Project Report - CHM684A

Molecular Dynamics Simulation of liquid Argon

Group Members:
Chitresh Bhaskar Chaudhari
Sharad Dattatray Gaikwad
Saarthak Shrivastava

Indian Institute of Technology, Kanpur

Contents

Problem Statement

In this project, the Molecular Dynamics (MD) Simulation of liquid Argon (Ar) has been performed and then the Radial Distribution Function, Mean Square Displacement has been calculated from the simulation trajectory and compared with reported plots in the reference article[1].

1 Introduction And Motivation

The calculations presented here are based on the assumption that classical dynamics with a two-body central-force interaction can give a reasonable description of the motion of atoms in liquid argon. For practical reasons, further assumptions have to be made, namely, the interaction potential has to be truncated beyond a certain range, the number of particles in the assembly has to be kept rather small, and suitable boundary conditions have to be imposed on the assembly. Finally, the equations of motion have to be solved as a set of difference equations, thus involving a certain increment of time to go from one set of positions and velocities to the next.

1.1 Lennard Jones Potential

The Lenard Jones potential is given by:-

$$U(R_1, R_2, R_3, ..., R_n) = 4\epsilon \sum_{I<J} [(\frac{\sigma}{R_{ij}})^{12} - (\frac{\sigma}{R_{ij}})^{6}]$$

$$R_{ij} = |Ri - Rj|$$

ϵ=Depth of potential

σ=Distance before which the repulsive term in the potential contributes

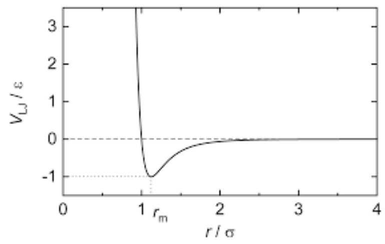


Figure 1: Lennard-Jones potential

[1]A. Rahman (1964) Correlations in the Motion of Atoms in Liquid Argon, Phys. Rev. 136, A405.

1.2 Box-Muller Sampling

Maxwell distribution of velocities at temperature T :-

$$f(v) = [(\frac{M}{2\pi k_b T})^{1/2} \exp(\frac{-Mv^2}{2k_b T})]$$

$$X = \sigma \cos(2\pi \xi_1) \sqrt{-2 \ln \xi_2'}$$

$$Y = \sigma \sin(2\pi \xi_1) \sqrt{-2 \ln \xi_2'}$$

$$\sigma = \sqrt{\frac{k_b T}{M}}$$

X & Y are two random velocities picking randomly from the f(v) at temperature T

1.3 Mean-Square Displacement

$$< r^2 >= \frac{1}{N} \frac{1}{n_{t_0}} \sum_{t_0} \sum_{i}^{N} (\mathbf{r}_i(t_0 - t) - \mathbf{r}_i(t_0))^2$$

Here, is N the number of particles, t is the time and t_0 is the time origin. Also, n_{t_0} is the number of time origins considered for averaging.

1.4 Velocity-Verlet Integrator

Velocity-Verlet Integrator has been used to integrate the equations of motion. The general algorithm for Velocity-Verlet Integrator is as follows:

$$v(t + \Delta t/2) = v(t) + \frac{\Delta t}{2} a(t)$$

$$r(t + \Delta t/2) = r(t) + \Delta t v(t + \Delta t/2)$$

$$v(t + \Delta t) = v(t + \Delta t/2) + \frac{\Delta t}{2} a(t + \Delta t)$$

2 Method and Model

2.1 Model

We used 4 modules and main project file for this project as follows:

- PROGRAM main
- MODULE variables
- MODULE md
- MODULE gr

- MODULE rmsdisplacement

All programs are appended in appendix section. Module variables contains all necessary variables required in program. Module md contains all the subroutines required to do molecular dynamics of argon gas. Module gr contains subroutine to calculate radial distribution function. module rmsdisplacement contains subroutine to calculate $< r^2 >$.

