

MOLECULAR DYNAMICS SIMULATION OF ARGON

ALEXANDER CRONHEIM¹ & ABOUBAKR EL MAHDAOUI¹

CONTENTS

1	Introduction	3
2	Theory	4
2.1	Lennard Jones Potential	4
2.2	Virial Theorem	4
2.3	Specific Heat	4
2.4	Pair correlation function	4
3	Methods for Simulation	5
3.1	Initialization	5
3.2	Dynamics	6
3.3	Information Processing	7
4	Results and Discussion	10
4.1	Pressure	10
4.2	Temperature	10
4.3	Specific Heat	10
4.4	Pair Correlation function	10
4.5	Phase Transitions	10
	Appendix A Main Fortran source code	12
	Appendix B Fortran source code for setting the initial conditions	14
	Appendix C Fortran source code for the dynamics of the system	16
	Appendix D Fortran source code for output of results	18

LIST OF FIGURES

LIST OF TABLES

¹ Faculty of Applied Physics, University of Technology, Delft, the Netherlands

ABSTRACT

The motion of a collection of 864 particles has been simulated using Molecular Dynamics techniques to compute values for pressure, specific heat and the static pair correlation function in reduced units. 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 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. Furthermore, the systems to be simulated can only be classical in the usual molecular dynamics approach.

In this report we discuss the results of simulations for a system of argon gas with 864 particles. Molecular dynamics simulations of argon are reported to be in good agreement with experimental results[1]. The particles are placed in a FCC lattice, considering the fact that the ground state configuration is a FCC lattice for argon. A Lennard-Jones Potential is implemented for the calculation of the force between the particles where a pair potential is assumed. The initial velocities of the particles are randomly generated for each velocity component while obeying a Maxwell-Boltzmann distribution for all particles.

After initialization the system's equations of motion are solved numerically after each time step using Verlet's algorithm. The system is allowed to relax to its equilibrium using a thermostat to rescale the velocities and enforce energy conservation. After the equilibrium has been reached, the simulation continues and collects results for calculating the time average of different properties of the system. The virial's theorem allows the calculation of the pressure, specific heat is evaluated using a formula derived by Lebowitz using the fluctuations of the kinetic energy[2].

2 THEORY

2.1 Lennard Jones Potential

2.2 Virial Theorem

2.3 Specific Heat

The specific heat at constant volume C_V is defined as:

$$C_V = \left(\frac{\delta E}{\delta T} \right)_V$$

In molecular dynamics simulations, a quantity that can be calculated is the ensemble average of the total energy $\langle E \rangle_{NVT}$:

$$\langle E \rangle_{NVT} = \frac{\sum_X e^{-\beta \mathcal{H}(X)} \mathcal{H}(X)}{\sum_X e^{-\beta \mathcal{H}(X)}} = -\frac{\delta \ln(Z)}{\delta \beta}$$

Using the ensemble average for the total energy, the formula for the specific heat can be rewritten as a function of the fluctuations in the total energy:

$$C_V = \frac{1}{k_B T^2} \frac{\delta^2 \ln(Z)}{\delta \beta^2} \quad (1)$$

$$= \frac{1}{k_B T^2} \left(\langle E^2 \rangle_{NVT} - \langle E \rangle_{NVT}^2 \right) \quad (2)$$

This is still difficult to use in a program simulating a system in the microcanonical ensemble as the total energy is kept fixed, but following the derivation by Lebowitz[2] this can be related to the fluctuation in kinetic energy:

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

2.4 Pair correlation function

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

$$g(\mathbf{r}, \mathbf{r}') = V^2 \frac{1}{N! h^{3N} Z} \int_V d^3 r_3 \dots d^3 r_N e^{-\beta V_N(\mathbf{r}, \mathbf{r}', \mathbf{r}_3, \dots, \mathbf{r}_N)}$$

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

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

3 METHODS FOR SIMULATION

3.1 Initialization

The system is initialized by creating an fcc lattice structure and assigning each position of the particles to the lattice points. This is done for 864 particles but can in principle be easily implemented for a number of particles N that can be written as $4M^3$ with M an integer and representing the number of fcc cells in one dimension.

```

subroutine cubic_fcc_lattice(N_cell_dim, L_side, pos)
  ....

  n = 0
5  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
10         pos(:,n) = L_side/N_cell_dim*((/ i-1, j-1, k-1 /) + fcc_cell(l))
        end do
      end do
    end do
  end do
15 end subroutine

```

Listing 1: Constructing the fcc lattice

Above can you see the code with i , j and k for the x , y and z component and a function `fcc_cell` is called that takes an argument l to add the relative position of each particle in a particular cell.

