Franz Edward Boas
Chem 161
20 May 1997

The Behavior of Small Clusters of Atoms in Thermal Equilibrium

Abstract

The thermodynamics of small clusters of atoms differs from that of bulk matter because of the discreteness of individual atoms. In particular, this paper considers the properties of small clusters of weakly interacting particles in a vacuum at thermal equilibrium. These liquid-like clusters can remain in stable quasi-equilibria at certain temperatures, although they will evaporate at any temperature given a sufficiently long time scale. Over the time scales considered in this paper, the binding energy of the ground state and the boiling point both increased with the number of atoms in the cluster. Evaporation of a cluster is often preceded by a long period of quasi-equilibrium, but the departure of a single particle causes the rest of the cluster to then quickly evaporate into individual atoms.

Introduction

This paper will examine the behavior of small clusters of atoms (N=2 to 14, and 50) interacting via a pair-wise Lennard-Jones potential (equilibrium radius and dissociation energy both normalized to 1). The clusters will be isolated at zero pressure and at thermal equilibrium. Such a situation might arise from the evaporation of a liquid into a vacuum, creating clusters with a Boltzmann distribution of energies.

Procedures

Clusters of N particles at temperature T were studied using the Metropolis algorithm for Monte Carlo simulation. At each time step, the three spatial coordinates of a single, randomly chosen particle were perturbed by a normally distributed random number with a standard deviation of 0.04 atomic diameters. Using normal deviates proposes moves in all directions with equal probability, unlike uniform deviates or moves along a discrete lattice which would both be anisotropic with respect to the coordinate axes. Each proposed move is always accepted if the total potential energy decreases, and accepted with probability $e^{-\Delta E/kT}$ if the energy increases.

The starting geometry of the clusters simulated was determined by a two-part equilibration scheme. First, randomly placed atoms were confined to a sphere with four times the total volume of the atoms. These atoms were cooled for 0.1 million steps per atom from a temperature equal to the dissociation energy exponentially decaying down to 0.005 of the dissociation energy. This simulated annealing procedure, performed with the Metropolis algorithm described above, perhaps qualitatively mimics the cooling procedure that could have generated the cluster in the real world. More importantly, it ensures that the simulation begins with an intact cluster near its ground state. Without this step, the initially configuration will likely get stuck in high energy local minima in low temperature simulations, and might spuriously evaporate at a higher temperature if the atoms randomly started too far apart.

The second part of the equilibration scheme involves running the Monte Carlo simulation at the desired temperature for half a million steps per atom without collecting any data. After equilibration, the simulation runs another half million steps per atom while recording information such as energy, dispersion in energy, the coordinates of the atoms, and cluster sizes. The working definition of a cluster employed in this project is a set of particles bonded together, where a "bond" forms between two particles centered within two atomic diameters of each other.

Each simulation is run 10 times and the results averaged.

Zero temperature simulations correspond to energy minimization, and are especially difficult because of the large number of local minima. In this project the minimization was done with 50 independent Monte Carlo simulations as described above at T=0.0001. These tentative solutions to the optimization problem were "polished" by following the energy gradient to the nearest minimum, and then only taking the best solution found. For all N, a majority of the randomly restarted runs converged to the same minimum, suggesting that it is the global minimum.

The C++ code for the simulations appears in the appendix.

Results

The geometry of the ground state configuration of Lennard-Jones clusters closely follows intuition up to at least N=6. (Please see graphs while reading this discussion!) For example, 4 particles form a pyramid, 5 form a trigonal bipyramid, and 6 form a square bipyramid. N=7 is an especially interesting case as its pentagonal bipyramidal geometry lies at the core of ground states at least up to N=14. N from 8 to 12 fill in adjacent dimples on the faces of one of the pentagonal pyramids to form another pentagonal ring of atoms.

