2009
Computer Exercises
Daniel Jost & Ralf Everaers

Session 1 : simple sampling

1 Calculation of π

Consider a circle of diameter d surrounded by a square of length l ($l \geq d$) (see Fig.1). Random coordinates within the square are generated. The value of π can be calculated from the fraction of points that fall within the circle.

1. How can π be calculated from the fraction of poins that fall in the circle? Remark: the "exact" value of π can be computed numerically using $\pi = 4\arctan(1)$.
2. Complete the Monte-Carlo program to calculate π using this method.
3. How does the accuracy of the result depend on the ration l/d and of the number of generated coordinates? Derive a formula to calculate the standard deviation of the estimate of π.
4. Is it a good idea to calculate many decimals of π using this method?

2 $2D$-Ising model

In this model, N spins $\{S_i = \pm 1\}$ are arranged on a $2D$-square lattice (each row and column contain P sites, *ie* $N = P^2$) (see Fig.2). The hamiltonian of the system is given by

$$H = -J \sum_{\langle i,j \rangle} S_i S_j \tag{1}$$

where J is the coupling constant and $\langle i, j \rangle$ represents the sum over the nearest neighbours with periodic boundary condition. Onsager in 1944 solved analytically this model and found a critical point at the temperature $T_c \approx 2.269 J/k_B$. In the following, we assume $J > 0$ and we note $T' = k_B T/J$.

1. What are the maximal and minimal values for the total magnetization $M = \sum_i S_i$ and the nearest-neighbour coupling $NNC = \sum_{\langle i,j \rangle} S_i S_j$ for a system of N spins?
2. Complete the subroutine *randomconfig.f90* to randomly generate a spin matrix and to compute the corresponding M and NNC values. Remark: a spin matrix is generated by randomly chosing between $\{-1, 1\}$ for each lattice site.
3. How can one evaluate the density of state $D(M, NNC)$ with simple sampling?
4. Complete the updating of D in *main.f90*.
5. How can one compute the thermodynamic average of a given variable $X(M, NNC)$ at the reduced temperature T' using D?
6. Why is it better to evaluate the root-mean squared magnetization $\bar{M}(T') = \sqrt{\langle M^2 \rangle}$ instead of the mean magnetization $\langle M \rangle$?

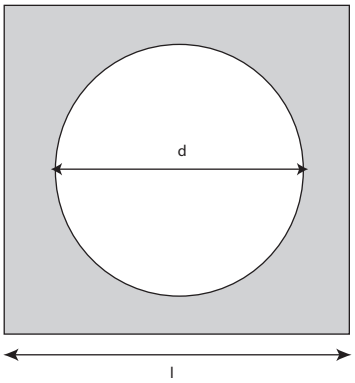

Figure 1: a circle of diameter d surrounded by a square of length l.

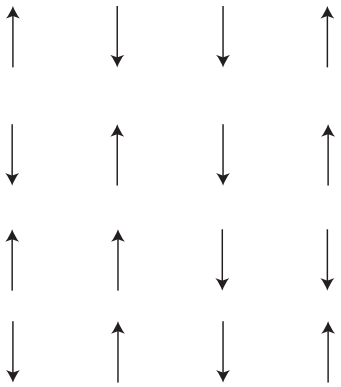

Figure 2: Example of a 4×4 Ising configuration.

7. Complete the subroutine *rmsm.f90* to compute $\bar{M}(T')$ from D.
8. Using the program, evaluate and trace $\bar{M}(T')$ and D for a 8×8 system using 100, 1000 or 10000 random configurations.
9. What are the probabilities to generate a configuration with $M = 0$ or with $M = M_{max}$?
10. Using the two previous questions, make comments on solving the 2D Ising model using simple sampling.

3 $3D$ Self-avoiding walks

We look at properties of a single chain molecule. We consider a molecule composed by N steps (or $N + 1$ beads) making a self-avoiding walk on a $3D$ cubic lattice (see Fig.3). Each lattice site has $z = 6$ neighbours. We are interested by the total number of different configurations $\mathcal{N}_{SAW}(N)$ and by the end-to-end root mean squared distance $R_{rms}(N)$.

