MOLECULAR DYNAMICS SIMULATION OF ARGON

ALEXANDER CRONHEIM[1] & ABOUBAKR EL MAHDAOUI[1]

[1] *Faculty of Applied Physics, University of Techonology, Delft, the Netherlands*

ABSTRACT

The motion of a collection of 864 particles has been simulated in Fortran using Molecular Dynamics techniques to compute values for pressure, specific heat and the static pair correlation function with density and temperature as input paramters. The computed properties were compared with experimental results for the properties of argon. The simulation was done with a Lennard-Jones pair-potential and the system was allowed to reach equilibrium. The computed results are in relative good agreement with the known properties of argon.

1 INTRODUCTION

Molecular dynamics is a method to simulate a many-particle system by numerically solving Newton's classical equations of motion for all particles for a period of time. The main limitations are the fact that simulations are only realizable for small systems and times compared to experimental systems in general. The simulations consist of monatomic particles subjected to pair interaction forces that result from a Lennard-Jones 1 potential. This is reported to be in good agreement with experimental results in the case of argon [1].

After initialization the system's equations of motion are solved numerically for each time step using the velocity Verlet algorithm. Because the systems initial state differs from the equilibrium state, the system is first allowed to relax to equilibrium. Because the simulation is run in the microcanonical ensemble, i.e. at constant energy and volume with a fixed number of particles, the system must be moved towards the intended temperature. For that a simple thermostat algorithm that rescales the velocities is used until the system has settled at the intended temperature.

When the desired equilibrium state has been reached, the simulation continues and collects results for calculating the time average of different properties of the system. The equipartition theorem is used for calculating the temperature, the virial theorem is applied in the calculation of the pressure, specific heat is evaluated using a formula derived by Lebowitz using the fluctuations of the kinetic energy[2].

2 METHODS FOR SIMULATION

The Lennard Jones pair potential is used to describe the interaction between a pair of atoms and models a repulsive term and an attractive term as a function of separation distance r_{ij}.

$$V_{ij} = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{1}$$

Here ϵ and σ determine respectively the depth of the potential well and the characteristic separation distance between two particles where the potential is zero. A particle pairs relative position is denoted by $\mathbf{r_{ij}}$. For argon this has been working quite well as a good mathematical model for all three phases[3].

In order to decrease numerical errors ϵ and σ reduced units are used throughout the simulation, with energy described in units of ϵ, distance in σ, and mass in units of particle mass m. Then the reduced unit of time is: $\sigma(m/\epsilon)^{1/2}$. By setting the Boltzman constant k_B equal to one in the simulation, the temperature is transferred to a description in units of ϵ/k_B.

2.1 Initialization

The particles are initialized in a cubic geometry according to an FCC-lattice structure, consistent with the fact that the ground state configuration is an FCC lattice for argon. Considering that a cubic FCC-cell contains four particles, and setting the number of cells per cartesian dimension to six results in 864 particles.

The particles are then assigned an initial velocity that is randomly distributed according to the Maxwell-Boltzmann distribution. A property of this distribution is that each (cartesian) velocity component obeys a Gaussian distribution with zero mean 2:

$$f(v_i) = \sqrt{\frac{m}{2\pi k_B T}} e^{\frac{-m v_i^2}{2 k_B T}} \tag{2}$$

Uniformly distributed random numbers are drawn from the intrinsic Fortran random-number subroutine. The seed for the random number generator is provided by a script from the ICCP coding manual[4] based on the entropy collected by the computer. The Box muller transform is applied to convert two random variables drawn from a uniform distribution to a random variable that is normally distributed.

After assignment of the random initial velocities the center of mass velocity is set to zero by substracting from each particle the system's total average velocity. This is in order to avoid introducing an offset in the kinetic energy calculation.

2.2 Boundary Conditions

As the system to be simulated is very small compared to real experimental systems, boundaries would probably have a profound effect in the simulation. To counter this periodic boundary conditions are used. This is implemented by copying the system in all directions creating a large box of 9 identical smaller boxes with the middle box our system of interest. When-

ever a particle leaves the original system, it must reappear on the exact opposite side with the same velocity vector.

As we calculate the forces the Lennard Jones pair potential is evaluated for all the particles inside the box and also all the other 8 copies that are closer than a predefined cut off distance. The latter is needed as long range interactions would otherwise result in a non periodic potential. This can be done because the potential tends to zero for large r_{ij}.

2.3 Interaction forces

Only pair wise interaction forces are considered between each particle pair: $\mathbf{F_{ij}} = -\nabla V_{ij}$. The force on each particle is then calculated as 3:

$$\mathbf{F_i} = \sum_{j \neq i} \epsilon \left(48 \frac{\sigma^{12}}{r_{ij}^{14}} - 24 \frac{\sigma^6}{r_{ij}^8} \right) \mathbf{r_{ij}} \tag{3}$$

The sum runs over all particle pairs within the cut off distance of the potential in the original system and its neighboring copies. Setting the potential cut off distance to half the length of the original system results in evaluating only the nearest instance of a particle and its copies.

