Grand Canonical Monte Carlo
Statistical Mechanics Project

CHEN 610

Applied Statistical Thermodynamics

Dr. Wu

Heather Barkley

April 26, 2006

Introduction

Simulations are important to the chemical engineering industry. They provide a way to obtain data less expensively, less dangerously, and less error due to human fault than in the laboratory. Therefore, much research has been done in the area of molecular simulation to study properties of many particle systems. The limitation of simulation is how accurately the mathematical model can describe the behavior of the system of interest and the information being gathered from the simulation. The cost involved in developing a detailed model to describe the system is computer time and memory. In research, accurate models are to be balanced with computer time and memory to best describe the systems of interest.

The statistical mechanics project looks at modeling simple monatomic atoms using Monte Carlo to study phase transitions. Phase transitions are used throughout chemical engineering by designing separations, manufacturing, and processing applications. Many methods have been developed to study phase transitions such as Gibbs ensemble, Gibbs-Duhem integration, and Histogram reweighting Grand Canonical Monte Carlo. The system and type of properties needed drives the method of choice for phase transitions. Also, choosing a force field to describe the intermolecular interactions is critical in obtaining usable data. This paper looks at these key factors using Monte Carlo simulations in the grand canonical ensemble.

Background

1. MonteCarlo, Grand Canonical Ensemble, and Phase Transitions

Molecular simulations are parted by two methods, time dependent Molecular dynamics which follow Newton's equations of motion and depend on momentum and positions of particles, and the stochastic approach called Monte Carlo which follow the random generation of configurations which depend only on the positions of particles. Monte Carlo is a better approach than Molecular Dynamics to determine phase transition properties because the integration process in calculating the equations of motion will overshoot a transition point because of the discontinuity in energy during a first order phase transition. Usually, two or more simulations at different initial starting points are utilized but consumes much more time without much increase in accuracy. The Monte Carlo code is found in Appendix1.

The grand canonical ensemble is utilized which simulates the system at constant chemical potential, μ, temperature, β (=1/k_BT), and volume, V. This ensemble works well when large amounts of phase equilibria data are needed because of the histogram reweighting technique to gather multiple information from a single simulation. Also, this method works better at the critical point than does the Gibbs Ensemble technique because the Gibbs Ensemble uses an interface. Near the critical point, the free energy for creating an interface becomes small because at the critical point there is no interface. When the free energy is small in the Gibbs Ensemble, the drive for inserting particles is gone. The disadvantages for using the grand canonical histogram reweighting is for large chain molecules because it is difficult to input a large chain if the shape and space are unavailable. It takes awhile for a configuration to open to a certain shape. Also, the gradual insertion technique cannot be used for the histogram reweighting because the energies obtained for half a molecule inserted is incorrect for describing the behavior of the system. Another disadvantage is for large system size, more simulations are needed for overlap in the histogram.

2. Histogram Reweighting

Statistical mechanics is used to obtain macroscopic properties like pressure from ensemble averages. The partition function for the grand canonical ensemble is shown in equation 1. The probability of a configuration to occur with an energy in this ensemble is shown in equation 2. Equation 3 shows the relationship from the partition function to the macroscopic property, pressure. Histogram reweighting technique (Non-Boltzmann Sampling) is a method to evaluate averages at a state of interest from a trajectory or a single simulation run. Using the factorability of the Boltzmann distribution and a reference energy distribution, equation 4, demonstrates the weighting of the histogram to obtain another probability distribution at another state point.

$$\Xi = \sum_{i,N} \exp[-\beta E_i + \beta\mu N] \quad (1)$$

$$P(A_i) = \frac{1}{\Xi}\left(A_i \exp[-\beta E_i + \beta\mu N\right) \quad (2)$$

$$P = \frac{1}{V\beta} Ln[\Xi] \quad (3)$$

$$P(N,E,V,\beta_1,\mu_1) = \exp[-(\beta_1(E-N\mu_1) - \beta_o(E-N\mu_o))]P(N,E,V,\beta_o,\mu_o) \quad (4)$$

3. Forcefields

Describing interactions between molecules is fundamental for determining the behavior of a system. Two forcefields looked at are the well-known and well used Lennard Jones two parameter potential shown in equation 5 and the four parameter potential developed by J.M. Hanley and M. Klein in J. Chem. Phys. 50, 4765 (1969) called the m-6-8 potential shown in equation 6. The parameters are used to describe magnitude of attraction and repulsion. For Lennard Jones, sigma (σ) is the distance of zero energy, and epsilon (ε) is the value of lowest energy. Distances less than the well depth is where repulsion is felt, distance greater than the well depth is where attraction is felt till a certain point to where there is no long range interactions. Manipulating the Lennard Jones form tries to change the repulsion form to a "softer" wall to explain attraction within solids and liquids. The fundamentals to obtaining the mathematical form rely on quantum mechanical perturbation theory. Figure 1 in appendix 2 shows the graphical form for the two potentials. The well depth is shifted closer to the center of the atom for the m-6-8 potential and has a lower well depth are the noticeable differences. Physically, the atoms have less energy closer together and are more incline to stay closer together for the m-6-8. A closer look shows the m-6-8 having a steeper attractive wall than the Lennard Jones which means more energy is needed for an atom to move away from each other. The parameters are determined by fitting them to experimental values. Table 1 shown below gives the parameters for argon.

$$u^* = 4\left[\left(\frac{1}{r^*}\right)^{12} - \left(\frac{1}{r^*}\right)^{6}\right] \quad (5)$$

$$u^* = \left[\frac{1}{m-6}\right][6+2\gamma]\left(\frac{1}{r^*}\right)^{m} - \left[\frac{1}{m-6}\right]\left[m-\gamma(m-8)\right]\left(\frac{1}{r^*}\right)^{6} - \frac{\gamma}{\left(r^*\right)^{8}} \quad (6)$$

$$r^* = \frac{r}{\sigma}, u^* = \frac{u}{\varepsilon}$$

Table 1: Parameters for Argon

	Lennard-Jones	M-6-8
sigma, σ	3.405	3.292
epsilon, ε	120	153
m		11
gamma, γ		3

Methodology

Argon is used as our monatomic atom. The grand canonical ensemble is used at a constant cubic box length of 24.35 Angstroms, and temperature of 101 K. To look at a region close to the critical point, Nigel B. Wilding, "*Computer simulation of fluid phase transitions*", in Am. J. Phys. **69** (11), November 2001, value of -2.778 βμ is recommended. Values for βμ used are -2.90, -2.66, -2.42, and -2.18. First, Lennard Jones was used to run at these four state points for 200,000 Monte Carlo (MC) steps. A distribution of energy and number of atoms is collected. Probability distribution is analyzed to determine phase transition. Second, the energy barrier height is analyzed by looking at the partition function versus density for values of βμ equal to -2.90 and -2.18. Third, histogram reweighting is used with βμ at -2.90 to plot a phase diagram of pressure versus temperature. Last, a comparison of the potentials is shown by looking at pressure and probability density for βμ at -2.42.

