AM3911G: Modelling and Simulation **(Due: 03/4/2020)**

Assignment #3

Instructor: Allan MacIsaac *Name:* Peter Akioyamen, *StudentID:* 250949002

Problem 1

– Points

Below is a C++ program to simulate the 2D Ising model on a square lattice using Monte Carlo.

Listing 1: Simulation of 2-Dimensional Ising Model Using Markov Chain Monte Carlo.

```cpp
/*
    Author: Peter Akioyamen
    Student #: 250949002
    Course: AM3911G - Modelling & Simulation
    Professor: Allan MacIsaac
    Assignment #: 3
    Due: March 4, 2020

    A program which simulates a 2-Dimensional Ising
    model using a 75x75 square lattice. Simulations
    of the system are done at various temperatures.
    Simulation is done using Markov chain Monte
    Carlo with a total of 12000 Monte Carlo steps
    for each temperature value. Average energy per
    spin and average absolute magnetization of the
    systems are computed and saved.
*/

// Incluse necessary dependencies
#include <iostream>
#include <random>
#include <math.h>
#include <fstream>
#include <string>
using namespace std;

// Define the number of spin sites in a row (periodic boundary)
// If the lattice desired is K => N = K + 2
const int N = 77;

// Function definitons
void initialize_lattice(int lattice[N][N]);
void update_boundary(int lattice[N][N], int i = 0, int j = 0);
```

```cpp
int init_state_energy(int lattice[N][N], int interaction);
int state_magnetization(int lattice[N][N]);
void show_lattice_state(int lattice[N][N]);

int main() {
    // Define file for saving data for plotting
    fstream ising_model("ising_model_2d_" + to_string(N - 2) +
        "by" + to_string(N - 2) + "_results.csv", ios::out | ios::app);

    // GS Energy: -2NJ (N should be the total number of spins) assuming periodic boundaries
    // Define parameters of Ising model
    int J = 1; // Define iteraction strength for a pair of neighbours
    int lattice[N][N]; // Define the lattice shape


    // Define parameters of Monte Carlo
    double K_b = 1.0; // Define the Boltzmann constant
    double T; // Define temperature
    int energy = 0; // Define the energy of the current state
    int magnetization = 0; // Define the magentization of the current state
    double delta_energy = 0.0; // Define the change in energy between current state and previous
    int trial_spin = 0; // Define variable for microstate change
    double transition_p = 0.0; // Define variable for transition probability
    double p = 0; // Define variable for generated probability
    int mc_steps = 12000; // Number of Monte Carlo steps to conduct Markov process
    int min_mc_step = 2000; // Minimum number of Monte Carlo steps before sampling
    double avg_energy_per_spin; // Average energy per spin of system
    double avg_abs_magnetization; // Average absolute magnetization of system
    double samples; // Number of samples taken for a temperature

    // Define dummy index variables
    int row_loc = 0;
    int col_loc = 0;

    // Define RNG
    default_random_engine generator;
    uniform_real_distribution<double> uniform(0.0, 1.0);

    // Compute for multiple temperatures
    for (T = 1.0; T <= 4.02; T += 0.05) {

        avg_energy_per_spin = 0.0;
        avg_abs_magnetization = 0.0;
        samples = 0.0;

        // Initialize new lattice and corresponding quantities
        initialize_lattice(lattice);
        update_boundary(lattice);
        energy = init_state_energy(lattice, J);
        magnetization = state_magnetization(lattice);

        // Compute for at least 5000 monte carlo steps
        for (int mc_step = 0; mc_step <= mc_steps; mc_step++) {


            // Compute new macrostate of entire lattice (one monte carlo step)
            for (int i = 1; i < (N - 1); i++) {
                for (int j = 1; j < (N - 1); j++) {
                    // Randomly select a spin site on lattice
                    row_loc = (int)(uniform(generator) * ((double)N - 2) + 1);
                    col_loc = (int)(uniform(generator) * ((double)N - 2) + 1);

                    // Flip the spin site and compute the change in energy
                    trial_spin = -1 * lattice[row_loc][col_loc];
                    delta_energy = -1.0 * trial_spin *
                        (lattice[row_loc][col_loc + 1] + lattice[row_loc][col_loc - 1]
                            + lattice[row_loc + 1][col_loc] + lattice[row_loc - 1][col_loc]) * 2.0;


                    // Compute the transition probability and accept or reject new state
                    transition_p = exp(-1 * delta_energy / (K_b * T)) / (1 + exp(-1 * delta_energy / (K_b * T)));
                    p = uniform(generator);
                    if (p <= transition_p) {
                        energy += (int)delta_energy; // Energy of new state
                        magnetization += 2 * trial_spin; // Magnetization of new state
                        lattice[row_loc][col_loc] = trial_spin; // Accept state