2.4 Time integration

The systems evolution of time is calculated by numerically integrating Newton's equations of motion over discrete timesteps h. The particles velocity and postion at each time step is calculated with the velocity Verlet algorithm:

$$\tilde{\mathbf{v}}(t) = \mathbf{v}(t) + h\mathbf{F}(t)/2 \tag{4}$$

$$\mathbf{r}(t+h) = \mathbf{r}(t) + h\tilde{\mathbf{v}}(t) \tag{5}$$

$$\mathbf{v}(t+h) = \tilde{\mathbf{v}}(t) + h\mathbf{F}(t+h)/2 \tag{6}$$

This algorithm belongs to the class of symplectic integration methods, which if implemented correctly conserve energy. The accumulated error in position and velocity is of order h^2, this is an order more accurate than the simpler semi-implicit Euler method.

2.5 Temperature

The systems temperature T is related to the kinetic energy K through the equipartition theorem 7 for a monoatomic ideal gas.

$$K = \frac{3}{2}(N-1)k_B T \tag{7}$$

The number of degrees of freedom is $3/2$ times the number of particles N. The center of mass degrees of freedom are substracted as these are fixed in a microcanonical system with periodic boundary conditions and no external interactions, following an argument in [5]

In order to move the system towards the intended temperature, all the particles velocities are rescaled by multiplying them with a scaling factor λ, which is the square root of the ratio between the kinetic energy correspond-

ing to the intended temperature T and the systems current kinetic energy, following [5]:

$$\lambda = \sqrt{\frac{3(N-1)k_B T}{\sum_{i=1}^{N} m v_i^2}} \tag{8}$$

2.6 Pressure

Because there are no walls in a system with periodic boundary conditions, the virial theorem is used for measuring the pressure in the simulation. In a pair potential the virial theorem can be described according to equation 9, with $\frac{\delta U(R)}{\delta r_{ij}} = \mathbf{r_{ij}} \cdot \mathbf{f_{ij}}$.

$$\frac{\beta P}{\rho} = 1 + \frac{\beta}{3N} \left\langle \sum_i \sum_{j>i} \mathbf{r_{ij}} \cdot \mathbf{f_{ij}} \right\rangle \tag{9}$$

Here β is shorthand for $1/k_B T$.The brackets denote an ensemble average, however, in our simulation, we take the time average instead. Actually, because the simulation is run in the microcanonical regime, the ensemble average, in our case the time average, must be used for the temperature, which for every timestep is calculated using the equipartition theorem 7.

2.6.1 *Specific Heat*

In the canonical ensemble one can derive the specific heat by measuring the fluctuations of the total energy, as is well known in the field of statistical mechanics. However in the microcanonical ensemble of which our simulated system is an example, there should be no fluctiations as the total energy is kept fixed. Following the derivation by Lebowitz[2] which relates the specific heat to fluctuations in kinetic energy, we can calculate the specific heat using the fluctuations in kinetic energy.

$$\frac{\langle \delta K^2 \rangle}{\langle K \rangle^2} = \frac{2}{3N} \left(1 - \frac{3N}{2C_V} \right) \tag{10}$$

2.6.2 *Pair Correlation Function*

The static pair correlation function in the canonical ensemble can be expressed as:

$$g(\mathbf{r}, \mathbf{r}') = V^2 \frac{1}{N! h^{3N} Z} \int_V d^3 r_3 .... d^3 r_N \, e^{-\beta V_N(\mathbf{r}, \mathbf{r}', \mathbf{r_3} ..... \mathbf{r_N})} \tag{11}$$

noindent If the system is homogeneous and has a large number of particles N, it can also be written as a function of the difference $\mathbf{\Delta r} = \mathbf{r} - \mathbf{r}'$:

$$g(\mathbf{\Delta r}) = \frac{V}{N(N-1)} \left\langle \int d^3 r' \sum_{i,j i \neq j}^{N} \delta(\mathbf{r} - \mathbf{r_i}) \delta(\mathbf{r}' + \Delta \mathbf{r} - \mathbf{r_j}) \right\rangle \tag{12}$$

The pair correlation function 13 can be evaluated in molecular dynamics systems if a histogram is recorded of the number of pairs of particles $n(r)$ for each distance r:

$$g(r) = \frac{2V}{N(N-1)} \left[\frac{\langle n(r) \rangle}{4\pi r^2 \Delta r} \right] \tag{13}$$