2.2 Method

- First the main program main.f90 calls SUBROUTINE create_lattice(). This subroutine asks for number of atoms, density, temperature, step size and number of steps.
- After this, program call subroutine get_initial_velocities() to assign initial velocities to each atom. This velocities are assigned from the Maxwell-Boltzmann equation using the Box-Muller Sampling.
- After this function calls SUBROUTINE energy_forces() to calculate initial energy and forces.
- After this there is do loop iterating from 1 to maximum number of steps given by user. In this loop Velocity-Verlet Integrator has been used to integrate the equations of motion. There are individual subroutines to calculate using velocity-Verlet integrator.
- In the same loop, temperature is calculated for each step. Also after every 50 iteration program call subroutine to print coordinates of atoms and energies in different files.
- After the loop, program calls SUBROUTINE calculate_gr() which calculates radial distribution function and prints required information to plot g(r) vs r in separate file.
- At last, program call SUBROUTINE rmsd() which calculates root mean square displacement according to equation mentioned before.

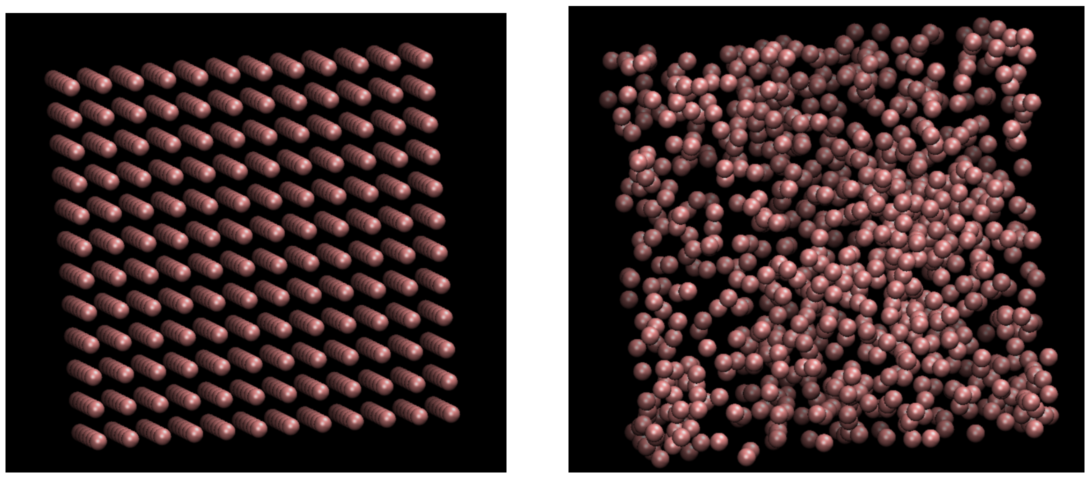
2.3 Simulation details

- **Number of atoms** : 864
- **Density** : 0.2
- **Temperature** : 0.786
- **Time step** : 0.001
- **Number of steps** : 10000

All mentioned quantities are in reduced units. Units are reduced according to the table in the lecture note.

