Monte Carlo Simulation of Colloidal Dispersion

By

Rohit Nikam

4th year undergraduate, Department of Chemical Engineering

Submitted as a course project (Multiscale Simulations - CL 671)

Indian Institute of Technology Bombay

Mumbai, India

April 2014

Abstract

Monte Carlo simulation has been used in the canonical ensemble to study the interaction of uniformly charged colloidal particles with oppositely charged monovalent ions using primitive model of 1:1 electrolyte solution. Ewald summation method was used to calculate the electrostatic interactions. Data is obtained for the configurational energy. Three cases have been studied viz. 2000 ions with 20 colloidal particles, 1000 ions with 10 colloidal particles and 100 ions with 1 colloidal particle. Radial distribution function of counter-ions with respect to colloidal particle and excess chemical potential of the counter-ions was calculated for the system of 100 ions with 1 colloidal particle. Widom insertion method was used to calculate excess chemical potential. All codes are written in MATLAB.

Table of Contents

1. Introduction

Typically, charged colloidal dispersions include the charged nanoparticles, their counter-ions and ions of electrolyte, all immersed in a dielectric solvent medium. According to the length scales of the components of the system, the system comes under mesoscopic length scale. Properties of materials at this length scale are significantly different from those of materials at higher length scales. This is predominantly due to the enormous interfacial area available at mesoscale which leads to the increased interfacial energy. Hence the magnitude of surface forces such as van der Waals interactions become comparable to that of body forces like electrostatic, magnetic or gravitational forces and thus become substantial factor in the consequential dynamics of the system. The laws of quantum mechanics rule the dynamics of objects smaller than mesoscopic scale instead of statistical mechanics.

Systems like microemulsions of water, oil and charged surfactants, micelles by charged surfactants, etc. and applications like drug delivery and design, aerosols and paints, DNA complexation etc. deal with the dynamics of charged colloidal dispersions.

The system in this study consists of following components-

2.1 Colloidal Particles

Uniform sized nanoparticles, each of 10 nm radius, having twice a negative charge are suspended in the dielectric medium.

2.2 Counter-ions

Colloidal particles have their counter-ions each having a single positive charge and 0.1 nm (1 A) radius.

2.3 Cations

Singly charged cations of 1:1 salt having uniform size of 0.1 nm (1 A) radius.

2.4 Anions

Singly charged anions of 1:1 salt having uniform size of 0.1 nm (1 A) radius.

2.5 Solvent

Solvent has been described on the coarse grained level as the continuum dielectric medium with no density gradients. Water is used as a solvent having a dielectric constant of 80. Charges on species are assigned so as to keep the total system electrically neutral.

Following three cases are considered:

Case 1: 1000 cations, 1000 anions, 20 nanoparticles and their 40 counter-ions

Case 2: 500 cations, 500 anions, 10 nanoparticles and their 20 counter-ions

Case 3: 50 cations, 50 anions, 1 nanoparticle and its 2 counter-ions

Counter-ions of colloidal particles have been assumed to identical to the cations of the salt. So these are basically three component systems.

3 Methodology

The hydrodynamic interactions between colloidal particles have been neglected. The solvation forces due to the background medium have been neglected. Solvent has been described on the coarse grained level as the continuum dielectric medium with no density gradients.

The system consists of two types of interactions among species

- van der Waals Interaction
- Electrostatic Interaction

3.1 van der Waals Interaction

This interaction is assumed to be pairwise additive and active among colloidal particles only and that between ions is approximated to be zero. Lennard-Jones potential model is employed to model this interaction.

Input parameters which need to be given are LJ parameters (σ, ε) and the cut-off distance or r-cut for this interaction.

3.2 Electrostatic Interaction

This is a long range interaction compared to van der Waals interaction. It is active both for ions and for colloidal particles. Both colloidal particles and ions are assumed to be hard spheres and from McMillan-Meyer solution theory, primitive model of electrolytes has been used which captures the electrostatic interaction between them. Role of the solvent medium in the model is only through its relative permittivity.

for a pair ij,

$$U_{ij} = \begin{cases} \infty & r_{ij} < R_i + R_j \\ \dfrac{z_i z_j e^2}{4\pi\varepsilon_0\varepsilon_r}\dfrac{1}{r_{ij}} & r_{ij} \geq R_i + R_j \end{cases}$$

where z_i is the charge on particle i, R_i is the radius of particle i, e is the electronic charge, ε_0 is the dielectric permittivity of vacuum, ε_r is the dielectric constant of the medium. r_{ij} is the center to center distance between the particles.

The Monte Carlo scheme in NVT ensemble is performed using the standard Metropolis algorithm to simulate the system. All components are in the cubic box of length 170 nm for cases 1 and 2 and 63 nm for case 3. Periodic boundary conditions are applied in all three directions. The electrostatic interaction is calculated using Ewald summation technique which gives rapidly convergent value of electrostatic energy regardless of any condition. Using the number of ions more than 2000 becomes more computationally expensive since Ewald sum calculates the electrostatic energy for the entire system for each move.

Since the movement of the colloidal particle is more likely to be rejected as compared to that of an ion due to possibility of overlap with other ions, the movement of ions is preferred. This was done by setting a small range from the range [0,1) of random number for the displacement of the colloidal particle and remaining longer range for the movement of any ion. The movement of each species gets recorded in one separate vector R, which is used in calculating the radial distribution function for each configuration.

Four separate subroutines are created for energy calculation:

- Real space part (short range part) of Ewald summation
- Reciprocal space part (long range part) of Ewald summation
- van der Waals interaction energy for whole system
- van der Waals interaction energy for a single species

Initially, van der Waals interaction energy was included in the real space sum of Ewald summation. The disadvantage of this form of the subroutine is that it calculates the van der Waals interaction for all colloidal particle system even after the movement of ion, which is actually the redundant calculation because this energy was updated last time when the colloidal particle was moved. So a separate function for van der Waals interaction energy for whole system was needed and that is invoked only when there is the displacement of a colloidal particle.

Again, in case of the displacement of a colloidal particle we don't have to calculate van der Waals interaction energy for all colloidal particles but just the total van der Waals interaction energy of the displaced colloidal particle. Hence the function for the interaction energy for single entity was created.

The maximum displacement of ion is set larger than that of a colloidal particle, again due to the reasons of the possible overlap with the adjacent configuration which is more likely for the colloidal particle. After every 50 moves, the acceptance ratio is calculated and if it is greater than 0.5, then the maximum displacement is increased by 5%. This is done for both the species. This will help in reaching the equilibrium configuration faster.

The factor by which the maximum displacement is increased should be chosen carefully. Using a very low value will result in very low velocity of reaching equilibrium, however the equilibrium state is guaranteed. Using a high value may help in reaching the equilibrium state faster, however the probability of the acceptance is very low, as a consequence, great amount of calculation done for the energy goes in vain.

Inputs needed for the calculation of the real space part of the Ewald sum

- Spacial configuration of the system
- Charges on species and no. of those species

- Length of box
- Width of the Gaussian charge distribution
- Radius of colloidal particle and ion
- Cut-off distance for the energy calculation

3.3 Some aspects regarding the simulation

3.3.1 Creating the initial configuration

Considerable amount of time was invested in creating the initial configuration of the system. Following were the small problems faced-

- Problem in executing the commands on the software PACKMOL in Ubuntu
- Problem in creating the desired configuration on the software VMD (Visual Molecular Dynamics)
- The resultant very low concentration of ion when packed like FCC in the cubic cell

A code was generated promising to keep the ions away from the colloidal particles by at least the distance equal to the addition radii of colloidal particle (10 nm) and that of the ion (0.1 nm).
Initially all 20 colloidal particles were assigned their position randomly from the FCC structure (which was having the unit cell distance equal to two times the radius of colloidal particle) and after that the code assigning the positions for ions without overlap was executed.