3.1.1 Velocity Distribution

The particles are then assigned an initial velocity that is randomly distributed but the distribution in each direction obeys a Maxwell-Boltzmann distribution:

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

```

subroutine init_vel(T, Kb, m, N_part, vel)
  ....

  do n = 1,N_part
5    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
10    end do
  end do

  ....
end subroutine

```

Listing 2: Generating initial velocities

```

subroutine init_vel(T, Kb, m, N_part, vel)
  ....
  !Set center of mass velocity to zero

```

```

5      do i = 1,3
        vel(i,:) = vel(i,:) - sum(vel(i,:))/N_part
      end do
end subroutine

```

Listing 3: Removing the center of mass degree of freedom

3.1.1.1 BOX MULLER ALGORITHM

3.2 Dynamics

3.2.1 Boundary Conditions

As the system to be simulated is very small compared to real experimental systems, boundaries would probably have a larger effect in the simulation. To counter this periodic boundary conditions are used and this is implemented by copying the system in all directions creating a large box of 9 identical smaller boxes with the box in the middle as our box of interest. As we calculate the forces the Lennard Jones pair potential is calculated for all the particles inside the box and also all the other 8 copies that are closer than a predefined cut off distance.

```

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)
  ....

  do n = 1, N_part
    F = 0
5    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
10         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
15             dF = e*(48*s**12/r**14 - 24*s**6/r**8) * r_vec
             F = F + dF
           end if
         end if
       end do
     end do
20   end do
  end do
end subroutine

```

Listing 4: Creating periodic boundary conditions

3.2.2 Lennard Jones Potential

3.2.3 Force Calculation

3.2.3.1 LEAP FROG/VERLET ALGORITHM

3.2.4 Pressure Calculation

3.2.4.1 VIRIAL THEOREM

3.2.5 Thermostat

3.3 Information Processing

3.3.1 Specific Heat

The implementation in our simulation is inside the algorithm for the dynamics of the particles. For each time step the new velocities are calculated, the kinetic energy is also calculated and the relevant sums of the kinetic energy are updated in a subroutine "calc_specific_heat":

```

subroutine calc_specific_heat(end_of_routine,N_part, kin_energy,
    sum_kin_energy, sum_kin_energy_sqr)
    ....

    sum_kin_energy_sqr = sum_kin_energy_sqr + kin_energy**2
    sum_kin_energy = sum_kin_energy + kin_energy

    ....
end subroutine

```

Listing 5: Updating the relevant sums of the kinetic energy

As discussed earlier, the specific heat is related to the fluctuations in kinetic energy using a formula derived by Lebowitz[2]:

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

Rewriting this to get an expression for the specific heat:

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

$$= \left(\frac{2}{3N} - \frac{\langle K^2 \rangle - \langle K \rangle^2}{\langle K \rangle^2} \right)^{-1} \quad (4)$$

Using the time average instead of the ensemble average, the averages can be expressed as a sum over all time steps n_t divided by the time steps:

$$\langle K^2 \rangle = \frac{\sum_{i=1}^{n_t} K(i)^2}{n_t}$$

$$\langle K \rangle^2 = \left(\frac{\sum_{i=1}^{n_t} K(i)}{n_t} \right)^2$$

Now the specific heat can be calculated as:

$$C_V = \left(\frac{2}{3N} - \frac{\langle K^2 \rangle - \langle K \rangle^2}{\langle K \rangle^2} \right)^{-1} \quad (5)$$

$$= \left(\frac{2}{3N} - \frac{\sum_{i=1}^{n_t} K(i)^2 * n_t - (\sum_{i=1}^{n_t} K(i))^2}{(\sum_{i=1}^{n_t} K(i))^2} \right)^{-1} \quad (6)$$

