Statistical Physics Summer 2019

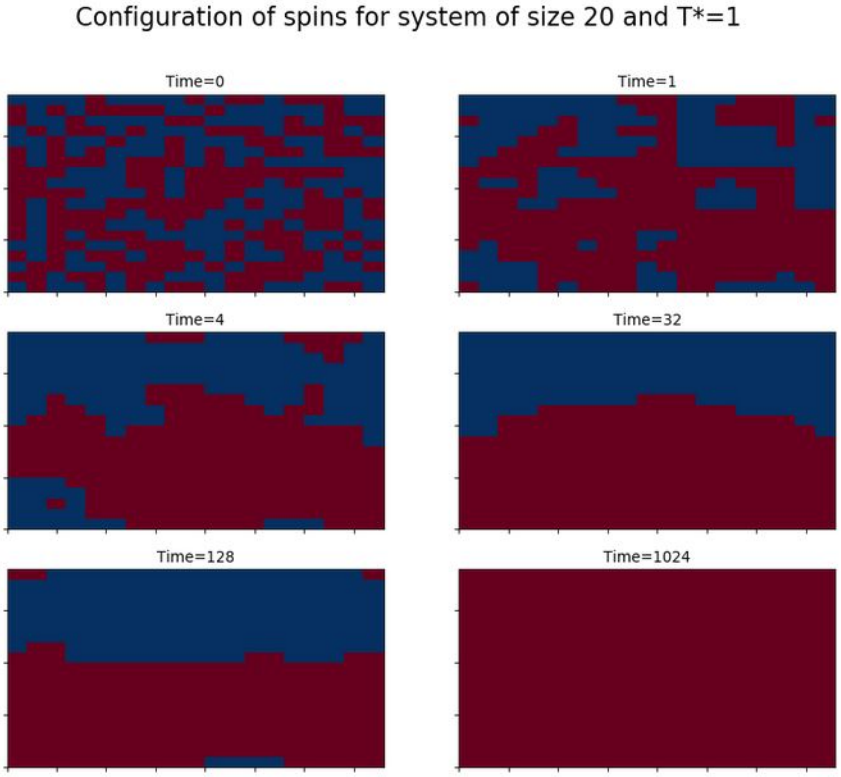
Task 3 - 2D Ising Model — 30.04, 2019

dr hab. inż. Grzegorz Pawlik *Author: Krzysztof Agieńczuk*

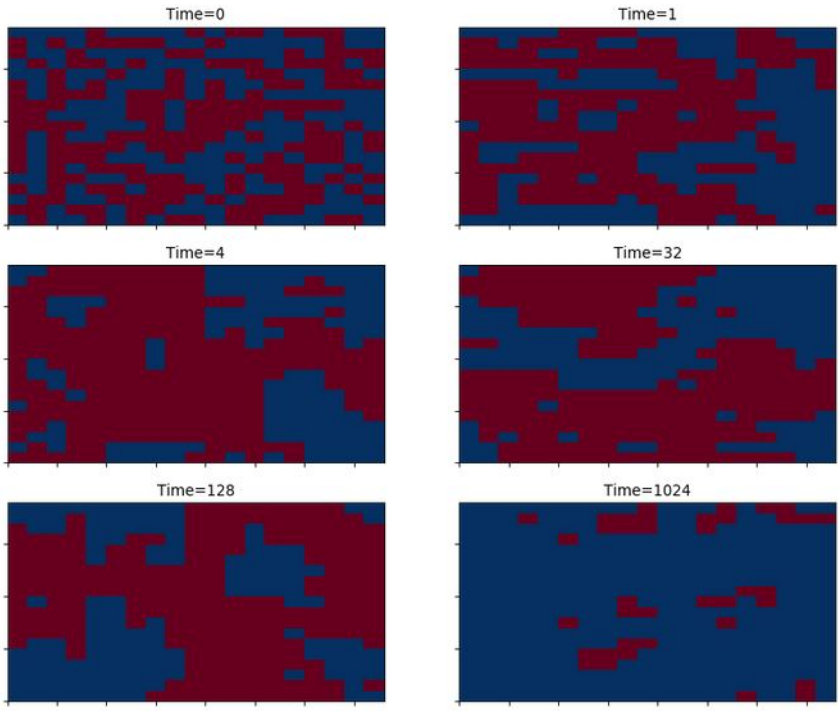
1 System behaviour

Note To improve readability of the document, all codes are placed in Appendix A.

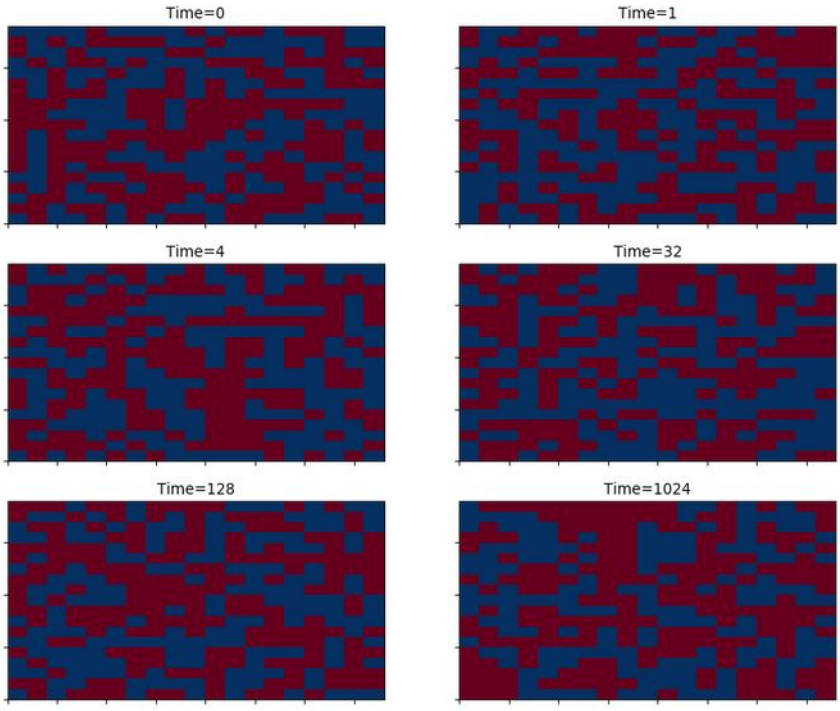
First task was to explore the behaviour of the system. Simulation was done for size `L=20 & L=100` and for temperature $T_C = 1$, $T_C = 2.26$ `&` $T_C = 5$.