N=13 places a final atom in the indentation at the center of this new pentagonal ring. This geometry is another interesting case as it consists of a central sphere tangent to 12 other spheres which are each tangent to 6 neighbors. 12 is called the "kissing number" for spheres since it is the maximum number of non-intersecting spheres that can be tangent to a central sphere. Each sphere in a cubic or hexagonal closest packing also features the maximum kissing number, but its unit sphere is not the ground state for a cluster of 13 particles since the non-central spheres have one fewer neighbor than the ground state. Conversely, the N=13 cluster does not form the unit cell for an infinite crystal since its pentagons cannot be tiled. Finally, N=14 consists of the N=13 geometry with a weakly bound particle stuck on the side.

The average binding energy per particle increases with the number of particles, approaching an asymptotic value of about 6 units per particle, corresponding to a closest-packed infinite lattice. The binding energy of the weakest bound particle also increases with the number of particles, but several values of N have anomalous values. For example, the ground states of N=8 and N=14 both have a single weakly bound particle.

Below the boiling point, all clusters exhibited a liquid-like phase where the cluster stays intact for the entire simulation and particles can diffuse through the cluster, exploring conformations largely reminiscent of the ground state. Higher temperatures result in greater fluctuations in energy about a slightly greater average energy. These trends reflect the increasing heat capacity and entropy with temperature.

Near or at the boiling point, the clusters begin to dissociate by losing a single particle at a time. It is uncertain whether multiple particles actually do not leave simultaneously, or whether this is just an artifact of a Monte Carlo move set that only moves a single particle at a time.

Clusters are fairly compact, even near the boiling point, and the loss of a particle is almost a discrete process occurring stochastically with little warning. Once the particle gets more than about two atomic diameters away from the cluster, it continues to move away from the cluster in a random walk. Simulations near phase transitions represent nonequilibrium results, with some clusters remaining intact, and some beginning to boil.

Above the boiling point, an initial equilibrium phase of cluster stability lasts for a randomly distributed interval, ended by the loss of a single particle. The loss of this single particle speeds up the loss of additional particles, after which the cluster quickly completes its evaporation. This observation is confirmed by plots of energy as a function of Monte Carlo step, and by cluster size distributions that record intact and completely evaporated clusters but few intermediate steps.

The boiling point of these small clusters, defined as the lowest temperature at which evaporation was observed in the simulations, increases almost linearly with N from about T=0.1 dissociation energy units for N=2 to almost T=0.3 units for N=10. After that, the boiling point stays around 0.3 units, even for N=50. The initially increasing boiling point with N parallels the increasing average binding energy of larger clusters at their boiling point, which parallels the increasing binding energy of the ground state. The leveling off of the boiling point curves at large N is a consequence of the average binding energy per particle, which also reaches an asymptotic value. The energy of the weakest bound particle in the ground state is less well correlated with the boiling point since it does not usually stay the weakest bound, but rather diffuses throughout the cluster and "feels" the average binding energy.

The increase in boiling point with N also explains why evaporation occurs so quickly after the loss of a single particle: the remaining cluster now has fewer particles and thus a lower boiling point.

Graphs of energy as a function of temperature exhibit the properties already discussed qualitatively. Below the boiling point, energy increases slowly with temperature, and larger clusters have more negative energies. No solid-liquid transition can be detected from the energy data. Around the boiling point, the average energy quickly increases to zero, but this represents the average of nonequilibrium clusters on the verge of boiling away. Above the boiling point, the clusters have completely evaporated and have zero potential energy.

Some of the $E(T)$ data points were rerun using Monte Carlo simulations an order of magnitude longer, with very similar results. However, over much greater changes in time scale, we would expect the $E(T)$ curves and the boiling point to change slightly. In fact, given an infinite amount of time, the clusters will boil at any temperature and reach their true equilibrium as a gas since the entropy of this state is much greater. Thus, intact clusters are in quasi-equilibrium and their thermodynamic properties will vary with the time scale, albeit slowly.

The entropy of intact quasi-equilibrium clusters can be calculated from the time dependence of the energy using $S(T) = \int_0^T \frac{1}{T}\frac{\partial E}{\partial T}dT$. This integral is slightly difficult to perform on discrete data points because of the apparent singularity at T=0. The singularity is removed by assuming the heat capacity $\frac{\partial E}{\partial T}$ to vary as T^3 for low temperatures, as in the Debye theory of solids. The heat capacity for larger temperatures can be inferred by assuming a linear relationship between E and T. As expected, entropy per particle increases with the N because of the greater number of possible places to put each subsequent particle. The Helmhotz free energy F=E-TS decreases with the number of particles, just like the energy, showing that there is an activation energy for losing particles.