3 Results

3.1 Lattice structure


(a) snapshot of lattice before simulation

(b) Snapshot of lattice at the end of simulation

Figure 2: Lattice structure at start and end of the simulation

3.2 Energy vs Time

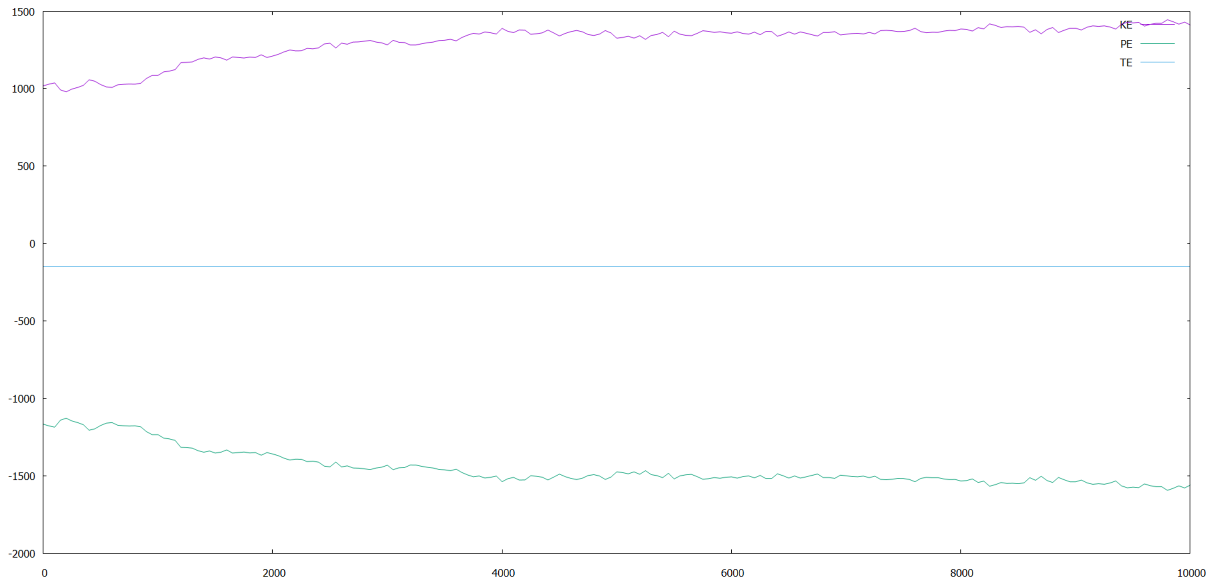


Figure 3: Energy as a function of time

Here we can observe that there is no shift in the total energy. Potential and kinetic energy varies with time but total energy remains same throughout the simulation. Temperature variance is as shown in fig 3:

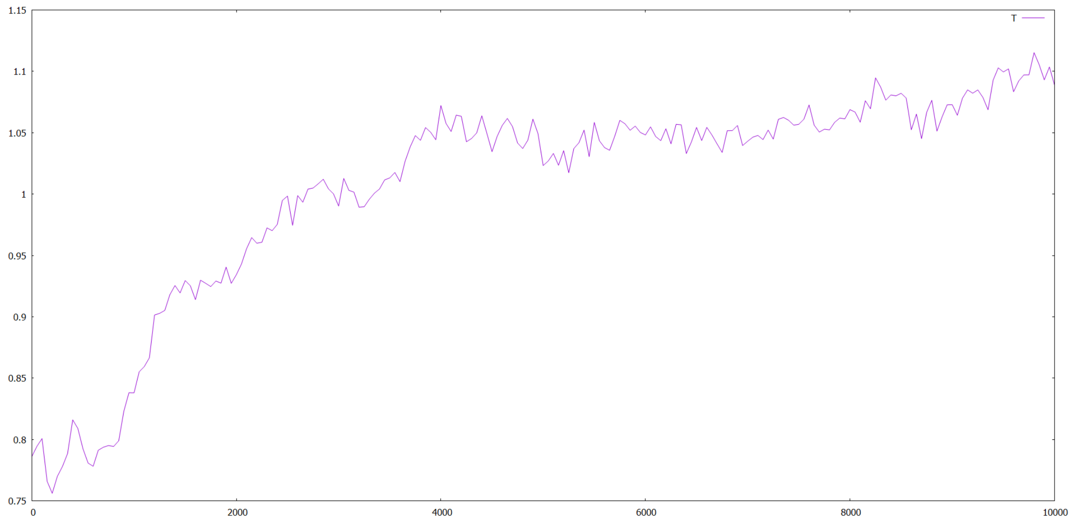


Figure 4: Temperature vs time in MD simulation

3.3 Radial Distribution function g(r)

We got rdf as following figure: This figure is not matching with fig 2 of reference article.