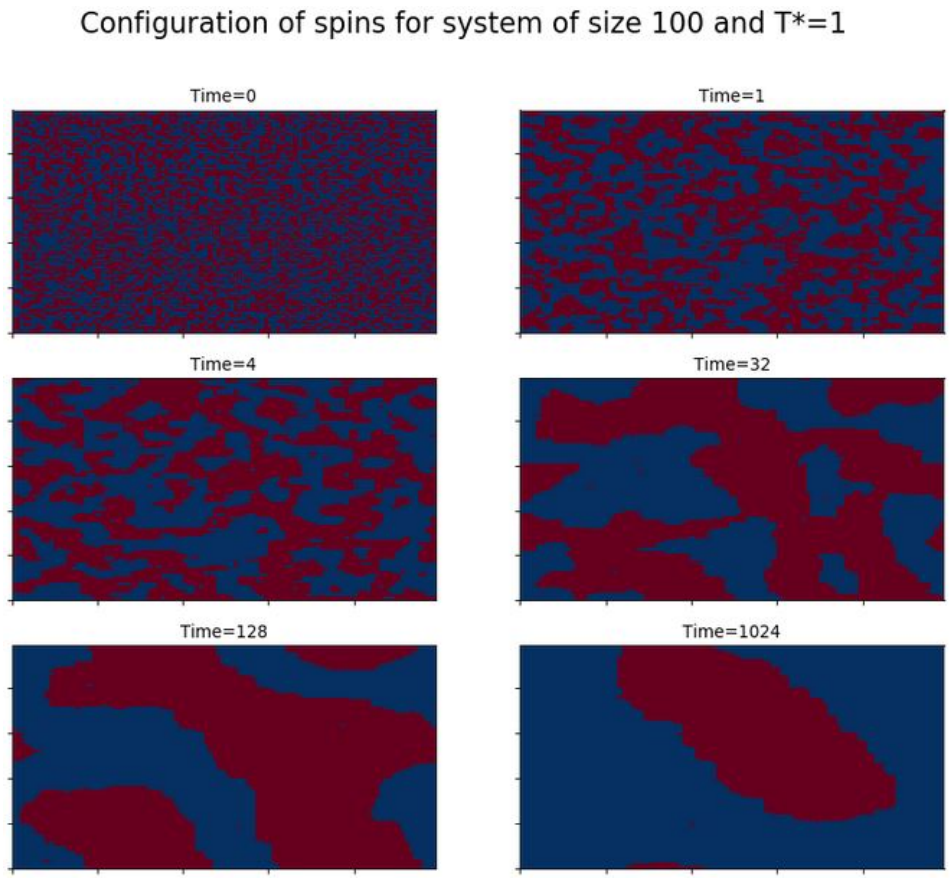
Configuration of spins for system of size 20 and T*=2.26