Where $V = L^3$ denotes the volume of the system. In our simulation a number of intervals n_i is defined which together with the length in each direction L sets the small Δr:

$$\Delta r = \frac{L}{n_i} \tag{14}$$

For a large enough number of intervals the distance r can be approximated by $i\Delta r$ with i the index of the corresponding interval. The number of pair occurrences $N(r,t)$ in a time step t and in a distance r can be expressed now as function of the interval you are in $N(i,t)$. This way we are able to count all pair occurrences in our program and calculate the pair correlation function:

$$g(r) \approx \frac{2V}{N(N-1)} \left[\frac{\langle n(r) \rangle}{4\pi r^2 \Delta r} \right] \tag{15}$$

$$= \frac{2L^3}{N(N-1)} \left[\frac{\sum_t^{n_t} N(i,t)}{n_t 4\pi i^2 \Delta r^2 \Delta r} \right] \tag{16}$$

$$= \frac{2L^3}{N(N-1)} \frac{\sum_t^{n_t} N(i,t)}{n_t 4\pi i^2} \frac{n_i^3}{L^3} \tag{17}$$

$$= \frac{2}{N(N-1)} \frac{n_i^3}{n_t 4\pi} \frac{\sum_t^{n_t} N(i,t)}{i^2} \tag{18}$$

This is implemented in the code as follows. For each interval i the number of pair occurrences in that interval is registered in a particular time interval. This is done while evaluating the forces of the particles.

2.7 Data blocking

When estimating the statistical measurement error, one would calculate the standard deviation of an ensemble of independent realisations of the same observable. In our case, every measurement is based on the time average of not uncorrelated realisations of instanteneous observables. Because of this one should also take the correlation time of these observables into account.

A simpler method exists however, it amounts to dividing the correlated dataset into blocks of equal length larger than the correlation time. Then, the time averages of the measurements over a single block can be regarded as independent from the other blocks. Subsequently the error can be estimated as the standard deviation of the averages of the different blocks.

3 RESULTS AND DISCUSSION

3.1 Energy

Since the simulation is run in the microcanonical regime, apart from some velocity rescaling in the equilibriation phase, the total energy should be conserved, while the potential and kinetic energy are allowed to fluctuate and this can be seen in Figure 1. Due to numerical errors, the total energy does fluctuate a bit, but to a much lesser degree than the kinetic and potential energy do.

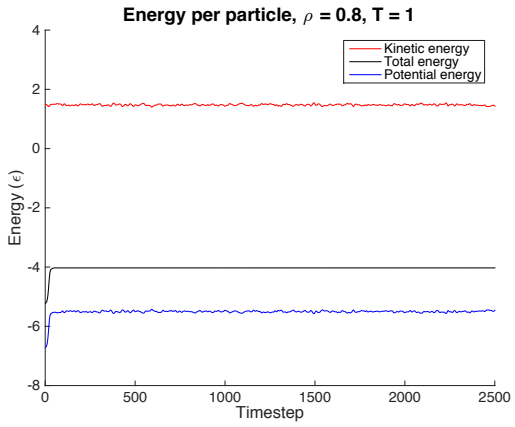


Figure 1: Time evolution of the energies of the system. Note the effect of velocity rescaling in the equilibration phase. Here the fluctuations of the potential and kinetic energy are in the order of $2.43 \cdot 10^{-02}$ ϵ, while those of the total energy are in the order of $6.34 \cdot 10^{-05}$ ϵ.

3.2 Macroscopic Thermodynamic Quantities

In this section the results for pressure, temperature and the specific heat are presented in Table 1 and is compared with the values of Verlet'spaper[1] and Table 2 on the next page for molecular dynamics simulations of a Lennard Jones liquid in [5]

Table 1: Thermodynamic quantities of the Lennard–Jones particles.

$\rho(1/\sigma^3)$	$T_0(\epsilon/k_B)$	T	$\beta P/\rho$	$U(\epsilon)$	C_V
0.3	3.0	3.233(7)	1.16(2)	−1.71(1)	1.64(4)
0.7	0.3	0.51(7)	−3(2)	−5.6(1)	−0(5)
0.7	0.5	0.58(3)	−2.8(8)	−5.29(5)	−0.3(7)
0.7	0.7	0.715(7)	−1.8(1)	−5.06(1)	2.8(4)
0.7	0.9	0.920(5)	−0.33(5)	−4.910(7)	2.1(2)
0.7	1.0	1.014(4)	0.17(3)	−4.842(5)	2(1)
0.7	1.1	1.090(7)	0.46(7)	−4.79(1)	2.2(2)
0.7	1.5	1.500(5)	1.54(5)	−4.546(8)	2.0(2)
0.8	0.4	0.45(2)	−5(2)	−6.23(3)	−0(1)
0.8	0.6	0.588(5)	−2.3(1)	−5.885(7)	2.6(4)
0.8	0.8	0.787(6)	0.05(7)	−5.678(9)	2.4(4)
0.8	1.0	0.984(3)	1.26(4)	−5.503(5)	2.3(4)
0.8	1.1	1.064(7)	1.66(8)	−5.43(1)	2.5(5)
0.8	2.0	2.018(5)	3.35(3)	−4.696(8)	2.2(3)
0.88	0.2	0.256(0)	−20.0(1)	−7.151(1)	2(1)
0.88	0.3	0.375(1)	−10.9(3)	−7.005(2)	2.6(5)
0.88	0.4	0.498(3)	−6.11(3)	−6.855(5)	3(1)
0.88	0.5	0.592(3)	−3.77(3)	−6.739(4)	3(1)
0.88	0.6	0.680(5)	−2.00(3)	−6.611(8)	2.7(8)
0.88	0.7	0.76(1)	0.2(2)	−6.37(2)	6(2)
0.88	0.8	0.79(2)	1.6(3)	−6.20(3)	−14.3(6)
0.88	0.9	0.847(8)	2.33(9)	−6.09(1)	3.0(7)
0.88	1.0	0.977(6)	2.90(7)	−5.936(9)	3(3)
0.88	1.1	1.095(7)	3.55(8)	−5.80(1)	2.4(7)
1.20	0.5	0.515(0)	25.45(3)	−7.373(0)	2(4)