The length of a Monte Carlo run is a concern, and the amount of data changes statistical output and in this case properties. Also, having randomness affects the output. Every run started with a configuration which was randomized by 1,000,000 MC steps at a constant number of particles (N = 256), energy (ε), and volume (box length = 24.35 Angstroms). Before attaining properties, the simulations ran for 100,000 MC steps at the specified state and ensemble before the 200,000 data points used for obtaining properties. The amount of data points is insufficient for this since the probability distributions are not smooth functions. If computational time was not an issue than the appropriate method to determine if the amount of data points were sufficient is to run another simulation at the same state but for longer and see if the properties changed.

Results and Discussion

1. Probability to find phase transition

Probability distributions are plotted in appendix 3, for values of $\beta\mu$ at -2.90, -2.66, -2.42, and -2.18. Interestingly, no phase transitions are shown. A phase transition would look like the figure 1 shown below. High probability for small densities show the gas region and high probability for large densities show the liquid region. For $\beta\mu$ of -2.90 and -2.18, a simulation run with 600 atoms as the initial configuration was tried but gave the same results. This was tried to see if the sampling was not running long enough to try inserting many particles and sampling the denser phase space. This however, was not a problem. The critical point would be observed if there were no double peak, but not one large one either, it would be a flat curve spanning the entire range of densities, which is not seen in these figures either. When the probabilities are at the same height, then the coexistence line is found.

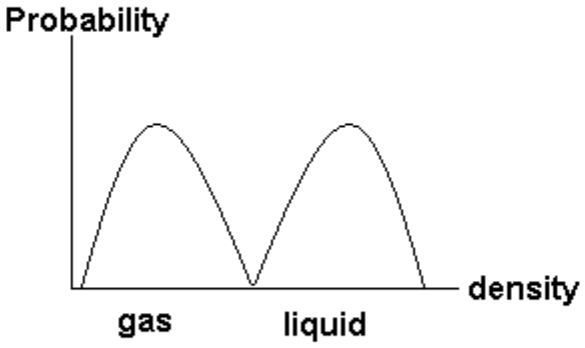


Figure 1: Probability density function for two phases at a state point.

2. Ξ versus ρ to find phase transition

Figure 2 shown below is used to describe the magnitude of energy barriers for different state points. A better description is shown in figure 3 where phase transitions are seen. For fewer atoms in the cube, gas phase, the energy is low because it is entropy motivated. The atoms want to be randomized and maximize the level of entropy. For many atoms in the cube, liquid phase, the atoms are more ordered which would lessen the level of entropy, however, the liquid phase is energy motivated. The way the potential is formed there is attraction which lowers the energy of the system. When atoms pull away from the distance of lowest energy, the energy increases and the atoms pull together creating a lower energy. Between these dips in energy is the energy barrier. Large energy barriers show a difficultly in transitioning from gas to liquid or vice versa. The lower energy well is the most likely phase the system will be in. The meta-stable phase is the higher energy well. When the wells are equal in magnitude, the coexistence line is found.

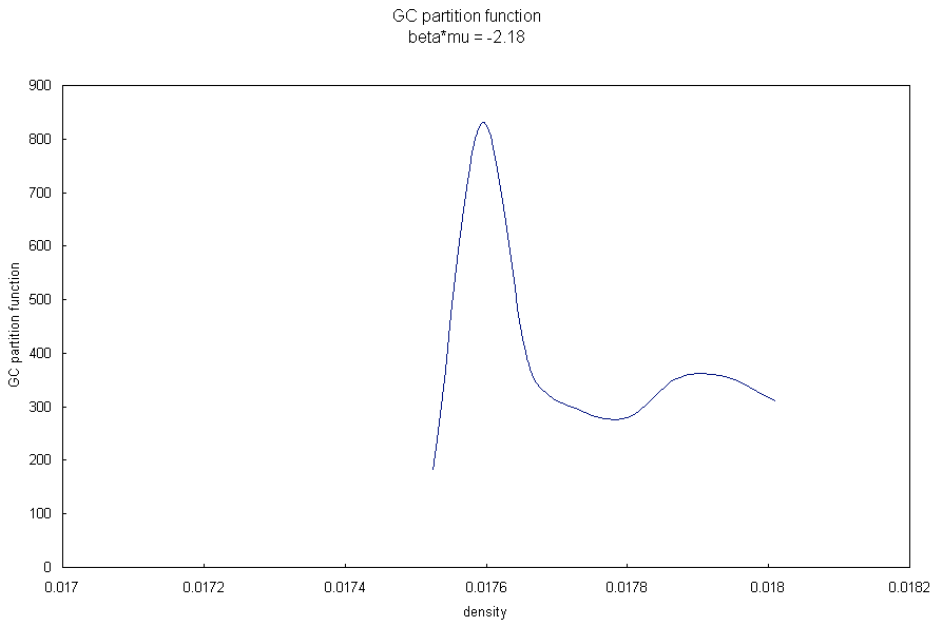


Figure 2: Shows calculations of the partition function for $\beta\mu = -2.18$

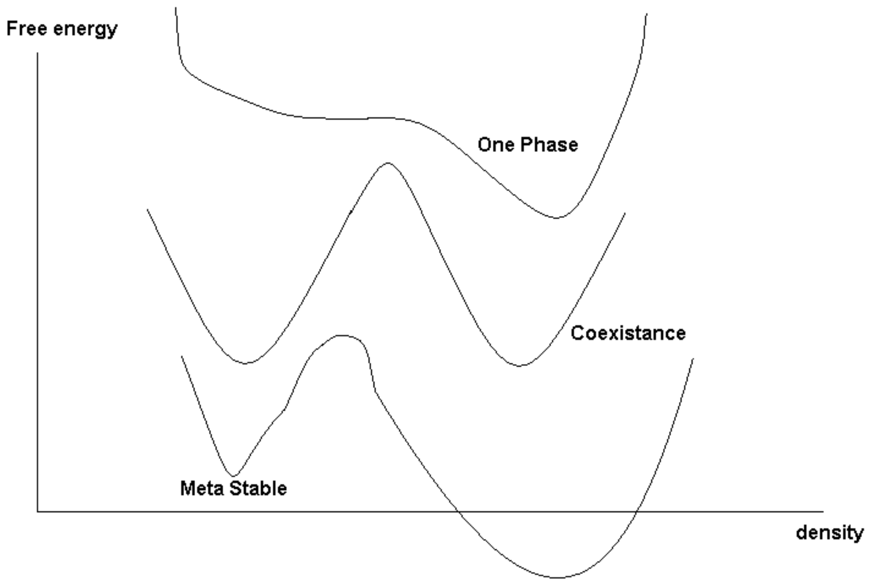


Figure 3: Better description of the energy barrier between phases

3. Histogram reweighting for $\beta\mu = -2.90$

Figure 4 shows the pressure and temperature for $\beta\mu = -2.90$. This was found through histogram reweighting. For simulations found near the coexistence line, this would be quite useful in developing a phase diagram. However, this simulation did not show a difference in the form of the probability density to show two different phases at this state so this method is not accurate. The ability to utilize one simulation run for more data points is quite useful in developing phase diagrams quickly with less computational memory. Also, if the computer simulations exhibit inaccurate data around the critical point, using data near the critical point and extrapolating between two sets of data could be beneficial and accurate for describing properties close to the critical point.