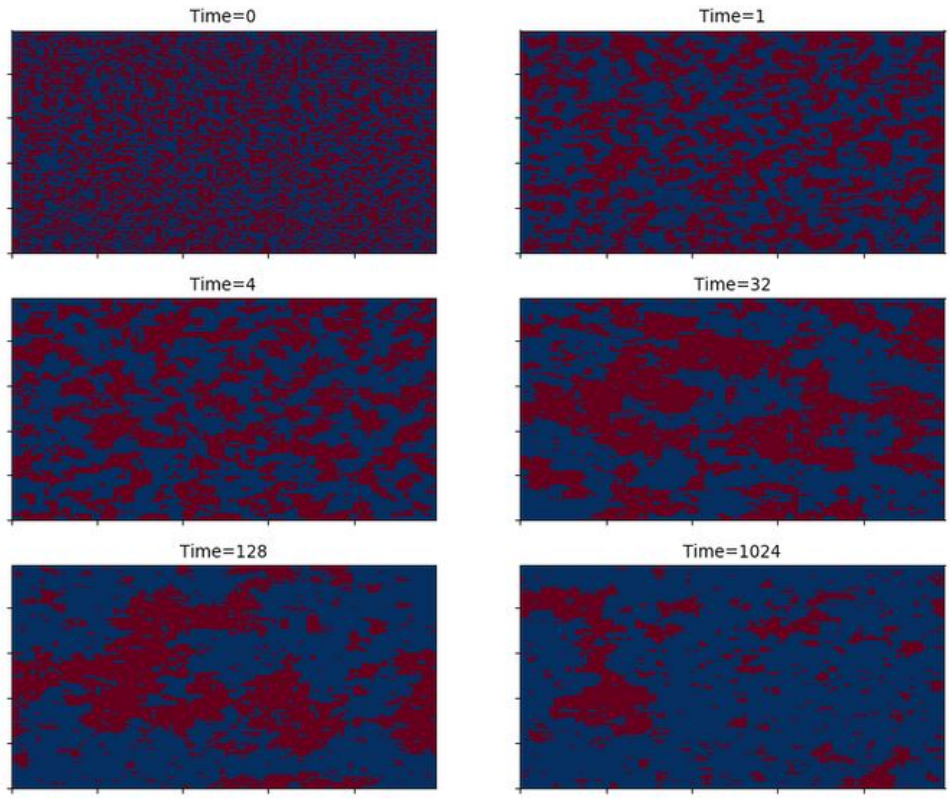


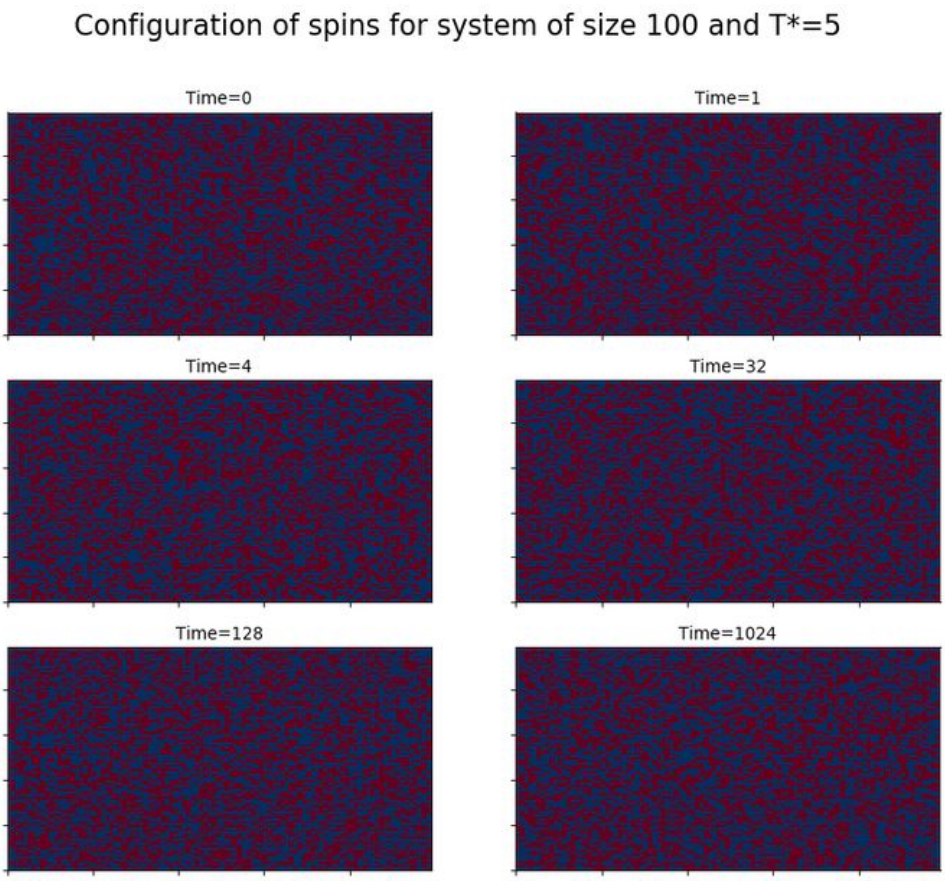
Configuration of spins for system of size 20 and T*=5



As seen above, for lower temperatures, small system cools down rather efficiently and quick. As we will see, it is not so efficient for larger systems. The higher the temperature and the bigger the size of the system, the longer it needs to cool down. For further experiments, 30000 Monte Carlo Steps were used to cool down the system, before calculating parameters.




Configuration of spins for system of size 100 and T*=2.26
Time=0
Time=1
Time=4
Time=32
Time=128
Time=1024

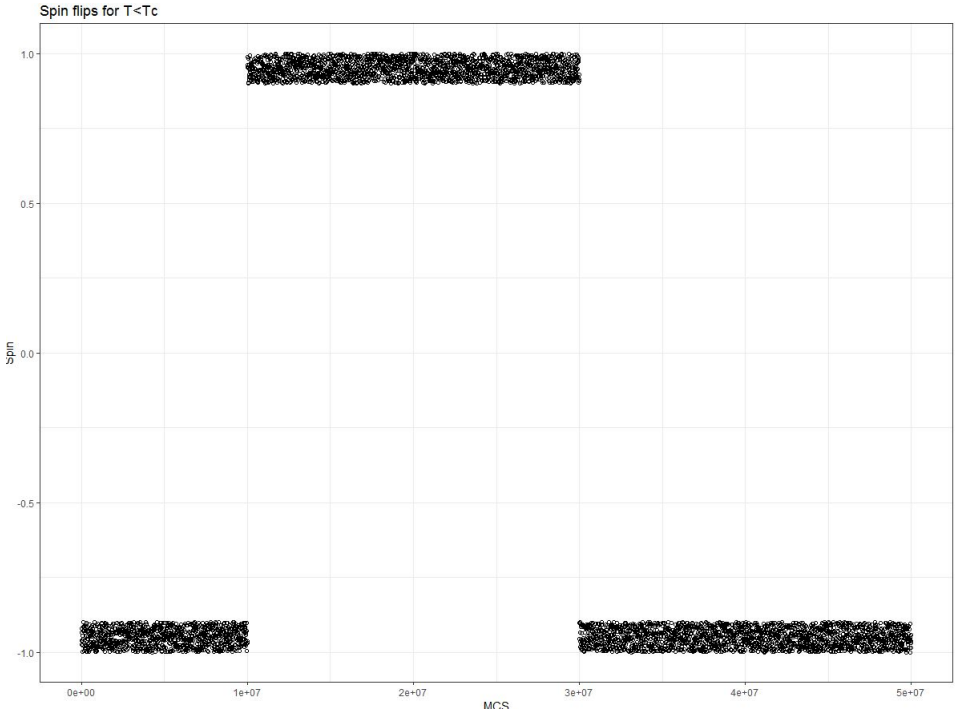


C++ was used for calculations. Code can be seen on listing 1.

Code for plotting is shown for only one of the plots, as it is highly repetitive. Initialisation of variables was skipped. As most of the plotting, this one was also done in Python. Unless specified otherwise, default programming languages were C++ for calculations and Python for plotting. Code can be seen in Appendix A, on listing 2

