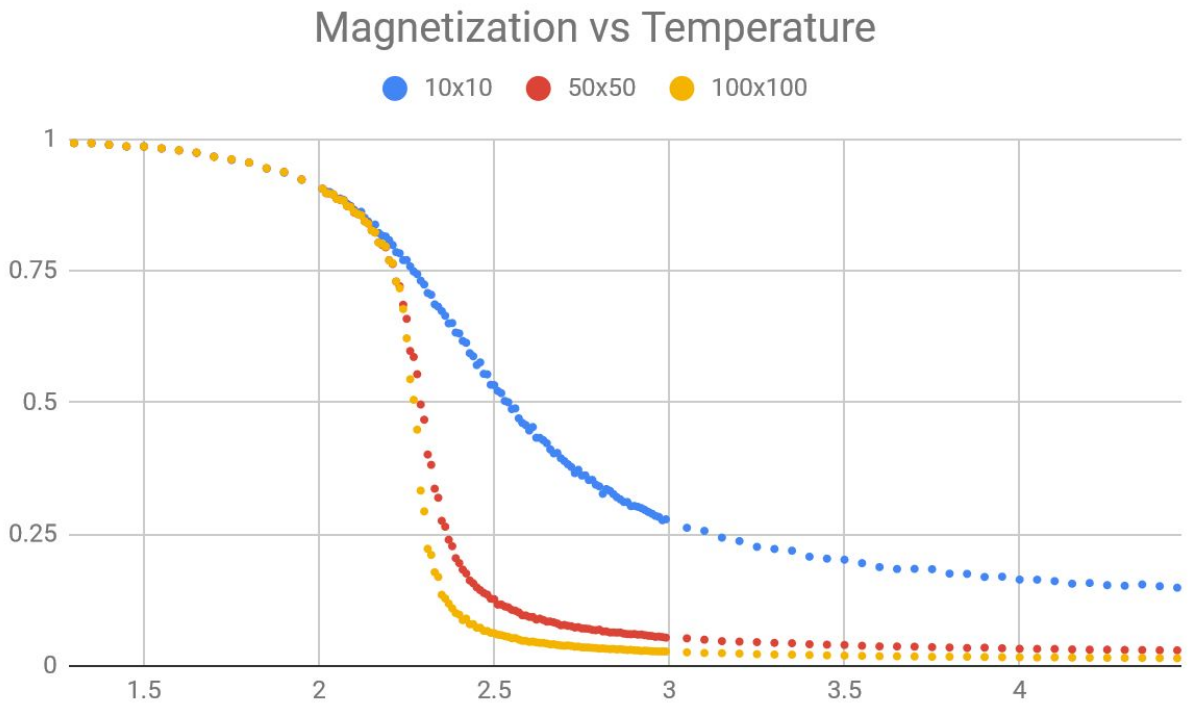


3. & 4. Dependence on temperature

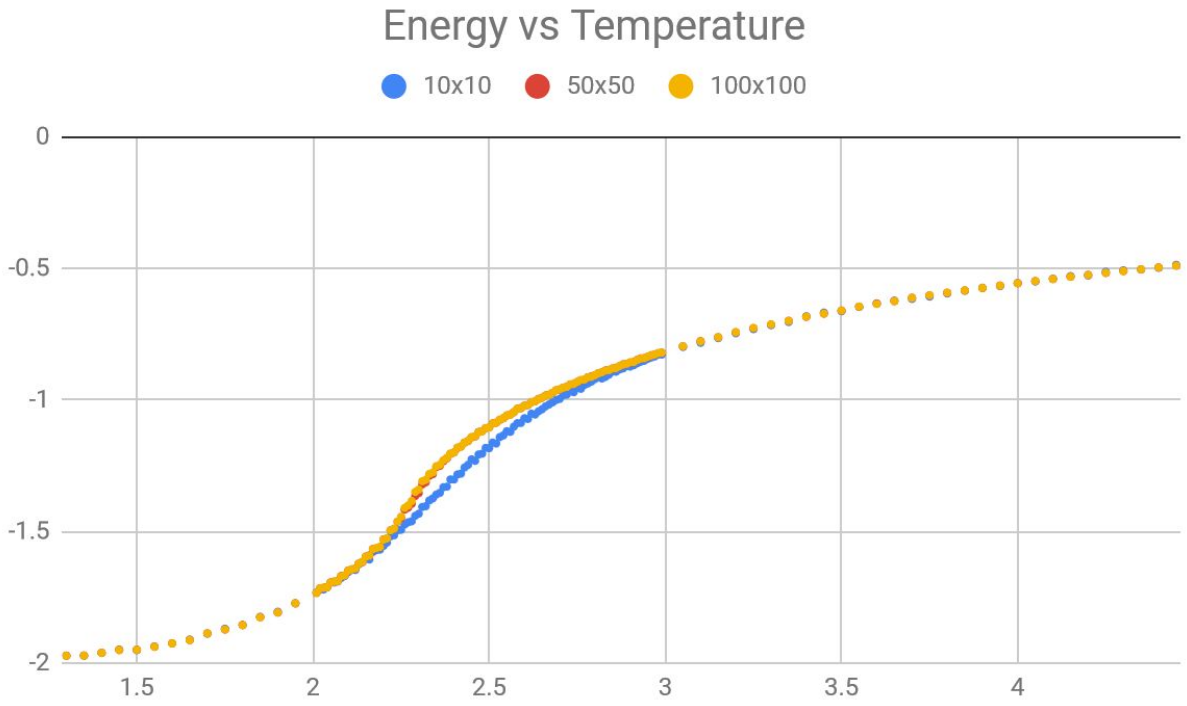
Dependence on temperature of the following parameters has been simulated: mean value of magnetization, energy, susceptibility, specific heat and Binder's cumulant. Systems of different sizes were considered. One simulation for each size and temperature were run and all the functions were calculated for it.