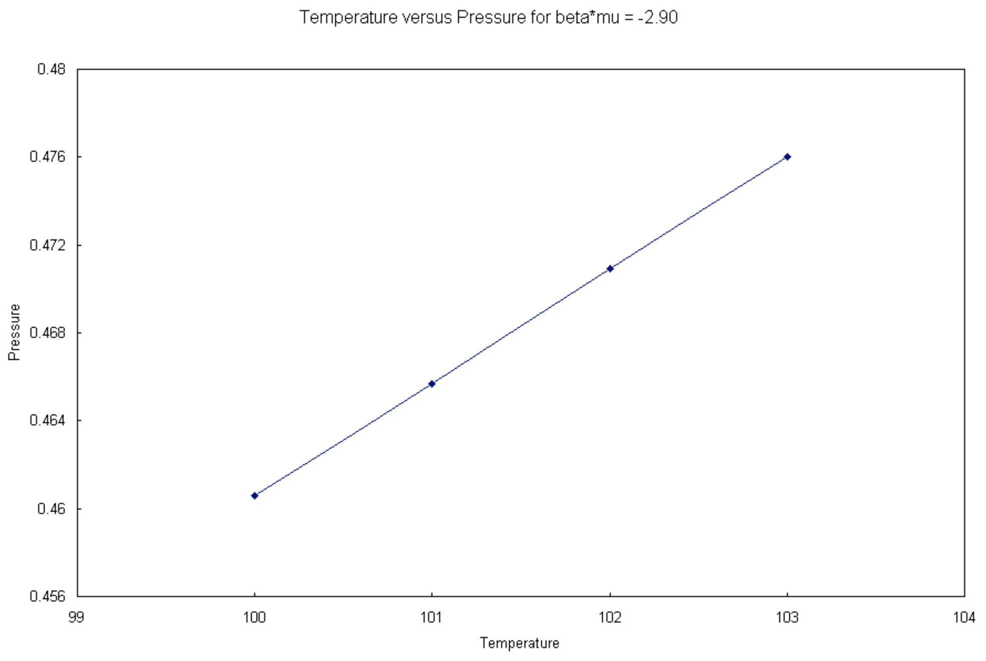


Figure 4: Shows the temperature versus Pressure attempt of a phase diagram using Histogram Reweighting for $\beta\mu = -2.90$

4. Comparing Force Fields

As expected, a higher number of atoms in the cube is more probable for the m-6-8 potential. Figure 5 shows during a simulation run, higher number of atoms are in the box for the m-6-8 as compared to the Lennard Jones. This is due to the lowest energy well depth to be closer to the center of the atom. Higher energy barriers between phases would be expected for the m-6-8 potential, therefore, for example higher temperatures are needed to boil liquid argon for m-6-8 potential than for the Lennard Jones. Since, there are more atoms in the cube for the m-6-8, it would be expected that the pressure be higher than in the Lennard Jones cube at the same temperature. This is true, for $\beta\mu = -2.42$, pressure for Lennard Jones is 0.706 while for m-6-8, P = 0.726. Noticeable differences for these potentials are shown, especially in choosing to design chemical reactors or separation tanks.

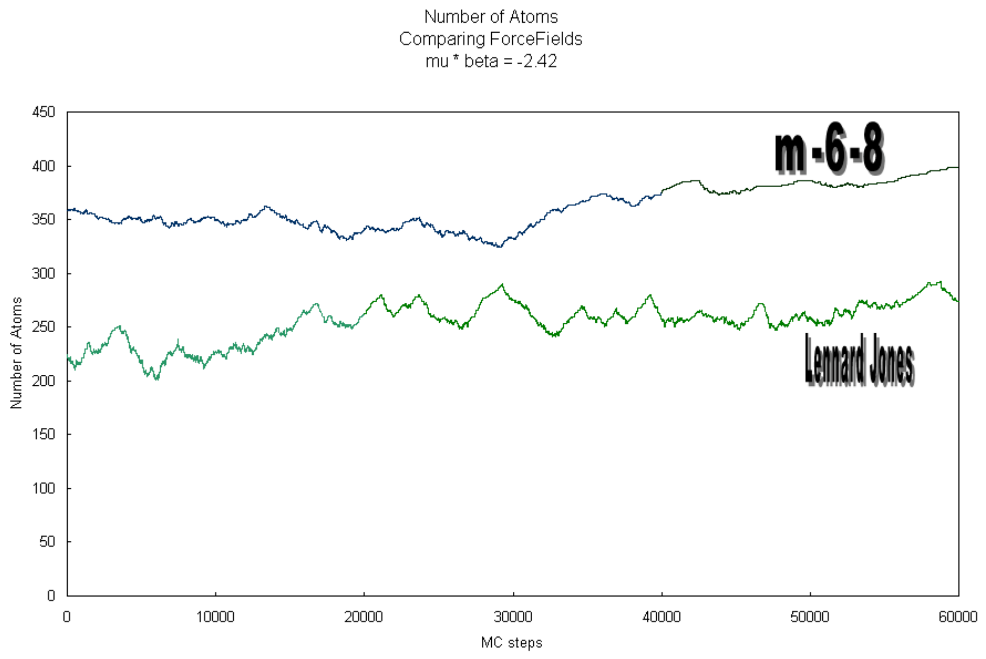


Figure 5: Shows the comparison of force fields at the same state point.

Conclusion

The need to develop quick and less computational memory to portray the behavior of real systems is great. This project shows one of many methods to describe phase behavior of a simple system. More complicated methods stems from these ideas and the fundamentals are drawn from the basics of statistical mechanics. Grand canonical ensemble is useful for phase transitions because the output of information such as the energy distribution which is used to collect data points at a different state. This method is called histogram reweighting and is used most often when collecting data for phase equilibria such as for phase diagrams. Choosing a mathematical model is critical for describing the intermolecular interactions which in turn describes the macroscopic behavior of the system. Lennard Jones is most often and widely used but others like the m-6-8 try changing the magnitude or strength of attraction and repulsion to obtain better results. Each parameter is unique to the system as well as each method and approach is unique to the information needed and gathered.

References

[1] Wilding, Nigel B. "*Computer simulation of fluid phase transitions*", Am. J. Phys. **69** (11) November 2001

[2] Panagiotopoulos, Athanassios Z. "*Topical Review, Monte Carlo methods for phase equilibria of fluids*", J. Phys.: Condens. Matter 1999 Vol **12**.

