Stochastic Simulations: Homework 2

Nathan Simonis

December 1, 2020

Abstract

The objective of this homework assignment is to apply and implement Markov Chain Monte Carlo in simulating an Ising model.

Exercise 1

Write a Python function that implements the Metropolis–Hastings algorithm for the Ising model.

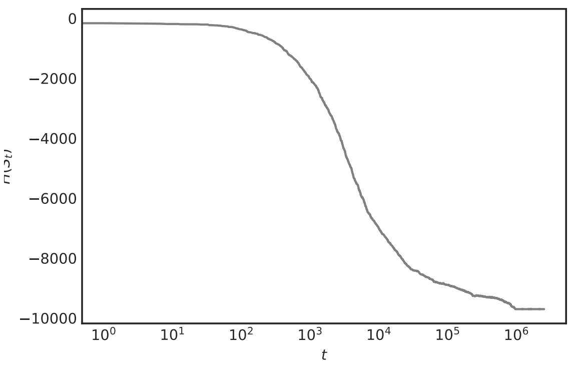
```
import numpy as np
from scipy.ndimage import convolve
from tqdm import tqdm

def initialize_Latice(m):
    """2D uniform square-lattice"""
    return np.random.choice([-1,1], size=(m,m))


def metropolisHastings(n, m, beta, J, B, S, seed=1):
    """Metropolis{Hastings algorithm for the Ising model"""
    np.random.seed(seed)
    def _H(S):
        """Compute the energy of a given system state of in the Ising model"""
        kernel = np.array([[1,1,1],
                           [1,0,1],
                           [1,1,1]])
        neighbors = convolve(S, kernel, mode="constant")
        return - np.sum(0.5*J*S*neighbors + B*S)

    def _update(S):
        """Randomly pick a spin, flip it, compute contribution and compare w/ old."""
        S_tilde = S.copy()
        cont = _H(S)

        i, j = np.random.randint(low=0, high=m, size=(2,))
        S_tilde[i,j] *= -1 # Flip atom
        new_cont = _H(S_tilde) # Compute new contribution

        acceptance = np.exp(-beta*(new_cont - cont))

        if new_cont < cont: # Keep spin if new_cont is smaller.
            return S_tilde
        elif np.random.uniform() < acceptance: #Keep with prob exp(-\beta(H_v-H_\mu))
            return S_tilde
        else: # Stay in same state.
            return S

    # Define Energies and Magnetization
    Hs = np.zeros(n)
    Ms = np.zeros(n)

    for ii in range(n):
        S = _update(S) # New state
        Hs[ii] = _H(S) # Compute & store energy for system
        Ms[ii] = np.sum(S) # Compute & store total magnetic moment of system
    return Hs, Ms, S
```

Exercise 2

Use your Python function with $\beta = 1$ and for n, such that both the energy and the total magnetic moment appear to have reached a stationary state. Plot also the final system configuration.

In order to obtain consistent and repeatable results, a 123 seed is applied in the pseudo-random number generation process. We also choose a lattice of 50×50 atoms, $J = 1$, and $B = 0$ for all simulations.

A simulation with $n = 5 \times 10^7$ is carried out to see when the energy and total magnetic moment stabilise. It can be seen that energy stabilises very quickly in comparison with the total magnetic moment, which reaches a stable value when all the atoms have the same spin. In this case, this happens after approximately 10^6 steps.



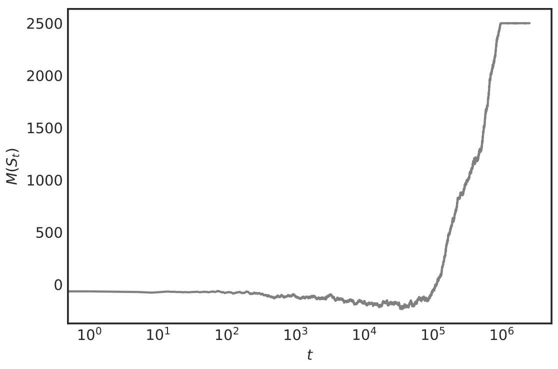


Figure 1: Energy (left) and total magnetic moment (right) w.r.t time.

