1 Introduction

The objective with the *aggregates.cpp* program is to evaluate the time evolution of aggregates from Langevin Dynamics simulations. The systems of interest are defined as a collection of individual particles (*beads*0 of size $\sigma = 2R$. These beads interact via two different particle-particle interactions. In one hand, in order to avoid overlapping between different beads, a Lennard-Jone type potential is defined, through two different parameters, σ_{LJ} the distance at which the energy decays to zero and ϵ_{LJ} corresponding to the depth of the potential well. In the other hand, magnetic dipole-dipole interaction between beads is implemented by defining a point dipolar moment for each bead, located at its geometrical center. The Lennard-Jones parameters are chosen to prevent the overlapping of beads but, at same time, to keep an aggregation contact distance close to its diameter, trying to emulate each bead as a hard sphere model.

The program uses the dump files obtained from LAMMPS in which, for each timestep selected, each particle position is stores as well as its id from LAMMPS.

2 Algorithm

The basic idea behind the algorithm is the following. First of all, we have to define a contact distance d_c between beads in the sense that if the distance between two beads is smaller or equal to this value, then we will say that they belong to the same aggregate. Lets say $d(i, j)$ is the distance between two beads. Then, if $d(i, j) \leq d_c$ then we add the bead j in the neighbor list of bead i, and vice versa. So, for each timestep we need to compute the neighbor list for any individual bead on the system.

All the beads are stored in a main array so, each element of the array is a bead object. The neighbor list is an array of pointers to bead, so, each element of this neighbor array is pointing tho the memory address of any neighbor bead. The function which performs this operation in the algorithm is called *GetConnections()*.

The next step is to count the size of each aggregate. The procedure is the following:

1. Select the (first) bead in the main array and set its internal state to *check*.

2. If its neighbor list is empty, the bead counts as an aggregate with size 1, and an aggregate id is assigned to it. We reset the aggregate size counter to zero and restart from 1. If the neighbor list is not empty, we sum 1 to the aggregate size counter, set the internal state of the bead to *check*, select the first neighbor of this last bead and we repeat this step.

The two functions used to accomplish this task are *GetSize(&bead[i],i,size,aggid)* and *CountAgregates()*. During the counting, the number of aggregates of each size is stored in an array called *hist* and results are stored at each timestep. So, after processing each configuration, all the variables are reset to default values.

3 Outputs

Four different output files are stored:

hist.txt Column-formatted file. The first column corresponds to the timestep. The, the second column corresponds to the number of aggregates of size 1 (i.e. individual beads), and so on.

norm.txt Is the same *hist.txt* file, but the number of aggregates have been normalized by the number of aggregates at this timestep.

stdmean.txt Stores the mean aggregate size at each timestep, defined as:

$$s(t) = \frac{\sum_s s n_s(t)}{\sum_s n_s(t)}$$

s-mean.txt Stores the average (mass) aggregate size defined as:

$$S(t) = \frac{\sum_s s^2 n_s(t)}{\sum_s s n_s(t)}$$