Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
Kai Meinerz, Jan Attig

Computational Many-Body Physics

Exercise Sheet 4

Summer Term 2021

Due date: Monday, **14th June** 2021, 10 am

Website: thp.uni-koeln.de/trebst/Lectures/2021-CompManyBody.shtml

This week's sheet is devoted to **quantum Monte Carlo** techniques. We will first guide you through an implementation of the stochastic series expansion and use it to study a *thermal* phase transition in a quantum spin model. In a second part, we will then look at a paradigmatic *quantum* phase transition – the field-driven phase transition in the transverse field Ising model – and study it using an open-source loop code implementation available as part of ALPS (Algorithms and Libraries for Physics Simulations).

Exercise 8: Stochastic Series Expansion

We start by looking for footprints of a thermal phase transition in the quantum spin-$\frac{1}{2}$ Heisenberg model on the cubic lattice, which is described by the Hamiltonian

$$H = -J \sum_{\langle i,j \rangle, \gamma} \hat{S}_i^\gamma \hat{S}_j^\gamma = -J \sum_{\langle i,j \rangle} \left(\hat{S}_i^z \hat{S}_j^z + \hat{S}_i^y \hat{S}_j^y + \hat{S}_i^x \hat{S}_j^x \right) , \tag{1}$$

where the spin operators $\hat{S}_i^\gamma$ now refer to Pauli matrices. We set the coupling constant to $J = -1$ for antiferromagnetic interactions and sum runs over all pairs of nearest neighbors of the lattice. To study this model we want using quantum Monte Carlo simulations ased on the stochastic series expansion (SSE). At its heart, SSE is based on rewriting the Boltzmann factor in the partition function as a Taylor expansion [1]

$$Z = \mathrm{Tr} e^{-\beta H} = \sum_\alpha \sum_{n=0}^{\infty} \left\langle \alpha \left| \frac{(-\beta H)^n}{n!} \right| \alpha \right\rangle . \tag{2}$$

It is particularly useful to rewrite the Hamiltonian in terms of contributions of every bond

$$\begin{aligned} H &= -\sum_b H_b & (3) \\ &= -\sum_b \left[\underbrace{\left(\hat{S}_{b_1}^z \hat{S}_{b_2}^z - \frac{1}{4} \right)}_{\text{diagonal}} + \underbrace{\frac{1}{2} \left(\hat{S}_{b_1}^+ \hat{S}_{b_2}^- + \hat{S}_{b_1}^- \hat{S}_{b_2}^+ \right)}_{\text{off-diagonal}} \right] + const. & (4) \end{aligned}$$

The constant absorbs the shift of the Hamiltonian induced by the artificially introduced addition of $\frac{1}{4}$ per bond. This form of the Hamiltonian is perfectly adapted to the needs of SSE, namely that all non-zero matrix elements ($\langle\uparrow\uparrow|H_b|\uparrow\uparrow\rangle, \langle\downarrow\downarrow|H_b|\downarrow\downarrow\rangle, \langle\uparrow\downarrow|H_b|\downarrow\uparrow\rangle, \langle\downarrow\uparrow|H_b|\uparrow\downarrow\rangle$) have the same value of $\frac{1}{2}$ in this case.

[1] A good reference for an introduction to SSE is this book chapter by Anders Sandvik at arXiv:1909.10591.

The first part of the exercise will focus on implementing the SSE algorithm. For this we will divide its implementation into multiple parts, separately discussing diagonal and off-diagonal updates.

a) To set the stage, you need to implement a **cubic lattice** and all the necessary data structures. For starters, you need an array containing all the spin information of the lattice and an array for all applied operators. What size does the operator array need to be? Since we will loop over all nearest bonds of the lattice, you need to define them (note that you might want to implement periodic boundary conditions).

b) In the **diagonal update** part of the simulation, diagonal operators are added/removed with a probability p_{add}/p_{rem}. How are the probabilities defined? There are no constraints on removing a diagonal operator, while inserting it is allowed only if the bond spins are antiparallel. Encountering an off-diagonal operator will cause the spin states at the bond to flip. Implement a function for the diagonal updates. To test your function, apply it repeatedly to an initial spin state and count the number of diagonal operators at the end. Repeat this process for different temperatures and create a histogram of the operator count.

c) The **off-diagonal updates** always involve updating at least *two* operators. To perform the updates efficiently, we will implement a loop update algorithm. For the loop construction, it is useful to introduce another data structure in which the *connectivity* of the operators is explicitly represented. This *vertex* structure is a spin operator object that contains information about the operator and the connected spins Fig. [1].
The vertices are connected by the operators applied to the mutual spins. This is best done by keeping a list that assigns an index of the connected vertex to each vertex. This list can then be updated by iterating over each vertex and checking for a connection.Write a linkvertex() function and apply it to a random spin/operator configuration. Check that the resulting vertex loops are closed.