3.3.2 Width of the Gaussian charge distribution

The width of Gaussian charge distribution has to be mentioned which decides the degree to which the charge is smeared or the variance of the distribution. This width is thought to be equal to the addition of the radius of the particle and the Debye length for that species.

Since we have two species (ions and nanoparticles) with different concentration and radii, the width of Gaussian charge distribution for ions is different than that for nanoparticles. To avoid complications, same width for both species is used in simulations. This width is given as

$$\kappa = \frac{5.714}{L} \quad ; \quad \sigma = \frac{1}{\kappa\sqrt{2}}$$ (Woodcock and Singer (1971), Allen and Tildesley)

σ is the width of Gaussian charge distribution.

3.3.3 Cut-off distance for the energy calculation

In total there are three energies which we are calculating: Real space Ewald sum, reciprocal space Ewald sum and van der Waals interaction energy. Hence we have to mention three types of barriers for truncation of energy calculation, one for each of them. Since real space Ewald sum and van der Waals interaction energy are calculated in a real space, the cut-off distances were assigned for both of them. Cut-off distance for van der Waals interaction energy was taken to be r = 3σ because till this distance, less than 1% of the maximum attractive potential energy remains while that for real space Ewald sum was obtained as

$$erfc\left(\frac{r}{\sigma\sqrt{2}}\right) \geq tolerance$$

which can also be written as $$exp\left(\frac{-r^2}{2\sigma^2}\right) \geq tolerance$$

tolerance was set to 0.001 and the value of r comes out to be : **r = 0.46L**

'r' is the cut-off distance for the real space Ewald sum.

In case of reciprocal space Ewald sum, the cut-off is in terms of the periodicity (n) of the parent simulation cell. Larger the periodicity, more accurate is the value of the long range part of the electrostatic energy.

Beyond n = 4, the magnitude of the wave vector becomes of the order of 10^{-9}, also to reduce the computational complexity, the value of n is restricted to 4.

Inputs needed for calculation of the reciprocal space part of the Ewald sum are same as those for the real space, but in addition, an array of wave vector is needed. The wave vector is related to the periodicity of the simulation cell (n) by

$$k = \frac{2\pi n}{L}$$

3.3.4 Sampling range

The system samples were taken when the potential energy became steady. Actual number of Monte Carlo moves needed for equilibrium condition is 50,000 to 80,000. Unfortunately such a large number of moves could not be executed due to low computational speed of MATLAB. For the system with case 3, the energy was sort of steady from 12,000 moves onwards till maximum number of moves (15,000). So 12,000 to 15,000 was decided as the sampling range. For the system with case 2, the steady form of energy was in very small range. However for system with case 1 the equilibrium condition could not be achieved and hence, sampling could not be done for the calculation of radial distribution function and the excess chemical potential of counter-ions.

3.3.5 Calculation of excess chemical potential

Widom insertion technique was used to calculate the excess chemical potential of counter-ions (cations) for case 3.

One extra counter-ion was given a random position in the simulation cell during each main loop execution in the sampling regime and resultant change in total potential energy of the system (ΔU) was calculated. This is related to excess chemical potential of the counter-ions as:

$$\boldsymbol{\mu}_{\mathbf{ex}} = -\mathrm{kTln}\left\langle \int e^{-\beta \Delta U} ds_{n+1} \right\rangle$$

where <x> = ensemble average of x , $\beta = 1/k_B T$ and k_B = Boltzmann constant

The value of exp[-β(ΔU)] was calculated during each main loop execution in the sampling regime and was averaged over the number of loop executions.

4 Results and Discussion

The programming for the simulation was done in MATLAB. The major issue in using MATLAB is its extremely slow computation speed. This is due to the extra background processes which cannot be terminated preferentially. The Ewald sum process takes most of the time. In total, it takes approximately

12 seconds to calculate the total energy including real space Ewald sum, reciprocal space Ewald sum and total van der Waals interaction energy for the system with case 1. Following are the exact figures for case 1:

	Function	Execution period (sec)
1	Real space Ewald sum	7.036092
2	Reciprocal space Ewald sum	5.172588
3	Total van der Waals interaction energy	0.000138

The execution period for each function was obtained by the built-in function tic/tok. Total time taken for 5000 Monte Carlo moves for the system with case 1 took 16.7 hours.

Considerable time was invested in improving the robustness of the simulation code. Taking the advantage of the specialty in MATLAB software, vectorized expressions were preferred instead of 'for' loops wherever possible.

Following are the results for a system with **case 3**:

Potential energy vs. number of Monte Carlo moves

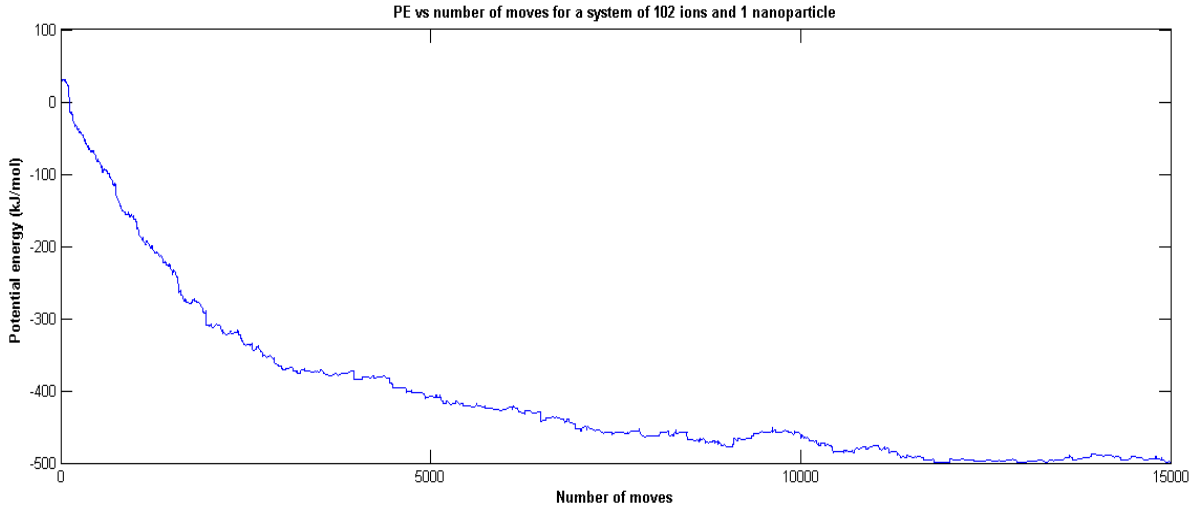

Fig. 1

Energy converges approximately to **-500 kJ/mol.**

Radial distribution function of counter-ions with reference to the colloidal particle

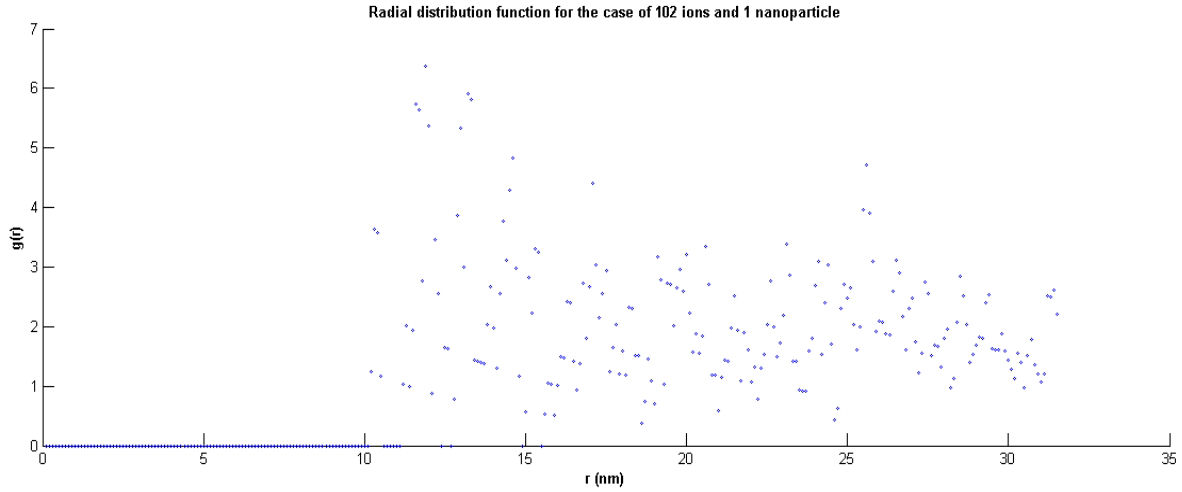

Fig. 2

Slight pattern in the distribution of counter-ions is seen which resembles to the exponential curve. This becomes clear in the following picture which is the rdf for the last configuration (n = 15,000)