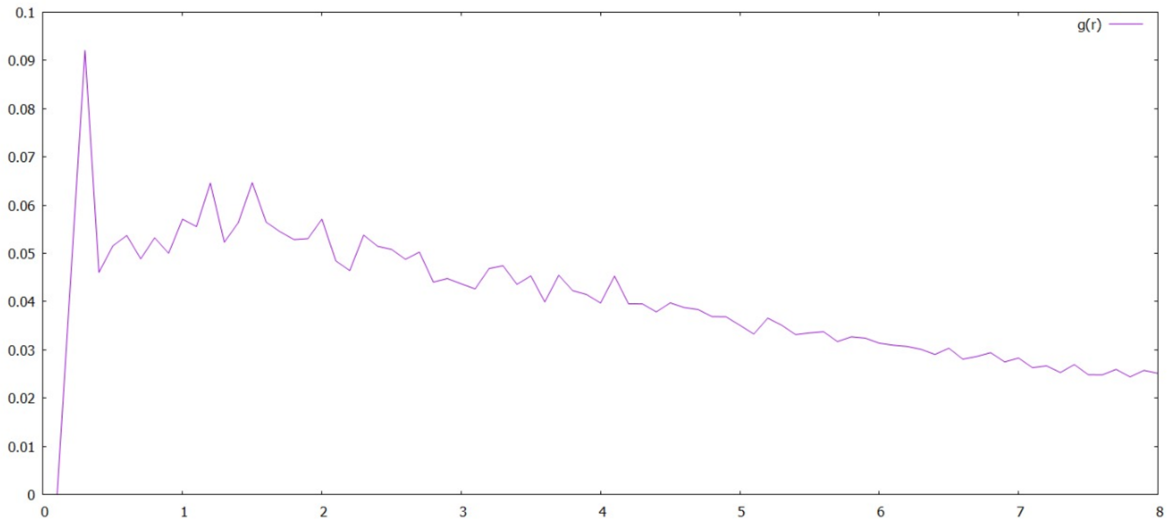


Figure 5: Radial Distribution Function g(r)

This means there is something wrong in our code.

3.4 Mean Square Displacement $< r^2 >$

We got following plot for $< r^2 >$.

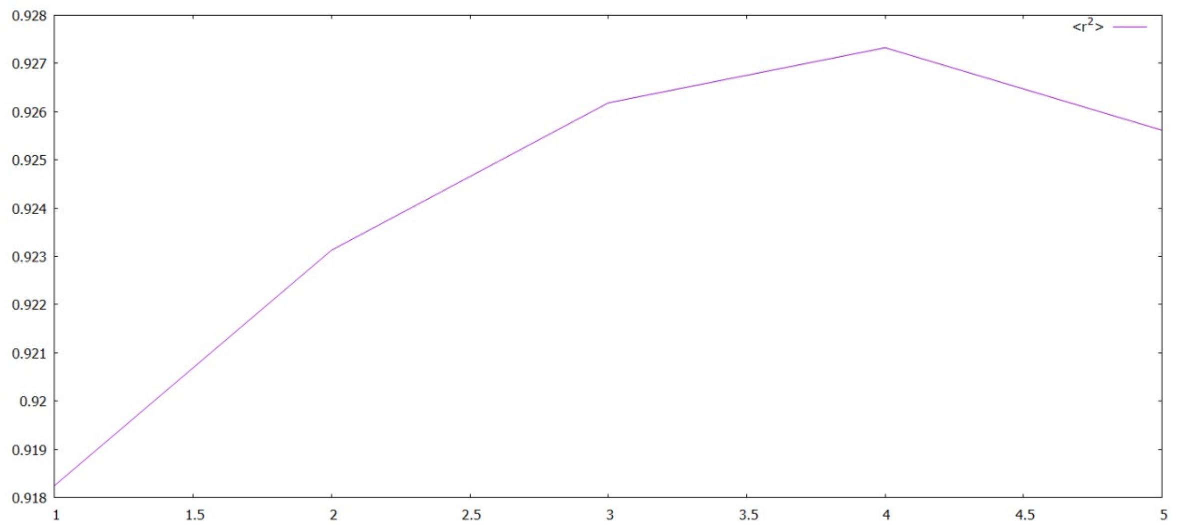


Figure 6: Root mean square displacement - $< r^2 >$

4 Conclusion

We tried to replicate results of paper by A. Rahman in this project. We succeeded in showing absence of drift in total energy but failed to replicate radial distribution function and $< r^2 >$ plot.