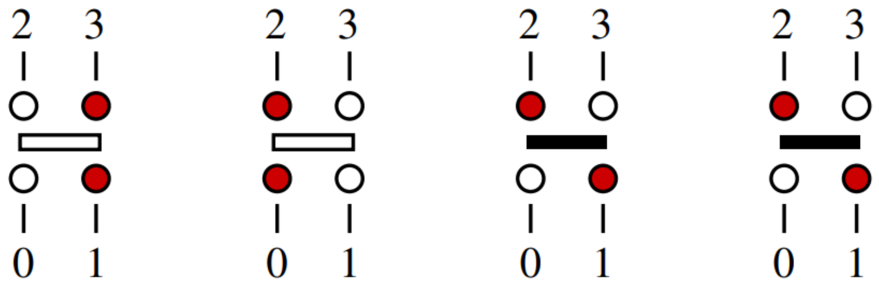

Figure 1: The different **allowed vertices**. The red/white represent the spins in the ±1 state and white/black bars represent the diagonal/off-diagonal operators. The numbering of the spins corresponds to the state of the connected spins before and after the operator acts.

d) Now we only need to perform the off-diagonal update, by flipping the **vertex loops** with a probability of 1/2. Flipping the loops will change the involved operators from diagonal to off-diagonal operators and vice versa, additionally all spins not connected to any operator are flipped.

In order to run the full simulations, you now need to put the different parts together. A single sweep of the simulation is done by performing the diagonal update, creating the vertices loop and performing the loop updates.

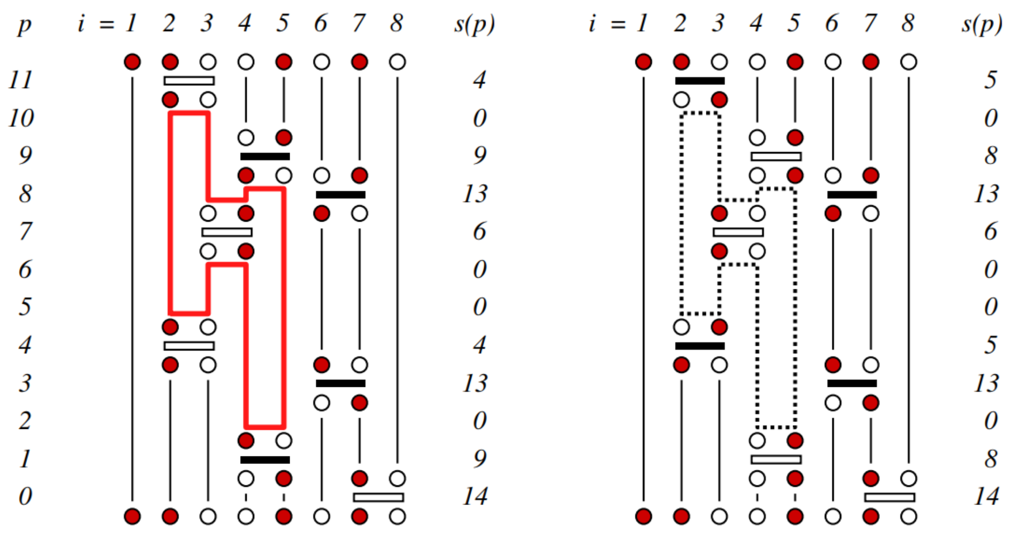

Figure 2: **A linked-vertex SSE configuration**. The left side shows a constructed loop of connected vertices before flipping it and the right side shows the configuration after flipping the loop. All spin orientations are flipped and the operators are changed from diagonal to off-diagonal and vice versa.

In general, we would have to determine the maximum expansion order Λ – a step which is typically done during a thermalization run. Here we will simply fix it to $\Lambda = 2\beta L^3$.

e) Perform simulations for inverse temperatures $\beta = 0.2, 0.4, 0.6, \ldots 5.0$ and varying system sizes $L = 8, 12, 16$. Measure the **staggered magnetization** m_s and plot $m_s L^3/(L^{(2-0.034)})$ against the inverse temperature β. The staggered magnetization is calculated by spatially alternating the sign of spins. For example, in a 2 by 2 square lattice the staggered magnetization would be $m_s = \pm(S_{11} - S_{12} - S_{21} + S_{22})$. It should be sufficient to do 10^4 measurement steps (sweeps) after a considerable thermalization time (usually 10% of the number of sweeps).

f) Remember that the **energy** can be elegantly obtained by simply measuring the expansion order n as $\langle E\rangle = -\frac{\langle n\rangle}{\beta}$. Plot the energy per spin against the inverse temperature β. What observable can you deduce from the energy that will allow you to clearly identify signatures of the thermal phase transition?

Similar to the lecture video, you might also want to plot the **average expansion order** $\langle n\rangle$ or, even better, a histogram of the sampled expansion orders as a function of the inverse temperature.

Exercise 9: Transverse-field Ising Model

Now we want to take look at a true *quantum* phase transition. For this, we consider the transverse field Ising model (TFIM) in one spatial dimension. Its Hamiltonian includes an external magnetic field perpendicular to the direction of the Ising interaction and reads

$$H = -J \sum_{\langle i,j \rangle} \hat{S}_i^z \hat{S}_j^z - h \sum_j \hat{S}_j^x \, . \tag{5}$$

We consider a ferromagnetic coupling $J = 1$ and vary the external field strength h.