Conclusion

The small clusters of atoms studied in this report exhibit a boiling point and average binding energy per particle increasing toward an asymptotic value. Other experimenters studying similar systems have discovered that certain clusters with a "magic number" of atoms tended to be more stable. The simulations of this project found no such magic numbers, perhaps because the magic numbers of the system are greater than 14, or perhaps because the Monte Carlo simulations are not sufficiently accurate. One possible improvement to the simulation would be to decrease the temperature appropriately after the evaporation of a particle. Evaporative cooling might be sufficient to halt the rapid evaporation process from the simulations in this paper.

C++ code for simulations

```
// program outputs average energy per particle
// as a function of N and T
// for small clusters of weakly interacting particles

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include <time.h>
#include "useful.hpp"
#include "useful.cpp"
#include "vector.hpp"
#include "vector.cpp"


// constants for potential energy
const float Req=1.0; // equilibrium radius
const float De=1.0; // dissociation energy
/*
const float potential_table_lower_bound=0.5*0.5;
const float potential_table_upper_bound=5*5;
const int potential_table_resolution=100;
*/

const float A=2*pow(Req, 6)*De; // coeff. for -1/r^6
const float B=pow(Req, 12)*De; // coeff. for +1/r^12

// end_N must be larger
int start_N=13;
int end_N=13;
int N;
vector *pos;
int *histogram; // of cluster sizes
float **energy;
float E;

// start_T must be larger
float start_T=0.05;
float end_T=0.05;
float step_T=0.001;

// parameters for initial cooling
float start_T_setup=De;
float end_T_setup=0.005*De;
float step_T_setup=0.95; // fraction multiplied to decrease temp.
long steps_setup=1000; // steps per particle per temperature

BOOL detailed=TRUE;
long nth_step=200;
long mth_of_nth_steps=4;

const float size_factor=4; // factor times packed volume
const float initial_step_size=0.04; // actual step_size is dynamically adjusted for a 50% accept rate
// #define ADAPTIVE_STEPSIZE
float min_step_size=0.04; //0.005;
float max_step_size=0.04; //0.5;
float step_increment=1.0; //1.001;
float step_decrement=1/step_increment;
float step_size;

// Note: "step" refers to Monte Carlo step
//       "trial" refers to independent Monte Carlo runs
long start_step=1000;
long end_step=8000;
int num_trials=3;
float ***detailed_data;

FILE *data_file;
FILE *coordinate_data_file;
FILE *cluster_data_file;
```

```
char data_file_name[256];
char coordinate_data_file_name[256];
char cluster_data_file_name[256];


// Lennard-Jones: potential(r) = -A/r^6 + B/r^12
// function expects r^2 (to save computation costs)
inline float potential(float rsquared) {
  float temp=1/(rsquared*rsquared*rsquared);
  return temp*(B*temp-A);
}


// calculate energy from scratch
void calculate_energy(void) {
  int i, j;
  E=0;
  for (i=0; i<N; i++) {
    for (j=i+1; j<N; j++) {
      energy[i][j]=potential(sqr(pos[i].x-pos[j].x)+sqr(pos[i].y-pos[j].y)+sqr(pos[i].z-pos[j].z));
      E+=energy[i][j];
    }
  }
}

// this is necessary to fix some unknown Borland C++ bug
int a__[5000];
int b__[5000];
int id__[100];
int count__[100];

// makes histogram of cluster sizes
// fresh = TRUE  -> new histogram
//       = FALSE -> increment existing histogram
