In the following, I give a little tutorial on how to compile and run the various Monte Carlo simulations that put out the (multicanonical) time series. First, we create a local copy from the remote `github`-repository executing the following line on a command shell.

```
$ git clone --recursive \
  https://github.com/four-spins/dissertation \
  dissertation_reproduce
```

This creates a directory called `dissertation_reproduce`, that contains several subdirectories, each one corresponding to a set of specific simulations. `CMake` eases the build process for the individual programs.

```
dissertation_reproduce
 |
 |__ original_periodic ............ 3d plaquette model with periodic boundary conditions
 |    |__ src
 |    |__ reproduce.py
 |    |__ CMakeLists.txt
 |    |__ CMakeLists.repro.txt
 |__ original_fixed ............... 3d plaquette model with fixed boundary conditions
 |    |__ src
 |    |__ CMakeLists.txt
 |__ Anisotropic_Ashkin_Teller .... 3d dual model with periodic boundary conditions
 |    |__ src
 |    |__ config
 |    |__ CMakeLists.txt
 |__ hysteresis ................... hysteresis curves for the dual model.
 |    |__ src
 |    |__ config
 |    |__ CMakeLists.txt
 |__ analyse_data ................. tools for data analysis
 |__ external_libraries ........... dlib, dsfmt, gzstream
```

1 Simulating the Three-Dimensional Plaquette Model with Periodic Boundary Conditions

We show how to reproduce the data for the $L = 4$ model under periodic boundary conditions. Since we want to simulate the original, $3d$ plaquette model, we need to enter the appropriate subdirectory by typing into a shell:

```
$ cd dissertation_reproduce/original_periodic
```

The simulations are configured by a header `config.hpp` in the subdirectory `src`. We set the lattice size in that file (and choose other parameters to our liking) with the line

Table 1: Seeds for the random number generator for reproducing the identical results of Section **??** for the $3d$ plaquette model.

L	seed	L	seed	L	seed	L	seed
4	877999	10	607688	16	604557	22	620240
5	793557	11	595393	17	978687	23	760296
6	887979	12	795316	18	38116	24	321213
7	105099	13	965577	19	745300	25	562871
8	545336	14	484040	20	569718	26	61145
9	684037	15	271837	21	447785	27	850676

```
const int L = 4;
```

If the want to reproduce the identical time series that was analysed in this thesis, we need the same sequence of random numbers to create the original Markov Chain. From the list of seeds for all lattice sizes in Table 1 we find that we need to set

```
const int seed = 877999;
```

for $L = 4$ in the configuration file `src/config.hpp`. `CMake` is then best used from a separate working directory, therefore we type

```
$ mkdir L4 && cd L4
```

to create and enter a novel subdirectory with the name `L4`. The commands

```
$ cmake .. -DCMAKE_BUILD_TYPE=release
$ make
```

create the executable of the simulation. We can also create an unoptimised executable that does very slow consistency checks (like calculating and comparing the energy after each update by going through the full lattice) by setting the parameter `-DCMAKE_BUILD_TYPE=debug`. This is meant for debugging only and should not be used for long simulations or large lattices.

Finally, we let run the simulation

```
$ ./goni3d_rec_muca.x
```

which will create several (human-readable text-) files in the current working directory[1] upon a successful run that are listed in the following.

[1]Keep in mind that, when starting several executables by a script, the executables will recklessly overwrite data in the current working directory.

L4

`simulation.dat`	overview: parameters, convergence, timings
`weights.dat`	multicanonical weights
`measurements.dat`	multicanonical time series of measurements
`energy_histogram.simu.dat`	multicanonical histogram
`energy_histogram.cano.dat`	canonical histogram for β=0
`measurements.full.dat`	full time series of the energy (i.e., measured every sweep)
`energy_histogram.simu.full.dat`	multicanonical energy histogram of the full time series
`energy_histogram.cano.full.dat`	canonical energy histogram at $\beta = 0$ for the full time series
`checkpoint_{`n`}.dump`	the n$^{\text{th}}$ checkpoint with full information to continue the simulation

In the configuration file we can set a variable named `measure_every` that defines how many sweeps are conducted between consecutive measurements. Since we calculate the system's energy for each update anyway, it makes sense to put the energy out more often, or calculate histograms with this "full" time series. Notice that the histograms and weights may have their values given logarithmically in the data files, please consult their respective file-headers. The simulation program also writes checkpoints onto the disk with full information that permit to continue from this point in case of problems, when, e.g., the simulation is cancelled due to runtime restrictions. In such a case, run the program with

```
$ ./goni3d_rec_muca.x n
```

with *n* being the number of a valid checkpoint-file `checkpoint_n.dat` lying in the current working directory.

