Simulation of the 2 D Ising Model

Background and Setup

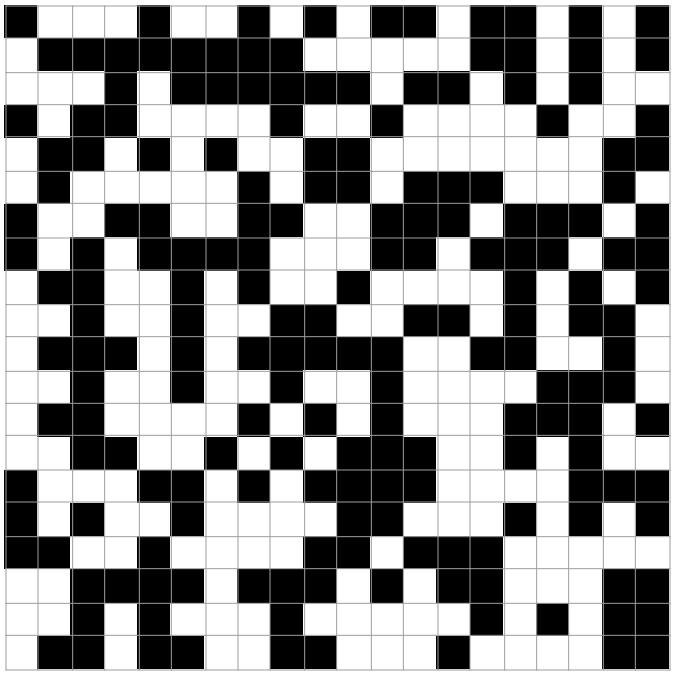
The idea of this program is to simulate, on a small scale, the internal structure and dynamics of a ferromagnet or and antiferromagnet.Below are snipets of Mathematica code that will be assembled into a working program that you will use to explore the 2 D Ising model.

Basic Metropolis Updater

Initialize and visualize the lattice

In[1]:=
```
ClearAll["Global`*"]
(* specifies the lattice size *)
n = 20;
(*generates a random initial setup*)
SeedRandom[1]; initLat = 2 Table[Random[Integer], {n}, {n}] - 1;
(* thie creates a pictoral plot of the spin structure with
white representing spin up and black representing spin down *)
ArrayPlot[initLat, ColorRules → { 1 -> White, -1 → Black}, Mesh → True]
```

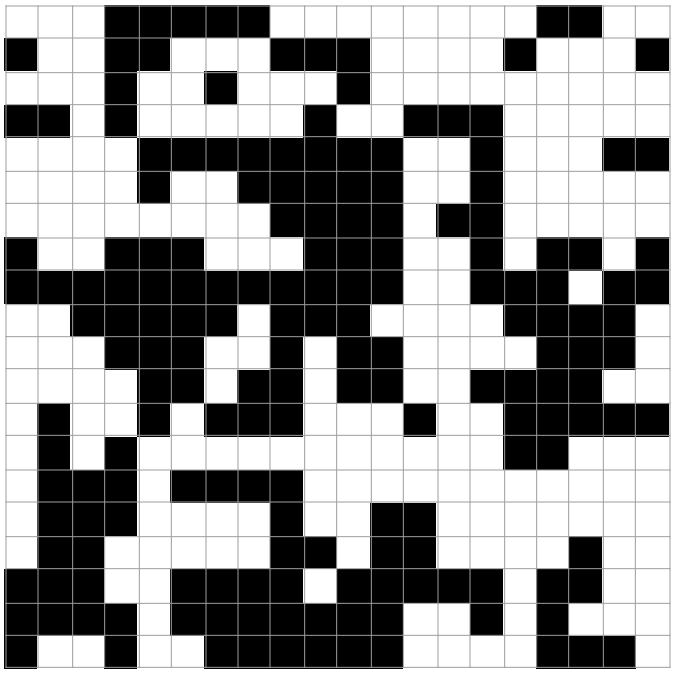
Out[4]=



In[5]:=
```
(*sets the initial values for the Heisenberg Hamiltonian,
external field and coupling constant.If J<0,then the system is a ferromagnet;
if J>0,the system is an antiferromagnet *)
B = 0; J = -0.3;
```

