1. Transfer matrices

Consider a 1D "spin" chain of length L and periodic boundary conditions. At each site i in the chain there is a spin σ_i that can take one of three values: $\{0, 1, 2\}$. The length L is measured in units of the lattice spacing and is a dimensionless integer. Thus there are in all $N = L$ different spins, numbered from 0 to $L - 1$. The periodic boundary condition makes spin L equivalent to spin 0. The Hamiltonian favors alignment of spins and is

$$H = -J \sum_{i=0}^{L-1} \delta_{\sigma_i, \sigma_{i+1}}$$

where $J > 0$.

The order parameter or "magnetization per site" for this system is defined as

$$m \equiv \frac{1}{N} \sum_{j=0}^{N-1} m_j$$

where the local "magnetization" m_j is the complex quantity

$$m_j \equiv e^{i(2\pi/3)\sigma_j}.$$

a) Use the transfer matrix approach to calculate the canonical partition function Z for this model. Use the obtained expression for Z to calculate the average total internal energy U. Also give approximate expressions for U valid for low and high temperatures. Explain briefly these limiting behaviors in words.

b) Use transfer matrices to show that $\langle m \rangle = 0$ at all temperatures.

c) Calculate using the transfer matrix approach the correlation function $C(r)$ for two spins separated by a distance r

$$C(r) \equiv \langle m_0^* m_r \rangle - \langle m_0^* \rangle \langle m_r \rangle.$$

Give both the exact expression for finite L and the limiting result for $L \to \infty$. What is $C(r)$ for $T = 0$?

2. Monte Carlo

a) Make a Monte Carlo (MC) program for the spin model in problem 1 that uses the **Wolff algorithm**, i.e. the single cluster version of the Swendsen-Wang algorithm. It should work both for the 1D chain with periodic boundary conditions as well as for the 2D square lattice with periodic boundary conditions in both directions. For your convenience a minimal-style, not entirely complete, C++ program for the 1D case is listed at the end of the problem set. You can either modify it, or make your own new code.

b) Check that your program reproduces the exact results you obtained for the 1D chain in problem 1c). To be concrete use $L = 16$ and make a plot of $C(r)$ where you plot the real part of C vs. r from both the MC simulation and the exact result from 1b) for two values of the temperature, $T/J = 0.25$ and $T/J = 0.5$.

Consider hereafter the spin model on a *two dimensional* square lattice with $N = L \times L$ sites and periodic boundary conditions in both directions.

c) Make a plot of the real part of the average magnetization per site $\langle m \rangle$ vs. T/J for $L = 16$. What is the value of $\langle m \rangle$ at $T/J = 0$? And what is it at $T/J \to \infty$?

d) Make a plot of the average magnetization squared per site $\langle |m|^2 \rangle$ vs. T/J for $L = 16$. What is the value of $\langle |m|^2 \rangle$ at $T/J = 0$? And what is it at $T/J \to \infty$?

A common way of finding the critical temperature of a phase transition is to consider a dimensionless ratio of moments of the magnetization, for instance

$$\Gamma \equiv \frac{\langle |m|^4 \rangle}{\langle |m|^2 \rangle^2}$$

and to plot Γ vs. T/J for different system sizes L.

e) Show using finite size scaling that the curves of Γ vs. T/J for values of L cross at the phase transition temperature. Assume that $\Gamma = \Gamma(t, L^{-1})$ i.e. that Γ is a function of the reduced temperature $t \equiv (T - T_c)/T_c$ and the inverse (linear) system size L^{-1} only.

f) Estimate the critical temperature T_c for the square lattice spin model by plotting Γ vs. T/J for different system sizes $L = \{8, 16, 32\}$. Compare with the exact result $T_c/J = \frac{1}{\ln(1+\sqrt{3})} \approx 0.9950$. Do the curves cross at a single point? If not, what can the reason be?

A. A minimal Wolff algorithm C++ program

```
#include<iostream>
#include<vector>
#include<cstdlib>
#include<math.h>
#include<complex>
using namespace std;

#define PI 3.14159265358979323846264 3
const int q=3;              // q spin states
const int L=8;              // linear system size
const double T=2;           // temperature in units of J

const int N=L;       // total number of spins
const double pconnect=???; // connection probability

const int NCLUSTERS=1; //     # of cluster builds in one MC step.
const int NESTEPS=10000; // # of equilibrium MC step.
const int NMSTEPS=10000; // # of measurement MC step.
const int NBINS    =10 ; //    #of measurement bins

vector<int> S(N);   // the spin array
vector<int> M(q);   // number of spins in different states.
vector<complex<double> > W(q); // order parameter weights

// lattice handling:
enum dirs{RIGHT,LEFT};
int indx(int x){return x;} // make an indx on every site
int xpos(int i){return i%L;}

int Nbr(int i,int dir)
```

```
{
  int x=xpos(i);
  switch(dir)
    {
    case RIGHT: return indx((x+1)%L);
    case LEFT:  return indx((x-1+L)%L);
    }
}

void FlipandBuildFrom(int s)
{
  int oldstate(S[s]),newstate((S[s]+1)%q);

  S[s]=newstate; // flip spin
  M[oldstate]--; M[newstate]++;  // update spin counts

  for(int dir=0; dir<2; dir++) // go thru neighbors
    {
      int j=Nbr(s,dir);
      if(S[j] == oldstate)
        if( rand()/(RAND_MAX+1.) <pconnect){FlipandBuildFrom(j);}
    }
}

int main()
{
  // initialize order parameter weights
  for(int s=0; s<q; s++)
    W[s]=complex<double>(cos(2*PI*s/q),sin(2*PI*s/q));
  for(int i=0; i<N; i++) S[i]=0;    // initialize to the spin=0 state
  for(int s=1; s<q; s++) M[s]=0;    // initialize counters.
  M[0]=N;
  srand((unsigned) time(0)); // initialize random number gen.

  // equilibriate
  for(int t=0; t<NESTEPS; t++)
    for(int c=0; c<NCLUSTERS; c++)
      {
        FlipandBuildFrom(rand()%N);
      }

  // measure
  for(int n=0; n<NBINS; n++)
    {
      complex<double> m(0.,0.);
      double m1=0, m2=0, m4=0; // measurement results

      for(int t=0; t<NMSTEPS; t++)
        {
          for(int c=0; c<NCLUSTERS; c++) FlipandBuildFrom(rand()%N);
          complex<double> tm(0.,0.);
          for(int s=0; s<q; s++){tm+= W[s]*double(M[s]);}
          tm/=N;
          double tm1=abs(tm);
          double tm2=tm1*tm1;
          m+=tm; m1+=tm1; m2+=tm2; m4+=tm2*tm2;
        }
      m/=NMSTEPS; m1/=NMSTEPS; m2/=NMSTEPS; m4/=NMSTEPS;
      cout << m << " " << m1 << " " << m2 << " " << m4 << endl;
    }
}
```