Parameters	
Lattice size	10x10, 50x50, 100x100
Temperatures	[1.3 : 1/20 : 2] [2 : 1/100 : 3] Higher grain range [3: 1/20 : 4.5]
MCSes skipped towards equilibrium	30 000
Configurations	5 000
Steps between configurations	50
Execution times	15.73s 6min 27s 25min 44s

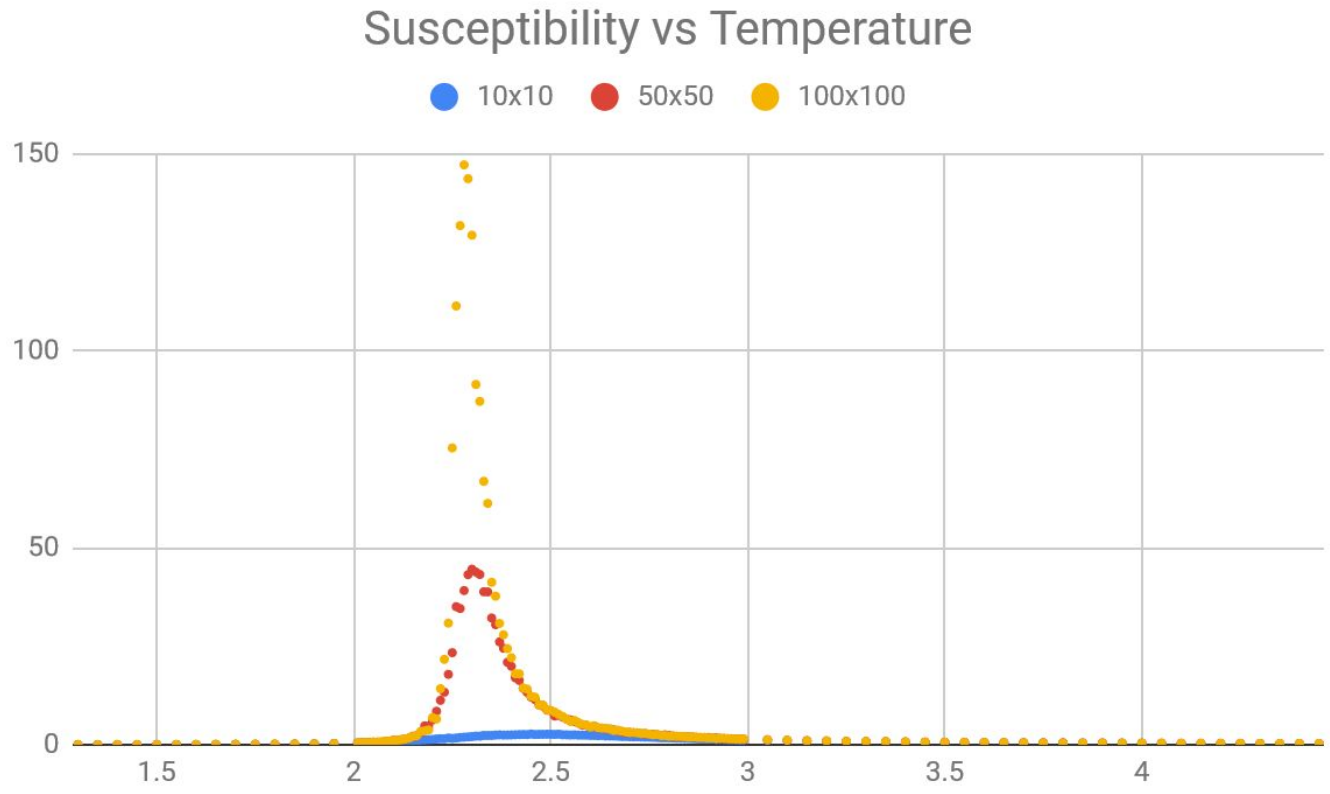
3.1 Magnetization