The results show relative good agreement with the reference data. The internal energies are very much comparable, the temperature shows that the thermostat is working quite well. As for the pressure, most data are in good agreement but some values diverge, for example at $\rho = 0.7$ and $T = 1.0$ there is a drop in pressure that isn't reported in Table 2 on the following page, but as will be discussed in the next section, it is clear that a phase transition is taking place going from $\rho = 0.8$ to $\rho = 0.6$ at $T = 1.0$ from liquid argon to solid argon. This would explain the pressure drop and Verlet[1] shows a similar drop at $\rho = 0.65$. The transition from solid to liquid occurs at a lower than expected temperature in our simulation. This suggests a higher preference for the liquid state in our simulations.

The specific heat shows unexpected values and is relatively constant for higher temperatures en densities. This has probably to do with the fact that Lebowitz's formula uses the fluctuations in kinetic energy to evaluate the specific heat, but in our simulations the kinetic energy is fairly constant after relaxing to its equilibrium.

Table 2: Reference data of thermodynamic quantities

$\rho(1/\sigma^3)$	$T_0(\epsilon/k_B)$	T	$\beta P/\rho$	$U(\epsilon)$
0.88	1.0	0.990	2.98	−5.704
0.8	1.0	1.010	1.31	−5.271
0.7	1.0	1.014	1.06	−4.662

$$C_V = \left(\frac{2}{3N} - \frac{\langle K^2 \rangle - \langle K \rangle^2}{\langle K \rangle^2} \right)^{-1}$$

3.3 Pair Correlation function

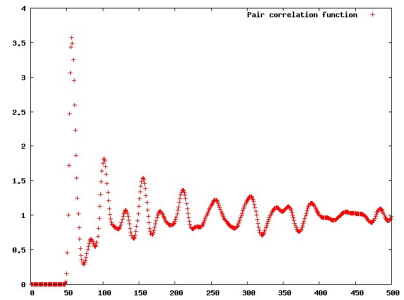
The pair correlation function is plotted for $\rho = 0.88$ and different values of T in Figure 2 on the next page. It can be seen that the correlation function tends to 1 as expected in all cases. Furthermore, the repulsive term of the Lennard Jones potential prevents particles being close together as can be seen in the pair correlation. For $\rho = 0.88$, there is also a phase transition taking place between the temperatures $T = 0.8$ and $T = 0.6$ where solid argon shows a different pair correlation function with respect to liquid argon.

For liquid argon the pair correlation function is simply oscillating indicating what the average distance is between particles. As you move further away from a particular particle, the fluctuations in the average distances cause the peaks to broaden as well as the peaks to be lower for longer distances in the liquid argon. The solid argon shows peaks where the first, second etc. neighbors are in a lattice structure and shows a more complex function, but the oscillations tend to be more persistent even along longer distances compared to liquid argon as can be expected because of the periodic lattice structure. The results presented show good agreement with other experiments[5].

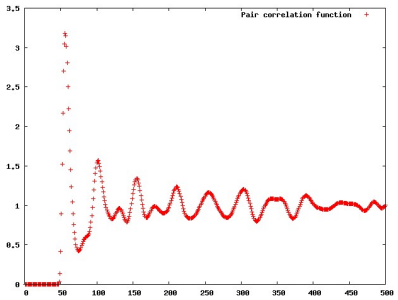
The simulation is also done for a higher temperature and lower density to observe the correlation function of argon in the gas phase. Results are shown in Figure 3 on the following page and show much less oscillations compared to liquid argon because of the increased disorder. This means that after the first peak where on average the first neighboring particle is present, the system is too disordered to show subsequent peaks.

3.4 Error estimation

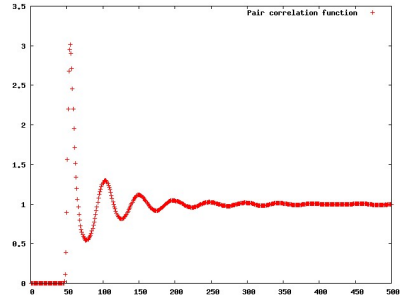
The estimated errors have been added in Table 1 on the previous page. We have chosen a data block size of 200 time steps, because we observed the relaxation time to its equilibrium to be around a time scale less than 200 steps. We have to note that the relatively short simulation time of 2500 time steps compared to the blocks chosen has resulted in significant errors in our results. For future simulations we would recommend a longer simulation time to reduce the errors.