Acknowledgement

We would like to thank Prof. Nisanth Nair and Mr. Saurabh Srivastav for Motivating and guiding us throughout the project.

Appendix

Main Program - main.f90

```
program main
use variables
use md
use gr
use rmsdisplacement
implicit none

integer :: istep, T= 0

call create_lattice()
call get_initial_velocities
```

```
CALL print_geo('traj.xyz',0)
 allocate(force(ndim,natoms))
CALL energy_forces
CALL print_energy(0)
DO istep=1,maxstep
!    Velocity Verlet Integrator
     CALL velocity_verlet_v
     CALL velocity_verlet_r
     CALL energy_forces
     CALL velocity_verlet_v
!    Properties: Temperature
     CALL temperature(temp)
     IF(MOD(istep,50).EQ.0)THEN
      CALL print_geo('traj.xyz',istep)
      CALL print_energy(istep)
      T = T + 1
     END IF
  END DO

call calculate_gr(T)
call rmsd(T)
end program main
```

Module for storing variables - variables.f90

```
module variables
implicit none
integer, parameter :: ndim = 3
integer :: natoms, maxstep
real*8 :: rho, L, dt, temp, energy
real*8, allocatable ::pos(:,:), vel(:,:), force(:,:)
REAL*8, PARAMETER :: pi=4.d0*ATAN(1.d0)
end module variables
```