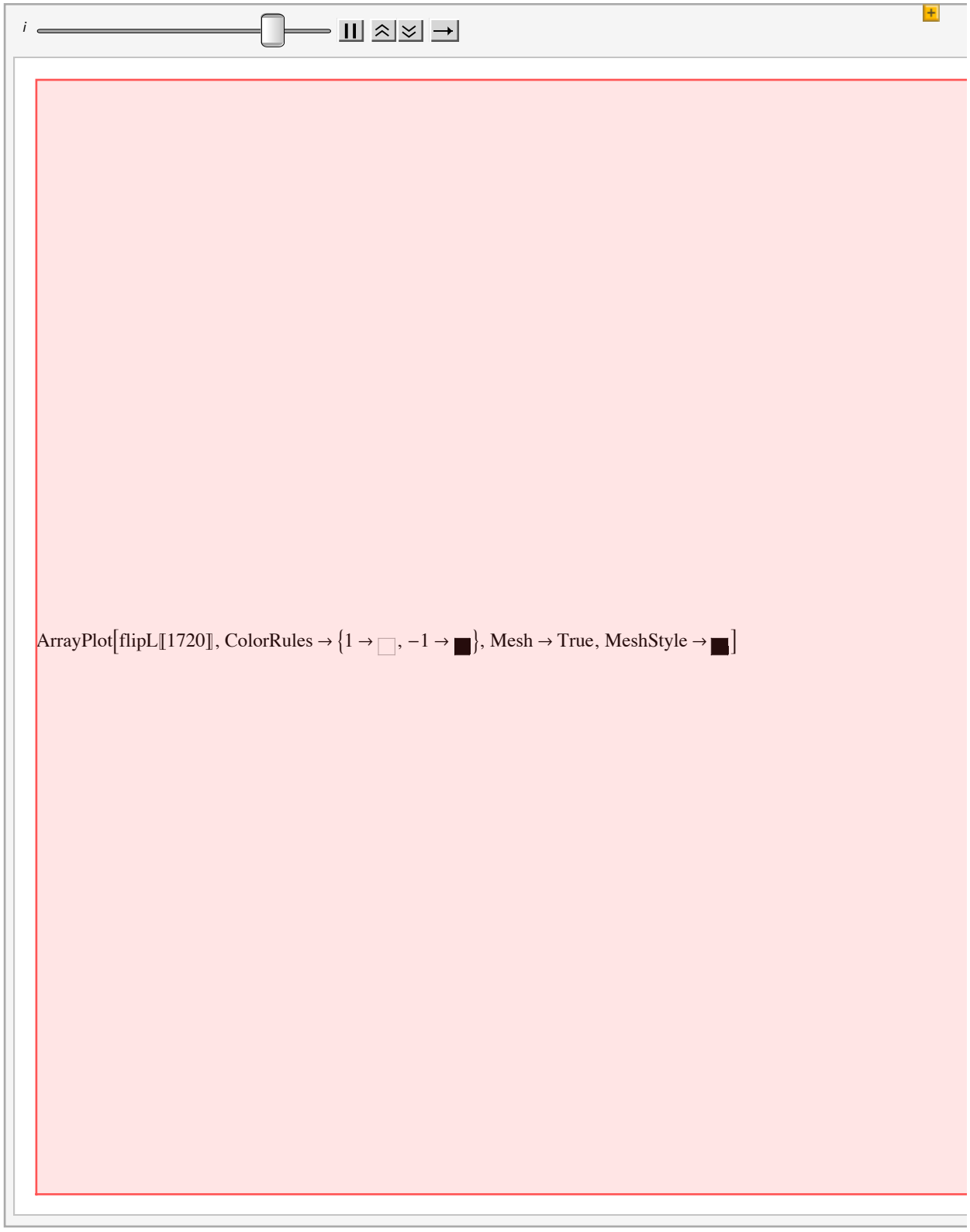
```
(*a table consisting of all possible energy changes if a single site is flipped
based on all possible configurations of the sites 4 nearest
 neighbors.  For example, if I am "up" and I have four "up" neighbors,
my energy goes from -B+4J to +B -4J, so the change in energy is 2(B-4J)*)
δE[1, 4] = 2 (B - 4 J);
δE[1, 2] = 2 (B - 2 J);
δE[1, 0] = 2 (B - 0 J);
δE[1, -2] =  2 (B + 2 J);
δE[1, -4] =  2 (B + 4 J);
δE[-1, 4] =  -2 (B - 4 J);
δE[-1, 2] =  -2 (B - 2 J);
δE[-1, 0] =  -2 (B - 0 J);
δE[-1, -2] =  -2 ( B + 2 J);
δE[-1, -4] =  -2 (B + 4 J);
(*Metropolis algorithm used to randomly select a site,test the energy,
and decide to flip the spin or not.Has periodic boundaries.*)
flipStep =
(
    S = #; (*the # symbol is a literal
call to the array that this function will work on*)
{i1, i2} = {RandomInteger[{1, n}], RandomInteger[{1, n}]};
    (*selects a site*)
(*the following 4 lines impose
     periodic boundary conditions on the nearest neigbors *)
    down = Mod[i1 + 1, n, 1];
    up = Mod[i1 - 1, n, 1];
    right = Mod[i2 + 1, n, 1];
    left = Mod[i2 - 1, n, 1];
(*calculates the energy
difference of flipping based on the 4 nearest neighbors*)

    eDiff = S[[down, i2]] + S[[up, i2]] + S[[i1, right]] + S[[i1, left]];

(*this statement is the implimentation of the Metropolis alrogithm.The
quantifier is an OR statement. If the energy difference dE<0,
    then the site is flipped.If it is positive,
    the a random number is generated and compared to exponential of the
energy difference and,if it is greater,the site is flipped "||" means "or"*)
    If[δE[S[[i1, i2]], eDiff] < 0 || RandomReal[] < Exp[-δE[#[[i1, i2]], eDiff]],
     S[[i1, i2]] = -S[[i1, i2]];
     S, S];

S (*this returns the new lattice configuration*)
) &;(*the& symbol is required at
the end of a literal function call like this*)
(*sets the number of iterations*)
count = 2000;
(*builds an nested array of lattices,one for each step*)
flipL = NestList[flipStep, initLat, count];
(*gives a visualization of the last lattice in the calculation*)
ArrayPlot[flipL[[count]], ColorRules → {1 -> White, -1 -> Black}, Mesh -> All]
```