void generate_cluster_size_histogram(BOOL fresh) {
  int i, j, numequiv=0;
  // a, b: lists of points in the same cluster
//  int *a, *b;
  // id: cluster number for each point
  // count: number of members for each cluster number
//  int *id;
//  int *count;
  // allocate enough for max number of equivalencies
//  a=new int[N*(N-1)/2];
//  b=new int[N*(N-1)/2];
//  id=new int[N];
//  count=(int*) calloc(N, sizeof(int));
  for (i=0; i<N; i++)
    count__[i]=0;
  if (fresh)
    for (i=0; i<=N; i++)
      histogram[i]=0;

  // build list of equivalencies
  for (i=0; i<N; i++) {
    for (j=i+1; j<N; j++) {
      if (sqr(pos[i].x-pos[j].x)+sqr(pos[i].y-pos[j].y)+sqr(pos[i].z-pos[j].z)<4*Req*Req) {
        // i and j in same cluster
        a__[numequiv]=i;
        b__[numequiv]=j;
        numequiv++;
      }
    }
  }
  // assign cluster numbers
  eclass(id__, N, a__, b__, numequiv);

//  for (i=0; i<N; i++)
//    cerr << id__[i] << " ";
//  cerr << "\n";
//  getch();


  // count members in each cluster
  for (i=0; i<N; i++)
```

```
    count__[id__[i]]++;
  // generate histogram
  for (i=0; i<N; i++)
    if (count__[i]) histogram[count__[i]]++;
//  delete[] a;
//  delete[] b;
//  delete[] id;
//  free(count);
}


// make single Monte Carlo step
// takes initialized pos, filled energy table,
// current energy E, number of particles N, temperature T
// returns new energy
void monte_carlo_step(float T) {
  int which, i, j;
  float old_E;
  vector oldpos;

  // make step
  which=floor(N*ran1());
  oldpos=pos[which];
  pos[which].x += step_size*gasdev();
  pos[which].y += step_size*gasdev();
  pos[which].z += step_size*gasdev();

  // calculate new energy
  old_E=E;
  for (i=which+1; i<N; i++)
    energy[which][i]=potential(sqr(pos[which].x-pos[i].x)+sqr(pos[which].y-pos[i].y)+sqr(pos[which].z-
pos[i].z));
  for (i=0; i<which; i++)
    energy[i][which]=potential(sqr(pos[which].x-pos[i].x)+sqr(pos[which].y-pos[i].y)+sqr(pos[which].z-
pos[i].z));
  E=0;
  for (i=0; i<N; i++)
    for (j=i+1; j<N; j++)
      E+=energy[i][j];

  // accept step ??
  float delta_E = E - old_E;
  #ifdef ADAPTIVE_STEPSIZE
    if (delta_E<0 || ran1()<exp(-delta_E/T)) { // accept
      step_size*=step_increment;
      if (step_size>max_step_size) step_size=max_step_size;
    } else { // reject
      step_size*=step_decrement;
      if (step_size<min_step_size) step_size=min_step_size;
      pos[which]=oldpos;
      E=old_E;
    }
  #else
    if (delta_E>0 && ran1()>exp(-delta_E/T)) { // reject
      pos[which]=oldpos;
      E=old_E;
    }
  #endif
}


void output_coordinates(FILE *f, float T, int trial, long step) {
  int i;
  fprintf(f, "%i %f %i %li %e ", N, T, trial, step, E/N);
  for (i=0; i<N; i++)
    fprintf(f, "%e %e %e ", pos[i].x, pos[i].y, pos[i].z);
  fprintf(f, "\n");
}

// run Monte Carlo once for fixed N, T
void monte_carlo(float T, int trial, float &E_bar, float &E2_bar) {
  double sum_E=0, sum_E2=0; // double to reduce rounding error when summing many terms
  float initial_size=size_factor*(Req/2)*pow(N, 1/3);
  step_size=initial_step_size;
  long step;
  vector oldpos;
  int i, j;

  // initialize with random positions
  for (i=0; i<N; i++) {
    pos[i].x=initial_size*gasdev();
    pos[i].y=initial_size*gasdev();
    pos[i].z=initial_size*gasdev();
  }

  // calculate initial energy
  calculate_energy();

  // Metropolis cooling to a low temperature
