2D_Ising

December 8, 2017

1 Metropolis Monte Carlo for 2D Ising Model

2D Ising Model: Ising Model is one of the simple yet effective model which consists of a lattice with fixed number of sites. Each Lattice site has a spin with two degree of freedom (Si = ± 1). Spins can either align themselves in upward direction(Si = +1) or in the downward(Si = +1) direction. The shape of the lattice here is s square . Thus, a two dimensional Ising Model with a size of L will consist of L^2 spins.

NOTE: Use *Shift+Enter* to run the codes written in the cells.

1.1 Need for Metropolis Algorithm

Probabaility of each microstate is given by: (

$$P_c = \frac{e^{-\beta \hat{H}_i}}{\sum_c e^{-\beta \hat{H}_i}} \tag{2.1}$$

To get the precise value of magnetization, we must calculate the magnetization for as many microstates/configurations as possible and then take the mean. Generating each and every microstate is a hopeless effort (For a small lattice size of L=10, number of possible microstates are 2^{100}).(

$$\langle M \rangle = \frac{\sum_{c=1}^{2^{100}} m_c \times P_c}{2^{100}} \tag{1.1}$$

But as in (1.1) the probability of every microstate is different. Without generating every single configuration, one can sample only those configurations which contribute most to the average (3.2). This is known as importance sampling. Here, the role of Metropolis Monte Carlo comes in. This importance sampling based algorithm reduces the computational time by a significant amount.

1.1.1 Steps for 2D Ising simulation:

1.Create a configuration with random alignment of spins.
2.Randomly select a spin and flip it, to generate a trial configuration.
3.Compute the energy difference ($\Delta E = E_B - E_A$).
4.If $\Delta E < 0$, accept the trial configuration with acceptance probability 1.
5.If $\Delta E > 0$, accept the trial configuration with acceptance probability $e^{-\Delta E}$.
6.Perform sampling for large number of cycles.

1.2 Code for equilibration

This function is called by the main file whenever necessary.

```
In [189]: function Equilibration(n_grid,T,J,L)

          #creating random arrangement
          grid=randn(n_grid,n_grid)
          for i in 1:n_grid^2
              if grid[i]>0.5
                  grid[i]=1
              else
                  grid[i]=-1
              end
          end

          numIters = (2^9)*length(grid)

          #pick a random spin
          for iter in 1:numIters
              row = rand(1:n_grid)
              col = rand(1:n_grid)

              #nearest neighbors
              if col==1
                  left=n_grid
              else
                  left=col-1
              end
              if col==n_grid
                  right=1
              else
                  right=col+1
              end
              if row==1
                  below=n_grid
              else
                  below=row-1
              end
              if row==n_grid
                  above=1
              else
                  above=row+1
              end

              neighbors=grid[above,col]+grid[row,left]+grid[row,right]+grid[below,col]

              #Energy change after spin flip
              dE = 2*(J*grid[row,col]*neighbors)
```

```
            #Spin flip condition
            if dE <= 0
                grid[row,col] = -grid[row,col]
            else
                prob=exp(-dE/T)
                r=rand(1)
                if  r[1,1] <= prob
                    grid[row,col] = -grid[row,col]
                end
            end
        end
        return grid
        end
```

```
WARNING: Method definition Equilibration(Any, Any, Any, Any) in module Main at In[187]:4 overwri
```

```
Out[189]: Equilibration (generic function with 2 methods)
```