2 Flips between states

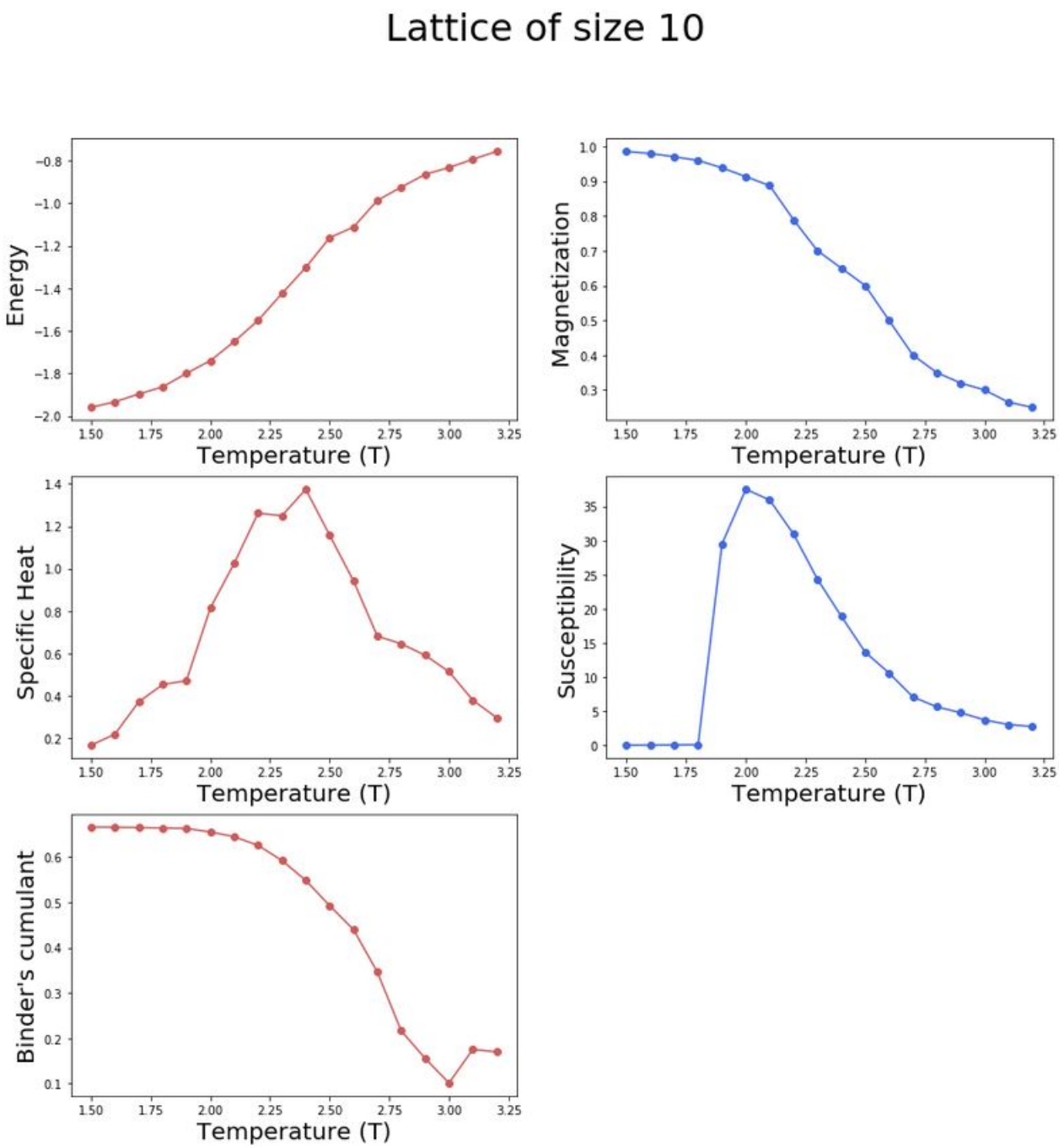
Below T_C system should spontaneously perform random flips in magnetisation from -1 to 1. This was observed only for huge number of MC steps. Every 1000th step was taken into account



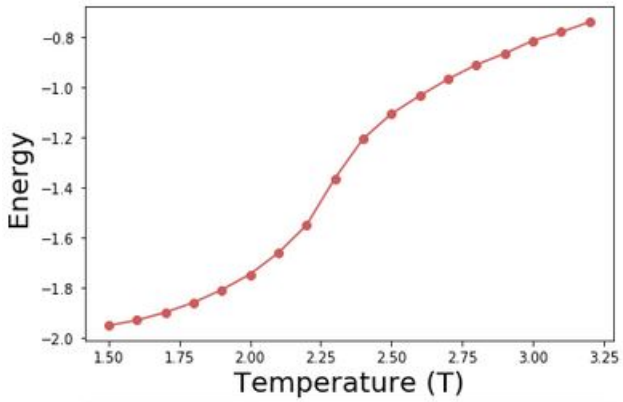
Code for plotting is available on listing 3 and is available in Appendix A.

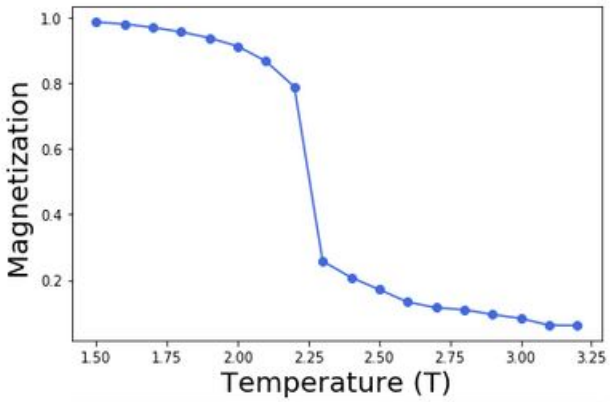
3 Temperature dependence on parameters

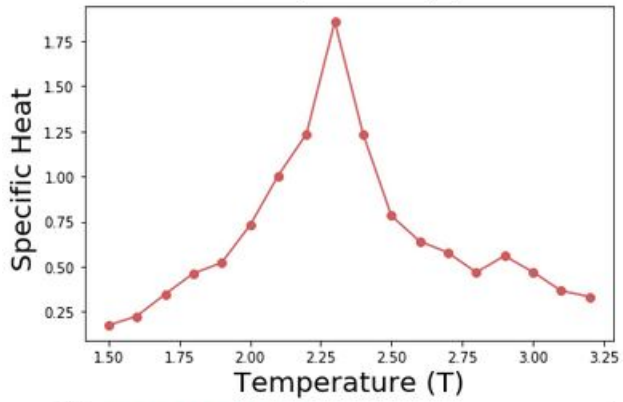
In this task, mean values of energy, magnetisation, susceptibility and specific heat were calculated. Also Binder's cumulant was used to determine T_C. Simulation lasted 230000 Monte Carlo Steps, including 30000 steps for cooling the system. Next every 1000th step was taken into account for calculations. Simulated lattices had sizes `L=10, L=50, L=100`.