Out[10]=



Important : before clicking to rerun the Animator Below turn off "Dynamic Updating Enabled" before running this.Once run turn it back on again to visualize the animation.

```
Animate[ArrayPlot[flipL[[i]], ColorRules → { 1 -> White, -1 → Black},
  Mesh → True, MeshStyle → Black] , {i, 1, count, 1}]
```



Program

This section is the chunk of code that you will run, the Ising2D function, to answer questions about the behavior of a FM and an AFM system.You will also be required to modify it a bit in some later problems.

The variable n controls the dimensions of the lattice while the variable count controls the number of iterations.Be warned that this program can get memory intenstive.The larger you make n and count, the longer you can expect to hang out. Also, if you want to use the last line of code in this section, the one that animates the result, I would recommend subsequently runtunring off Dynamic Update (do this from the menu bar at the top : Evaluation > Dynamic Updating Disabled/Enabled). If you don' t, then every-time you run the program, Mathematica will try and update it as well, which gets very memory intensive.

In[1]:=
```
ClearAll "Global` "
(*in the actual program,
temperature is implimented.The value of the Boltzman constant,k=1 *)
(* Calculate the energy.  The funtion Mod[i+1,n,1] takes the site to the right,
imposing periodic boundary conditions. *)
energy[S_, n_, J_, B_] := Module[{energ}, energ = 0;
   Do[energ = energ + (S[[i, j]] (J (S[[Mod[i + 1, n, 1], j]] + S[[i, Mod[j + 1, n, 1]]]) -
         B)), {i, 1, n}, {j, 1, n}];
   energ];
magnetization[S_] := Total[S, 2];
initialize[n_] := 2 Table [Random [Integer], {n}, {n}] - 1;
Ising2D[n_, count_, B_, J_, T_] :=
 Module[{}, (* Module just groups all of the commands together*)
  (* Define the energy cost/temp of a spin flip *)
  (* δE[spin,mag] = change in energy/temp to flip from +spin to -spin
     given a nearest neigbor magnetization mag*)
  δE[1, 4] = 2 (B - 4 J) / T;
  δE[1, 2] = 2 (B - 2 J) / T;
  δE[1, 0] = 2 (B - 0 J) / T;
  δE[1, -2] =  2 (B + 2 J) / T;
  δE[1, -4] =  2 (B + 4 J) / T;
  δE[-1, 4] =  -2 (B - 4 J) / T;
  δE[-1, 2] =  -2 (B - 2 J) / T;
  δE[-1, 0] =  -2 (B - 0 J) / T;
  δE[-1, -2] =  -2 ( B + 2 J) / T;
  δE[-1, -4] =  -2 (B + 4 J) / T;
  flipStep =
(
     S = #[[1]]; mg = #[[2]]; en = #[[3]]; (*the # symbol is a literal
call to the array that this function will work on*)
{i1, i2} = {RandomInteger[{1, n}], RandomInteger[{1, n}]};
     (*selects a site*)
     (*the following 4 lines create statements to isolate
the 4 neighest neighbors to the
      chosen site.The Mod structure imposes periodic BCs *)
     down = Mod[i1 + 1, n, 1];
     up = Mod[i1 - 1, n, 1];
     right = Mod[i2 + 1, n, 1];
     left = Mod[i2 - 1, n, 1];
(*calculates the energy
```

```
difference of flipping based on the 4 nearest neighbors*)

      eDiff = S[[down, i2]] + S[[up, i2]] + S[[i1, right]] + S[[i1, left]];

(*this statement is the implimentation of the Metropolis alrogithm.The
quantifier is an OR statement. If the energy difference dE<0,
      then the site is flipped.If it is positive,
      the a random number is generated and compared to exponential of the
energy difference and,if it is greater,the site is flipped "||" means "or"*)
      If[δE[S[[i1, i2]], eDiff] < 0 ||
        RandomReal[] < Exp[-δE[S[[i1, i2]], eDiff]], mg = mg - 2 S[[i1, i2]];
       en = en + δE[S[[i1, i2]], eDiff];
       S[[i1, i2]] = -S[[i1, i2]]; S, S];

      {S, mg, en}(*this returns the new
       lattice configuration and magnetization and energy *)
) &; (*the& symbol is required at
the end of a literal function call like this*)
  (*sets the number of iterations*)
(* count sets the number of iterations*)
  (* first thermalize. You can eliminate this if you prefer *)
  thermal = Nest[flipStep,
    {initLat, magnetization[initLat], energy[initLat, n, J, B] / T}, count / 5];
  (* now do the run *)
  flipList = NestList[flipStep, thermal, count];
  (* List of magnetizations *)
  mag = Table[flipList[[i, 2]], {i, 1, count + 1}];
  (* List of energies *)
  eng = T Table[flipList[[i, 3]], {i, 1, count + 1}];
  (* update initLat for next run *)
  initLat = flipList[[count + 1, 1]];
] (* end Module *)
```

Out[1]= ClearAll Global`

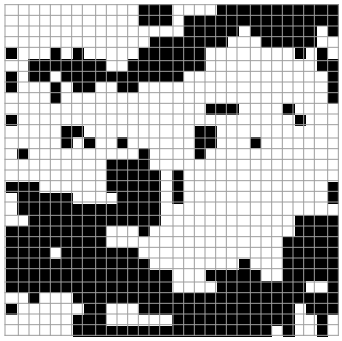
Visualization of run

Heres a run used to illustrate the approach.Note that I only initialize the lattice on the first run - this allows for better thermalization.

In[6]:=
```
(* Notice that this is only 90000/900~ 100 steps per site *)
nSteps = 90 000;
(* Small positive field to induce an "up" magnetization *)
B = 0.01; J = -1; npoints = 30;
temp = 2.4;
initLat = initialize[npoints];
Ising2D[npoints, nSteps, B, J, temp];
```

In[9]:=
```
ArrayPlot[initLat, ColorRules → { 1 -> White, -1 → Black},
 Mesh → True, ImageSize → Small] (* This is the initial arrangement*)
```