The graph 4 (appendix) shows the state of the initial system as well as its state after n simulations when all atoms have the same spins.

Compute the mean total magnetic moment $\bar{M}(\beta)$ for different values of $\beta \in \left[\frac{1}{3}, 1\right]$ and $n = 5 \times 10^6$.

We use the **linspace** function from numpy which returns numbers regularly spaced over the requested interval. We choose to use 20 values within this interval.

For this exercise, given the computation time required, m and n is greatly reduced. This will have an impact on the values obtained but not on the effect of β on the system. It can be seen in figure 5 that for higher values of β a stationary state is reached more quickly and therefore the average is higher/lower (depending on the initial state) here, the same initial state is used for each simulation.

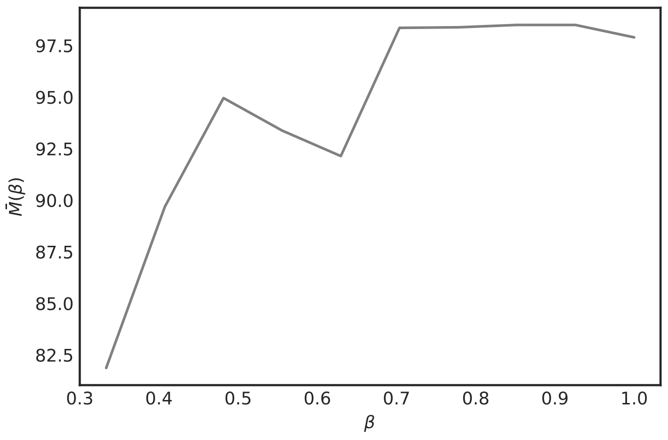


Figure 2: Mean of total magnetic moment for different values of β

Exercise 3

Set $m = 4$ and compute the exact expected value of the total magnetic moment $\bar{M}(\beta)$

$$\bar{M}(\beta) = \frac{1}{Z_\beta} \sum_{S \in \mathcal{K}} M(S) e^{-H(S)\beta} \tag{1}$$

where $Z_\beta = \sum_{i=1}^{M} e^{-H(S_i)\beta}$ is the normalization constant, M is the number of all states of the system. Thus, for $m = 4$, the number of possible states is $2^{4*4} = 65536$. This makes it unfeasible to compute when m is large.

Computing the exact value of $\bar{M}(\beta)$ we obtain 0. This makes sense because each state of the system has a magnetisation that has the same probability as the flipped magnetisation.

We can see from figure 3 that by increasing n, when using Markov Chain Monte Carlo, we are getting closer to the exact expected value.