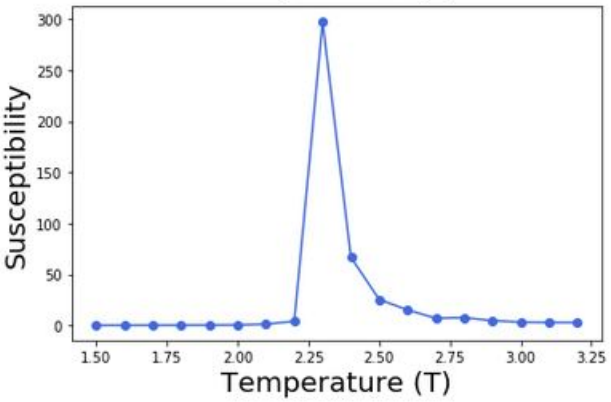


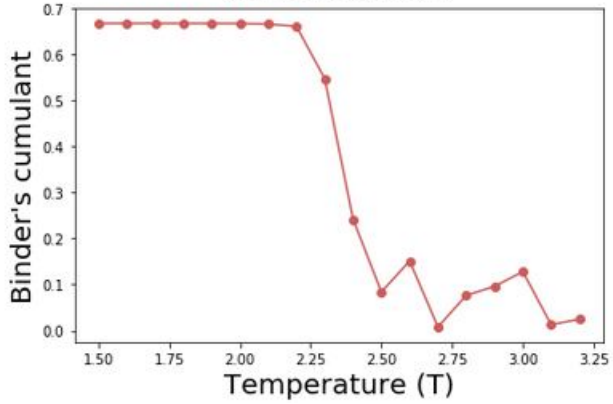
Lattice of size 50

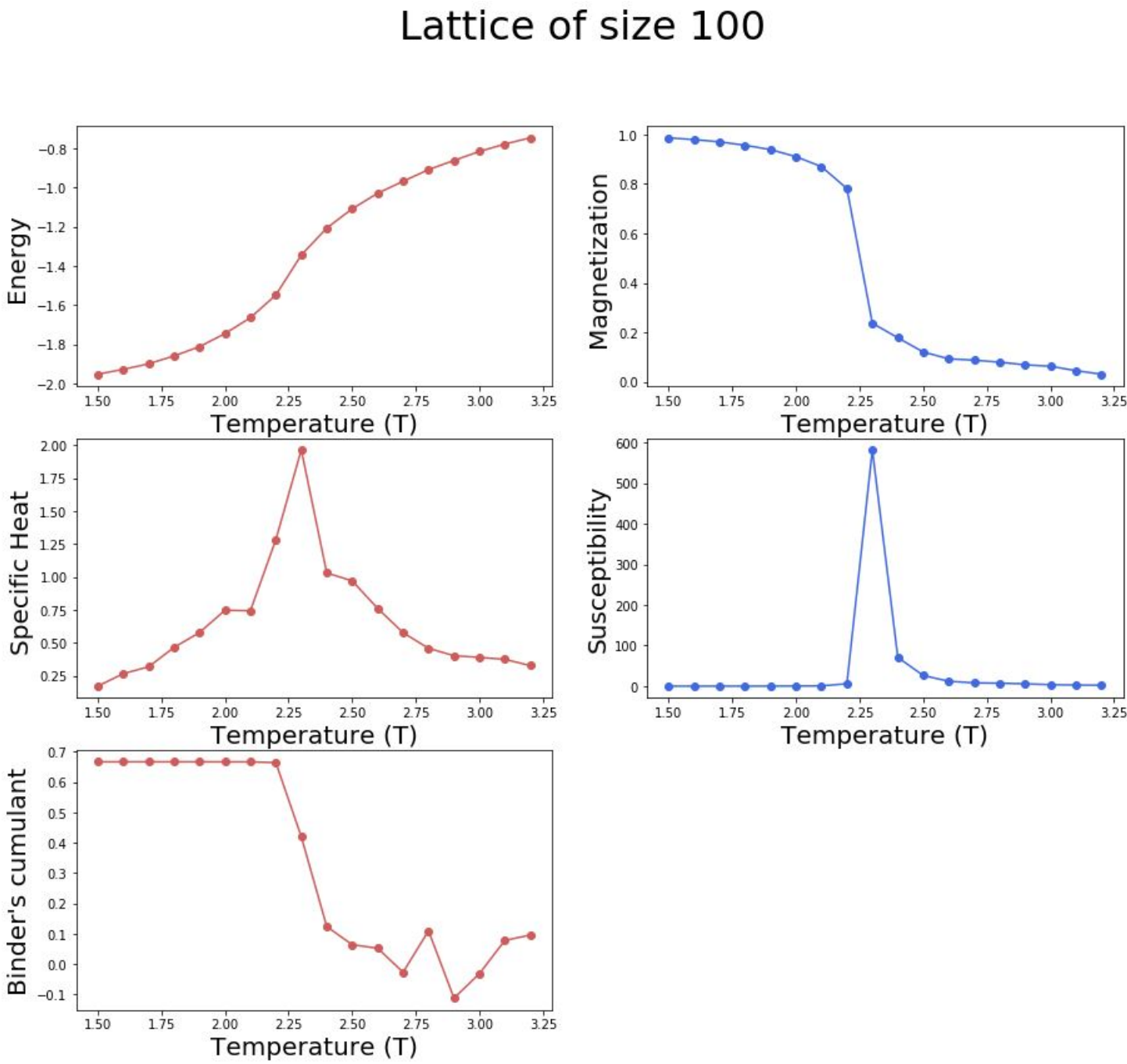










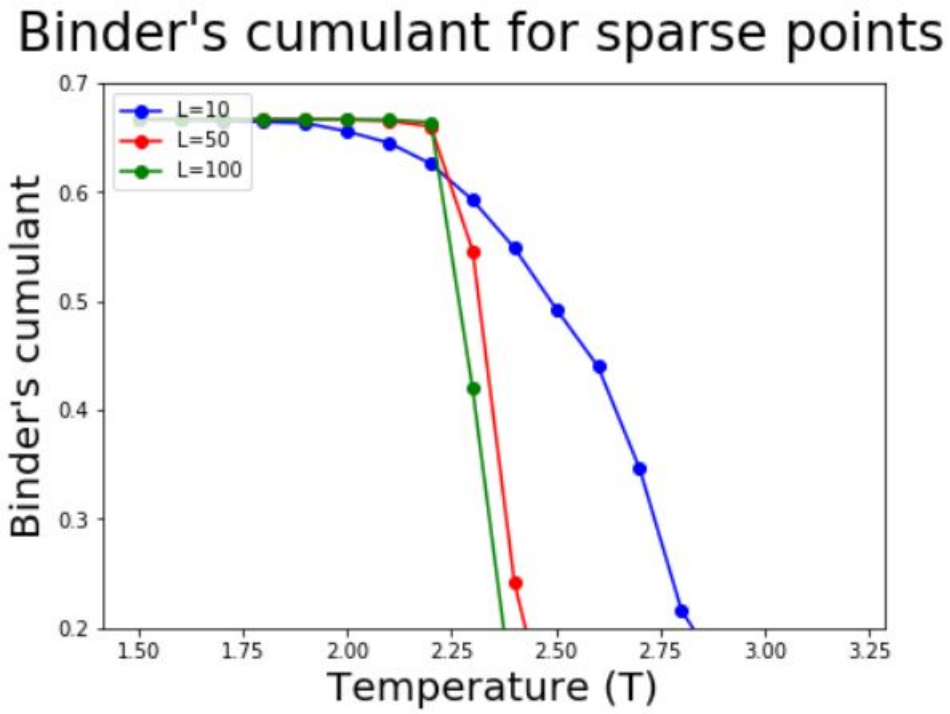


Code used for calculations was prepared in C++ and implemented in method testModel(). It is shown on listing 4

Code used for producing plots is presented on listing 5. Again, only one of plots is shown, due to repetitiveness of the code. Initialisation of variables is skipped.