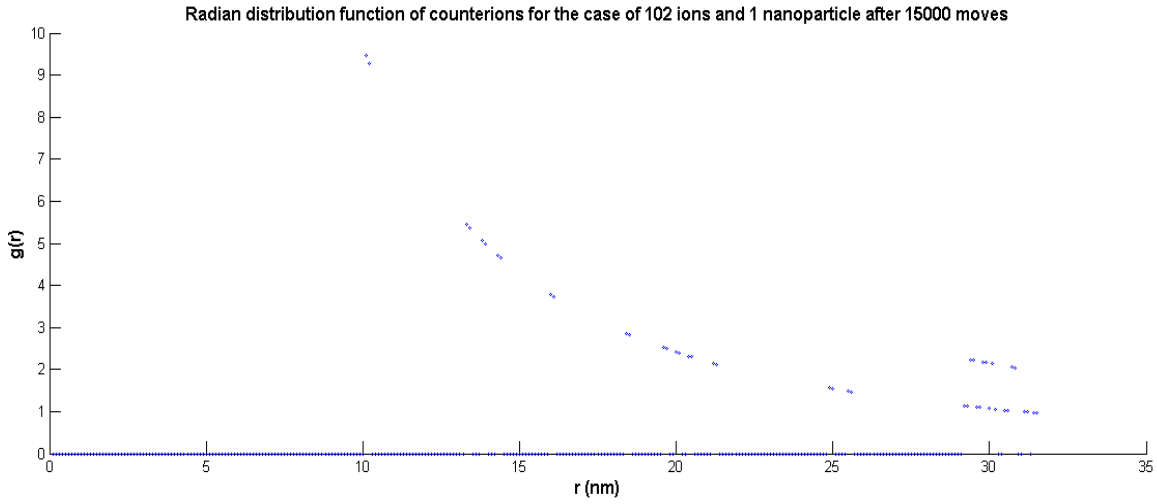

Fig. 3

It can also be seen that there is no ion within the range of 0 to 10 nm, which is the radius of the nanoparticle.

The excess chemical potential of counter-ions was calculated to be -219.1284 kJ/mol.

Although the relevant data could not be sampled for the system with **case 1,** following is the energy profile obtained:

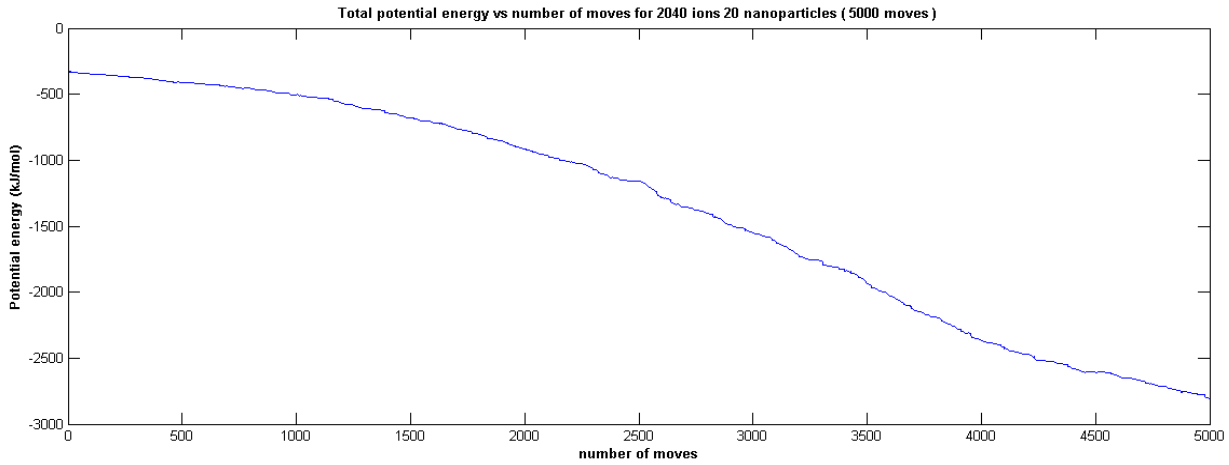

Fig. 4

The potential energy after 5000 moves was reported to be **-2803.9 kJ/mol**

Following are the results for a system with **Case 2**:

Potential energy vs. number of Monte Carlo moves

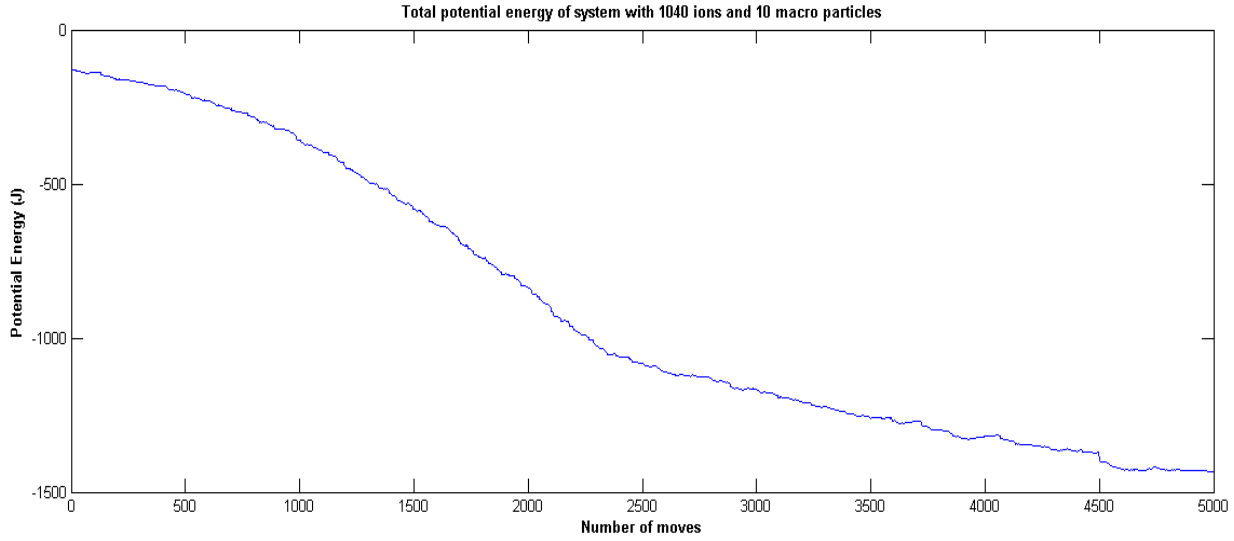

Fig. 5

Energy converges approximately to **-1435 kJ/mol.**

Radial distribution function of counter-ions with reference to the colloidal particle