[3] Hanley, H.J.M. and Klein, M. "*On the Utility of the m-6-8 Potential Function*", Nat. Bur. Stand.(U.S.), Tech. Note 628, November 1972

[4] Hanley, H.J.M. and Klein, M. "*m-6-8 Potential Function*", J. Chem. Phys., Vol **53**, 1970

[5] Frenkel, D. and Smit, B. Understanding Molecular Simulation From Algorithms to Applications, Academic Press 2002

APPENDIX 1: Monte Carlo Grand Canonical Code in Fortran90/95

```
Program GrandCanonicalMC
Use DFport
Implicit DoublePrecision (a-h,o-z)
!
!Heather Barkley started March 24th
!This is Monte Carlo for a grand
! canconical ensemble. (constant chemical potential,
! volume, and temperature)
! Uses LJ 6-12 potential argon atoms, also m-6-8 potential is also used //commented out
!                    Number of atoms
integer, parameter :: MaxN = 1000
!                    Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)              !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)          !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                    Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405             !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                   !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
real(8), parameter :: temperature = 101. !K
real(8), parameter :: chemicalPotential = -2033.7 !Joules
!
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /Potential/ distance1, distance2, potential1, potential2, cutoff_dist, sqr_r1, sqr_r2
Common /num_of_atoms/ count_atoms, potential
Common /LJenergy/ potentialTemp
Common /DeleteAccept/ deleteParticle
!
integer :: i_step, atom_num, k
Character * 25, StatusFile
accept_t = 0
accept_i = 0
accept_d = 0
iSeed = -4.
Call ReadingInitial() !brining in number of atoms and configuration from file
Call Initialize() !finding initial energy
!
OPEN(UNIT = 10, FILE = 'GC_Data1.DAT', STATUS = 'replace', ACTION = 'write')
step = 200000.
        Do i_step = 1, step
                if (randnum(iSeed) < 0.5 ) then !translation
```

```
                        Call Move_Atom()
                        Call LennardJonesEnergy()
                        Call AcceptanceTranslation()
                else !insertion/deletion
                        if (randnum(iSeed) < 0.5 ) then !insert
                                Call Insert()
                                if (count_atoms < MaxN-1) then
                                        Call LennardJonesEnergy()
                                        Call AcceptanceAdd()
                                else
                                endif
                        else !delete
                                Call Deletion()
                                if (count_atoms > 0) then
                                        Call LennardJonesEnergy()
                                        Call AcceptanceDelete()
                                else
                                endif
                        endif
                endif
                !
                Do k = 1, count_atoms
                        X_temp(k) = rx(k)
                        Y_temp(k) = ry(k)
                        Z_temp(k) = rz(k)
                enddo
                !
                if (Mod(i_step, 10000) .eq. 0) then
                        print *, i_step, nint(count_atoms), 'a_T= ', nint(accept_t), ' a_I= ', nint(accept_i),
' a_D= ', nint(accept_d)
                endif
                if (Mod(i_step, 125000) .eq. 0) then
                        write (StatusFile, '(I25)') i_step
                        StatusFile = AdjustL(StatusFile)
                        StatusFile = 'StatFile' // Trim(StatusFile)// '.dat'
                        open(30, File = StatusFile)
                        do k = 1, count_atoms
                                write(30, *) rx, ' ', ry,  ' ', rz
                        enddo
                        close(30)
                endif
        170 Format (1X, I8, 1X, F5.0, 1X, F20.2)
        180 Format (1X, I8, 2X, F10.0, 2X, F10.0, 2X, F10.0)
        enddo
        do k = 1, count_atoms
                write(15, 190) rx(k), ry(k), rz(k)
        enddo
        190 Format (1X, F10.6, 2X, F10.6, 2X, F10.6)
End Program
!-----------------------------------------------------------------------------------------------
Subroutine Initialize()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will initialize the parameters of the box the atoms are located.
! and initialize the potential energy of the system.
!
```

```
!                    Number of atoms
integer, parameter :: MaxN = 1000
!                    Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)              !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)          !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                    Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405             !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                   !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7     !reduced density, (n*sigma^3/V)
!
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /SetUp/ lengthS, mass
Common /num_of_atoms/ count_atoms, potential
!
!          Variables
real(8) :: distance, sqr_r, cutoff_dist
integer(4) :: i, j
real(8) :: const_m, gamma
const_m = 11.0
gamma = 3.0
potential = 0.
cutoff_dist = 2.*sigma
          do i = 1, (count_atoms - 1)
                    do j = i+1, count_atoms
                              sqr_r = (rx(i) - rx(j))**2. + (ry(i)-ry(j))**2. + (rz(i)-rz(j))**2.
                              distance = (sqr_r)**(1./2.)
                                        if (distance <= cutoff_dist) then
                                                  !potential = 4.*((1./distance)**(12.)-(1./distance)**(6.)) +
potential
                                                  potential = (1./(const_m-
6.))*(6.+2.*gamma)*(1./distance)**(const_m)-(1./(const_m-6.))*(const_m-gamma*(const_m-
8.))*(1./distance)**(6.)-gamma*(1./distance)**(8.) + potential
                                        else
                                                  potential = potential
                                        endif
                    enddo
          enddo
          do i = 1, MaxN
                    X_temp(i) = rx(i)
                    Y_temp(i) = ry(i)
                    Z_temp(i) = rz(i)
          enddo
end
!-----------------------------------------------------------------------------------------------
Subroutine Move_Atom()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will randomly choose an atom in the box and generate a new coordinate
! randomly, apply periodic boundary condition and checks for overlap on other atoms
! Calls function randnum ( random number generator )
!
```

```
!                       Number of atoms
integer, parameter :: MaxN = 1000
!                       Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)                 !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)             !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                       Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405              !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                    !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
!
real(8) :: rxold, ryold, rzold, rxnew, rynew, rznew
real(8) :: atom_number
real(8) :: delta_length, old_dist, new_dist
integer :: i, j, k, flag
!
        flag = 1
        new_dist = 0.
        delta_length = 0.05
!Choose a random atom
        atom_number = NINT(count_atoms*randnum(iSeed))
                if (atom_number == 0) then
                        atom_number = 1
                endif
!save old coordinates of atom
        rxold = X_temp(atom_number)
        ryold = Y_temp(atom_number)
        rzold = Z_temp(atom_number)
!generate new coorinates for atom
        do
                rxnew = rxold + (randnum(iSeed)-0.5)*delta_length
                rynew = ryold + (randnum(iSeed)-0.5)*delta_length
                rznew = rzold + (randnum(iSeed)-0.5)*delta_length
!apply periodic boundary conditions
                do
                        if ( rxnew > lengthS ) then
                                rxnew = rxnew - lengthS
                        elseif (rxnew < 0d0) then
                                rxnew = rxnew + lengthS
                        elseif (rynew > lengthS ) then
                                rynew = rynew - lengthS
                        elseif (rynew < 0d0 ) then
                                rynew = rynew + lengthS
                        elseif (rznew > lengthS ) then
                                rznew = rznew - lengthS
                        elseif (rznew < 0d0 ) then
                                rznew = rznew + lengthS
                        else
                                exit
                        endif
                enddo !trying to get the new atom's coordinate into the box
        !check for overlap
                do i = 1, count_atoms
                        new_dist = ((X_temp(i)-rxnew)**2.+(Y_temp(i)-
rynew)**2.+(Z_temp(i)*rznew)**2.)**(1./2.)
                        if ( i .ne. atom_number) then
                                if  (new_dist == 0.)  then
                                        flag = 0