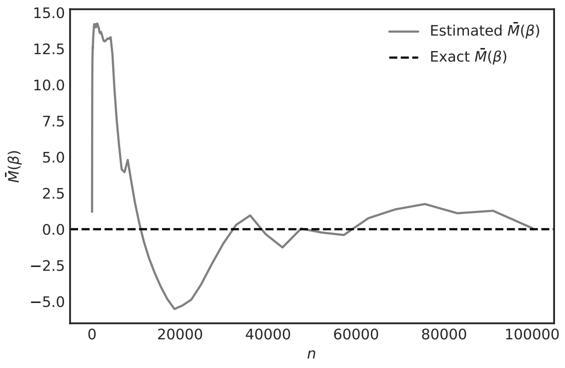


Figure 3: Expected total magnetic moment as a function of n

Appendix A Plots

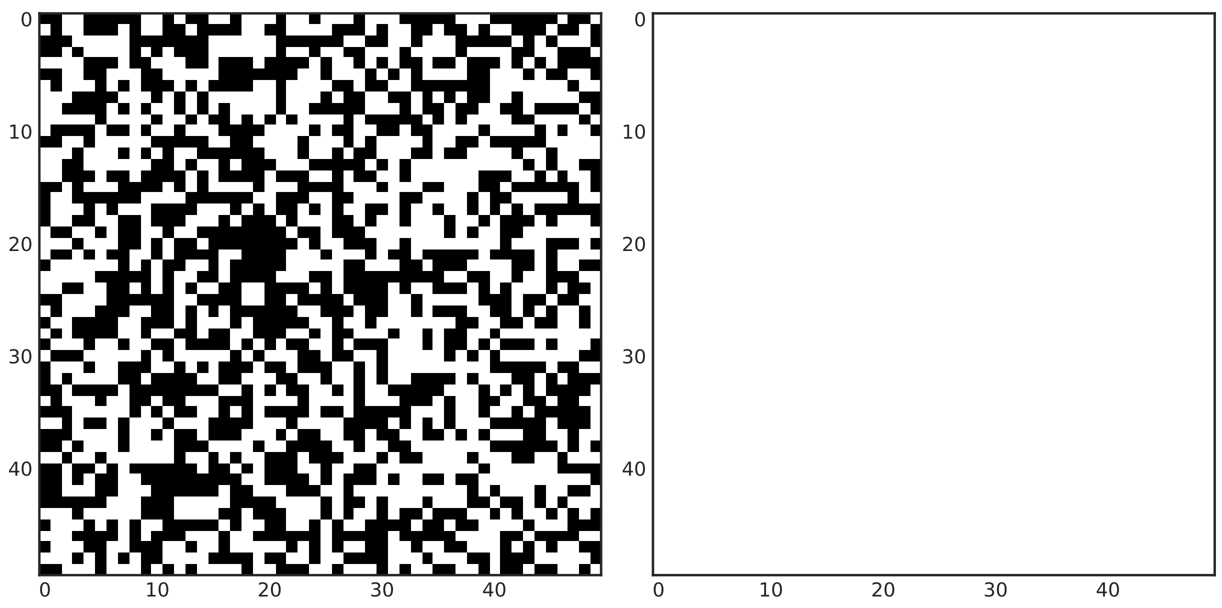


Figure 4: Initial (left) and final (right) state of the system

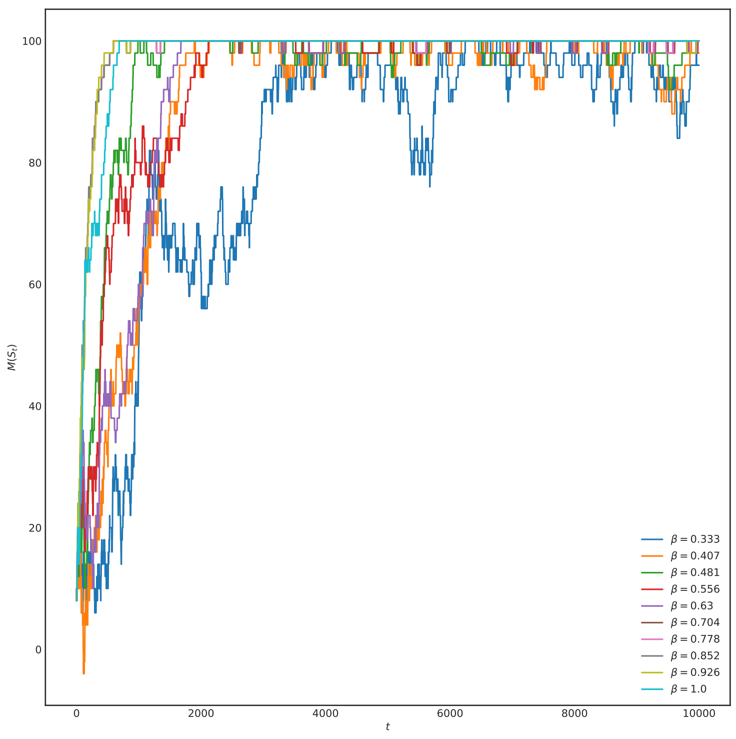


Figure 5: Evolution of total magnetic moment for different values of β

Appendix B Python code

Exercise 2.1

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from itertools import product
import hasting
plt.style.use("seaborn-white")
np.random.seed(1)

# Define parameters.
m = 50
beta = 1
J = 1
B = 0
S0 = hasting.initialize_Latice(m)

# Run simulations
n = 10000
H, M, S = hasting.metropolisHastings(n, m, beta, J, B, S0)
```

```
## Plots
# Energy
plt.plot(H, color="grey")
plt.xlabel(r"$t$")
plt.ylabel(r"$H(S_t)$")
plt.xscale("log")
plt.savefig("../figures/ex21_energy.pdf")
plt.show()