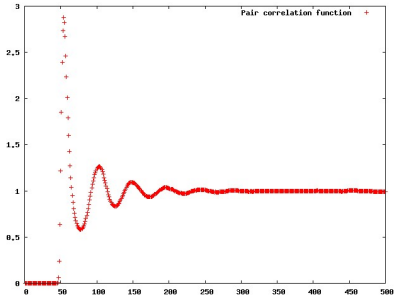
(a) Pair correlation function of Argon($\rho = 0.88$, $T = 0.4$)



(b) Pair correlation function of Argon($\rho = 0.88$, $T = 0.6$)



(c) Pair correlation function of Argon($\rho = 0.88$, $T = 0.8$)



(d) Pair correlation function of Argon($\rho = 0.88$, $T = 1.0$)

Figure 2: Phase transition shown in pair correlation

REFERENCES

[1] L. Verlet. 'Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules'. *Physical Review*, 159, 1967.

[2] S. Duane. 'Stochastic quantization versus the microcanonical ensemble – getting the best of both worlds'. *Nucl. Phys.*, 275:398–420, 1985.

[3] A. Rahman. 'Correlations in the motion of atoms in liquid argon'. *Phys. Rev.*, 136A:405–411, 1964.

[4] Glosser C. 'ICCP Coding Manual'. 2015.

[5] J. Thijssen. 'Computational Physics'. 2013.

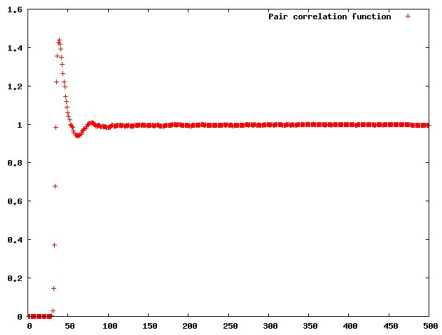


Figure 3: Pair correlation function of solid Argon ($\rho = 0.30$, $T = 3.0$)