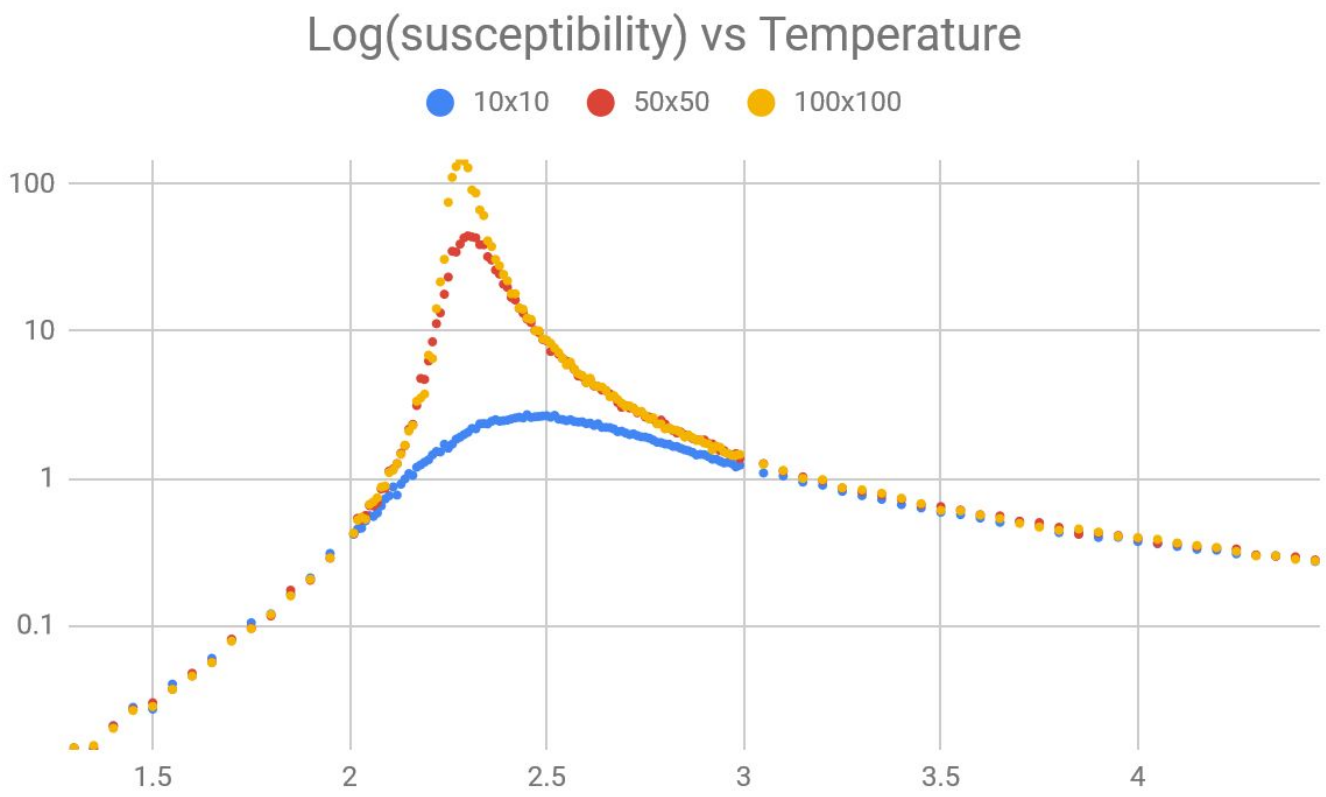
3.2 Energy



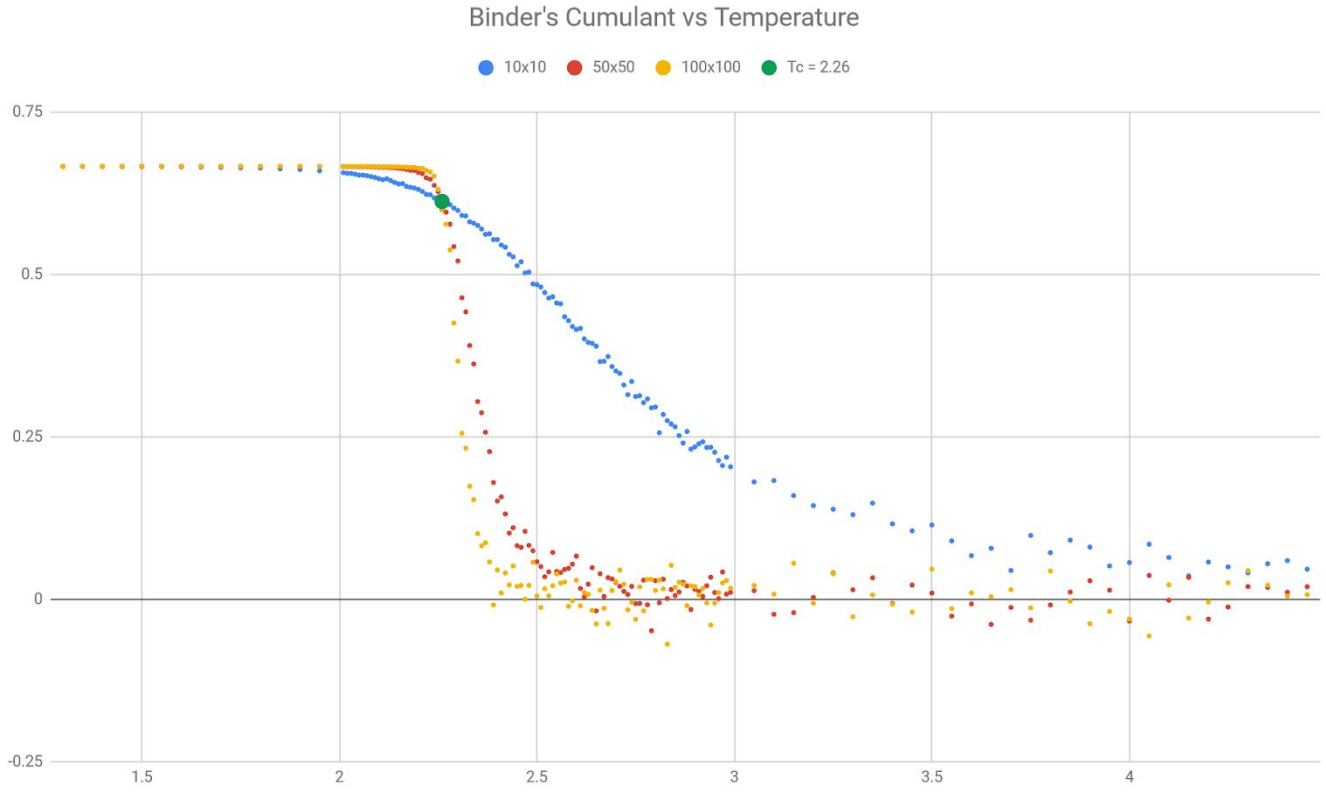
3.3 Susceptibility



(A log plot added for readability)