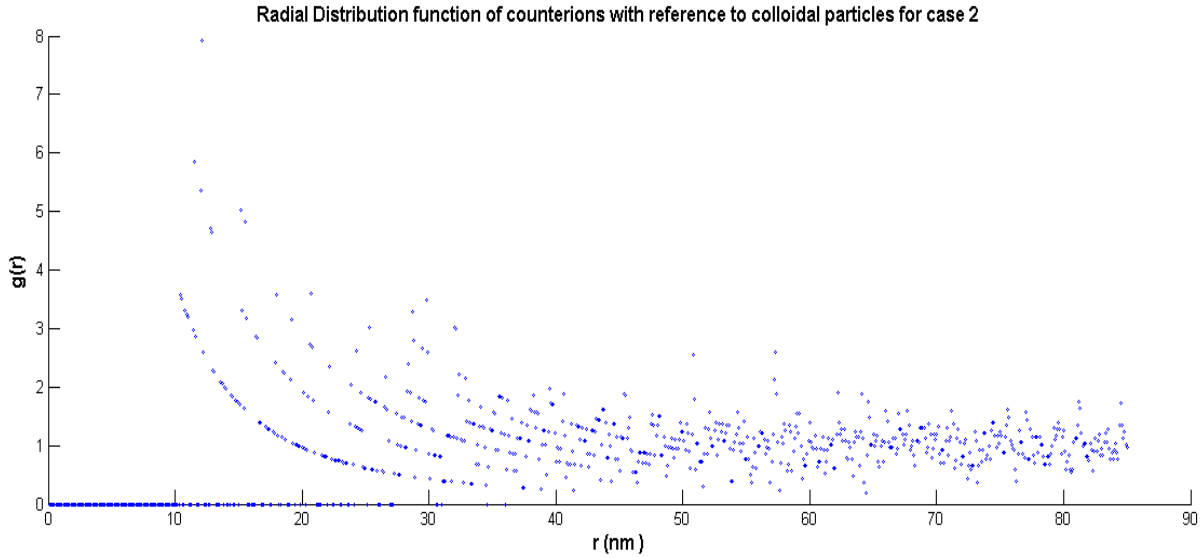

Fig. 6

5 Summary and Conclusion

The Monte Carlo simulations were performed in NVT ensemble to analyze the interaction between the colloidal particles and their counter-ions. The radial distribution function of counter-ions with reference to nanoparticles was evaluated for cases 2 and 3. Plots of radial distribution function were created by averaging out radial distributions functions of all system configurations in the sampling range. The reason for existence of multiple patterns in the plot of radial distribution instead of one single curve could not be found. Excess chemical potential was calculated for case 3. The sampling could not be done for case 1 since the system could not reach equilibrium state due to insufficient number of Monte Carlo moves. Small amount of ions was used in order to decrease the computational load. The value of the excess potential in case 3 may contain some error since the system energy may become steady at some other value on doing more number of Monte Carlo moves. The Monte Carlo simulation technique proves to be an effective and adaptable way to analyze the equilibrium in colloidal systems.

6 Challenges and Future work

Since the concentration of ions used in this work is very low, Debye Hückel theory needs to be verified. This work will be extended in future to include the variation in the dielectric constant of the medium, including polyelectrolyte solution and hydrodynamic interactions between nanoparticles using Brownian dynamic simulations. Hydration of charged species also needs to be studied using different water models. The code will be translated in FORTRAN to avoid using MATLAB and decrease the execution time so that larger concentration of ions can be used in the system which will lead to the cluster formation between nanoparticles due to significant decrease in the Debye length and thus low overlap of two Debye layers.

Acknowledgements

I sincerely thank Dr. Ateeque Malani (IIT Bombay) for helpful advice and to clear my doubts regarding the work and Prof. Peter Schall (University of Amsterdam) for the code creating a set of FCC lattice points to create initial configuration of colloidal particles.
(https://www.science.uva.nl/research/wzi/scm/MatlabRoutines.html)

7 References

1) Simulation of Charged Colloids in Solution (Per Linse, 2005)
2) Ewald Summation for Coulomb Interactions in a Periodic Supercell (Hark Lee, Wei Cai , 2009)
3) Monte Carlo Simulation Of Colloidal Systems (Eric Dickinson, Stephen Euston, 1992)
4) Physical meaning of the Ewald sum method (T. R. S. Prasanna)
5) Computer simulation of liquids (Allen, Tildesley, 1989)
6) Understanding molecular simulation (Frenkel, Smit)
7) Monte Carlo Study of the Thermodynamics of Electrolyte Solutions (Card, Valleau, 1970)
8) Statistical Mechanics of Fluid Mixtures (John G. Kirkwood, 1935)
9) Thermodynamic and Structural Properties of Liquid Ionic Salts Obtained by Monte Carlo Computation (Woodcock, Singer, 1970)

8 Appendix

8.1 Function for the calculation of real space part of Ewald sum

```
%%******************** Real space sum for whole system ********************%%

% Calculates the real part of the Ewald sum
% q  :            charge vector containing charge on each species
% eps :           dielectric constant
% rx, ry, rz  :   vectors containing positions of species
% L  :            length of side of cubic central box
% n  :            number of species
% sigma_gauss  :  width of Gaussian
% sigma :         sigma for macro - macro van der Waal interactions
% epsilon :       epsilon for macro - macro van der Waal interactions


function [ pot_real ] = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg,
n_pos, n_macro,   kappa_gauss, rcut2_ewald, R_macro, R_micro  )


u   =  0.0 ;


for i = 1 : n_neg + n_pos + n_macro - 1

    rxi = rx(i) ;
    ryi = ry(i) ;
    rzi = rz(i) ;

    for j = i + 1 : n_neg + n_pos + n_macro

        rxij  =  rxi   -  rx(j) ;
        ryij  =  ryi   -  ry(j) ;
        rzij  =  rzi   -  rz(j) ;

        rxij  =  rxij  -  round( rxij / L ) * L ;
        ryij  =  ryij  -  round( ryij / L ) * L ;
        rzij  =  rzij  -  round( rzij / L ) * L ;




        rij2  =  rxij^2  +  ryij^2  +  rzij^2 ;


        if  rij2 <= rcut2_ewald

            rij   =  sqrt( rij2 ) ;
            krij  =  kappa_gauss * rij ;
```

```
            if i <= n_neg

                if j <= n_neg

                    if rij < 2 * R_micro
                        pot_real = inf ;
                        return
                    end

                    u  =  u  +   q(1) * q(1) * erfc( krij ) / rij ;    %%
interaction between two negative ions  ( electrostatic )

                elseif j <= n_neg + n_pos

                    if rij < 2 * R_micro
                        pot_real = inf ;
                        return
                    end

                    u  =  u  +   q(2) * q(1) * erfc( krij ) / rij ;    %%
interaction between negative and positive ion  ( electrostatic )

                else

                    if rij < ( R_macro + R_micro )
                        pot_real = inf ;
                        return
                    end

                    u  =  u  +   q(3) * q(1) * erfc( krij ) / rij ;  %%
interaction between colloidal particle and negative ion  ( electrostatic )

                end


            elseif i <= n_neg +  n_pos

                if j <= n_neg +  n_pos

                    if rij < 2 * R_micro
                        pot_real = inf ;
                        return
                    end


                    u  =  u  +   q(2) * q(2) * erfc( krij ) / rij ;    %%
interaction between two positive ions  ( electrostatic )


                else
                    if rij < ( R_macro + R_micro )
                        pot_real = inf ;
```

```
                    return
                end

                u  =  u  +   q(3) * q(2) * erfc( krij ) / rij ;  %%
interaction between colloidal particle and positive ion  ( electrostatic )

            end

        else
            if rij < 2 * R_macro
                pot_real = inf ;
                return
            end

            u  =  u  +   q(3) * q(3) * erfc( krij ) / rij ;    %%
interaction between two colloidal particles ( electrostatic + van der Waals )

        end

    end

end
pot_real  =  k * u  / eps ;

end
```