This is implemented in our code within the same subroutine "calc_specific_heat" after an if statement is enabled at the end of the simulation:

```

subroutine calc_specific_heat(end_of_routine,N_part, kin_energy,
    sum_kin_energy, sum_kin_energy_sqr, step)
    ....

    if (end_of_routine .eqv. .true.) then
        specific_heat = ((2d0/(3d0*N_part)) - (((sum_kin_energy_sqr * step) -
            sum_kin_energy**2) / (sum_kin_energy**2)))*(-1)

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

```

Listing 6: Calculating the specific heat

3.3.2 Pair Correlation Function

The pair correlation function 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 :

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

In our simulation a number of intervals n_i is defined which together with the length in each direction L sets the small Δr :

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

Again a time average is used and the average number of pairs $\langle n(r) \rangle$ is calculated as a sum of all the pair occurrences $N(r, t)$ in a time step t and in a distance r over all time steps n_t :

$$\langle n(r) \rangle = \frac{\sum_t^{n_t} N(r, t)}{n_t}$$

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:

$$g(r) \approx \frac{2V}{N(N-1)} \left[\frac{\langle n(r) \rangle}{4\pi r^2 \Delta r} \right] \quad (7)$$

$$= \frac{2L^3}{N(N-1)} \left[\frac{\sum_t^{n_t} N(i, t)}{n_t 4\pi i^2 \Delta r^2 \Delta r} \right] \quad (8)$$

$$= \frac{2L^3}{N(N-1)} \frac{\sum_t^{n_t} N(i, t)}{n_t 4\pi i^2} \frac{n_i^3}{L^3} \quad (9)$$

$$= \frac{2}{N(N-1)} \frac{n_i^3}{n_t 4\pi} \frac{\sum_t^{n_t} N(i, t)}{i^2} \quad (10)$$

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:


```

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)

    histogram_vector = 0

5
    do n = 1,N_part
        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
10             do l = -1, 1
                    if (n/=i) then
                        r_vec = .... !Calculation of the difference vector
                        r = sqrt(dot_product(r_vec, r_vec))

15                     ! 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
20                             end if
                        end if
                    end if
                end do
            end do
        end do
25     end do
end do

end subroutine

```

Listing 7: Updating the histogram for each interval

As the final time step is done and the routine for the dynamics is finished, the sum for each interval is passed to a routine for calculating the pair correlation function that is called at the end:

```

subroutine write_histogram_file(average_number, num_intervals, N_part, step
)

    temp_factor = 2d0 * num_intervals / N_part
    temp_factor2 = 1d0 * num_intervals / (N_part - 1)
5    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
10        write (6,"(I3, 4F18.6)") i, (constant_factor * average_number(i) )/ (i
            **2)
    end do
end subroutine

```

Listing 8: Factors are needed to prevent overflow errors

4 RESULTS AND DISCUSSION

4.1 Pressure

4.2 Temperature

4.3 Specific Heat

Some text with a citation [1] The other citation [3] and another[4]

4.4 Pair Correlation function

4.5 Phase Transitions

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] Glosser C. 'ICCP Coding Manual'. 2015.
- [4] J. Thijssen. 'Computational Physics'. 2013.

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,
5 !number of particles is N, (4 particles per cube)

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

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

20

integer, parameter :: N_cell_dim = 6, velocity_rescale_steps = 50,
equilibration_steps = 120
real(8), parameter :: dt = 0.004_8, T_initial = 1d0, rho = 0.88_8, t_stop
= 1d0

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

30 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
35 integer :: step = 0
real(8) :: time = 0, kin_energy, pot_energy, virial
real(8) :: Pressure, Temperature, tot_energy
real(8) :: sum_kin_energy_sqr = 0, sum_kin_energy = 0, sum_virial = 0,
sum_potential_energy = 0

40

! 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)

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

do while (time < t_stop)
time = time + dt
50 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)

```

```

55  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
60  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 subtracted..
65  Pressure = 1 + 1/(3*Kb*Temperature*N_part)* virial !P/(Kb T rho) + TODO:
        correction cutoff

    tot_energy = tot_energy/N_part
    pot_energy = pot_energy/N_part
70  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
75  !print *, histogram_vector

    if (equilibration_steps < step) then
        tot_histogram_vector = tot_histogram_vector + histogram_vector
        sum_virial = sum_virial + virial
80  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)
85  end do
    ! call plot_end