Module for Molecular dynamics - module1.f90

```
module md
use variables
implicit none

contains
!------------------------------------------------------------------------------
subroutine create_lattice()
```

```
    implicit none

    integer :: index, i, j, k
    real*8 :: dx


    print *,"Please enter number of atoms"
    read(*,*)natoms
    print *,"Please enter density"
    read(*,*)rho
    print *,"Please enter Temperature"
    read(*,*)Temp
    print *,"Please enter step size and number of steps"
    read(*,*)dt, maxstep

    L = (real(natoms)/rho)**(1.d0/3)
    dx = L/12
    print *,L
    allocate(pos(ndim,natoms))
    index = 0
    do i = 1, 12
        do j = 1, 12
            do k = 1,6
                index = (i-1)*72 + (j-1)*6 + k
                pos(1, index) = 0.5 + (i-1)*dx
                pos(2, index) = 0.5 + (j-1)*dx
                pos(3, index) = 0.5 + (k-1)*dx*2
            end do
        end do
    end do
end subroutine create_lattice
!-----------------------------------------------------

SUBROUTINE get_initial_velocities
    IMPLICIT NONE
    REAL*8, PARAMETER :: pi=4.d0*ATAN(1.d0)
    REAL*8 :: dum(2), sigma
    INTEGER :: ii, i, k

    allocate(vel(3,natoms))
    sigma=DSQRT(temp)
    ii=0
    DO i=1,natoms
      DO k=1,ndim
        CALL RANDOM_NUMBER(dum(1:2))
```

```
          ii=ii+1
          IF(MOD(ii,2)==0)THEN
             vel(k,i)=SIGMA*DSQRT(-2.D0*DLOG(dum(2)))*DCOS(2.D0*pi*dum(1))
          ELSE
             vel(k,i)=SIGMA*DSQRT(-2.D0*DLOG(dum(2)))*DSIN(2.D0*pi*dum(1))
          END IF
        END DO
      END DO
      CALL rescale_velocities(temp)
      WRITE(6,*) "Initial velocities are assigned for T=", TEMP
    END SUBROUTINE get_initial_velocities
!------------------------------------------------------------
SUBROUTINE rescale_velocities(t)
      IMPLICIT NONE

      REAL*8,INTENT(IN)    ::  t

      REAL*8    ::  t0,scal

      CALL temperature(t0)
      print *, t0
      scal=DSQRT(t/t0)
      vel(:,:)=vel(:,:)*scal

    END SUBROUTINE
!----------------------------------------------
SUBROUTINE temperature(t)
      IMPLICIT NONE
      REAL*8, INTENT(out) :: t

      INTEGER :: i
      REAL*8 :: mv2

      mv2=0.D0
      DO i=1,natoms
        mv2=mv2+DOT_PRODUCT(vel(1:ndim,i),vel(1:ndim,i))
      END DO
      t=(real(1)/3.d0)*mv2/DFLOAT(natoms)
    END SUBROUTINE temperature
!----------------------------------------------------------
    SUBROUTINE velocity_verlet_r
      IMPLICIT NONE

      INTEGER :: i
!
```

```
    DO i=1,natoms
       pos(1:ndim,i)=pos(1:ndim,i)+vel(1:ndim,i)*dt
    END DO
  END SUBROUTINE velocity_verlet_r
!------------------------------------------------------------------
  SUBROUTINE velocity_verlet_v
    IMPLICIT NONE

    INTEGER :: i
!
    DO i=1,natoms
       vel(1:ndim,i)=vel(1:ndim,i)+0.5d0*dt*force(1:ndim,i)
    END DO
  END SUBROUTINE velocity_verlet_v
!--------------------------------------------------------------
SUBROUTINE print_geo(filen,iacc)
    IMPLICIT NONE

    CHARACTER (LEN=*), INTENT(IN)     :: filen
    INTEGER, INTENT(IN)               :: iacc

    INTEGER :: i,j

    IF(iacc.EQ.0)THEN
      OPEN(1,FILE=FILEN,STATUS='UNKNOWN',FORM='FORMATTED')
    ELSE
      OPEN(1,FILE=FILEN,STATUS='UNKNOWN',FORM='FORMATTED',ACCESS='APPEND')
    END IF

    !IF(ndim/=2)STOP 'print_geo() is not implemented for ndim/=2'
    WRITE(1,'(I10)')natoms
    WRITE(1,*)"comment"
    DO i=1,natoms
      DO j=1,ndim
         IF(pos(j,i)>L)pos(j,i)=pos(J,I)-L
         IF(pos(j,i)<0.d0)pos(j,i)=pos(j,i)+L
      END DO
      WRITE(1,"(A,3F16.6)") "Ar", pos(1:3,i)
    END DO
    CLOSE(1)
  END SUBROUTINE print_geo
!---------------------------------------------------------------
  SUBROUTINE energy_forces
    IMPLICIT NONE
    INTEGER  :: i, j
    REAL*8    :: dd(ndim), d, d2, d6, d12, fjk(ndim), fik(ndim)
!

    force(:,:)=0.d0
    energy=0.D0

!   Loop of all the atom pairs (i,j)
    DO i=1,natoms-1
      DO j=i+1,natoms

         dd(1:ndim)=pos(1:ndim,i)-pos(1:ndim,j)
         dd(1:ndim)=dd(1:ndim)-L*NINT(dd(1:ndim)/L)

!        Calculate pair distance
         d=DSQRT(DOT_PRODUCT(dd(1:ndim),dd(1:ndim)))

        IF(d<1.0e-4 )THEN  ! if the distance is too small, STOP
          PRINT *, "i=", i, " j=", j, " d=", d
          PRINT *, "Atom i = ", pos(1:3,i)
          PRINT *, "Atom j = ", pos(1:3,j)
          STOP 'ERROR! Atoms i and j are overlapping!'
        END IF

!        Calculate Energy
         d2=1.d0/(d**2)
         d6=d2*d2*d2
         d12=d6*d6
         energy=energy+d12-d6

!        Calculate Force
         fik(1:ndim)=-(-2.D0*d12+d6)*dd(1:ndim)*d2
         fjk(1:ndim)=-fik(1:ndim)

         force(1:ndim,i)=force(1:ndim,i)+fik(1:ndim)
         force(1:ndim,j)=force(1:ndim,j)+fjk(1:ndim)

      END DO
    END DO
    energy=energy*4.D0
    !print *, "energy =", energy
    force(:,:)=force(:,:)*24.d0
  END SUBROUTINE energy_forces
!---------------------------------------------------------------
  SUBROUTINE print_energy(istep)
    IMPLICIT NONE
    INTEGER :: istep
    REAL*8  :: KE, TE
    INTEGER :: ICALL=0
    SAVE    :: ICALL
!
    icall=icall+1
    IF(ICALL.EQ.1)THEN
      OPEN(11,FILE='energy.dat',STATUS='UNKNOWN')
      WRITE(*, '(5A12)')'ISTEP', 'TEMP.', 'K.E.', 'P.E.', 'T.E'
    END IF
!
    ke=1.5*dfloat(natoms)*temp
    te=ke+energy
  !
    WRITE(*,*)istep, temp, ke, energy, te
    WRITE(11,*)istep, temp, ke, energy, te
END SUBROUTINE print_energy
!------------------------------------------------------------------------

end module md
```