8.2 Function for creating the wave vector for the sum in Fourier space

```
% K       : the matrix containing all three components of each wave vector
% kvector : the vector containing the term exp(-ksqr)/ksqr for each wave
vector
% size_kvector = (2*nmax + 1) * (2*nmax + 1) * (2*nmax + 1)  - 1
% L       : length of box decided by number of secies and density
% kappa_gauss controls the width of Gaussian distribution


function [ kvector  K ] = setup_kvector( kappa_gauss, L )


nmax = 4 ;      %% max integer component of k-vector

b = 1.0 / 4.0 / kappa_gauss / kappa_gauss ;


count = 0 ;  %%  total number of k-vectors stored


for nx = -nmax : nmax
```

```
    kx = 2 * pi * nx / L ;

    for ny = -nmax : nmax

        ky = 2 * pi * ny / L ;

        for nz = -nmax : nmax

            kz = 2 * pi * nz / L ;

            n2  =  nx * nx  +  ny * ny  +  nz * nz ;

            if  n2 ~= 0

                count  = count + 1 ;

                K( count, : ) = [ kx  ky  kz ] ;

                k2 =  kx * kx  +  ky * ky  +  kz * kz ;

                kvector( count ) =  exp( -b * k2 ) / k2 ;

            end
        end
    end
end

K

kvector'

fprintf(' Number of wave vectors is %d.\n',count)

end
```

8.3 Function for the calculation of reciprocal space part of Ewald sum

```
   %%***************** Reciprocal space sum *******************%%

% Calculates the reciprocal part of the Ewald sum
% q  :            charge vector containing charge on each species
% eps :           dielectric constant of background medium
% rx, ry, rz  :   vectors containing positions of species
% L  :            length of side of cubic central box
% kappa_gauss  :  width of Gaussian
% k :             1 / 4 / pi / epsilon zero
% kvector :       parameter storing the term 2*pi*exp(-k_sqr/4kappa_sqr)/ksqr
```

```
% eik :           vector containing exp(ik.(rj)) for each species j
% K   :           all components of each wave vector kx ky kz
% kvector :       vector containing exp(-b*k^2)/k^2 for each wave vector


function [ pot_resip ] =  ewald_reciprocal_new( q, eps, k, rx, ry, rz, eik,
L, n_neg, n_pos, n, kappa_gauss, K, kvector )


pot_resip = 0.0 ;


%****** sum over all k-vectors ******%

    rows = size( K, 1 ) ;  % rows of matrix K

            for  i = 1 : rows  %% loop for each k

                sum = 0 ;

                for j = 1 : n

                    rx( j ) =  rx( j )  -  round( rx( j ) / L ) * L ;
                    ry( j ) =  ry( j )  -  round( ry( j ) / L ) * L ;
                    rz( j ) =  rz( j )  -  round( rz( j ) / L ) * L ;

                    eik( j ) =  exp( 1i * (  K( i, 1 ) * rx( j )  +  K( i, 2
) * ry( j )  +  K( i, 3 ) * rz( j ) )) ;

                    if j <= n_neg

                        add = q(1) * eik ( j ) ;

                    elseif j <= n_neg + n_pos

                        add = q(2) * eik ( j ) ;

                    else

                        add = q(3) * eik ( j ) ;

                    end

                    sum  =  sum  +  add ;
                end

                 pot_resip  =  pot_resip +  sum * conj( sum ) * kvector( i );

            end


            pot_resip = 2 * pi * k * pot_resip / L / L / L / eps ;  %
```

```
%********** Self part of the reciprocal space sum ********** %%

pot_self = 0 ;

for temp = 1 : n

    if temp <= n_neg

        add = q(1) * q(1) ;

    elseif temp <= n_neg + n_pos

        add = q(2) * q(2) ;

    else

        add = q(3) * q(3) ;

    end

    pot_self  =  pot_self  +  add ;

end

pot_self  =  (pi^-0.5) * k * kappa_gauss / eps * pot_self   ;


%********** Total reciprocal space energy ************ %

pot_resip =  pot_resip -  pot_self ;


end
```

8.4 Function for the calculation of total van der Waals interaction energy for colloidal particles

```
% Calculates the total van der Waals interaction energy using LJ pair
potential
% n  :            total number of species
% eps :           maximum attractive interaction energy between two
nanoparticles
% sig :           second LJ parameter
% rx, ry, rz  :   vectors containing positions of species
% L  :            length of side of cubic central box
% rcut2    :      square of cut-off distance
```

```
function [pot_total]  =  vanderwaals_total( rx, ry, rz, eps, sig, rcut2, L,
n)

pot_total = 0.0;


for i = 1 : n

    rxi = rx( i ) ;
    ryi = ry( i ) ;
    rzi = rz( i ) ;


    for j = i + 1 : n

        rxij = rxi - rx( j ) ;
        ryij = ryi - ry( j ) ;
        rzij = rzi - rz( j ) ;

        rxij = rxij - round( rxij / L ) * L ;
        ryij = ryij - round( ryij / L ) * L ;
        rzij = rzij - round( rzij / L ) * L ;

        rij2 = rxij^2 + ryij^2 + rzij^2 ;

        if rij2 <= rcut2

            sr = ( sig^2 / rij2 )^3 ;

           pot_total =  pot_total  +  sr * ( sr - 1 ) ;

        end

    end
end

 pot_total = 4 * pot_total * eps ;

end
```

8.5 Function for the calculation of total van der Waals interaction energy of a single nanoparticle

```
% Calculates the total van der Waals interaction energy of a single
nanoparticle using  LJ pair potential
% p  :            index of a particle
% n  :            total number of species
% eps :           maximum attractive interaction energy between two
nanoparticles
% sig :           second LJ parameter
% rx, ry, rz  :   vectors containing positions of species
% L  :            length of side of cubic central box
% rcut2    :      square of cut-off distance


function [pot_single] = vanderwaals_single( p, rx, ry, rz, eps, sig, rcut2,
L, n )

pot_single = 0.0;

    rxi = rx( p ) ;
    ryi = ry( p ) ;
    rzi = rz( p ) ;


    for j = 1 : p - 1

            rxij = rxi - rx( j ) ;
            ryij = ryi - ry( j ) ;
            rzij = rzi - rz( j ) ;

            rxij = rxij - round( rxij / L ) * L ;
            ryij = ryij - round( ryij / L ) * L ;
            rzij = rzij - round( rzij / L ) * L ;

            rij2 = rxij^2 + ryij^2 + rzij^2 ;

            if rij2 <= rcut2

                sr = ( sig^2 / rij2 )^3 ;

                pot_single = pot_single +  sr * ( sr - 1 ) ;

            end
    end

    for j = p+1:n

            rxij= rxi-rx(j);
            ryij= ryi-ry(j);
            rzij= rzi-rz(j);

            rxij = rxij - round( rxij / L ) * L ;
            ryij = ryij - round( ryij / L ) * L ;
```

```
            rzij = rzij - round( rzij / L ) * L ;

            rij2 = rxij^2 + ryij^2 + rzij^2 ;

            if rij2 <= rcut2

                sr = ( sig^2 / rij2 )^3 ;

                pot_single = pot_single +  sr * ( sr - 1 ) ;

            end
    end

    pot_single = 4 * pot_single * eps ;

end
```

