Simulation Methods in Physics I

Worksheet 5: Monte-Carlo

Students:	Michael Marquardt	Cameron Stewart
matriculation numbers:	3122118	3216338

1 Implementation in Cython

In order to make the given program faster one part of the monte_carlo_ising() function was implemented in Cython. Because of the already efficient numpy operations it is not necessary to implement the functions compute_energy() and compute_magnetization() in C. The function compute_act_error() may be faster in C but as we see later it will not be necessary.

The really time intensive part of the program is the trial move. In this case it is a spin-flip which is done L^2 times, with system size $L \times L$, between the num_sweeps measurements.

In order to make this faster the function spin_flip() is implemented in Cython. The according C-function is called c_spin_flip() and can be seen in code block 1. The function takes the Temperature in form of beta and the system size L as arguments. Furthermore it uses pointer on the energy E, the magnetization mu and the actual state sigma.

Code block 1: Spin flip script: cising_impl.c

```c
void c_spin_flip(double beta, long L, double* E, double*
    mu, int* sigma) {
    long V = L*L;
    for (long step=0; step<V; step++) {
        // k = i + j*L   =>   i = k%L   j = k//L
        long k = gsl_randint(V);
        // Test deltaE before flipping spin
        int deltaE = 2*sigma[k]*(sigma[k/L*L+(k+1+L)%L]+
            sigma[k/L*L+(k-1+L)%L]+sigma[(k+L+V)%V]+sigma[(
            k-L+V)%V]);
        if (gsl_rand() < exp(-beta*deltaE)) {
            sigma[k] *= -1;     // spin flip
            *E += deltaE;
            *mu += 2*sigma[k];
        }
    }
}
```

In fact the function does nothing different from the original python code, but because of C can only use plane arrays the indices will look different.

$$\sigma_{i,j} \rightarrow \sigma_{i+L\cdot j} = \sigma_k \tag{1}$$

$$i = k\%L \qquad \text{(modulo)} \tag{2}$$

$$j = k/L \qquad \text{(integer division)} \tag{3}$$

This leads to:

$$\sigma_{i\pm1,j} \to \sigma_{(i\pm1)\%L+L\cdot j} = \sigma_{(k\pm1)\%L+k/L\cdot L} \tag{4}$$

$$\sigma_{i,j\pm1} \to \sigma_{i+[(j\pm1)\%L]\cdot L} = \sigma_{(k\pm L)\%L^2} \tag{5}$$

But in C the modulo of negative numbers is defined in a different way than in python and because of this when using the modulo $a\%b$ we have to add b to a to gain $(a+b)\%b$.

In order to save disk space no new state sigma is generated but the old one will be overwritten. Same for E and mu. They need not to be calculated again but can be simply changed by their change for one spin flip.

The function can now replace the inner for-loop in monte_carlo_ising().

2 Simulation

In order to make easier simulations ising.py was extended to take the command line parameter −−L <L1> <L2> ... which takes all system sizes for which you want to simulate as arguments. Furthermore the script can now save figures and data.

3 Binder Parameter

The Binder parameter U is defined by:

$$U = 1 - \frac{1}{3}\frac{\langle\mu^4\rangle}{\langle\mu^2\rangle^2} \tag{6}$$

And it is implemented in python just like this:

Code block 2: Binder parameter script: ising.py

```python
def binder_parameter(ms):
    '''
    computes the binder parameter U=1-1/3*<mu^4>/<mu^2>^2
    '''
    return 1-1./3.*(ms**4).mean()*(ms**2).mean()**-2
```

4 Actual Results

When looking on the states in figure 1 to 3 you can see, that for lower temperatures there is much more order into the arrangement of spins. There are big areas which have the same spin all over them. But what is also the case is that it must not be the whole area of simulation which has the same spin. Because of this there are some low temperatures in figure 4 for which the magnetization has an very low value.