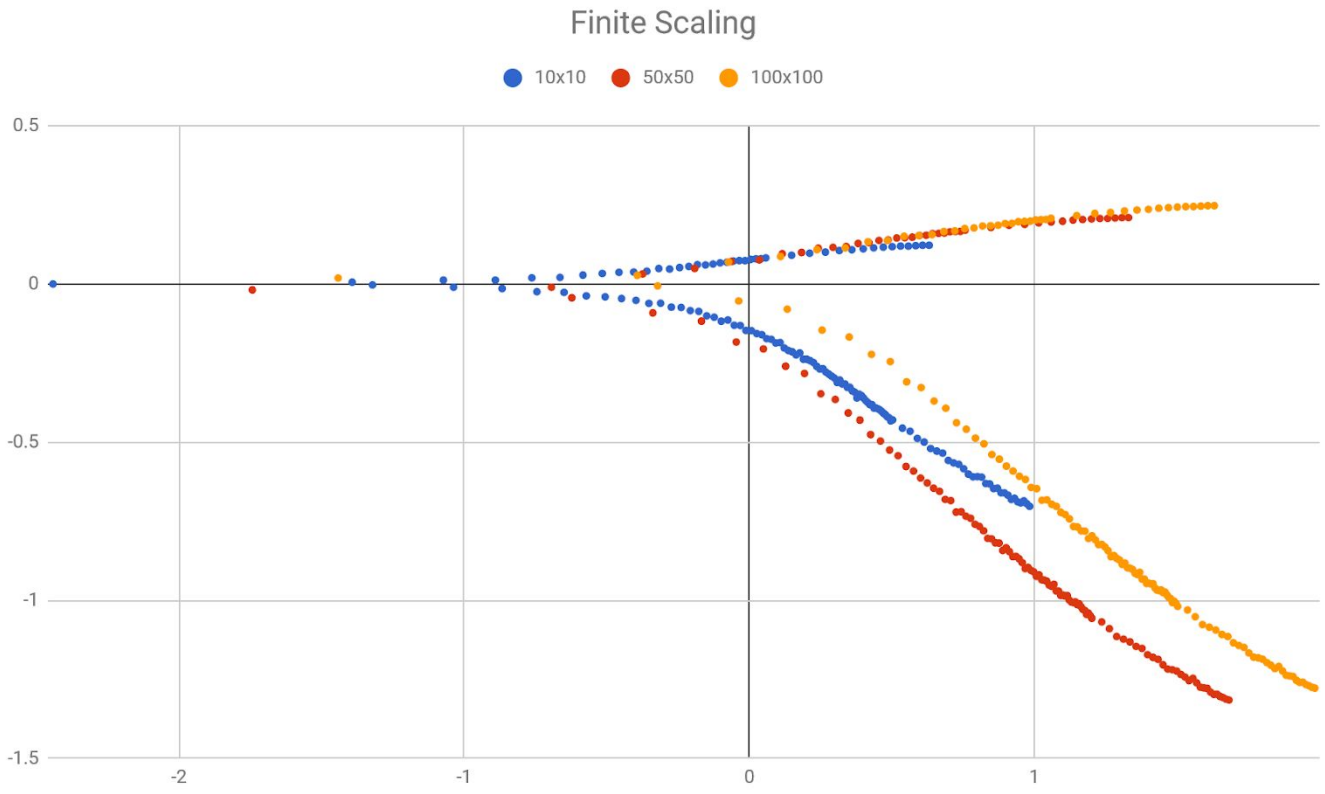


4.1. Binder’s Cumulant



5. Finite scaling

Finite scaling data has been calculated based on data provided by earlier simulations - hence no source code for it exists. Calculations for it and all plotting available at https://tinyurl.com/y443csrr