While we could, in principle, use the stochastic series expansion code from above, we suggest that you instead use an open-source code implementing highly efficient loop updates – the `loop` code from the ALPS project (Algorithms and Libraries for Physics Simulations) .

a) The first step therefore is to install the ALPS libraries. Head to the ALPS website and follow the installation instructions or you can follow along our prerecorder installation video on vimeo, where we show the installation and usage of ALPS. (For Windows users: We recommend using the stable ALPS release version 2.1)

b) In order to get acquainted with the general ALPS framework of input and output files you should repeat your *classical* Ising Monte Carlo calculations from one of the earlier exercise sheets. Use the ALPS `spinmc` code to calculate once again the magnetization, energy, specific heat and the Binder cumulant for the ferromagnetic ordering transition in a two-dimensional Ising model.

To get you started, we provide you with an example for setting up the simulation for a square lattice Ising model using the raw text input 1 and an example using a python wrapper 2.

c) In a next step, you can now perform a simulation using the ALPS `loop` code for the 1D transverse field Ising model of size $L = 16$ for an external field in the range $h \in [0, 3]$ and temperatures $T \in [0.01, 1]$. Plot the resulting magnetization $M(h)$ and susceptibility $\chi(h)$.

Again, we provide with an example simulation for the loop algorithm 3.

d) Now you can zoom in on the quantum phase transition by fixing a low temperature $T = 0.01$ and simulate chains of varying system sizes $L = [8, 16, 32, 64]$, again for a range of field strengths $h \in [0, 3]$. Plot the resulting magnetization $M(h)$ and susceptibility $\chi(h)$.

Listing 1: Example of a text input file for a classical Monte Carlo simulation of the Ising model at its critical temperature for different system sizes.

```
LATTICE="square lattice"
T=2.269186
J=1
THERMALIZATION=10000
SWEEPS=50000
UPDATE="local"
MODEL="Ising"
{L=2;}
{L=4;}
{L=8;}
{L=16;}
```

Listing 2: Example of a python wrapper that runs the ALPS `spinmc` code for an Ising model on the honeycomb lattice for different system sizes and temperatures and creates a plot of the energy versus temperature.

```python
import pyalps
import matplotlib.pyplot as plt
import pyalps.plot
import numpy as np

Ls = [4,8]
Ts = np.arange(1.0, 4.0, 0.1)

parms = []
for L in Ls:
    for T in Ts:
        parms.append(
            {
                'LATTICE'         : "honeycomb lattice",
                'T'               : T,
                'J'               : 1 ,
                'THERMALIZATION' : 10000,
                'SWEEPS'          : 50000,
                'UPDATE'          : "local",
                'MODEL'           : "Ising",
                'L'               : L
            }
        )

input_file = pyalps.writeInputFiles('parm1a',parms)
pyalps.runApplication('spinmc',input_file,Tmin=1,writexml=True)

data = pyalps.loadMeasurements(pyalps.getResultFiles(prefix= \
                'parm1a'),'|Magnetization|')
magnetization = pyalps.collectXY(data,x='T',
                y='|Magnetization|',foreach=['L'])

plt.figure()
pyalps.plot.plot(magnetization)
plt.xlabel('Temperature $T/J$')
plt.ylabel('Magnetization $M$')
plt.xlim(1,4)
plt.legend()
plt.show()
```

Listing 3: Example of a python wrapper for running the ALPS `loop` code for the one-dimensional transverse field Ising model.

```python
import pyalps
import matplotlib.pyplot as plt
import pyalps.plot
import numpy as np

Ls = [8]
hs = np.linspace(0.0, 1.5, 31)
parms = []
for h in hs:
    for L in Ls:
        parms.append(
            {
                  'LATTICE'        : "chain lattice",
                  'MODEL'          : "spin",
                  'local_S'        : 0.5,
                  'L'              : L,
                  'Jxy'            : 0,
                  'Jz'             : -1,
                  'Gamma'          : h,
                  'BETA'           : 100.0,
                  'THERMALIZATION' : 10000,
                  'SWEEPS'         : 10000,
                  'ALGORITHM'      : 'loop'
            }
        )
input_file = pyalps.writeInputFiles('1D-SSE-TFIM', parms)
res = pyalps.runApplication('loop', input_file, Tmin=1)


data = pyalps.loadMeasurements(pyalps.getResultFiles(prefix= \
                '1D-SSE-TFIM'),'|Magnetization|')
magnetization = pyalps.collectXY(data,x='Gamma',
                y='|Magnetization|',foreach=['L'])

magnetization[0].y /= (L/2)
magnetization[0].x *= 2

plt.figure()
pyalps.plot.plot(magnetization)
plt.xlabel('Magnetic Field h')
plt.ylabel('|Magentization|')
plt.legend()
plt.show()
```