  // while confining to a sphere of radius initial_size
//  cerr << "Initial energy: " << E << "\n";
  float T_setup;
  vector center;
  BOOL moved_particle;
  float current_size;
  for (T_setup=start_T_setup; T_setup>=end_T_setup; T_setup*=step_T_setup) {
    current_size=initial_size-(Req/2)*(start_T_setup-T_setup)/(start_T_setup-end_T_setup);
    for (step=0; step<N*steps_setup; step++) {
      monte_carlo_step(T_setup);
      center=vector(0);
      for (i=0; i<N; i++)
        center+=pos[i];
      center/=N;
      moved_particle=FALSE;
      for (i=0; i<N; i++) {
        pos[i]-=center;
        if (abs(pos[i])>current_size) {
           pos[i]*=current_size/abs(pos[i]);
          moved_particle=TRUE;
        }
      }
      if (moved_particle) calculate_energy();
    }
//    cerr << "Energy at temperature " << T_setup << ": " << E << "\n";
  }

  // start Metropolis stepping at correct temperature
  for (step=0; step<end_step; step++) {

    monte_carlo_step(T);

    // record energy and coordinates
    if (detailed && (step % nth_step == 0)) {
      detailed_data[step/nth_step][trial][0]=E;
      detailed_data[step/nth_step][trial][1]=step_size;
      if (step % (nth_step*mth_of_nth_steps) == 0) {
        output_coordinates(coordinate_data_file, T, trial, step);
        generate_cluster_size_histogram(TRUE);
        fprintf(cluster_data_file, "%i, %f, %i, %li, %e, ", N, T, trial, step, E/N);
        for (i=1; i<=N; i++)
          fprintf(cluster_data_file, "%i, ", histogram[i]);
        fprintf(cluster_data_file, "\n");
      }
    }

    if (step>=start_step) {
      sum_E+=E;
      sum_E2+=E*E;
    }
  }

  output_coordinates(coordinate_data_file, T, trial, end_step-1);

  E_bar=sum_E/(end_step-start_step);
  E2_bar=sum_E2/(end_step-start_step);
}


void open_files(char *mode) {
  data_file=fopen(data_file_name, mode);
  coordinate_data_file=fopen(coordinate_data_file_name, mode);
  cluster_data_file=fopen(cluster_data_file_name, mode);
```

```
}

void close_files(void) {
  fclose(data_file);
  fclose(coordinate_data_file);
  fclose(cluster_data_file);
}

void flush_files(void) {
  fflush(data_file);
  fflush(coordinate_data_file);
  fflush(cluster_data_file);
}


void main(void){
  int trial;
  float T;
  float E_bar, E2_bar, E_bar_sum, E2_bar_sum;
  float *saved_E2_bar; // to save <E^2> for columns at end of line
  int i, j, numtemps;
  char c;
  cout << "\nDetailed? (y/n): ";
  cin >> c;
  detailed = (c=='y' || c=='Y');
  if (detailed) {
    cout << "N: ";
    cin >> start_N;
    end_N=start_N;
    cout << "T: ";
    cin >> start_T;
    end_T=start_T;
    step_T=1; // arbitrary
    start_step=0; // arbitrary
    cout << "Steps: ";
    cin >> end_step;
    cout << "Record every nth step: ";
    cin >> nth_step;
    cout << "Output coordinates on every mth recorded step: ";
    cin >> mth_of_nth_steps;
    cout << "Number of independent trials: ";
    cin >> num_trials;
  } else {
    cout << "start N: ";
    cin >> start_N;
    cout << "end N: ";
    cin >> end_N;
    if (start_N>end_N) SWAP(start_N, end_N);
    cout << "start T: ";
    cin >> start_T;
    cout << "end T: ";
    cin >> end_T;
    if (end_T>start_T) SWAP(end_T, start_T);
    cout << "step T: ";
    cin >> step_T;
    cout << "Warm-up steps: ";
    cin >> start_step;
    cout << "Averaged steps: ";
    cin >> end_step;
    end_step+=start_step;