A MAIN FORTRAN SOURCE CODE

```
!argon gas in a box simulation, molecular dynamics.

!the cubic geometry sides of length = L
!initial positions initialized according to fcc lattice structure
!number of fcc cells per cartesian dimension = Ncell,
!number of particles is N, (4 particles per cube)

!velocity verlet method: v' = v+1/2*F(x)/m*dt; x = x+v'*dt; v = v'+1/2*F/m*
    dt.
! (coverted from initially an implementation of the semi iplicit euler
    method)
!time evolution for particles in lennard jones potential: U = 4*e*((s/r)
    **12-(s/r)**6),
!Fij =-du/dx =-du/dr*dr/dx = e*(48*s**12/r**13 - 6*s**6/r**7) * x/r,
!r = sqrt(x**2+y**2+z**2)

program argon_box
  use argon_box_init
  use argon_box_dynamics
  use argon_box_results
  !use md_plot
  implicit none

  integer, parameter :: N_cell_dim = 6, velocity_rescale_steps = 50,
      equilibration_steps = 200
  real(8), parameter :: dt = 0.004_8, T_initial = 3.0_8, rho = 0.30_8,
      t_stop = 10d0

  integer, parameter :: N_cell = N_cell_dim**3, N_part = N_cell*4
  real(8), parameter :: L_side = (N_part/rho)**(1._8/3)

  real(8), parameter :: s = 1d0, e = 1d0, r_cut = 5d-1*L_side ! lennard
      jones potential
  real(8), parameter :: m = 1d0, Kb = 1d0   !mass and boltzman constant

  integer, parameter :: hist_num_intervals = 500
  integer, dimension(1:hist_num_intervals) :: histogram_vector,
      tot_histogram_vector = 0

  real(8), dimension(1:3, 1:N_part) :: pos, vel
  integer :: step = 0
  real(8) :: time = 0, kin_energy, pot_energy, virial
  real(8) :: Pressure, Temperature, tot_energy
  real(8) :: sum_kin_energy = 0, sum_virial = 0, sum_potential_energy = 0

  real(8), dimension(1:(int(t_stop/dt)-equilibration_steps+1)) ::
      kin_energy_vector


  ! Create initial state
  call cubic_fcc_lattice(N_cell_dim, L_side, pos)
  call init_random_seed
  call init_vel(T_initial, Kb, m, N_part, vel)

  !call plot_init(0d0, L_side,0d0, L_side,0d0, L_side)

  do while (time < t_stop)
    time = time + dt
    step = step + 1

    !velocity verlet integration method, .true. triggers the calculation of
        thermodynamic quantities.
    call calc_dynamics(.false., N_part, L_side, dt, m, e, s, r_cut, pos,
        kin_energy, pot_energy, &
```

```
                                    & virial, vel, hist_num_intervals,
                                         histogram_vector)
    call new_pos(N_part, L_side, dt, vel, pos)
    call calc_dynamics(.true., N_part, L_side, dt, m, e, s, r_cut, pos,
        kin_energy, pot_energy, &
                                    & virial, vel, hist_num_intervals,
                                         histogram_vector)

    !Temperature control
    if (step < velocity_rescale_steps) then
      call rescale_vel(T_initial, kin_energy, Kb, N_part, vel)
    end if

    tot_energy = pot_energy + kin_energy
    Temperature = 2*kin_energy/(3* (N_part-1) *Kb)  !Center of mass degrees
        of freedom substracted..
    Pressure = 1 + 1/(3*Kb*Temperature*N_part)* virial !P/(Kb T rho) + TODO:
         correction cuttoff

    tot_energy = tot_energy/N_part
    pot_energy = pot_energy/N_part
    kin_energy = kin_energy/N_part

    !call plot_points(pos)
    !print *, step,  "t=", time, "H=", tot_energy, "K=", kin_energy, "U=",
        pot_energy, "virial=", virial, &
    !     & "T=", Temperature, "P=", Pressure
    !print *, histogram_vector

    if (mod(step,25) == 0) then
      print *, "K=", kin_energy, "U=", pot_energy, "virial=", virial, "T=",
          Temperature, "P=", Pressure
    end if
    if (equilibration_steps < step) then
      kin_energy_vector(step - equilibration_steps) = kin_energy
      tot_histogram_vector = tot_histogram_vector + histogram_vector
      sum_virial = sum_virial + virial
      sum_potential_energy = sum_potential_energy + pot_energy
      ! variables sum_kin_energy and sum_kin_energy_sqr calculated in
          subroutine calc_specific_heat
      ! call calc_specific_heat(.false., N_part, kin_energy, sum_kin_energy,
           sum_kin_energy_sqr, step - equilibration_steps)
    end if
    call write_energy_file(kin_energy, pot_energy, virial, time, step)
  end do
  !call plot_end

  ! results
  call calc_specific_heat(N_part, step - equilibration_steps,
      kin_energy_vector, sum_kin_energy)
  pot_energy = sum_potential_energy/(step - equilibration_steps)
  kin_energy = sum_kin_energy/(step - equilibration_steps)
  tot_energy = kin_energy + pot_energy
  virial = sum_virial/(step - equilibration_steps)
  Temperature = 2*kin_energy*N_part/(3* (N_part-1) *Kb) !Center of mass
      degrees of freedom substracted..
  Pressure = 1 + 1/(3*Kb*Temperature*N_part)* virial !P/(Kb T rho) + TODO:
      correction cuttoff
  !print *, "H=", tot_energy, "K=", kin_energy, "U=", pot_energy, "virial=",
       virial, "T=", Temperature, "P=", Pressure
  call write_histogram_file(tot_histogram_vector, hist_num_intervals, N_part
      , step - equilibration_steps)
end program
```