                                endif
                        endif
                enddo !iterating for overlap on all atoms expect once they become the same atom
                         !exit to get new atom instead of looping through 1 to N
                if ( flag .ne. 0. ) then
                        exit
                endif
        enddo !trying to change the new atom's coordinate
!collect coordinates in temporary coordinates to check valid move through statistics
        do i = 1, MaxN
                if ( i == atom_number ) then
                        X_temp(i) = rxnew
                        Y_temp(i) = rynew
                        Z_temp(i) = rznew
                else
                        X_temp(i) = X_temp(i)
                        Y_temp(i) = Y_temp(i)
                        Z_temp(i) = Z_temp(i)
                endif
        enddo
!
End
!----------------------------------------------------------------------------------------------
Subroutine ReadingInitial()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will initialize the parameters of the box the atoms are located.
! This routine will read in initial configuration from a file (config.dat).
!
!                       Number of atoms
integer, parameter :: MaxN = 1000
!                       Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)                  !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)              !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                       Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405               !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                     !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
!
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /SetUp/ lengthS, mass
Common /num_of_atoms/ count_atoms, potential
!
!           Variables
real(8) :: LoopConfigEnd, box_length, space
integer(4) :: i, status, j
!
!BOX LENGTH DOES NOT CHANGE !!!
LoopConfigEnd = (real(256)/4.)**(1./3.)
count = 0
box_length = ((real(256)*sigma**(3.))/(rhostar))**(1./3.) !angstroms
mass = real(256)*MW*(1./Avogadro)*(1./1000.) !kg
lengthS = box_length/sigma !used in MD simulation
count_atoms = 0
i = 0
j = 0
!
Do i = 1, MaxN
            rx(i) = 0.
            ry(i) = 0.
            rz(i) = 0.
enddo
!read in coordinates
OPEN (UNIT = 20, FILE = 'config.dat', STATUS= 'OLD', IOSTAT= status)
readloop: DO
            if ( status .ne. 0 ) exit
            j = j + 1
            count_atoms = count_atoms + 1
            READ (20, *, IOSTAT = status) rx(j), ry(j), rz(j)
            enddo readloop
CLOSE (UNIT = 20)
count_atoms = count_atoms - 1
END
!-----------------------------------------------------------------------------------------------
Subroutine LennardJonesEnergy()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will calculate the potential energy of the system.  The addition of the
! m-6-8 potential was added later and commented out, but name of function is still
! LennardJonesEnergy but will also calculate the m-6-8 potential (Hanley and Klein)
!
!                       Number of atoms
integer, parameter :: MaxN = 1000
!                       Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)                  !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)             !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                       Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405              !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                    !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
!
Common /SetUp/ lengthS, mass
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
Common /LJenergy/ potentialTemp
integer :: i, j
real(8) :: cutoff_dist, distance, sqr_r
real(8) :: const_m, gamma
const_m = 11.0
gamma = 3.0
potentialTemp = 0.
cutoff_dist = 2.*sigma
        do i = 1, (count_atoms - 1)
                do j = i+1, count_atoms
                        sqr_r = (X_temp(i) - X_temp(j))**2. + (Y_temp(i)-Y_temp(j))**2. +
(Z_temp(i)-Z_temp(j))**2.
                        distance = (sqr_r)**(1./2.)
                                if (distance <= cutoff_dist) then
                                        !potentialTemp = 4.*((1./distance)**(12.)-(1./distance)**(6.))
+ potentialTemp
                                        potentialTemp = (1./(const_m-
6.))*(6.+2.*gamma)*(1./distance)**(const_m)-(1./(const_m-6.))*(const_m-gamma*(const_m-
8.))*(1./distance)**(6.)-gamma*(1./distance)**(8.) + potentialTemp
                                else
                                        potentialTemp = potentialTemp
                                endif
        enddo
enddo
!
End
!------------------------------------------------------------------------------------------------
Subroutine AcceptanceTranslation()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will calculate if the translation move from Move_Atom is acceptable by
! comparing energy of the old system to the energy of the new system dependent on the
! configuration.  If the energy is less, the move is accepted and if the probability is large
! based on the Boltzmann factor than it is also accepted.
!
!               Number of atoms
integer, parameter :: MaxN = 1000
!               Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)           !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)       !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!               Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405           !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                 !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
real(8), parameter :: temperature = 101. !K
real(8), parameter :: chemicalPotential = -2033.7 !Joules
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
Common /LJenergy/ potentialTemp
!
integer(4) :: i,j,k
real(8) :: probability
if ((potentialTemp - potential) < 0. ) then !accepted
        potential = potentialTemp
        do k = 1, count_atoms
                rx(k) = X_temp(k)
                ry(k) = Y_temp(k)
                rz(k) = Z_temp(k)
        enddo
        accept_t = accept_t + 1
else
        probability = Exp( -1./(ConstantR*temperature)*(potentialTemp-potential)*epsilon*ConstantR)
        if (randnum(iSeed) < probability) then !accepted
                do i = 1, count_atoms
                        rx(i) = X_temp(i)
                        ry(i) = Y_temp(i)
                        rz(i) = Z_temp(i)
                enddo
                accept_t = accept_t + 1
                potential = potentialTemp
        else
                i = i
        endif
endif
end
!-----------------------------------------------------------------------------------------------
Subroutine Insert()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will add a single atom at a random position
!
!               Number of atoms
integer, parameter :: MaxN = 1000
!               Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)             !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)         !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!               Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405            !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                  !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7     !reduced density, (n*sigma^3/V)
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
!
```

```
!
X_temp(count_atoms + 1) = lengthS * randnum(iSeed)
Y_temp(count_atoms + 1) = lengthS * randnum(iSeed)
Z_temp(count_atoms + 1) = lengthS * randnum(iSeed)
count_atoms = count_atoms + 1
end
!-----------------------------------------------------------------------------------------------
Subroutine Deletion()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will randomly choose an atom in the box to delete
!
!                       Number of atoms
integer, parameter :: MaxN = 1000
!                       Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)                  !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)              !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                       Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405              !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                    !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7     !reduced density, (n*sigma^3/V)
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
Common /DeleteAccept/ deleteParticle
!
!
integer(4) :: i,j,k
deleteParticle = int(randnum(iseed)*count_atoms)
if (deleteParticle == 0) then
        deleteParticle = 1
endif
Do i = 1, count_atoms
        if ( i == deleteParticle) then
                Do j = i, (count_atoms-1)
                        X_temp(j) = X_temp(j+1)
                        Y_temp(j) = Y_temp(j+1)
                        Z_temp(j) = Z_temp(j+1)
                enddo
                Do k = count_atoms, MaxN
                        X_temp(k) = 0.
                        Y_temp(k) = 0.
                        Z_temp(k) = 0.
                enddo
        else
                X_temp(i) = X_temp(i)
                Y_temp(i) = Y_temp(i)
                Z_temp(i) = Z_temp(i)
        endif
```

