Numerical Methods
PHYS 777, Core 2026
Dustin Lang, Geoffrey Ryan, Naren Manjunath

Tutorial 4: Monte Carlo and Ising

January 28

In this tutorial we'll cover a famous physics application of the MCMC method: calculating observables of the Ising model in one and two dimensions. (The 1d Ising model is exactly solvable, as you know, but this method comes into its own in two and higher dimensions.) You can find incomplete code for this in the Python notebook here: *https://github.com/nnath94/PSI-Numerical-Methods-2026-Tutorial-4.git*. This code generates samples of spin states and computes some important expectation values, such as the average magnetization and the two-point correlation function $\langle s_i s_j \rangle$.

1 1d Ising model

a) Complete the four unfinished code snippets of the 1d Ising section.

b) What is the expected behaviour of the 2-point correlation function in the 1d Ising model? In particular, what is the correlation length in terms of the parameter J used in the code? You can extract this correlation length from one of the plots available in the code. Does the numerical result agree with your theoretical expectation?

c) Benchmarking: make a rough estimate of how the number of Monte Carlo steps and the burn-in time scales with the linear system size L.

d) Plotting exercise: The code outputs, among other things, the final spin configuration after Monte Carlo sampling. This is a vector of 1s and -1s. The code implements a very naive way of plotting this vector: just use the `plot` function in matplotlib. Can you think of/find a better method to visualize this spin state?

e) *Do this only if you have finished the Symmetry exercise*: modify the code so that it stores the average magnetization of each spin state after the burn-in time, and returns an array rather than a single number. Plot a histogram of the resulting magnetization values. What do you observe?

2 Interlude: login to Symmetry!

a) Go to `symmetry.pi.local` and login with your username and password (this should have been stated in the email you got from IT).

b) Choose the head node (this is the default choice).

c) In order to run a job on Symmetry, you need two files: the actual job file (e.g. the Ising MCMC code) which we'll call `myjob.py` and a job script (text file) ending in `.sh`, for example `job.sh` .

Create a Python file (any code will do to begin with, but ideally you should finally be able to run some Ising MCMC code).

d) Paste the following into the `job.sh` file:

```
#!/bin/bash
#SBATCH --job-name= "myjob"
#SBATCH --nodes=1
#SBATCH --time=24:00:00
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=30
#SBATCH -p defq

module load python
python myjob.py
```

Option: you can replace `defq` with `amdq` if there is a long delay in running (see below on how to check the status of your job).

e) Open the Terminal in a new tab.

f) Type `module load slurm` .

g) To run the job, type `sbatch job.sh` . This creates a job with a number (e.g. 42364).

h) To view the status of your job, type `squeue --me` in Terminal.

i) If everything runs correctly, you should see an output file e.g. `slurm-42364.out` in your directory with whatever text output you asked for. Data saved to a file should also appear here.

j) To cancel a job, type in Terminal `scancel ##42364` .

k) **Note:** to run code on Symmetry you should define a `main()` function in your .py file and include the lines

```
if __name__ == "__main__":
    main()
```

(this has been done in the 1d Ising code given to you but not the 2d Ising code). You also need to save the data you collected, e.g. the 2-point correlation function, to a text or csv file, and plot it separately on your local machine.

3 2d Ising model

a) Complete the unfinished code snippets of the 2d Ising section of the Python notebook. This should be very similar to the 1d case, with a slight change in how we number the points on a two-dimensional grid.

b) What is the expected behaviour of the 2-point correlation function in the 2d Ising model? You should find some signature of a phase transition upon tuning the inverse temperature J (set $B = 0$). What is this signature? Can you estimate the critical inverse temperature J^*?

c) Plotting exercise: The code implements one way of visualizing the final state after sampling: just use the `matshow` function in matplotlib. Can you spot any special feature of the samples close to the critical temperature you obtained in the previous part? Can you think of/find a better method to visualize the spin state?