1.3 Code for Calculating average properties (Production Run)

This function is called by the main file whenever necessary.

```
In [190]: #Production
          function Production(n_grid,T,J,L,grid)
          Mmean=zeros(1,L)
          Emean=zeros(1,L)
          for iter in 1:L
              row = rand(1:n_grid)
              col = rand(1:n_grid)

              #nearest neighbors
              if col==1
                  left=n_grid
              else
                  left=col-1
              end
              if col==n_grid
                  right=1
              else
                  right=col+1
              end
              if row==1
                  below=n_grid
              else
                  below=row-1
              end
              if row==n_grid
```

```
            above=1
        else
            above=row+1
        end

        neighbors=grid[above,col]+grid[row,left]+grid[row,right]+grid[below,col]

        #Energy change after spin flip
        dE = 2*(J*grid[row,col]*neighbors)

        #Spin flip condition
        if dE <= 0
            grid[row,col] = -grid[row,col]
        else
            prob=exp(-dE/T)
            r=rand(1)
            if  r[1,1] <= prob
                grid[row,col] = -grid[row,col]
            end
        end

        #Calculating Properties
        Mmean[1,iter]=mean(grid)

        #sumofneighbors=circshift(grid,[0 1])+circshift(grid,[0 -1])+circshift(grid,[1 0])
        #Em = - J*grid.*sumofneighbors
        #E=0.5*sum(Em)
        #Emean[1,iter]=E/length(grid)
    end
    gridpr=grid;
    Ms=mean(Mmean);
    #Es=mean(Emean)
    xs=(mean(Mmean.^2)-mean(Mmean)^2)/T;
    #Cs=(mean(Emean.^2)-mean(Emean)^2)/T^2
    return gridpr, Ms, xs
    end
```

```
WARNING: Method definition Production(Any, Any, Any, Any, Any) in module Main at In[188]:3 overw
```

1.4 Code for generating equilibrated arrangements by calling Equilibration

This function generates equilibrated arrangement for different temperatures. It is called by the main code whenever necessary.

```
In [193]: function ising_over_temp(n_grid,J,L,Tmin,Tinc,Tmax)
          len = floor((Tmax-Tmin)/Tinc);
          len=convert(Int64,len);
```

```
    #allocating required memory
    gridqm=Array{Array{Int8}}(1,len+1);
    Ts=Array{Float64}(1,len+1);
    i=1;

    # The temperature loop
    print("equilibration started!","\n");
    print("Number of steps = ",(2^8)*n_grid^2,"\n");
    for T in Tmin:Tinc:Tmax
        grid = Equilibration(n_grid, T, J, L);

        #storing equilibrated arrangements with temperature
        gridqm[1,i] = grid;
        Ts[1,i] = T;
        i=i+1;
    end
    print("equilibration finished!","\n");
    len = length(Ts);
    return gridqm, Ts, len
    end
```

```
WARNING: Method definition ising_over_temp(Any, Any, Any, Any, Any, Any) in module Main at In[19
```

```
Out[193]: ising_over_temp (generic function with 1 method)
```

1.5 Main Code with input parameters

NOTE: Run all the codes in above cells before running this main code as it uses functions which are defined above.

This is the main code that calculates magnetisation and susceptibility.

Change the input parameters within next cell and press *Shift+Enter* to run the simulation.

```
In [200]: n_grid=40;     #Lattice Size
          L=1000000;     #MC Steps
          J=1;           #J Constant
          P=3            #Production Runs
          Tmin=1;      #Minimum Temp
          Tinc=0.05;     #Increment in Temp
          Tmax=3;        #Maximum Temp

          gridqm, Ts, len = ising_over_temp(n_grid,J,L,Tmin,Tinc,Tmax)

          Mp=zeros(P,len);
          x=zeros(P,len);

          for Pr in 1:P
              print("production run = ",Pr,"\n")
              for h in 1:len
```

```
                gridpr,Ms,xs = Production(n_grid,Ts[1,h],J,L,gridqm[1,h])
                #print(h,"\n")
                Mp[Pr,h] = Ms;
                x[Pr,h] = xs;
            end
        end

        Mp_avg=mean(Mp,1);
        x_avg=mean(x,1);



        using Plots
        plt1 = Plots.scatter(Ts,Mp_avg,color="red",legend=false,xaxis="Temp",yaxis="<M>/spin")
        plt2 = Plots.scatter(Ts,x_avg,color="green", reuse= false,legend=false,xaxis="Temp",ya
        display(plt1)
        display(plt2)

        print("FINISHED!")
```

```
equilibration started!
Number of steps = 409600
equilibration finished!
production run = 1
production run = 2
production run =

3
FINISHED!
```