3.1 Binder's cumulant

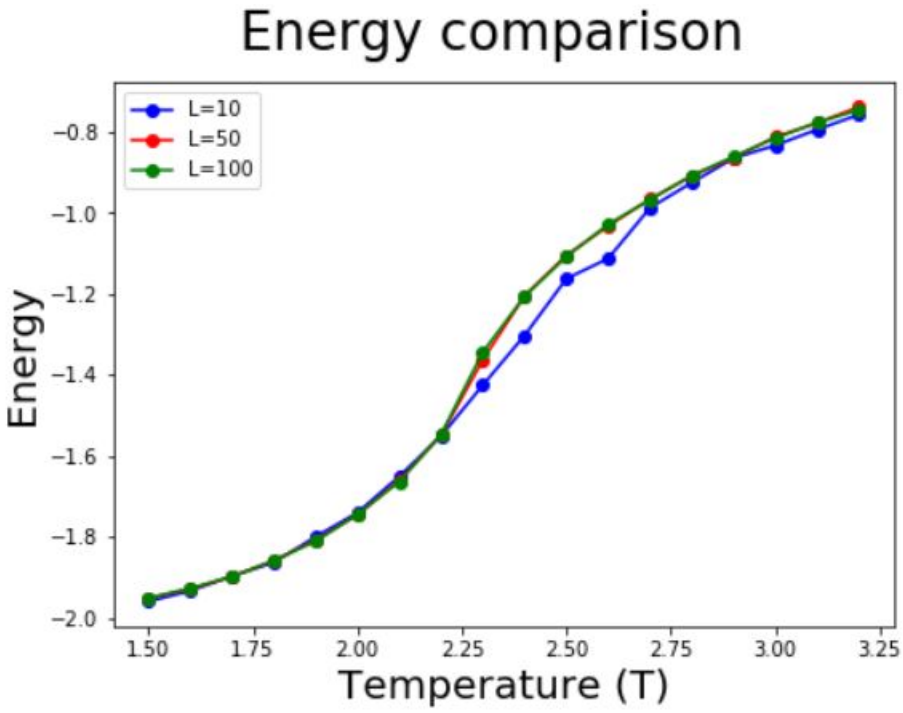
To determine critical temperature, Binder's cumulant was plotted on one graph for all tested system sizes (10,50,100). The result is shown below. Large drop can be seen for bigger systems around temperature 2.25-2.26. This is the critical temperature.

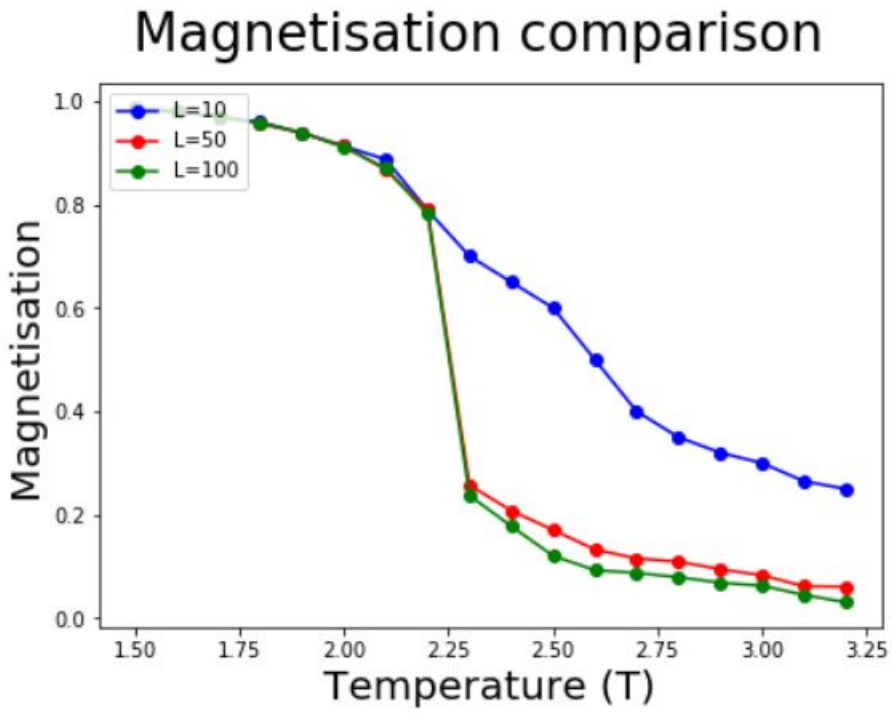


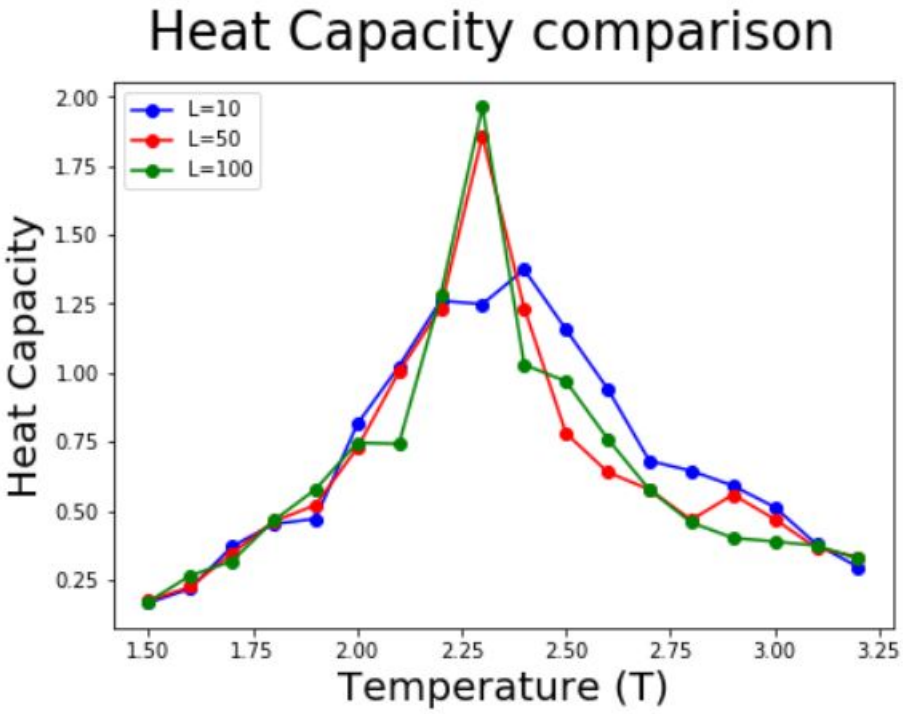
Code used for plotting is available on listing 6