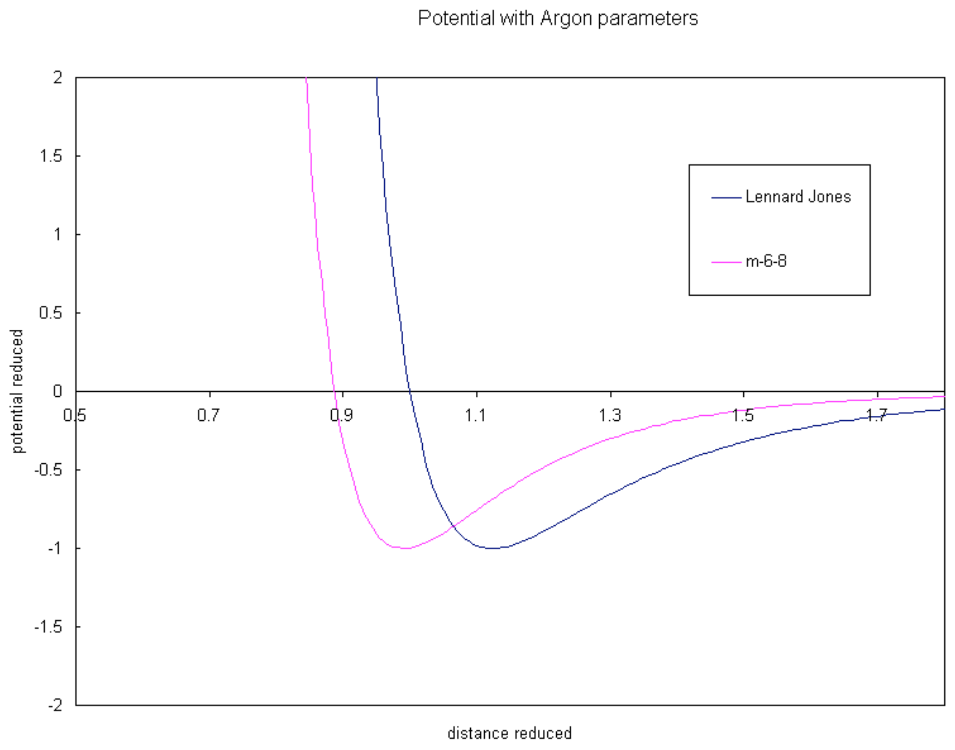
```
enddo
end
!-----------------------------------------------------------------------------------------------
!
Subroutine AcceptanceDelete()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will calculate if the deletion from Deletion() is acceptable by
! comparing energy of the old system to the energy of the new system dependent on the
! configuration.  If the energy is less, deletion is accepted and if the probability is large
! based on the Boltzmann factor than it is also accepted.
!
!                       Number of atoms
integer, parameter :: MaxN = 1000
!                       Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)                  !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)              !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!                       Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405               !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                     !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7      !reduced density, (n*sigma^3/V)
real(8), parameter :: temperature = 101. !K
real(8), parameter :: chemicalPotential = -2033.7 !Joules
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
Common /LJenergy/ potentialTemp
Common /DeleteAccept/ deleteParticle
!
!
real(8) :: zeta, probability, beta, volume
integer(4) :: i,j,k
beta = 1./(ConstantR*temperature)
zeta = exp(beta*chemicalPotential)
volume = lengthS*lengthS*lengthS
        count_atoms = count_atoms - 1
        do k = 1, MaxN
                rx(k) = X_temp(k)
                ry(k) = Y_temp(k)
                rz(k) = Z_temp(k)
        enddo
        accept_d = accept_d + 1
        potential = potentialTemp
else
        probability = ((count_atoms)/(zeta*volume))*exp(-beta*(potentialTemp-
potential)*epsilon*ConstantR)
        if ( probability > randnum(iSeed) ) then
                count_atoms = count_atoms - 1
                do k = 1, MaxN
                        rx(k) = X_temp(k)
                        ry(k) = Y_temp(k)
                        rz(k) = Z_temp(k)
                enddo
                accept_d = accept_d + 1
                potential = potentialTemp
        else
                count_atoms = count_atoms
        endif
        count_atoms = count_atoms
endif
end
!-----------------------------------------------------------------------------------------------
!
Subroutine AcceptanceAdd()
Use DFport
Implicit DoublePrecision (a-h,o-z)
! This routine will calculate if the addition from Insert() is acceptable by
! comparing energy of the old system to the energy of the new system dependent on the
! configuration.  If the energy is less, the addition is accepted and if the probability is large
! based on the Boltzmann factor than it is also accepted.
!
!               Number of atoms
integer, parameter :: MaxN = 1000
!               Constants
real(8), parameter :: Avogadro = 6.022*10.**(23.)           !molecules/g.mol
real(8), parameter :: Boltzmann = 1.38066*10.**(-23.)       !J/K
real(8), parameter :: ConstantR = Boltzmann * Avogadro  !J*molecules/K * mol
!               Initial Set up for parameters of Argon
real(8), parameter :: sigma = 3.292   !Angstroms
real(8), parameter :: epsilon = 153. !K (reduced, Epsilon/Temperature)
!real(8), parameter :: sigma = 3.405           !angstroms
!real(8), parameter :: epsilon = 120.  !K
real(8), parameter :: MW = 40.                 !molecular weight g/g.mol
real(8), parameter :: rhostar = 0.7     !reduced density, (n*sigma^3/V)
real(8), parameter :: temperature = 101. !K
real(8), parameter :: chemicalPotential = -2033.7 !Joules
!
Common /SetUp/ lengthS, mass
Common /Nrand/iSeed
Common /TemporaryCoord/ X_temp(MaxN), Y_temp(MaxN), Z_Temp(MaxN)
Common /Coordinates/ rx(MaxN), ry(MaxN), rz(MaxN)
Common /runs/ step, accept_i, accept_d, accept_t
Common /num_of_atoms/ count_atoms, potential
Common /LJenergy/ potentialTemp
!
!
real(8) :: zeta, probability, beta, volume
integer(4) :: i,j,k
beta = 1./(ConstantR*temperature)
zeta = exp(beta*chemicalPotential)
volume = lengthS*lengthS*lengthS
if ( (potentialTemp-potential) < 0.) then
        count_atoms = count_atoms
        accept_i = accept_i + 1
```

```
            potential = potentialTemp
            do i = 1, count_atoms
                        rx(i) = X_temp(i)
                        ry(i) = Y_temp(i)
                        rz(i) = Z_temp(i)
            enddo
else
            probability = (zeta*volume)/(count_atoms)*exp(-beta*(potentialTemp-
potential)*epsilon*ConstantR)
            if (probability > randnum(iSeed)) then
                        count_atoms = count_atoms
                        accept_i = accept_i + 1
                        potential = potentialTemp
                        do i = 1, count_atoms
                                    rx(i) = X_temp(i)
                                    ry(i) = Y_temp(i)
                                    rz(i) = Z_temp(i)
                        enddo
            else
                        count_atoms = count_atoms - 1
            endif
endif
end
!----------------------------------------------------------------------------------------------------
            FUNCTION randnum(idum)
            IMPLICIT NONE
            INTEGER, PARAMETER :: K4B=selected_int_kind(9)
            INTEGER(K4B), INTENT(INOUT) :: idum
            REAL(8) :: randnum
!
! This routine was obtained from F90 numerical recipes.
!
! "Minimal" random number generator of Park and Miller combined with a Marsaglia shift
! sequence. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint
! values). This fully portable, scalar generator has the "traditional" (not Fortran 90) calling
! sequence with a random deviate as the returned function value: call with idum a negative
! integer to initialize; thereafter, do not alter idum except to reinitialize. The period of this
! generator is about 3.1E18.
!
            INTEGER(K4B), PARAMETER :: IA=16807,IM=2147483647,IQ=127773,IR=2836
            REAL(8), SAVE :: am
            INTEGER(K4B), SAVE :: ix=-1,iy=-1,k
            if (idum <= 0 .or. iy < 0) then                ! Initialize.
              am=nearest(1.0,-1.0)/IM
              iy=ior(ieor(888889999,abs(idum)),1)
              ix=ieor(777755555,abs(idum))
              idum=abs(idum)+1                             ! Set idum positive.
            end if
            ix=ieor(ix,ishft(ix,13))        ! Marsaglia shift sequence with period 2**32 - 1.
            ix=ieor(ix,ishft(ix,-17))
            ix=ieor(ix,ishft(ix,5))
            k=iy/IQ                        ! Park-Miller sequence by Schrage's method,
                                                                                      ! period 2**31 - 2
            iy=IA*(iy-k*IQ)-IR*k
            if (iy < 0) iy=iy+IM
```