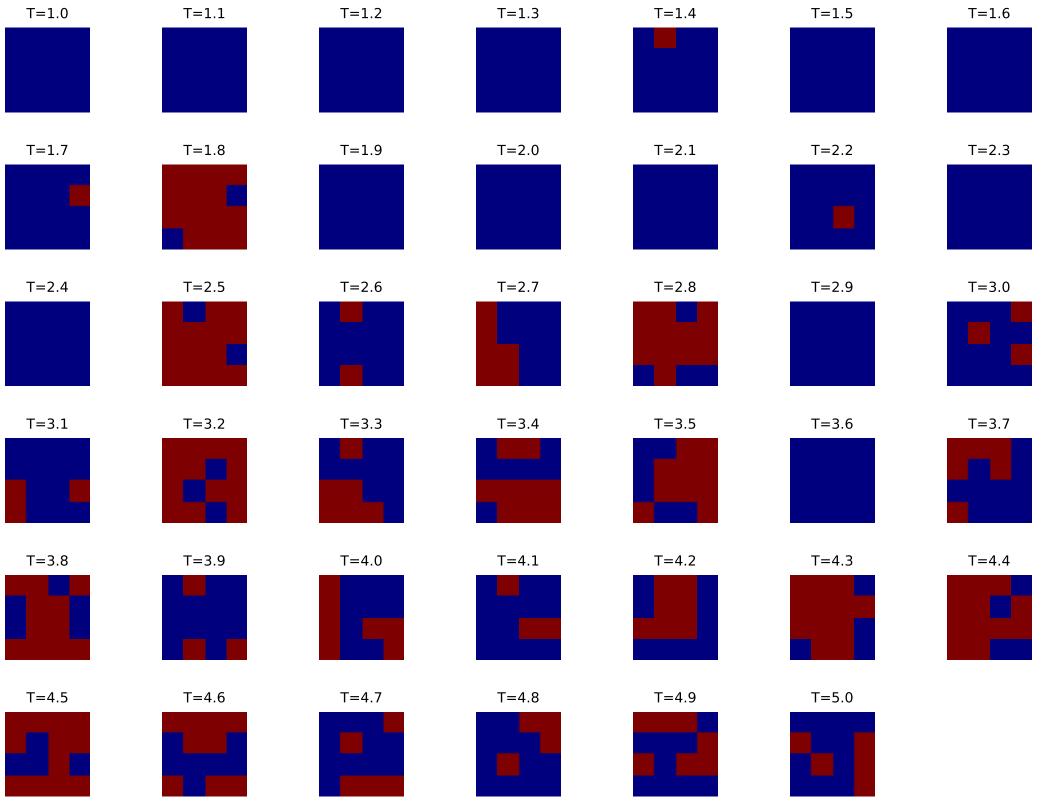

Fig. 1: Final states for L=4.

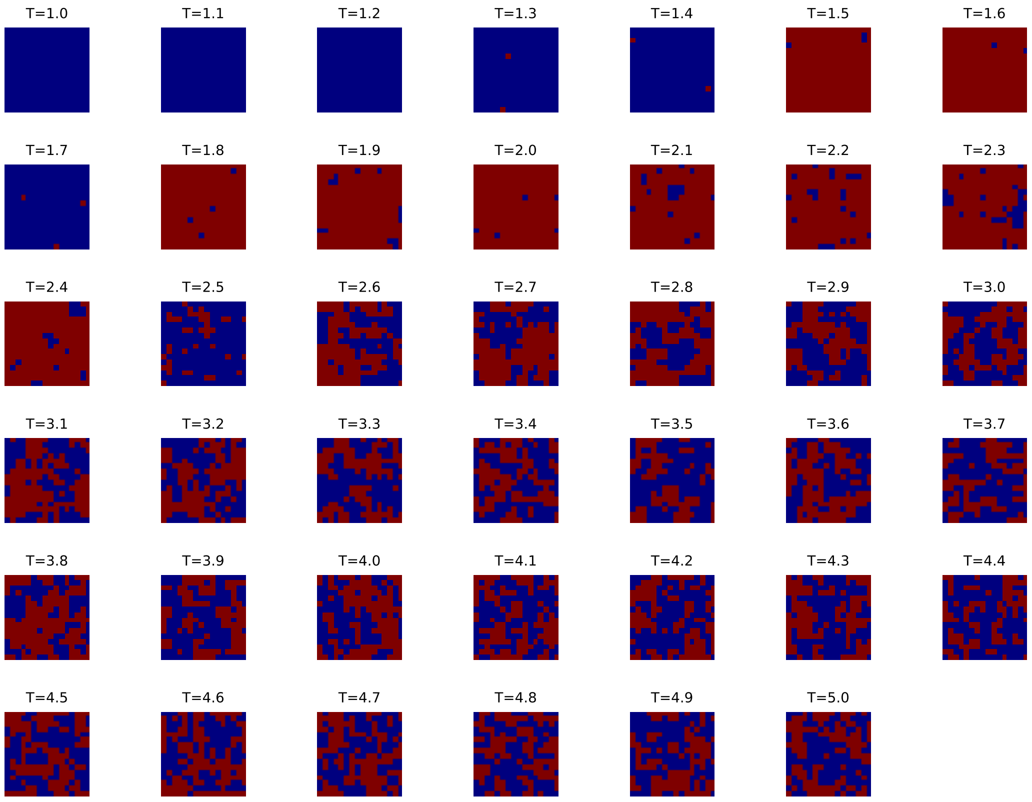

Fig. 2: Final states for L=16.