8.6 Main Execution File for systems with cases 1 and 2 (sample is shown for case 1)

```
% Monte Carlo simulation of charged colloidal dispersion

%%%%%%***************** MAIN EXECUTION FILE ******************%%%%%

% The following code calculates the total interaction energy of the system
% consisting of colloidal particles and ions of 1:1 ionic salt

% System description

% Cubic box of length 170 nm


% Species               Dimension (nm)              Number              Charge

% Colloidal particle        20                        20                  -2
% Counter-ion               0.1                       40                  +1
% Positive ion              0.1                      1000                 +1
% Negative ion              0.1                      1000                 -1
% Solvent (water)            -                      continuum             0

% All length units in nanometers
% All energy units in kJ/mol

% R_macro          radius of each colloidal particle
% R_micro          radius of each ion ( same for cation and anion )
% n_macro          number of colloidal particles
% n_neg            number of negative ions
% n_pos            number of positive ions
```

```
% eps              dielectric constant of background medium ( water )
% delr_micro       displacement for ion
% delr_macro       displacement for colloidal particle
% epsilon,sigma    van der Waals interaction parameters
% rcut_lj          cutoff distance for van der Waals interaction
% L                length of side of cubic simulation box
% q                charge on each ion and colloidal particle
% n_loop           total Monte Carlo moves
% kappa_gauss      width of Gaussian distributions
% K                all components of each wave vector : kx, ky, kz
% eik              vector containing exp(ik.(rj)) for each species j
% kvector          vector containing exp(-b*k^2)/k^2 for each wave vector
% T                temperature
% R                vector recording the changes in the position after each
move
% sigma_ewald      width of Gaussian
% k                1 / 4 / pi / epsilon zero


clc
clear all

load r_init_1000ions.txt  % contains initial configuration of system

n_neg = 1000 ;
n_pos = 1040 ;
n_macro = 20 ;
n = n_neg + n_pos + n_macro ;

load K.mat
load kvector.txt
eik = zeros ( n , 1 ) ;

R_macro = 10.0 ;
R_micro = 0.1 ;
L   =  170.0 ;


kappa_gauss = 5.714 / L ;
sigma_ewald = 1 / sqrt(2) / kappa_gauss ;
eps = 80 ;
k = 1382.5 ;

q = [ -1  +1 -2 ] ;
rcut_ewald  =  0.46 * L ;
rcut2_ewald = ( rcut_ewald )^2 ;

rx = r_init_1000ions( : , 1 ) ;
ry = r_init_1000ions( : , 2 ) ;
rz = r_init_1000ions( : , 3 ) ;


epsilon = 1 ;  %%  kJ/mol
sigma = 0.5 * ( 0.3 + 2 * R_macro ) ; % sigma is the van der Waals radius and
equals half the internuclear distance between non-bonded particles
```

```
rcut_lj  =  3 * sigma ;
rcut2_lj = ( rcut_lj )^2 ;

T  =  108 ;
beta  =  1.0 / ( 8.314472 * 0.001 * T ) ;

delr_micro = 1.5 ;
delr_macro = 0.5 ;

n_loop = 5000 ;
R = zeros( n_loop , 4 ) ;


realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos, n_macro,
kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;

u_sys =  realpart + ewald_reciprocal_new( q, eps, k, rx, ry, rz, eik, L,
n_neg, n_pos, n, kappa_gauss, K, kvector ) ;


upot  =  zeros( n_loop , 1 ) ;


naccept_micro = 0 ;  nint_micro = 50;
naccept_macro = 0 ;  nint_macro = 50;


for nmc = 1 : n_loop

    % *********** preferential movement ************* %

    g = rand () ;

    if g >= 0.2

        p  =   min( int32( rand() * (n_neg + n_pos) ) + 1  , n_neg + n_pos )
;  % an ion is chosen

        % ********** storing the old configuration ********** %
        xold  =  rx( p ) ;
        yold  =  ry( p ) ;
        zold  =  rz( p ) ;



        % ************* Giving a displacement *************** %

        rx( p )  =  rx( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;
        ry( p )  =  ry( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;
        rz( p )  =  rz( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;
```

```
        realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos,
n_macro,  kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;

        if realpart == inf
            rx( p )  =  xold ;
            ry( p )  =  yold ;
            rz( p )  =  zold ;

            R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;


else

            upnew_micro =  realpart + ewald_reciprocal_new( q, eps, k, rx,
ry, rz, eik, L, n_neg, n_pos, n, kappa_gauss, K, kvector ) ;


            delu_micro  =  upnew_micro  -  u_sys ;

            % ****************** Metropolis ******************* %

            if  delu_micro <= 0

                u_sys  =  upnew_micro ;

                naccept_micro  =  naccept_micro  +  1 ;

                R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move

            else

                if  exp( -beta * delu_micro )  >  rand()

                    u_sys  =  upnew_micro ;

                    naccept_micro  =  naccept_micro  +  1 ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move

                else
                    rx( p )  =  xold ;
                    ry( p )  =  yold ;
                    rz( p )  =  zold ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;
                end
            end
        end


        upot( nmc ) =  u_sys + vanderwaals_total(rx(n - n_macro + 1 : n ),
ry(n - n_macro + 1 : n ),rz(n - n_macro + 1 : n ), epsilon, sigma, rcut2_lj,
L, n_macro ) ;
```

```
        % ******** Re-adjusting the maximum displacement ******** %

        if mod( nmc , nint_micro ) == 0

            if naccept_micro / nint_micro  >=  0.5

                delr_micro  =  min( 1.05 * delr_micro , 0.5 * L ) ;

            else

                delr_micro  =  0.95 * delr_micro ;

            end

            naccept  =  0 ;

        end


    else
        % if colloidal particle is chosen, then

        p  =  min( int32( rand() * n_macro ) + 1 + n - n_macro , n ) ;


        pot_colloid_old = vanderwaals_single( p - n + n_macro, rx(n - n_macro
+ 1 : n ), ry(n - n_macro + 1 : n ),rz(n - n_macro + 1 : n ), epsilon, sigma,
rcut2_lj, L, n_macro);

        % ********** storing the old configuration ********** %
        xold  =  rx( p ) ;
        yold  =  ry( p ) ;
        zold  =  rz( p ) ;


        % ************* Giving a displacement*************** %

        rx( p )  =  rx( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;
        ry( p )  =  ry( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;
        rz( p )  =  rz( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;

        pot_colloid_new = vanderwaals_single( p - n + n_macro, rx(n - n_macro
+ 1 : n ), ry(n - n_macro + 1 : n ),rz(n - n_macro + 1 : n ), epsilon, sigma,
rcut2_lj, L, n_macro);


        realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos,
n_macro, kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;

        if realpart == inf
```

```
            rx( p )  =  xold ;
            ry( p )  =  yold ;
            rz( p )  =  zold ;

            R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;

        else

            upnew_macro =  realpart + ewald_reciprocal_new( q, eps, k, rx,
ry, rz, eik, L, n_neg, n_pos, n, kappa_gauss, K, kvector ) ;


            delu_macro  =  upnew_macro  -  u_sys  +  pot_colloid_new  -
pot_colloid_old;



            % ****************** Metropolis ******************* %

            if  delu_macro <= 0

                u_sys  =  upnew_macro ;

                naccept_macro  =  naccept_macro  +  1 ;

                R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move

            else

                if  exp( -beta * delu_macro )  >  rand()

                    u_sys  =  upnew_macro ;

                    naccept_macro  =  naccept_macro  +  1 ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording move

                else
                    rx( p )  =  xold ;
                    ry( p )  =  yold ;
                    rz( p )  =  zold ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;
                end
            end
        end


        upot( nmc ) =  u_sys + vanderwaals_total(rx(n - n_macro + 1 : n ),
ry(n - n_macro + 1 : n ), rz(n - n_macro + 1 : n ), epsilon, sigma, rcut2_lj,
L, n_macro ) ;
```

```
        % ******* Re-adjusting the maximum displacement ********* %

        if mod( nmc , nint_macro ) == 0
            if naccept_macro / nint_macro  >=  0.5
                delr_macro  =  min( 1.05 * delr_macro , 0.5 * L ) ;

            else
                delr_macro  =  0.95 * delr_macro ;
            end
            naccept_macro  =  0 ;
        end
    end
end
  plot ( upot )
```

Above code gives following outputs:

- Array of potential energy (Useful for Widom insertion)
- Vector 'R' consisting the sequence changes in position of particles (useful for RDF)
- Final configuration of the system