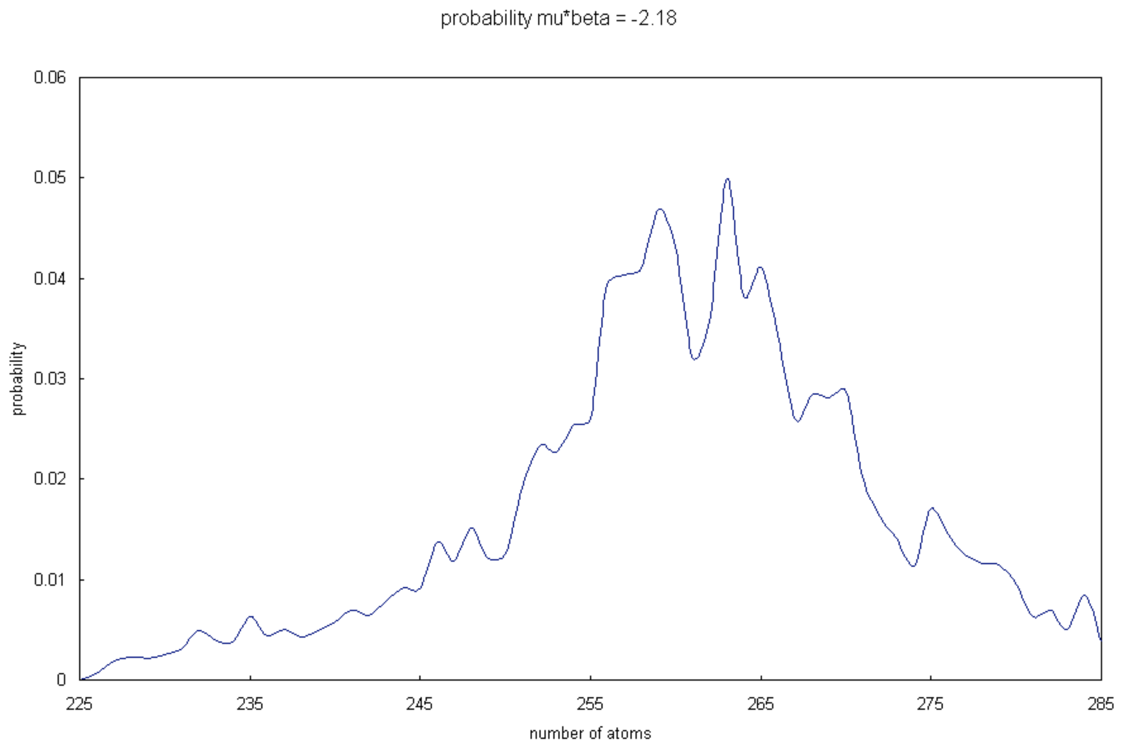
```
        randnum=am*ior(iand(IM,ieor(ix,iy)),1)              ! Combine the two generators with masking
to ran ensure nonzero value.

        END FUNCTION
!------------------------------------------------------------------------------------------------
```