To ease the editing of configuration files, etc., a python script is delivered that is started with

```
$ python ./reproduce.py
```

to properly create subdirectories for *all* lattice sizes in one bunch, to modify the configuration files and compile the programs along the way.

2 Simulating the Three-Dimensional Plaquette Model with Fixed Boundary Conditions

The source code for fixed boundary conditions is the exact analogue to Section 1, but the source is located in the subdirectory

```
dissertation_reproduce/original_fixed
```

and with the exception that these programs do not support checkpoints. The seeds for the random number generator of the original simulations are collected in Tab. 2.

Table 2: Seeds for the random number generator for reproducing the identical results of Section **??** for the $3d$ plaquette model.

L	seed	L	seed	L	seed	L	seed
4	939473	11	813331	18	294434	25	834743
5	345069	12	310935	19	686090	26	392241
6	511689	13	60773	20	111805	27	457700
7	796661	14	276500	21	700454	28	836983
8	25568	15	942277	22	332738	29	754355
9	972061	16	839182	23	649927		
10	846271	17	630843	24	200310		

3 Simulating the Three-Dimensional Anisotropic Ashkin-Teller Model

The source code for the anisotropic Ashkin-Teller model, contained in the subdirectory `Anisotropic_Ashkin_Teller`, works a little bit different than that for the plaquette model. Only one binary file is needed to simulate the different lattice sizes (this flexibility comes as a trade for speed). It is best to compile out of source by creating and entering a new subdirectory:

```
$ mkdir build && cd build
```

and start the compilation by

```
$ cmake .. -DCMAKE_BUILD_TYPE=release
$ make
```

which creates a binary called `AniAshkTell.x`. The simulation can be started with one of the configuration files in the `config` subdirectory, e.g.,

```
$ ./AniAshkTell.x ../config/L4_config.ini
```

would reproduce the data for the $L = 4$ lattice. The configuration files contain the random number seed of 1369295284 for the original simulation (which was the same for all lattice sizes).

4 Hysteresis

The hysteresis curve for the dual lattice follows the same pattern as that of the anisotropic Ashkin-Teller model in Section 3. Just follow the steps, but in subdirectory `hysteresis`. In its subdirectory `config` the two configuration files are located to heat up or cool down the system, `heatup.ini` or `cooldown.ini`, respectively. The random number seeds are lost in this case, unfortunately.

5 Data analysis

As an example I deliver code for time series reweighting of multicanonical data in the subdirectory `analyse_data`. Again, enter this directory and build the executable with

```
$ mkdir build && cd build
$ cmake .. -DCMAKE_BUILD_TYPE=release
$ make
```

which will create an executable called `reweight_time.x`. Re-enter the data directory from our example in Section 1. Executing

```
$ ../../analyse_data/build/reweight_timeseries.x \
> measurements.dat -w weights.dat
```

in that directory calculates the canonical internal energy, (unnormalised) heat capacity and Binder's energy cumulant via time series reweighting from the multicanonical time series that is stored in `measurements.dat` and has multicanonical weights `weights.dat`. The executable is designed such that it awaits the inverse temperature of interest from the standard input. This allows to calculate the observables for arbitrary temperatures, e.g. equally-spaced, optimized such that peaks are displayed nicely in plots, or external minimisation algorithms.

Conducting the full data analysis in this thesis was quite involved and I used a huge number of scripts, and small helper programs to automatically

- factorise the measurements into jackknife blocks
- calculate microcanonical expectation values from these blocks
- minimise the respective observables for each jackknife block
- combine the data from the jackknife blocks to yield an error estimate.

Some of these tasks are simple one-liners on a linux shell (employing, e.g., `awk` or `sed`), others are more involved. Managing this complexity is left as an exercise for the reader. Also, the reweighting-program must be adjusted for other observables, such as the Fuki-Nuke order parameters. Please also refer to the comments in the respective source files and online documentation.