# Magnetic

plt.plot(M, color="grey")
plt.xlabel(r"$t$")
plt.ylabel(r"$M(S_t)$")
#plt.xscale("log")
plt.savefig("../figures/ex21_magnetic.pdf")
plt.show()

# System

fig, axs = plt.subplots(1,2, figsize=(9,9))
axs[0].imshow(S0, aspect="equal", cmap = "Greys")
axs[1].imshow(S, aspect="equal", cmap = "Greys")
plt.tight_layout()

plt.savefig("../figures/ex21_system.pdf")
plt.show()
```

Exercise 2.2

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
from tqdm import tqdm
from itertools import product
import hasting
plt.style.use("seaborn-white")
np.random.seed(1)

# Parameters.
np.random.seed(1)
m = 10
S0 = hasting.initialize_Latice(m)
```

```
params = {"n":[(10**3)],
          "m":[m],
          "beta":np.linspace(1/3, 1, 10),
          "J":[1],
          "B":[0],
          "S":[S0]}

# All possible combinations of parameters. (In this case only \beta changes)
keys, values = zip(*params.items())
experiments = [dict(zip(keys, v)) for v in product(*values)]

M_bar = []
fig = plt.figure(figsize = (10,10))
# Iterate each value of beta.
for comb in experiments:
    H, M, S = hasting.metropolisHastings(**comb) # Run simulation
    plt.plot(M, label=r"$\beta = $"+str(round(comb["beta"],3))) # Plot magnetization fo
    plt.xlabel(r"$t$")
    plt.ylabel(r"$M(S_t)$")
    M_bar.append(np.mean(M)) # Compute average total magnetic moment of simulation.
plt.legend()
plt.tight_layout()
plt.savefig("../figures/ex22_magnetic_sims.pdf")
plt.show()

# Plot average total magnetic moment as a function of beta.

plt.plot(params["beta"], M_bar, color="grey")
plt.xlabel(r"$\beta$")
plt.ylabel(r"$\bar{M}(\beta)$")
plt.savefig("../figures/ex22_mbar.pdf")
plt.show()
```

Exercise 3

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
from tqdm import tqdm
from itertools import product
import hasting
plt.style.use("seaborn-white")
np.random.seed(1)
```

```
# Define energy computations from ex1.

def energy(S, J=1, B=0):
    """The energy of a given system state of in the Ising model"""
    kernel = np.array([[1,1,1],
                       [1,0,1],
                       [1,1,1]])
    neighbors = convolve(S, kernel, mode="constant") # constant --> fill edges w/ 0
    return - np.sum(0.5*J*S*neighbors + B*S)

# Parameters.
m = 4
beta = 1/3

permutations = list(product([-1,1],repeat=m*m)) # Create all states of system.

n = len(permutations)
Bs = np.zeros(n)
Hs = np.zeros(n)
Ms = np.zeros(n)

for i in range(n):
    S = np.array(permutations[i]).reshape(m,m) # Reshape to m*m 2D latice.
    Hs[i] = energy(S) # Compute & store energy
    Ms[i] = np.sum(S) # Compute & store TMM
    Bs[i] = np.exp(-Hs[i]*beta) # Boltzman

Z = np.sum(Bs) # Normalization constant
mbar = 1/Z * np.sum(Ms*Bs) # Exact mbar
print(mbar)


N = np.logspace(1,5, 100)  # Multiple runs of N
est_mbar = []
S0 = hasting.initialize_Latice(m) # Same initial state.

for n in N:
    n = int(np.ceil(n))
    H, M, S = hasting.metropolisHastings(n=n, m=4, beta=1/3, J=1, B=0, S=S0, seed=3)
    est_mbar.append(np.mean(M)) # Compute estimated Mbar

# Plot
plt.plot(N,est_mbar, c="grey", label='Estimated '+r'$\bar{M}(\beta)$')
plt.axhline(mbar, ls="--", c="black", label='Exact '+r'$\bar{M}(\beta)$')
plt.xlabel(r"$n$")
plt.ylabel(r"$\bar{M}(\beta)$")
plt.legend()
plt.savefig("../figures/ex3_mbar_n.pdf")
plt.show()
```