    cout << "Number of independent trials: ";
    cin >> num_trials;
  }
  cout << "Data file: ";
  cin >> data_file_name;
  cout << "Coordinate data file: ";
  cin >> coordinate_data_file_name;
  cout << "Cluster data file: ";
  cin >> cluster_data_file_name;
  open_files("wb");

  // allocate memory
  pos=(vector*)malloc(end_N*sizeof(vector));
  energy=(float**)malloc(end_N*sizeof(float*));
  for (i=0; i<end_N; i++)
    energy[i]=(float*)malloc(end_N*sizeof(float));
```

```
if (detailed) {
  detailed_data=(float***)malloc(((end_step-1)/nth_step+1)*sizeof(float**));
  for (i=0; i<((end_step-1)/nth_step+1); i++) {
    detailed_data[i]=(float**)malloc(num_trials*sizeof(float*));
    for (j=0; j<num_trials; j++)
      detailed_data[i][j]=(float*)malloc(2*sizeof(float));
  }
}
// allocates one more than necessary just in case of roundoff error
saved_E2_bar=(float*)malloc( int(2+round((start_T-end_T)/step_T)) * sizeof(float) );
histogram=(int*)malloc((N+1)*sizeof(int));

if (detailed) {
  fprintf(data_file, "\"Step\", \"Energy (%i columns)\", \"Step size (%i columns)\"\n",
                      num_trials, num_trials);
  fprintf(cluster_data_file, "N, T, trial, step, E, histogram\n");
} else {
  fprintf(data_file, "\"\", ");
  for (T=start_T; T>=end_T-step_T/2; T-=step_T)
    fprintf(data_file, "%f, ", T);

  fprintf(cluster_data_file, "N, T, histogram\n");
}

for (N=start_N; N<=end_N; N++) {

  if (!detailed) {
    fprintf(data_file, "\n");
    fprintf(data_file, "%i, ", N);
  }

  cerr << "\n" << N << " ";

  // cool cluster
  for (T=start_T, numtemps=0; T>=end_T-step_T/2; T-=step_T, numtemps++) {

    cerr << " *";
    E_bar_sum=0;
    E2_bar_sum=0;

    if (!detailed) {
      for (i=0; i<=N; i++)
        histogram[i]=0;
    }

    for (trial=0; trial<num_trials; trial++) {
      cerr << ".";
      monte_carlo(T, trial, E_bar, E2_bar);
      if (!detailed) {
         generate_cluster_size_histogram(FALSE);
        E_bar_sum+=E_bar;
        E2_bar_sum+=E2_bar;
      }
    }

    if (detailed) {
      for (i=0; i<=(end_step-1)/nth_step; i++) {
        fprintf(data_file, "%li, ", i*nth_step);
        for (trial=0; trial<num_trials; trial++)
          fprintf(data_file, "%e, ", detailed_data[i][trial][0]/N);
        for (trial=0; trial<num_trials; trial++)
          fprintf(data_file, "%f, ", detailed_data[i][trial][1]);
        fprintf(data_file, "\n");
      }
    } else {
      fprintf(data_file, "%e, ", E_bar_sum/num_trials/N);
      saved_E2_bar[numtemps]=E2_bar_sum/num_trials/N;
      fprintf(cluster_data_file, "%i, %f, ", N, T);
      for (i=1; i<=N; i++)
        fprintf(cluster_data_file, "%.1f, ", float(histogram[i])/num_trials);
      fprintf(cluster_data_file, "\n");
    }

    // save that precious data before being logged out!
    flush_files();
    }

    if (!detailed) {
      for (i=0; i<numtemps; i++)
        fprintf(data_file, "%e, ", saved_E2_bar[i]);
    }

    // save that precious data before being logged out!
    flush_files();

  }

  // close files
  close_files();

  // free memory
  free(pos);
  for (i=0; i<end_N; i++)
    free(energy[i]);
  free(energy);
  if (detailed) {
    for (i=0; i<((end_step-1)/nth_step+1); i++) {
      for (j=0; j<num_trials; j++)
        free(detailed_data[i][j]);
      free(detailed_data[i]);
    }
    free(detailed_data);
  }
  free(saved_E2_bar);
  free(histogram);
}
```