6. Source code

Source code is divided into two parts: C++ engine that runs the code reasonably fast and python script that makes for a portable multithreading driver.

```
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <cstdio>
#include <ctime>

using namespace std;
// a new binary must be compiled for each lattice size
#define L 10
int LATTICE [L][L];
int SKIPSTEPS = 30000;
int CONFIGS = 5000;

double calculate_energy(int trial, int n1, int n2, int n3, int n4){
    return 2 * trial * (n1 + n2 + n3 + n4);
}
void print_lattice(){
    for(int i = 0; i<L; i++){
        for(int j = 0; j<L; j++){
            cout<<LATTICE[i][j]<<"\t";
        }
        cout<<"\n";
    }
}
void MCS(int steps, double T){
    int trial;
    double energy;
    double diff;
    for(int i = 0; i < steps; i++){
        for(int x = 0; x < L; x++){
            for(int y = 0; y < L; y++){
                //Single MCS
                //TRIAL ENERGY
                if(x == 0 and y ==0) energy =
calculate_energy(LATTICE[x][y], LATTICE[((x+1)%L)][y]
```

```
                                                , LATTICE[L-1][y]
                                                ,
LATTICE[x][((y+1)%L)]
                                                , LATTICE[x][L-1]);
            else if(x == 0) energy = calculate_energy(LATTICE[x][y],
LATTICE[((x+1)%L)][y]
                                                , LATTICE[L-1][y]
                                                , LATTICE[x][((y+1)%L)]
                                                ,
LATTICE[x][((y-1)%L)]);
            else if(y == 0) energy = calculate_energy(LATTICE[x][y],
LATTICE[((x+1)%L)][y]
                                                , LATTICE[((x-1)%L)][y]
                                                , LATTICE[x][((y+1)%L)]
                                                , LATTICE[x][L-1]);
            else energy = calculate_energy(LATTICE[x][y],
LATTICE[((x+1)%L)][y]
                                            , LATTICE[((x-1)%L)][y]
                                            , LATTICE[x][((y+1)%L)]
                                            , LATTICE[x][((y-1)%L)]);
            //FLIPPING
            if(energy < 0) LATTICE[x][y] *= -1;
            else if(rand()%1000/1000.0 < exp(-energy/T))
LATTICE[x][y] = -LATTICE[x][y];
         }
      }
   }
}
double calculate_avg_energy(){
   double energy = 0;
   for(int x = 0; x < L; x++){
      for(int y = 0; y < L; y++){
         if(x == 0 and y ==0) energy -= LATTICE[x][y] *
(LATTICE[((x+1)%L)][y] + LATTICE[L-1][y] + LATTICE[x][((y+1)%L)]
+ LATTICE[x][L-1]);
         else if(x == 0) energy -= LATTICE[x][y] *
(LATTICE[((x+1)%L)][y] + LATTICE[L-1][y] + LATTICE[x][((y+1)%L)]
+ LATTICE[x][((y-1)%L)]);
         else if(y == 0) energy -= LATTICE[x][y] *
(LATTICE[((x+1)%L)][y] + LATTICE[((x-1)%L)][y] +
LATTICE[x][((y+1)%L)] + LATTICE[x][L-1]);
         else energy -= LATTICE[x][y] * (LATTICE[((x+1)%L)][y] +
LATTICE[((x-1)%L)][y] + LATTICE[x][((y+1)%L)] +
LATTICE[x][((y-1)%L)]);
      }
   }
   return energy/(2*pow(L, 2));
}
double calculate_avg_spin(){
   double sum = 0;
   for(int i = 0; i < L; i++)
      for(int j = 0; j< L; j++) sum += LATTICE[i][j];
   return sum/pow(L, 2);
}
double calculate_tot_mag(){
   double sum = 0;
   for(int i = 0; i < L; i++)
      for(int j = 0; j< L; j++) sum += LATTICE[i][j];
   return sum;
}

int main(int argc, char *argv[]) {
   // Temperature read & initialisation
   string arg = argv[1];
   double T = atof(arg.c_str());
   for(int i = 0; i < L; i++){
      for(int j = 0; j < L; j++) LATTICE[i][j] = 1;
   }
   srand(time(NULL));

  // Skip to stable
   MCS(SKIPSTEPS, T);
   // Calculations
   double mag = 0, mag2 = 0, magtmp = 0;
   double ener = 0, ener2 = 0, enertmp = 0;
   double binder = 0, binder2 = 0, binder4 = 0, bindertmp = 0;
   double susc = 0;
   double sheat = 0;

   for(int n = 0; n < CONFIGS; n++){
      MCS(50, T);
      magtmp = abs(calculate_avg_spin());
      mag += magtmp;
      mag2 += pow(magtmp, 2);
      enertmp = calculate_avg_energy();
      ener += enertmp;
      ener2 += pow(enertmp, 2);
      bindertmp = calculate_tot_mag();
      binder2 += pow(bindertmp, 2);
      binder4 += pow(bindertmp, 4);
   }
   mag /= CONFIGS;
   mag2 /= CONFIGS;
   ener /= CONFIGS;
   ener2 /= CONFIGS;
   binder2 /= CONFIGS;
   binder4 /= CONFIGS;
   // Thermodynamics quantities
   susc = (pow(L, 2)/T)*(mag2-pow(mag, 2));
   sheat = pow(L, 2)/pow(T, 2)*(ener2-pow(ener, 2));
   binder = 1 - (binder4/(3*pow(binder2, 2)));
   cout<<T<<", "<<mag<<", "<<ener<<", "<<susc<<", "<<sheat<<",
"<<binder<<"\n";

   // print_lattice();
   /* FLIPS CODE
   for(int i = 0; i < CONFIGS; i++){
      MCS(100, T);
      cout<<avg_spin()<<"\n";
   }
   FLIPS CODE */
   return 0;
}
```

And for the driver code:

```
from multiprocessing import Pool
import os
import numpy as np
import time


def drive(i):
            os.system('./Ising-100 '+str(i))


if __name__ == '__main__':
            start_time = time.time()
            temps = [i/100 for i in range(130, 200, 5)]+[i/100 for i in
range(201, 300, 1)]+[i/100 for i in range(305, 450, 5)]
            pool = Pool(60)
            pool.map(drive, temps)
            pool.close()
            pool.join()
            print("--- %s seconds ---" % (time.time() - start_time))
```