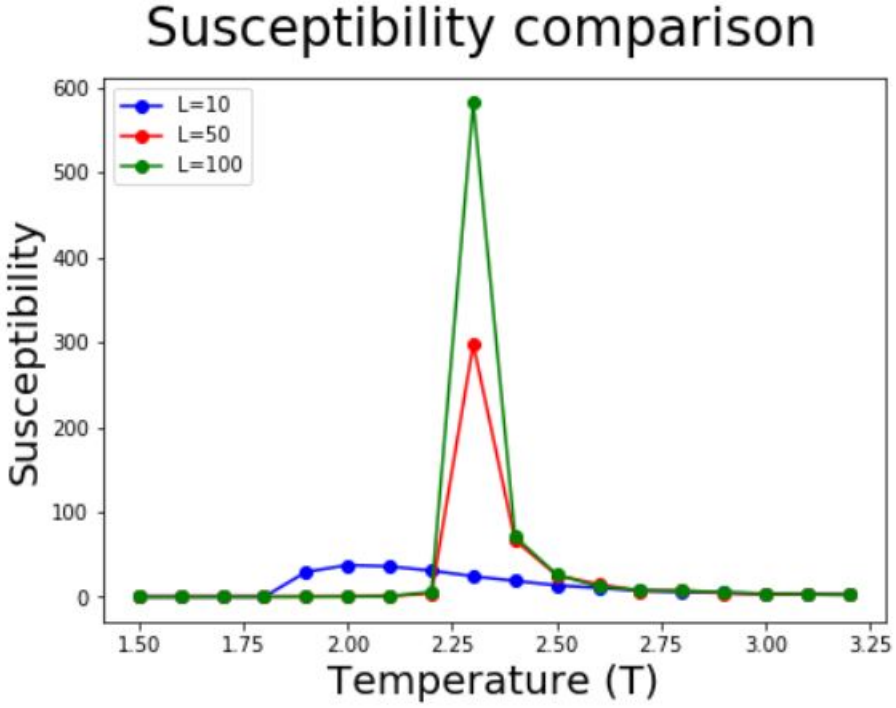
3.2 Comparing parameters

For easier comparison, all parameters were also plotted against each other. Code is analogous as one to plot Binder's cumulant and will be omitted.

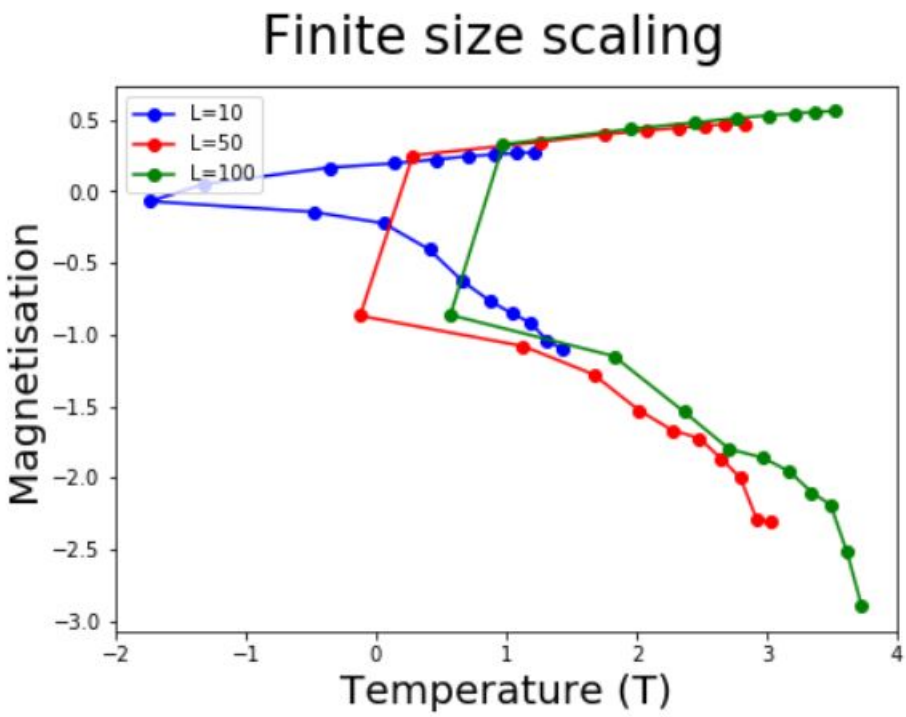

Energy comparison
L=10
L=50
L=100
Energy
−0.8
−1.0
−1.2
−1.4
−1.6
−1.8
−2.0
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25
Temperature (T)


Magnetisation comparison
L=10
L=50
L=100
Magnetisation
1.0
0.8
0.6
0.4
0.2
0.0
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25
Temperature (T)


Heat Capacity comparison
L=10
L=50
L=100
Heat Capacity
Temperature (T)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25


Susceptibility comparison
L=10
L=50
L=100
Susceptibility
Temperature (T)
600
500
400
300
200
100
0
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25

4 Finite size scaling



5 Appendix A

Listing 1: "C++ code used for observing of magnetisation flips"

```
void countSpins(double arr[LATTICE_SIZE][LATTICE_SIZE]){
   std::fstream m;
   m.open("spins.txt");

   double M11 =0;
   double mag = 0;
   double tab[2000] = {0};
   int position = 0;
   for(int i=0; i<OMITTED_STEPS; i++) mcStep(arr, 1.9);

   for(int i=0; i<1; i++){
      for(int ii=0; ii<MONTE_CARLO_STEPS; ii++) {
         mcStep(arr, 1.9);
         if(ii % 1000 == 0){
            mag = calculateMagnetisation(arr);
            M11 = mag/(LATTICE_SIZE*LATTICE_SIZE);
            tab[position] = M11;
            position++;
         }
      }

   for(int ii=0; ii<2000; ii++) m<<tab[ii]<<",";
   std::cout<<i<<std::endl;
   }

   m.close();
}
```