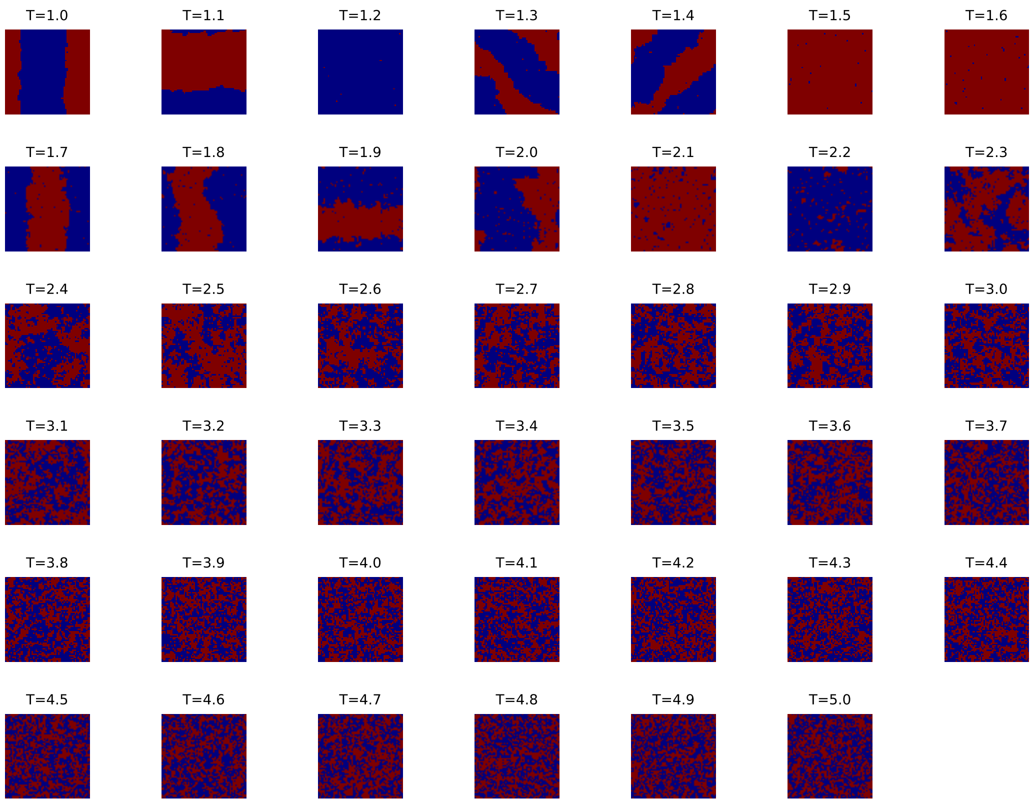

Fig. 3: Final states for L=64.

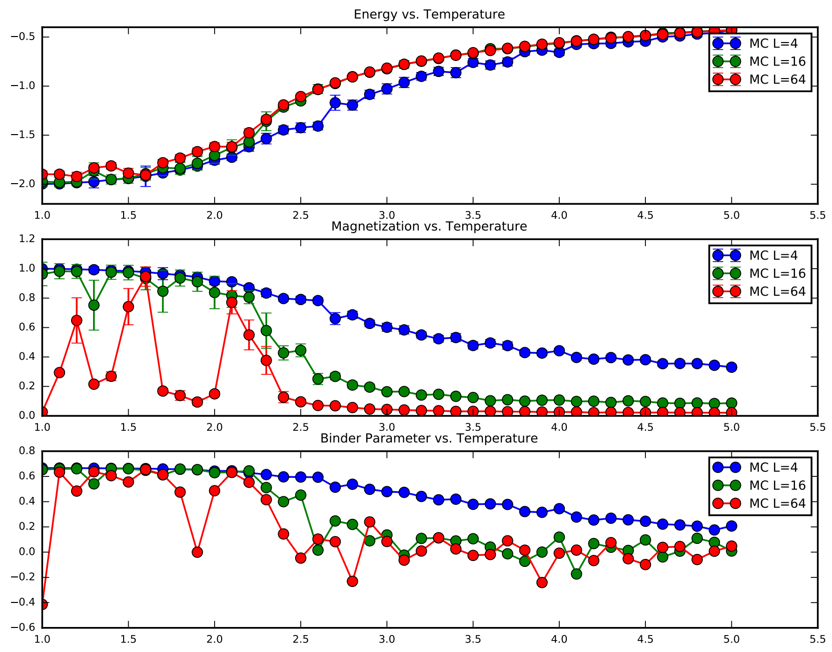

Fig. 4: Measurements for the different system sizes L.