1. What is the number of configurations and the end-to-end root mean squared distance for an ideal walk?
2. Complete the subroutine *randomchain.f90* to generate a random SAW and to compute its end-to end distance. Remark: a configuration is generated by iteratively growing the chain from bead 1 to bead $N+1$. At each step, a random neighbour is chosen among the z possible directions. If the site is not occupied, the process continues otherwise the chain is throw out and a new growing process begins. The self-avoidance checking is made using hash coding techniques (see Annexe).
3. From the fraction of well-growth chains, how can one estimate $\mathcal{N}_{SAW}(N)$? Estimate the error made after P growing trys.
4. How can one evaluate $R_{rms}(N)$? Estimate the error with M random chains.
5. Complete the computation of $\mathcal{N}_{SAW}(N)$ and $R_{rms}(N)$ in *main.f90*.

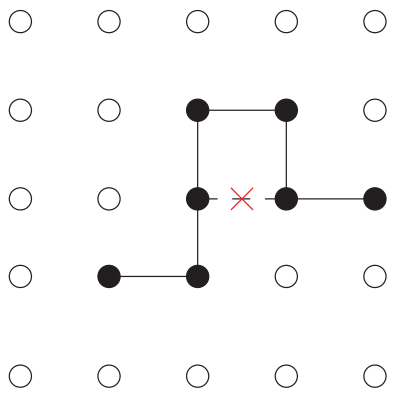

Figure 3: A self avoiding walk on a $2D$ square lattice for $N = 6$.

6. Trace $\mathcal{N}_{SAW}(N)$ and $R_{rms}(N)$ (up to $N = 100$). What happens for long chains? Any comments? Remark: one can show that for a self-avoiding walk $R_{rms}(N) \sim N^{0.588}$ and $\mathcal{N}_{SAW}(N) \sim \mu^N N^{1/6}$ with $\mu = 4.68$ for a cubic lattice.

Annexe - *the method of hash-coding (Madras & Sokal. 1988. J. Stat. Phys.)* : An array of M words is assigned, and each position x is assigned a primary address $h(x)$ in this array. Since in general M is much smaller than the number of possible positions ,the "hash function" h is necessarily many-to-one, i.e., many distinct positions may share the same primary address, leading to the possibility of collisions. The various hash-coding algorithms are distinguished by the method they use to resolve collisions, i.e., to decide where to store a position if its primary address happens to be occupied by some other position. One of the simplest collision-resolution schemes, and the one we use, is linear probing: if the primary address $h(x)$ is occupied, the algorithm searches successively in addresses $h(x) + 1$, $h(x) + 2$ .... (modulo M) until it finds either the position x or an empty slot. In the worst possible case, a single query or insertion into a hash table containing N entries could take a time of order N. However, it can be shown that as long as the hash table does not get close to full (i.e., N does not get near M), then the average time (i.e., if the points are randomly distributed) for a single query or insertion is of order 1. So the hash-coding method is far more effective than the naive method for which a new position is compared to the other already former positions. In choosing the hash function h, we want the image set $h[x_1, ..., x_N]$ to be "sparse": that is, if x_i happens to be close to x_j, then we want $h(x_i)$ to be far from $h(x_j)$. This is particularly important, since the occupied lattice sites in a self-avoiding walk are close together. We used hash functions of the form $h(x_{i,1}.....x_{i,d}) = (a_1 x_{i,1} + ... + a_d x_{i,d}) mod M$ where d is the lattice dimension, and, a_1,..., a_d, M are chosen to be relatively prime and satisfy $a_k \approx M^{k/(d+1)}$. Thus, the a_k are all of different magnitude, which helps ensure the desired behavior of h. [To understand this, think about why $h(x_{i,1}, ..., x_{i,d}) = (x_{i,1} + ... + x_{i,d}) mod M$ is a bad hash function.] In particular, because we are using linear probing, we want to avoid near-collisions as well as collisions; this is why we insist on $a_k > 1$ for all k.