```

```cpp
                        // Update boundary if the new state has a changed spin on the edges of lattice
                        if (row_loc == 1 || row_loc == (N - 2) || col_loc == 1 || col_loc == (N - 2)) {
                            update_boundary(lattice, row_loc, col_loc);
                        }
                    }
                }
            } // End of one Monte Carlo step

            // Beginning of computations for average energy per spin
            // and average absolute magnetization
            if ((mc_step > min_mc_step) && (mc_step % 10 == 0)) {
                avg_energy_per_spin += energy;
                avg_abs_magnetization += magnetization;
                samples++;
            }
        } // end Monte Carlo for given temperature - mc_step

        // Finish computations for average energy per spin and
        // average absolute magnetization
        avg_energy_per_spin = (avg_energy_per_spin / samples) / (((double)N - 2) * ((double)N - 2));
        avg_abs_magnetization = (abs(avg_abs_magnetization) / samples) / (((double)N - 2) * ((double)N - 2));

        // Save the results
        ising_model << T << "," << avg_energy_per_spin << "," << avg_abs_magnetization << "\n";
    } // end one temperature computation - T

    // Close file
    ising_model.close();
    return 0;
}


// A function which initializes the 2D lattice - this gives the first state
void initialize_lattice(int lattice[N][N]) {
    default_random_engine gen;
    uniform_real_distribution<double> distribution(0.0, 1.0);

    // Initialize the square lattice randomly
    for (int i = 1; i < (N - 1); i++) {
        for (int j = 1; j < (N - 1); j++) {
            //lattice[i][j] = 1; for ground state configuration
            if (distribution(gen) < 0.5) {
                lattice[i][j] = 1;
            }
            else {
                lattice[i][j] = -1;
            }

        }
    }

    // Set the corners which are not interacted with
    lattice[0][0] = 0;
    lattice[0][N - 1] = 0;
    lattice[N - 1][0] = 0;
    lattice[N - 1][N - 1] = 0;
}


// A function to update the boundaries of the lattice
// based on the periodic boundary condition
void update_boundary(int lattice[N][N], int i, int j) {
    // Update on initialization of lattice
    if (i == 0 && j == 0) {
        for (int h = 1; h < (N - 1); h++) {
            lattice[h][0] = lattice[h][N - 2]; // Column 1 periodic ↩
               boundary
            lattice[h][N - 1] = lattice[h][1]; // Column 2 periodic ↩
               boundary
            lattice[0][h] = lattice[N - 2][h]; // Row 1 periodic ↩
               boundary
            lattice[N - 1][h] = lattice[1][h]; // Row 2 periodic ↩
               boundary
        }
    }
    // Update on state change
    else {
        if (i == (N - 2)) {
            // New spin in last row - update row 1 periodic boundary
            lattice[0][j] = lattice[i][j];
        }
        if (i == (1)) {
            // New spin in first row - update row 2 periodic boundary
            lattice[N - 1][j] = lattice[i][j];
        }
        if (j == (N - 2)) {
            // New spin in the right column - update column 1 periodic↩
                boundary
            lattice[i][0] = lattice[i][j];
        }
        if (j == (1)) {
            // New spin in the LEFT column - update column 2 periodic ↩
               boundary
            lattice[i][N - 1] = lattice[i][j];
        }
    }
}

// A function which computes and returns the energy
// of the initialization state of the lattice
int init_state_energy(int lattice[N][N], int interaction) {
    int energy = 0;
    int J = interaction;
    // Compute the hamiltonian of initilization state
    for (int i = 1; i < (N - 1); i++) {
        for (int j = 1; j < (N - 1); j++) {
            // Computation for spin on the bottom corner of the ↩
               lattice
            if ((j == (N - 2)) && (i == (N - 2))) {
                // Compute energy contribution between current spin ↩
                   and row periodic boundary spin
                energy += -J * lattice[i][j] * lattice[i][N - 1];

                // Compute energy contribution between current spin ↩
                   and column periodic boundary spin
                energy += -J * lattice[i][j] * lattice[N - 1][j];
            }
            // Computation for spins on right boundary of lattice
            else if (j == (N - 2)) {
                // Compute energy contribution between current spin ↩
                   and row periodic boundary spin
                energy += -J * lattice[i][j] * lattice[i][N - 1];

                // Compute energy contribution between current spin ↩
                   and bottom-adjacent spin
                energy += -J * lattice[i][j] * lattice[i + 1][j];
            }
            // Computation for spins on bottom boundary of lattice
            else if (i == (N - 2)) {
                // Compute energy contribution between current spin ↩
                   and right-adjacent spin
                energy += -J * lattice[i][j] * lattice[i][j + 1];

                // Compute energy contribution between current spin ↩
                   and column periodic boundary spin
                energy += -J * lattice[i][j] * lattice[N - 1][j];
            }
```

```cpp
            // Computation for all other spins
            else {
                // Compute energy contribution between current spin and right-adjacent spin
                energy += -J * lattice[i][j] * lattice[i][j + 1];

                // Compute energy contribution between current spin and bottom-adjacent spin
                energy += -J * lattice[i][j] * lattice[i + 1][j];
            }
        }
    }
    return energy;
}

// A function which computes the magnetization of the
// current state of the lattice
int state_magnetization(int lattice[N][N]) {
    int magnetization = 0;
    // Compute magnetization of initialization state
    for (int i = 1; i < (N - 1); i++) {
        for (int j = 1; j < (N - 1); j++) {
            magnetization += lattice[i][j];
        }
    }
    return magnetization;
}

// A function which prints out the currentstate of the lattice
// as a grid containing its spin values at each site
void show_lattice_state(int lattice[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (lattice[i][j] != -1) {
                cout << " " << lattice[i][j] << " ";
            }
            else {
                cout << lattice[i][j] << " ";
            }
        }
        cout << "\n";
    }
}
```