Listing 2: "Python code for plotting behaviour of the system."

```
f = plt.figure(figsize=(12, 10), dpi=80);
f.suptitle("Configuration of spins for system of size 20 and T*=1", fontsize = 20)
X, Y = np.meshgrid(range(20),range(20))
sp =  f.add_subplot(3,2,1)
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_0, cmap=plt.cm.RdBu);
plt.title('MCS=0'); plt.axis('tight')

sp =  f.add_subplot(3,2,2)
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_1, cmap=plt.cm.RdBu);
plt.title('MCS=1'); plt.axis('tight')

sp =  f.add_subplot(3,2,3)
```

```
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_4, cmap=plt.cm.RdBu);
plt.title('MCS=4'); plt.axis('tight')

sp = f.add_subplot(3,2,4)
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_32, cmap=plt.cm.RdBu);
plt.title('MCS=32'); plt.axis('tight')

sp = f.add_subplot(3,2,5)
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_128, cmap=plt.cm.RdBu);
plt.title('MCS=128'); plt.axis('tight')

sp = f.add_subplot(3,2,6)
plt.setp(sp.get_yticklabels(), visible=False)
plt.setp(sp.get_xticklabels(), visible=False)
plt.pcolormesh(X, Y, L20T1_1024, cmap=plt.cm.RdBu);
plt.title('MCS=1024'); plt.axis('tight')

plt.show()
```

Listing 3: "R code for plotting spin flips"

```
dt <- read.table("spins.txt", header = F, sep = ",")
df <- data.frame(dt)
z = seq(from = 1, to = 50000000, by=10000)

ggplot() +
geom_point(data = df, aes(x=z, y=dt), shape = 1) +
ggtitle("Spin flips for T<Tc") +
xlab("MCS") +
ylab("Spin") +
ylim(-1,1) +
xlim(0,50000000) +
theme_bw()
```

Listing 4: "C++ code for main calculations"

```
void testModel(double lattice[LATTICE_SIZE][LATTICE_SIZE]){
   // Prepare time series
   double T[TEMP_POINTS];
   for(int i=0; i<TEMP_POINTS; i++){
      T[i] = 2 + i*0.05;
   }
```

```
7
8     // Prepare helper variables
9     double ene =0, mag=0, E1=0, E2=0, M1=0, M2=0, M4=0, n1=0, n2=0, n4=0, iT=0, iT2=0;
10    n1 = 1/((MONTE_CARLO_STEPS/1000) * LATTICE_SIZE * LATTICE_SIZE);
11    n2 = 1/((MONTE_CARLO_STEPS/1000) * (MONTE_CARLO_STEPS/1000) * LATTICE_SIZE *
          LATTICE_SIZE);
12    n4 = 1/(MONTE_CARLO_STEPS/1000);
13
14    // Start model
15    for(int point=0; point<TEMP_POINTS; point++){
16       E1=0;E2=0;M1=0;M2=0;M4=0;
17
18       iT = 1/T[point];
19       iT2 = iT*iT;
20
21       // Omit steps
22       for(int i=0; i<OMITTED_STEPS; i++) mcStep(lattice, iT);
23
24       // "Real" calculation
25       for(int ii=0; ii<MONTE_CARLO_STEPS; ii++){
26          mcStep(lattice, iT);
27
28          // Take every 1000th configuration
29          if(ii%1000 == 0){
30             ene = calculateEnergy(lattice);
31             mag = calculateMagnetisation(lattice);
32
33             E1 += ene;
34             E2 += ene*ene;
35             M1 += mag;
36             M2 += mag*mag;
37             M4 += mag*mag*mag*mag;
38          }
39       }
40
41       E[point] = n1 * E1;
42       M[point] = abs(n1 * M1);
43       C[point] = (n1*E2 - n2*E1*E1)*iT2;
44       X[point] = (n1*M2 - n2*M1*M1)*iT;
45       U[point] = 1 - n4*M4/(3*M2*M2*n4*n4);
46       std::cout<<"Temp point: "<<point<<std::endl;
47    }
48
49    writeToFile();
50 }
```

Listing 5: "Python code for plotting main parameters"

```
1  f = plt.figure(figsize=(15, 15)); # plot the calculated values
2  f.suptitle("Lattice of size 50", fontsize = 32)
3
```

```
sp = f.add_subplot(3, 2, 1 );
plt.plot(T, E50, marker='o', color='IndianRed')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Energy ", fontsize=20);     plt.axis('tight');

sp = f.add_subplot(3, 2, 2 );
plt.plot(T, M50, marker='o', color='RoyalBlue')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Magnetization ", fontsize=20); plt.axis('tight');

sp = f.add_subplot(3, 2, 3 );
plt.plot(T, C50, marker='o', color='IndianRed')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Specific Heat ", fontsize=20); plt.axis('tight');

sp = f.add_subplot(3, 2, 4 );
plt.plot(T, X50, marker='o', color='RoyalBlue')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Susceptibility", fontsize=20); plt.axis('tight');

sp = f.add_subplot(3, 2, 5 );
plt.plot(T, U50, marker='o', color='IndianRed')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Binder's cumulant", fontsize=20); plt.axis('tight');
```

Listing 6: "Python plot for plotting Binder cumulants against each other"

```
fig = plt.figure(figsize=(7, 5))
fig.suptitle("Binder's cumulant for sparse points", fontsize = 26)
ax1 = fig.add_subplot(111)
axes = plt.gca()
axes.set_ylim([0.2,0.7])

ax1.plot(T, U10, c='b', marker="o", label='L=10')
ax1.plot(T, U50, c='r', marker="o", label='L=50')
ax1.plot(T, U100, c='g', marker="o", label='L=100')
plt.xlabel("Temperature (T)", fontsize=20);
plt.ylabel("Binder's cumulant", fontsize=20)

plt.legend(loc='upper left');
plt.show()
```

Listing 7: "C++ code for Monte Carlo Step"

```
void mcStep(double arr[LATTICE_SIZE][LATTICE_SIZE], double beta){
   for(int i=0; i<LATTICE_SIZE; i++){
      for(int ii=0; ii<LATTICE_SIZE; ii++){
         int a = rand() % LATTICE_SIZE;
         int b = rand() % LATTICE_SIZE;
```

```
6           double s = arr[a][b];
7           double nb = arr[mod(a+1, LATTICE_SIZE)][b]
8                   + arr[a][mod(b+1, LATTICE_SIZE)]
9                   + arr[mod(a-1, LATTICE_SIZE)][b]
10                  + arr[a][mod(b-1, LATTICE_SIZE)];
11          double cost = 2*s*nb;
12          double r = (((double) rand() / (RAND_MAX)));
13          if (cost<0) s *= -1;
14          else if ( r < exp(-cost*beta)) s*= -1;
15          arr[a][b] = s;
16      }
17   }
18 }
```