Listing 1: argon_box.f90

B FORTRAN SOURCE CODE FOR SETTING THE INITIAL CONDITIONS

```
module argon_box_init
  implicit none
  private

  public cubic_fcc_lattice, init_vel, init_random_seed

contains

  function fcc_cell(i) result(output)
    implicit none
    integer, intent(in) :: i
    real(8) :: output(3)
    ! face centered cubic unit cell with basis particle positions:
    real(8), dimension(1:3), parameter :: &
    fcc_part1 = (/0d0,  0d0,  0d0/),  &
    fcc_part2 = (/0d0,  5d-1, 5d-1/), &
    fcc_part3 = (/5d-1, 0d0,  5d-1/), &
    fcc_part4 = (/5d-1, 5d-1, 0d0/)
    real(8), dimension(1:3,1:4), parameter :: &
    R_cell = reshape( (/fcc_part1, fcc_part2, fcc_part3, fcc_part4/),
         (/3,4/))
    output = R_cell(:,i)

    ! print *, "face centered cubic unit cell"
    ! do i = 1,3    !   print *, (R_cell(i,j), j=1,4)
    ! end do
  end function fcc_cell

  subroutine cubic_fcc_lattice(N_cell_dim, L_side, pos)
    !initial positions of all particles according to an fcc lattice
        structure
    integer :: i,j,k,l,n
    integer, intent(in) :: N_cell_dim
    real(8), intent(in) :: L_side
    real(8), intent(out), dimension(1:3, 1:(4*N_cell_dim**3)) :: pos

    n = 0
    do i = 1,N_cell_dim
      do j = 1,N_cell_dim
        do k = 1,N_cell_dim
          do l = 1,4
            n = n + 1
            pos(:,n) = L_side/N_cell_dim*((/ i-1, j-1, k-1 /) + fcc_cell(l))
            !     print *, "particle:", n, "/",  N_part, "position:", pos(:,
                n)
          end do
        end do
      end do
    end do
  end subroutine

  subroutine init_vel(T, Kb, m, N_part, vel)
    !initial particles velocities according to the maxwell distribution
    ! in the maxwell boltzman distribution each velocity component is
        normally distributed:
    ! Box muller transform used for converting uniform dist to normal dist
    !
    ! f(v) = sqrt(m/(2*PI*Kb*T)) * exp(-(v**2)/2 *m/(*Kb*T))
    ! sigma**2 = Kb*T/m, and zero mean
    real(8), intent(in)  :: T, Kb, m
    integer, intent(in) :: N_part
    real(8), intent(out), dimension(1:3, 1:N_part) :: vel
    real(8), parameter :: pi = 4*atan(1d0)
```

```
    real(8) :: xs(2) !two random numbers
    integer :: n, i

    do n = 1,N_part
      do i = 1,3
        CALL RANDOM_NUMBER(xs(1))
        CALL RANDOM_NUMBER(xs(2))
        vel(i,n) = sqrt(Kb*T/m) * sqrt(-2d0*log(xs(1)))*cos(2*pi*xs(2)) !
             sigma * box_muller
      end do
    end do
    !Set center of mass velocity to zero
    do i = 1,3
      vel(i,:) = vel(i,:) - sum(vel(i,:))/N_part
    end do
  end subroutine

  ! copied from ICCP coding-notes
  subroutine init_random_seed()
    implicit none
    integer, allocatable :: seed(:)
    integer :: i, n, un, istat, dt(8), pid, t(2), s
    integer(8) :: count, tms

    call random_seed(size = n)
    allocate(seed(n))
    open(newunit=un, file="/dev/urandom", access="stream",&
    form="unformatted", action="read", status="old", &
    iostat=istat)
    if (istat == 0) then
      read(un) seed
      close(un)
    else
      call system_clock(count)
      if (count /= 0) then
        t = transfer(count, t)
      else
        call date_and_time(values=dt)
        tms = (dt(1) - 1970)*365_8 * 24 * 60 * 60 * 1000 &
        + dt(2) * 31_8 * 24 * 60 * 60 * 1000 &
        + dt(3) * 24 * 60 * 60 * 60 * 1000 &
        + dt(5) * 60 * 60 * 1000 &
        + dt(6) * 60 * 1000 + dt(7) * 1000 &
        + dt(8)
        t = transfer(tms, t)
      end if
      s = ieor(t(1), t(2))
      pid = getpid() + 1099279 ! Add a prime
      s = ieor(s, pid)
      if (n >= 3) then
        seed(1) = t(1) + 36269
        seed(2) = t(2) + 72551
        seed(3) = pid
        if (n > 3) then
          seed(4:) = s + 37 * (/ (i, i = 0, n - 4) /)
        end if
      else
        seed = s + 37 * (/ (i, i = 0, n - 1 ) /)
      end if
    end if
    call random_seed(put=seed)
  end subroutine init_random_seed

end module
```