Problem 2 & 3 – Points

The system used for the Markov chain Monte Carlo (MCMC) simulation of the Ising model was a 75×75 square lattice. Results were recorded for various temperatures: $T = 1, \ldots, 4$ at intervals of 0.05. 12000 Monte Carlo steps were used in the simulation. Figures: 1 & 2 below show the computed and analytical solutions of the system energy and magnetization, respectively, as temperature increases.

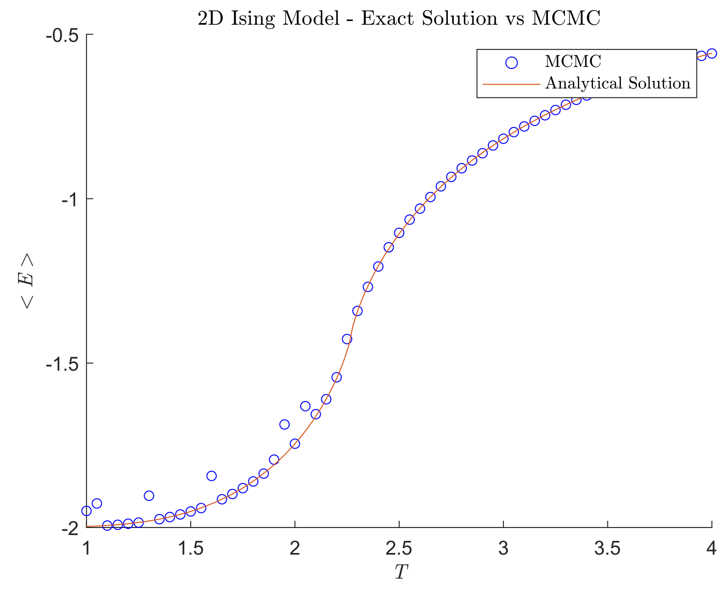

Figure 1: Average energy per spin of 75×75 Ising model as temperature varies

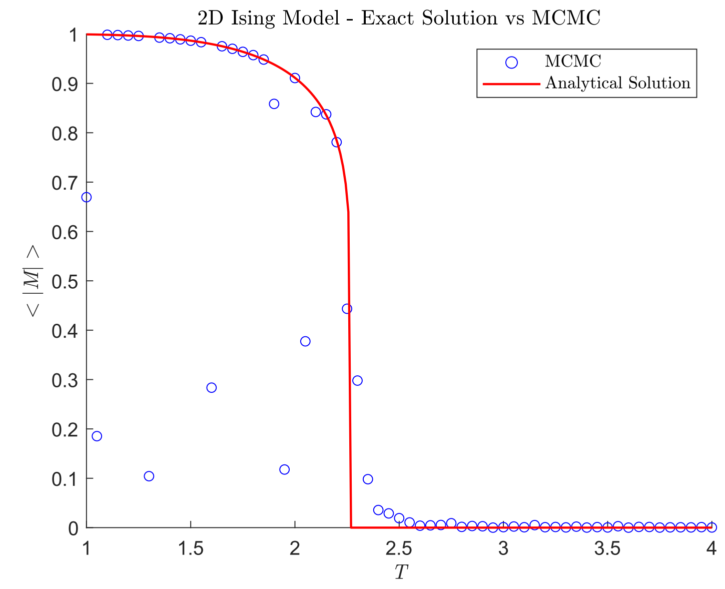

Figure 2: Average absolute magnetization of 75×75 Ising model as temperature varies