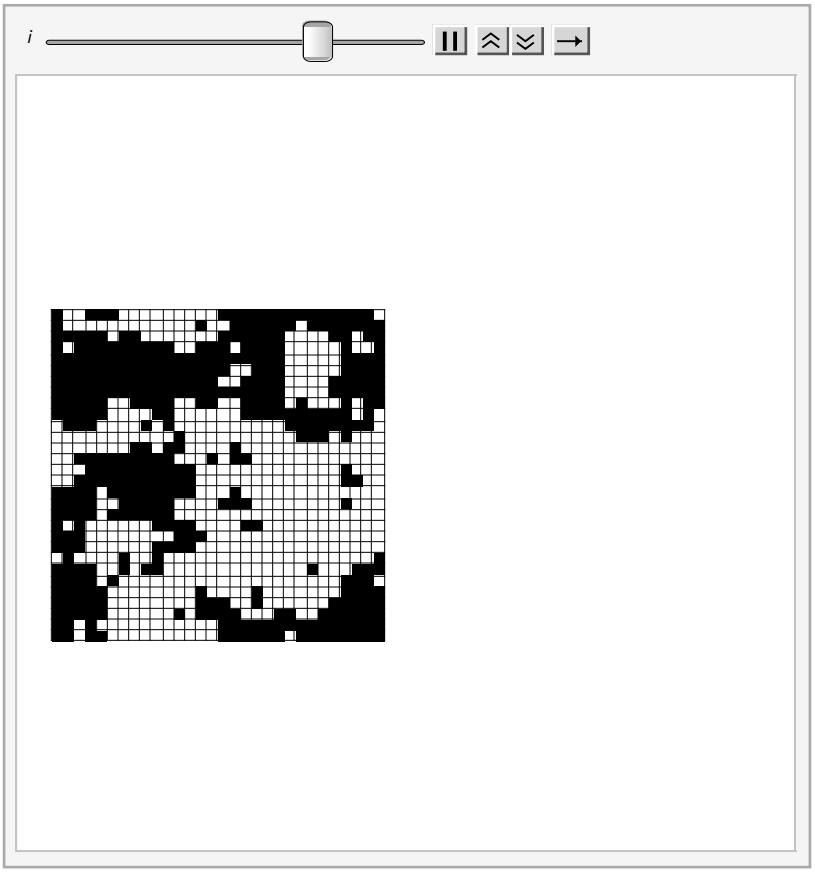
Out[9]=



Notice how in this animation, the fully magnetized cluster has a size comparable or greater than the size of
the simulation - this is because at T=2.4J one is close to the phase transition at T=2.26J.

In[10]:=
```
Animate[ArrayPlot[flipList[[i, 1]], ColorRules → { 1 -> White, -1 → Black},
  Mesh → True, MeshStyle → Black, ImageSize → Small] , {i, 1, nSteps, 1}]
```

Out[10]=



Problem 1

a.) Run the program (Visualization of Run) with n = 20 at a variety of temperatures, say T = 10, 5, 2.5, 1, and 0.1, using about 2000 steps or so. (Use J=1 in all calculations). At each temperature, roughly estimate the size of the emerging clusters.

b.) Repeat part (a) with n = 50. Do the cluster sizes change? Explain.

c.) Run the program with n = 20 at tempertures T = 2, 1, and 0.5 with a variety of step values.Estimate the average magnetization of the lattice at each temperature. Discuss the results.Do you ever see symmetry breaking, the point where the whole system takes on one spin value?

d.) Run the program with n = 10 at T = 2.5 using at least 100, 000 steps.Using the animation code, watch the system evolve. Discuss and explain this behavior.e.) Based on the previous parts, estimate where the critical temperature of the system is.Try using larger lattices with more iterations (as much as your computer can handle) to get a good feeling for what is happening.

Problem 2.

a.) Use the energy data to calculate the specific heat capacity.

b.) Calculate and plot the specific heat capacity for a sequence of temperatures from T=5 to T=1. What do you observe? Is your system in thermal equilibrium at all temperatures?

c.) Plot the equilibrium magnetization in a small field as a function of temperature.

d.) Calculate and plot the magnetic susceptibility for a sequence of temperatures from T=5 to T=1.

f.) Use your results to estimate the transition temperature.

e.) How you could improve the code to run for much longer times and lattice sizes (think memory)?