8.7 Function for radial distribution function of counter-ions with reference to nanoparticles

```
%************ Radial Distribution Function ***************%

% n_neg             number of negative ions
% n_pos             number of positive ions
% r                 probing distance vector
% dr                step size of r
% rho               number density of system
% n                 total no. of species
```

```
function [ gr ] = rdffunction( rx, ry, rz, L, r, dr, n_neg, n_pos, n, rho )

gr = zeros( numel(r), 1) ;


for k = 1 : numel( r )

    rk = r( k ) ;

    NGRk = 0 ;

    for i = n_neg + n_pos + 1 : n

        rxi = rx( i ) ;
        ryi = ry( i ) ;
        rzi = rz( i ) ;

        ngri = 0 ;

        for j = n_neg + 1 : n_neg + n_pos

            rxij = rxi - rx( j ) ;
            ryij = ryi - ry( j ) ;
            rzij = rzi - rz( j ) ;


            rxij = rxij - round( rxij / L ) * L ;
            ryij = ryij - round( ryij / L ) * L ;
            rzij = rzij - round( rzij / L ) * L ;

            rij2 = rxij^2  +  ryij^2  +  rzij^2 ;

            roundij2 = 0.001 * round( 1000 * rij2 ) ;

            rij = sqrt(roundij2) ;

            if rij < rk + dr && rij > rk - dr

                ngri  =  ngri + 1 ;

            end

        end

        NGRk = NGRk + ngri ;

    end

    NGRk = 0.25 * NGRk / n / pi / dr / rho ;

    gr(k) = NGRk / (rk^2) ;
```

```
end
end
```

8.8 Code for averaging out the RDF for each configuration after sampling period

```
clear all
clc

load r_init_500ions.txt % initial configuration of system
load R_500ions_5000moves.txt

r_init = r_init_500ions ;
R = R_500ions_5000moves ;

rx = r_init( : , 1 ) ;
ry = r_init( : , 2 ) ;
rz = r_init( : , 3 ) ;

n = numel( rx ) ;

L  = 170 ;
dr = 0.1 ;
r  = 0 : dr : 0.5 * L ;
n_neg = 500 ;
n_pos = 520 ;

n_loop = 5000 ;
n_sampling = 4900 ; % number from which the sampling starts
nint   = 100 ;

sum = zeros( numel(r), 1 )  ;
count = 0 ;

for i = 1 : n_sampling

    p = R( i , 1 ) ;

    rx( p ) = R( i, 2 ) ;
    ry( p ) = R( i, 3 ) ;
    rz( p ) = R( i, 4 ) ;

end


for j = n_sampling + 1 : n_loop


    p = R( j, 1 ) ;
```

```
    rx( p ) = R( j, 2 ) ;
    ry( p ) = R( j, 3 ) ;
    rz( p ) = R( j, 4 ) ;

    if mod( j , nint ) == 0

        gr = rdffunction( rx, ry, rz, L, r, dr, n_neg, n_pos, n, rho ) ;

        sum = sum + gr ;

        count = count + 1 ;
    end
end

avg_gr = sum / count ;

scatter( r, avg_gr, 2.1 )
```

8.9 Main execution code for the simulation of a system of case 3 along with code for Widom insertion

The structure of the code remains the same as the code for cases 1 and 2. The difference is that the van der Waals interaction energy need not be calculated since there is only one colloidal particle.

```
% Monte Carlo simulation of charged colloidal dispersion

%%%%%%***************** MAIN EXECUTION FILE ******************%%%%%%


% The following code calculates the total interaction energy of the system
% consisting of a colloidal particle and ions of 1:1 ionic salt and the
% excess chemical potential of counter-ions
 % System description

% Cubic box of length 63 nm

% Species               Dimension (nm)              Number              Charge

% Colloidal particle        20                        1                   -2
% Counterion                0.1                       2                   +1
% Positive ion              0.1                      50                   +1
% Negative ion              0.1                      50                   -1
% Solvent (water)            -                     continuum               0

% All length units in nanometers
% All energy units in kJ/mol
```

```
% R_macro          radius of each colloidal particle
% R_micro          radius of each ion ( same for cation and anion )
% n_macro          number of colloidal particles
% n_neg            number of negative ions
% n_pos            number of positive ions
% eps              dielectric constant of background medium ( water )
% delr_micro       displacement for ion
% delr_macro       displacement for colloidal particle
% epsilon,sigma    van der Waals interaction parameters
% rcut_lj          cutoff distance for van der Waals interaction
% L                length of side of cubic simulation box
% q                charge on each ion and colloidal particle
% n_loop           total Monte Carlo moves
% kappa_gauss      width of Gaussian distributions
% K                all components of each wave vector : kx, ky, kz
% eik              vector containing exp(ik.(rj)) for each species j
% kvector          vector containing exp(-b*k^2)/k^2 for each wave vector
% T                temperature
% R                vector recording the changes in the position after each
move
% sigma_ewald      width of Gaussian
% r_widom          array containing coordinates of system plus additional
inserted cation
% muexsum          sum of terms exp(-beta*delu_widom)
% muexcess         excess chemical potential of positively charged ions
% k                1 / 4 / pi / epsilon zero

clc
clear all

load r_init_50ions_7000moves.txt  % contains initial configuration of system

n_neg = 50 ;
n_pos = 52 ;
n_macro = 1 ;
n = n_neg + n_pos + n_macro ;

load K.mat
load kvector.txt
eik = zeros ( n , 1 ) ;

R_macro = 10.0 ;
R_micro = 0.1 ;
k = 1382.5 ;

L   =  63 ;
kappa_gauss = 5.714 / L ;
sigma_ewald = 1 / sqrt(2) / kappa_gauss ;
eps = 80 ;

q = [ -1  +1 -2 ] ;
rcut_ewald  = 0.46 * L ;
rcut2_ewald = ( rcut_ewald )^2 ;
```

```
rx = r_init_50ions_7000moves(:,1);
ry = r_init_50ions_7000moves(:,2);
rz = r_init_50ions_7000moves(:,3);

rxw = zeros( n+1 , 1 ) ;
ryw = rxw ;
rzw = rxw ;

rxw( 1 : n_neg + n_pos ) = rx( 1 : n_neg + n_pos ) ;
rxw( n + 1 ) = rx( n ) ;

ryw( 1 : n_neg + n_pos ) = ry( 1 : n_neg + n_pos ) ;
ryw( n + 1 ) = ry( n ) ;

rzw( 1 : n_neg + n_pos ) = rz( 1 : n_neg + n_pos ) ;
rzw( n + 1 ) = rz( n ) ;

muexsum = 0 ;

T  =  108 ;
beta  =  1.0 / ( 8.314472 * 0.001 * T ) ;

delr_micro = 1.5 ;
delr_macro = 0.5 ;

n_loop = 15000 ;
n_sampling = 10200 ;
R = zeros( n_loop , 4 ) ;


realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos, n_macro,
kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;


u_sys =  realpart + ewald_reciprocal_new( q, eps, k, rx, ry, rz, eik, L,
n_neg, n_pos, n, kappa_gauss, K, kvector ) ;

upot  =  zeros( n_loop , 1 ) ;

naccept_micro = 0 ;  nint_micro = 50;
naccept_macro = 0 ;  nint_macro = 50;

format short

for nmc = 1 : n_loop

    % ********** preferential movement ************ %

    g = rand () ;

    if g >= 0.2
```

```
        p  =   min( int32( rand() * (n_neg + n_pos) ) + 1  , n_neg + n_pos )
;  % an ion is chosen

        % ********** storing the old configuration ********** %
        xold  =  rx( p ) ;
        yold  =  ry( p ) ;
        zold  =  rz( p ) ;

        % ************* Giving a displacement *************** %

        rx( p )  =  rx( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;
        ry( p )  =  ry( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;
        rz( p )  =  rz( p )  +  delr_micro * ( 2.0 * rand() -  1.0 ) ;

        realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos,
n_macro, kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;


        if realpart == inf
            rx( p )  =  xold ;
            ry( p )  =  yold ;
            rz( p )  =  zold ;

            R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;

        else

            upnew_micro =  realpart + ewald_reciprocal_new( q, eps, k, rx,
ry, rz, eik, L, n_neg, n_pos, n, kappa_gauss, K, kvector ) ;


            delu_micro  =  upnew_micro  -  u_sys ;

            % ****************** Metropolis ******************* %

            if  delu_micro <= 0

                u_sys  =  upnew_micro ;

                naccept_micro  =  naccept_micro  +  1 ;

                R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move

            else

                if  exp( -beta * delu_micro )  >  rand()

                    u_sys  =  upnew_micro ;

                    naccept_micro  =  naccept_micro  +  1 ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move
```

```
                else
                    rx( p )  =  xold ;
                    ry( p )  =  yold ;
                    rz( p )  =  zold ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;
                end
            end
        end

            rxw(p) = rx(p);
            ryw(p) = ry(p);
            rzw(p) = rz(p);


% ******** Re-adjusting the maximum displacement *********** %

        if mod( nmc , nint_micro ) == 0

            if naccept_micro / nint_micro  >=  0.5

                delr_micro  =  min( 1.05 * delr_micro , 0.5 * L ) ;

            else

                delr_micro  =  0.95 * delr_micro ;

            end

            naccept  =  0 ;

        end

        upot( nmc )  =  u_sys ;

    else
        % if colloidal particle is chosen, then

        p  =  n ;

        % ********** storing the old configuration ********** %
        xold  =  rx( p ) ;
        yold  =  ry( p ) ;
        zold  =  rz( p ) ;

        % ************* Giving a displacement*************** %

        rx( p )  =  rx( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;
        ry( p )  =  ry( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;
        rz( p )  =  rz( p )  +  delr_macro * ( 2.0 * rand() -  1.0 ) ;
```

```
        realpart = ewald_real_new( q, eps, k, rx, ry, rz, L, n_neg, n_pos,
n_macro, kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;

        if realpart == inf
            rx( p )  =  xold ;
            ry( p )  =  yold ;
            rz( p )  =  zold ;

            R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;

        else

            upnew_macro =  realpart + ewald_reciprocal_new( q, eps, k, rx,
ry, rz, eik, L, n_neg, n_pos, n, kappa_gauss, K, kvector ) ;


            delu_macro  =  upnew_macro  -  u_sys  ;

            % ****************** Metropolis ******************* %

            if  delu_macro <= 0

                u_sys  =  upnew_macro ;

                naccept_macro  =  naccept_macro  +  1 ;

                R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording the move

            else

                if  exp( -beta * delu_macro )  >  rand()

                    u_sys  =  upnew_macro ;

                    naccept_macro  =  naccept_macro  +  1 ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ; % recording move

                else
                    rx( p )  =  xold ;
                    ry( p )  =  yold ;
                    rz( p )  =  zold ;

                    R(nmc,:) = [ p rx(p) ry(p) rz(p) ] ;
                end
            end
        end

        rxw(n+1) = rx(p);
        ryw(n+1) = ry(p);
        rzw(n+1) = rz(p);
```

```
    % ******** Re-adjusting the maximum displacement ************ %

        if mod( nmc , nint_macro ) == 0

            if naccept_macro / nint_macro  >=  0.5

                delr_macro  =  min( 1.05 * delr_micro , 0.5 * L ) ;

            else

                delr_macro  =  0.95 * delr_macro ;

            end

            naccept_macro  =  0 ;

        end

        upot( nmc )  =  u_sys ;


    end

    %%%%%%%%%%%%%%% WIDOM INSERTION ************%%%%%

    if nmc > n_sampling

        Rxw = 0.5 * L * ( 2 * rand() - 1 ) ;
        Ryw = 0.5 * L * ( 2 * rand() - 1 ) ;
        Rzw = 0.5 * L * ( 2 * rand() - 1 ) ;

        rxw( n ) = Rxw ;
        ryw( n ) = Ryw ;
        rzw( n ) = Rzw ;

        realpartw = ewald_real_new( q, eps, k, rxw, ryw, rzw, L, n_neg, n_pos
+ 1, n_macro, kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;

        while realpartw == inf

            Rxw = 0.5 * L * ( 2 * rand() - 1 ) ;
            Ryw = 0.5 * L * ( 2 * rand() - 1 ) ;
            Rzw = 0.5 * L * ( 2 * rand() - 1 ) ;

            rxw( n_neg + n_pos + 1 ) = Rxw ;
            ryw( n_neg + n_pos + 1 ) = Ryw ;
            rzw( n_neg + n_pos + 1 ) = Rzw ;


            realpartw = ewald_real_new( q, eps, k, rxw, ryw, rzw, L, n_neg,
n_pos + 1, n_macro, kappa_gauss, rcut2_ewald, R_macro, R_micro ) ;
        end
```

```
        u_sysw =  realpartw + ewald_reciprocal_new( q, eps, k, rxw, ryw, rzw,
eik, L, n_neg, n_pos + 1, n + 1, kappa_gauss, K, kvector ) ;

        deluw = u_sysw - upot( nmc  ) ;

        factor = exp( -beta * deluw ) ;

        muexsum = muexsum + factor ;

    end

end

plot ( upot )

ncount = n_loop - n_sampling - 1 ;

muexcess = - muexsum / ncount / beta ;
```