    ! results
    pot_energy = sum_potential_energy/(step - equilibration_steps)
90  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 subtracted..
    Pressure = 1 + 1/(3*Kb*Temperature*N_part)* virial !P/(Kb T rho) + TODO:
        correction cutoff
95  print *, "H=", tot_energy, "K=", kin_energy, "U=", pot_energy, "virial=",
        virial, "T=", Temperature, "P=", Pressure
    call calc_specific_heat(.true., N_part, kin_energy, sum_kin_energy,
        sum_kin_energy_sqr, step - equilibration_steps)
    call write_histogram_file(tot_histogram_vector, hist_num_intervals, N_part
        , step - equilibration_steps)
end program

```

Listing 9: argon_box.f90

B FORTRAN SOURCE CODE FOR SETTING THE INITIAL CONDITIONS

```

module argon_box_init
  implicit none
  private

5   public cubic_fcc_lattice, init_vel, init_random_seed

contains

  function fcc_cell(i) result(output)
10    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 :: &
15    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 :: &
20    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)
25    ! 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
30    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

35    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
40            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
45    end do
end subroutine

  subroutine init_vel(T, Kb, m, N_part, vel)
50    !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
55    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)

```

```

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

    do n = 1,N_part
        do i = 1,3
65          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
70    !Set center of mass velocity to zero
    do i = 1,3
        vel(i,:) = vel(i,:) - sum(vel(i,:))/N_part
    end do
end subroutine

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

    call random_seed(size = n)
    allocate(seed(n))
85    open(newunit=un, file="/dev/urandom", access="stream",&
        form="unformatted", action="read", status="old", &
        iostat=istat)
    if (istat == 0) then
        read(un) seed
90    close(un)
    else
        call system_clock(count)
        if (count /= 0) then
            t = transfer(count, t)
95        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 &
100            + 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
110            seed(2) = t(2) + 72551
            seed(3) = pid
            if (n > 3) then
                seed(4:) = s + 37 * (/ (i, i = 0, n - 4) /)
            end if
        end if
115    else
        seed = s + 37 * (/ (i, i = 0, n - 1) /)
    end if
end if
    call random_seed(put=seed)
120 end subroutine init_random_seed

end module

```

Listing 10: argon_box_init.f90

C FORTRAN SOURCE CODE FOR THE DYNAMICS OF THE SYSTEM

```

module argon_box_dynamics
  implicit none
  private

5   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,&
10      & 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
15    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)

20    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
30    histogram_vector = 0

    do n = 1,N_part
      F = 0
      do i = 1,N_part !integrate over all particles inside box except i = n
35        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)
40                  &)/ + 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
45                  ! 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)
50                    end if
                  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)
75              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
    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
    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)
    ! Position 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 11: argon_box_dynamics.f90

D FORTRAN SOURCE CODE FOR OUTPUT OF RESULTS

```

module argon_box_results
    implicit none
    private

5    public calc_specific_heat, write_energy_file, write_histogram_file

contains

    subroutine write_energy_file(kin_energy, pot_energy, virial, time, cnt)
10        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

15    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
20        integer :: i

        temp_factor = 2d0 * num_intervals / N_part
        temp_factor2 = 1d0 * num_intervals / (N_part - 1)
        temp_factor3 = 1d0 * num_intervals / step
25        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

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

        end do
    end subroutine

35    subroutine calc_specific_heat(end_of_routine, N_part, kin_energy,
        sum_kin_energy, sum_kin_energy_sqr, step)

        logical, intent(in) :: end_of_routine
        real(8), intent(in) :: kin_energy
40        real(8), intent(out) :: sum_kin_energy, sum_kin_energy_sqr
        integer, intent(in) :: N_part, step
        real(8) :: specific_heat

        sum_kin_energy_sqr = sum_kin_energy_sqr + kin_energy**2
45        sum_kin_energy = sum_kin_energy + kin_energy

        if (end_of_routine .eqv. .true.) then
            specific_heat = ((2d0/(3d0*N_part)) - (((sum_kin_energy_sqr * step) -
                sum_kin_energy**2) / (sum_kin_energy**2)))*(-1)

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

55 end module
    
```

Listing 12: argon_box_results.f90