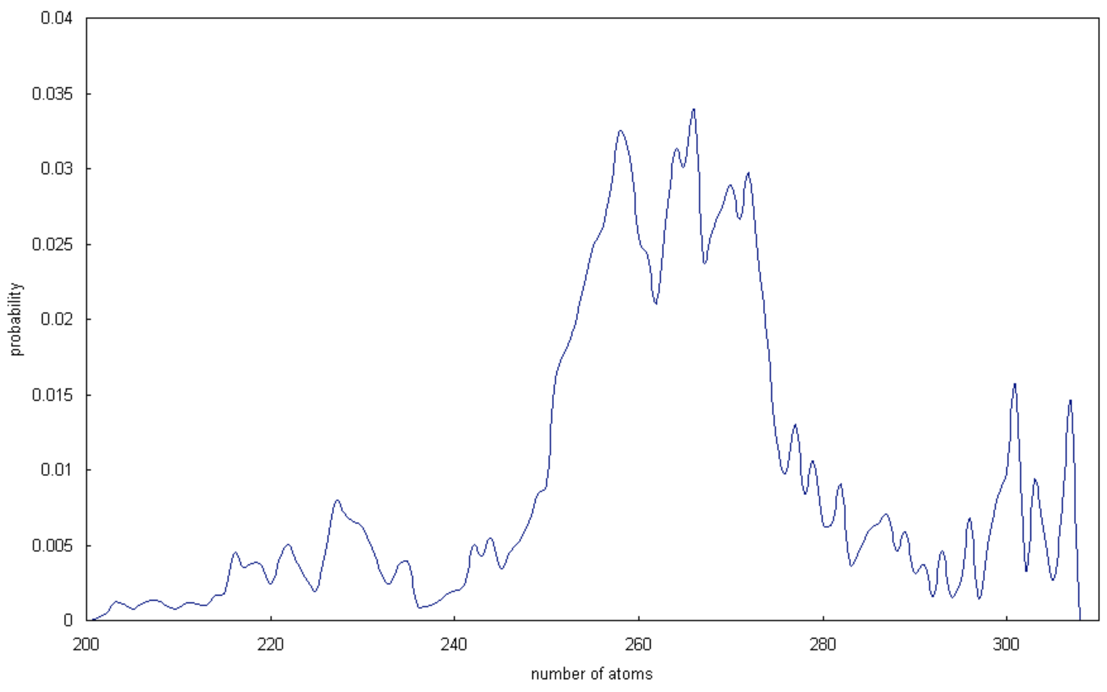
APPENDIX 2: Comparison of mathematical shape between potentials



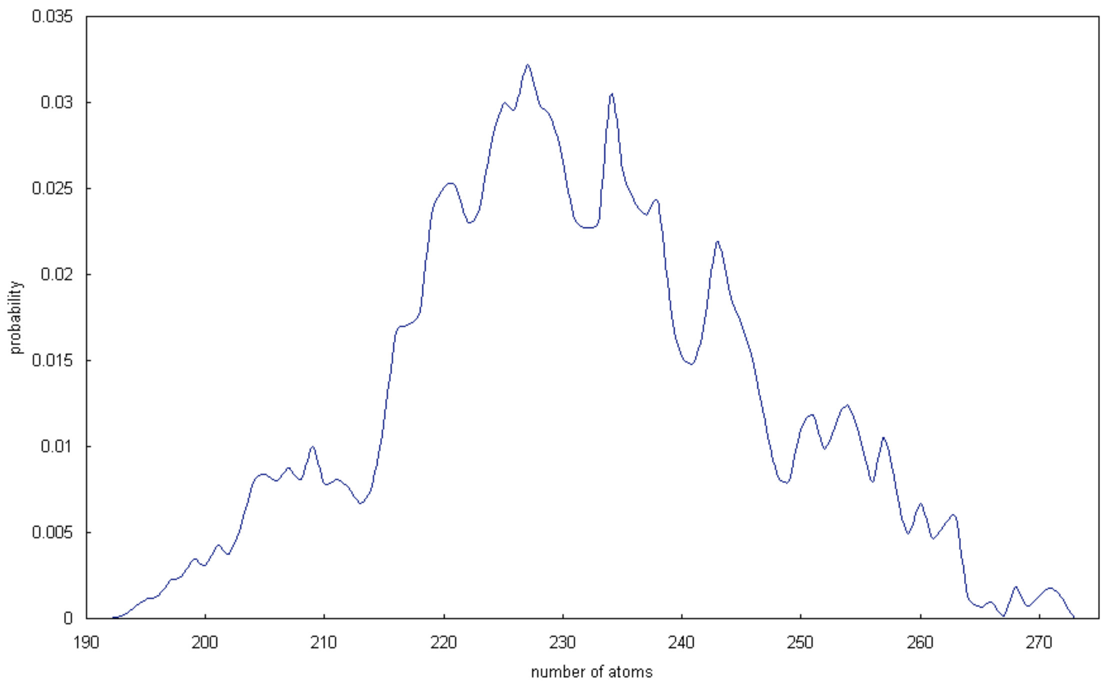
APPENDIX 3: Graphical data produced from code in form of probability density function

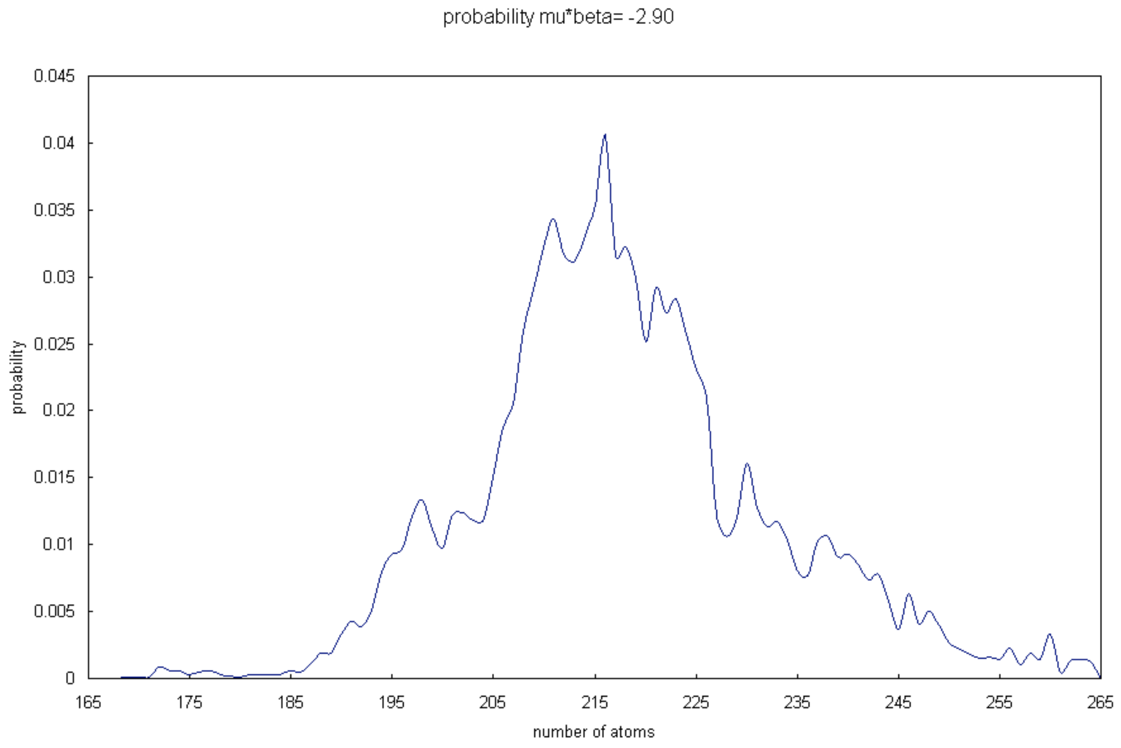


Probability mu*beta = -2.42



Probability mu*beta = -2.66




probability mu*beta= -2.90
probability
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
165
175
185
195
205
215
225
235
245
255
265
number of atoms