Listing 2: argon_box_init.f90

C FORTRAN SOURCE CODE FOR THE DYNAMICS OF THE SYSTEM

```
module argon_box_dynamics
  implicit none
  private

  public calc_dynamics, rescale_vel, new_pos

contains

  subroutine calc_dynamics(calc_quant, N_part, L_side, time_step, m, e, s,
       r_cut, pos, kin_energy, pot_energy,&
                                & virial, vel, num_intervals,
                                     histogram_vector)

    logical, intent(in) :: calc_quant !for improving efficiency with
        velocity verlet method
    integer, intent(in) :: N_part, num_intervals
    real(8), intent(in) :: e, s, r_cut !Lennard Jones
    real(8), intent(in) :: m, time_step, L_side
    real(8), intent(inout), dimension(1:3, 1:N_part) :: vel
    real(8), intent(in), dimension(1:3, 1:N_part) :: pos
    integer :: i,j,k,l,n
    real(8), intent(out) :: kin_energy, pot_energy, virial
    real(8) :: sum_v_2, F(3), dF(3), r, r_vec(3)

    integer, intent(out), dimension(1:num_intervals) :: histogram_vector
    integer :: hist_i
    real(8) :: delta_r_hist
    delta_r_hist = L_side / num_intervals

    virial = 0
    pot_energy = 0
    sum_v_2 = 0
    histogram_vector = 0

    do n = 1,N_part
      F = 0

      do i =  1,N_part !integrate over all particles inside box except i = n
        do j = -1, 1
        do k = -1, 1 !periodic boundary condition
        do l = -1, 1
          if (n/=i) then
            r_vec = (/pos(1,n)-pos(1,i), pos(2,n)-pos(2,i), pos(3,n)-pos(3,i
                 )/) + L_side*(/j,k,l/)
            r = sqrt(dot_product(r_vec, r_vec))
            !force calculation
            if (r<r_cut) then
              dF = e*(48*s**12/r**14 - 24*s**6/r**8) * r_vec
              F = F + dF
              ! calculate other quantities:
              if (calc_quant .eqv. .true.) then
                if (n > i) then
                  pot_energy = pot_energy + 4*e*((s/r)**12-(s/r)**6)
                  virial =  virial + dot_product(r_vec, dF)
                  !print *, "Fij", e*(48*s**12/r**14 - 24*s**6/r**8), "r", r
                end if
              end if
            end if
            ! histogram for the pair correlation function
            if ((n > i) .and. (calc_quant .eqv. .true.)) then
              hist_i = 1 + floor(r/delta_r_hist)
              if (hist_i < num_intervals + 1) then ! defines a cut off
                   distance
```

```
                histogram_vector(hist_i) = histogram_vector(hist_i) + 1
              !else
              ! histogram_vector(num_intervals) = histogram_vector(
                  num_intervals) + 1
            end if
          end if
        end if
      end do
      end do
      end do
    end do

    vel(:,n) = vel(:,n) + F/m*time_step/2 ! velocity verlet method -->
        factor 1/2 !
    if (calc_quant .eqv. .true.) then
      sum_v_2 = sum_V_2 + dot_product(vel(:,n),vel(:,n))
    end if
  end do
  !print *, "virial", virial
  kin_energy = m/2*sum_v_2

end subroutine



subroutine rescale_vel(T_intended, kin_energy, Kb, N_part, Vel)
  !rescale velocities in order to move system to desired temperature

  real(8), intent(in) :: T_intended, kin_energy, kb
  integer, intent(in) :: N_part
  real(8), intent(inout), dimension(1:3, 1:N_part) :: vel
  real(8) :: scaling_factor

  scaling_factor = sqrt((N_part - 1)*3/2*kb*T_intended/kin_energy)
  vel = scaling_factor*vel

end subroutine

subroutine new_pos(N_part, L_side, dt, vel, pos)
  ! Postion calculation

  real(8), intent(in) :: L_side, dt
  integer, intent(in) :: N_part
  real(8), intent(in), dimension(1:3, 1:N_part) :: vel
  real(8), intent(inout), dimension(1:3, 1:N_part) :: pos
  integer :: n, i

  do n = 1,N_part
    pos(:,n) = pos(:,n) + vel(:,n)*dt
    do i = 1,3 !implements periodic boundary conditions
      if (pos(i,n) < 0d0) then
        pos(i,n) = pos(i,n) + L_side
      else if (pos(i,n) > L_side) then
        pos(i,n) = pos(i,n) - L_side
      end if
    end do
  end do

end subroutine

end module
```

Listing 3: argon_box_dynamics.f90

D FORTRAN SOURCE CODE FOR OUTPUT OF RESULTS

```
module argon_box_results
  implicit none
    private

  public calc_specific_heat, write_energy_file, write_histogram_file

contains

  subroutine write_energy_file(kin_energy, pot_energy, virial, time, cnt)
    real(8), intent(in) :: virial, kin_energy, time, pot_energy
    integer, intent(in) :: cnt
    open (unit=1,file="energy_matrix.dat",action="write")
    write (1,"(I6, 4F18.6)")  cnt, time, kin_energy, pot_energy, virial
  end subroutine

  subroutine write_histogram_file(average_number, num_intervals, N_part,
      step )
    integer, intent(in) :: num_intervals, N_part, step
    integer, intent(in), dimension(1:num_intervals) :: average_number
    real(8) :: constant_factor, temp_factor, temp_factor2, temp_factor3
    integer :: i

    temp_factor = 2d0 * num_intervals / N_part
    temp_factor2 = 1d0 * num_intervals / (N_part - 1)
    temp_factor3 = 1d0 * num_intervals / step
    constant_factor = (temp_factor * temp_factor2 * temp_factor3) / ( 4 *
        abs(atan(1d0)) * 4)

    open (unit=6,file="histogram.dat",action="write")
    do i=1,num_intervals

      write (6,"(I3, 4F18.6)")  i, (constant_factor * average_number(i) )/ (
          i**2)

    end do
  end subroutine


  subroutine calc_specific_heat(N_part, step, kin_energy_vector,
      sum_kin_energy)
    integer, intent(in) :: N_part, step
    real(8), intent(in), dimension(1:step) :: kin_energy_vector
    real(8) :: specific_heat, kin_average, kin_average_sqr
    integer :: i
    real(8), intent(out) :: sum_kin_energy




    kin_average = 0d0
    kin_average_sqr = 0d0

    do i=1,step
      kin_average = kin_average + kin_energy_vector(i)
    end do

    sum_kin_energy = kin_average
    kin_average = kin_average/step


    do i=1,step
      kin_average_sqr = kin_average_sqr + (kin_energy_vector(i) -
          kin_average)**2
```

```
    end do

    kin_average_sqr = kin_average_sqr/step


    specific_heat = ((2d0/(3d0*N_part)) - ((kin_average_sqr) / (kin_average)
         )**2 )**(-1)


    open (unit=7,file="specific_heat.dat",action="write")

    write (7,"(4F18.6)")  specific_heat

    print *, "The specific heat is ", specific_heat
  end subroutine


end module
```

Listing 4: argon_box_results.f90