8.10 Code creating the initial configuration of ions

Input file for the initial configuration of colloidal particles(r_colloids.txt)is created using the code creating the fcc lattice points

Sample code is given for the system with case 1.

```
clear all
close all
clc

L = 170 ;

load r_colloid.txt

r = r_colloid ;

n = 2040 ; % total number of ions

h = zeros(n,3);


for j = 1 : n

    x = 0.5*L*( 2 * rand() -1 ) ;
```

```
    y = 0.5*L*( 2 * rand() -1 ) ;
    z = 0.5*L*( 2 * rand() -1 ) ;

    h(j,:) = [x y z] ;

end

a = size(r,1) ;

for i = 1 : a

    for k = 1 : n

        rxij  =  r(i,1) - h(k,1) ;
        ryij  =  r(i,2) - h(k,2) ;
        rzij  =  r(i,3) - h(k,3) ;

        rxij  =  rxij  -  round( rxij / L ) * L ;
        ryij  =  ryij  -  round( ryij / L ) * L ;
        rzij  =  rzij  -  round( rzij / L ) * L ;

        rij2  =  rxij^2  +  ryij^2  +  rzij^2 ;

        rij = sqrt(rij2);

        if rij <= 10 + 0.1

            h(k,:) = [0 0 0] ;

        end
    end
end

k = 1 ;

for i = 1 : n

    if h(i,2) ~= 0
        r_ions(k,:) = h(i,:);
        k = k + 1 ;
    end
end

r_ions
```

9 Input parameters

9.1 Case 1

Number of anions = 1000

Charge on each anion = -1e

Number of cations = 1040

Charge on each cation = +1e

Number of nanoparticles = 20

Radius of ions = 0.1 nm (both for anions and cations)

Radius of nanoparticles = 10 nm

Length of box = 170 nm

Value of kappa for Gaussian charge distribution = 5.714 / L = 0.0336

Width of Gaussian charge distribution = 1 / sqrt(2) / kappa = 21.0379 nm

Dielectric constant of medium = 80

Cut-off distance = 0.46 * L = 78.2 nm

LJ parameter ε = 1 kJ/mol

LJ parameter σ = 0.5 * (0.3 + 2 * Radius of nanoparticle) = 10.15 nm

Temperature = 108 K

Maximum initial displacement of ion = 1.5 nm

Maximum initial displacement of nanoparticle = 0.5 nm

Maximum periodicity for wave vector (n) = 4

9.2 Case 2

All parameters are kept same except

Number of anions = 500

Number of cations = 520

Number of nanoparticles = 10

9.3 Case 3

All parameters are kept same except

Length of box = 63 nm

Number of anions = 50

Number of cations = 52

Number of nanoparticles = 1

Value of kappa for Gaussian charge distribution = 5.714 / L = 0.0907

Width of Gaussian charge distribution = 1 / sqrt(2) / kappa = 7.7962 nm

Cut-off distance = 0.46 * L = 28.98 nm