Module for rdf - module2.f90

```
module gr
use variables
implicit none

contains
subroutine calculate_gr(T)
integer, intent(in) :: T
integer :: maxiter, i, j, k, y, bin, nat
real*8 :: r = 0, d(3), rjk
real*8, allocatable :: dum(:), g(:), xyz(:,:,:)
character(len=2) :: comment, at

maxiter = int((L/2)/0.1d0)+1
print *, maxiter
allocate(dum(maxiter))
allocate(g(maxiter))
dum(1:maxiter) = 0.d0
allocate(xyz(T, natoms, 3))
open(15,file="traj.xyz",status='unknown')
 do i = 1,T
    read(15,*)nat
```

```
     read(15,*)comment
     do j = 1,nat
        read(15,'(a2,3f15.6)') at, xyz(i,j,1:3)
     enddo
enddo

!do z = 1, maxiter
do i = 1, T
do j = 1, natoms-1
     do k = j+1,natoms
          d(1:3) = xyz(i, k, 1:3) - xyz(i, j, 1:3)
          d(1:3) = d(1:3) - L*nint(d(1:3)/L)
          rjk = sqrt(d(1)**2 + d(2)**2 + d(3)**2)
          if(rjk <0.5*L) then
          bin = int(rjk/0.1d0) + 1
          if (bin > maxiter) exit
          dum(bin) = dum(bin) + 1.d0
          !write(11,*) bin
          endif
       end do
    end do
end do


open(3, file="hist.txt", status="unknown", form='formatted')
do bin = 1, maxiter-1
     r = DFLOAT(bin)*0.1d0
     g(bin) = real(dum(bin))/(2.d0*rho*r*r*0.1d0*pi*T*natoms)
     write(3,*)r, g(bin)
end do
close(15)
end subroutine
end module gr
```

Module for rms displacement - module3.f90

```
module rmsdisplacement
use variables
implicit none

contains
subroutine rmsd(T)
integer, intent(in) :: T
real*8, allocatable :: xyz(:,:,:)
```

```
real*8 :: rdum(3), rj, r2=0
integer :: i, j, k, nat, t0_max
character(len=2):: comment, at

t0_max = T/2
allocate(xyz(T,natoms,3))
open(13,file="traj.xyz",status='unknown')
 do i = 1,T
    read(13,*)nat
    read(13,*)comment
    do j = 1,nat
       read(13,'(a2,3f15.6)') at, xyz(i,j,1:3)
    enddo
enddo

open(17, file = "r2.txt", status = 'unknown', form = 'formatted')
    do i = 1, t0_max
        r2 = 0
        if(mod(i,20).eq.0)then
            do j = 1, natoms
                do k = 1, 15
                    rdum(1:3) = xyz(i, j, 1:3) - xyz(i+k, j, 1:3)
                    rdum(1:3) = rdum(1:3) - L*nint(rdum(1:3)/L)
                    rj = sqrt(rdum(1)**2 + rdum(2)**2 + rdum(3)**2)
                    r2 = r2 + rj
                end do
            end do
            r2 = r2/(natoms*t0_max)
            write(17,*)i/20, r2
        end if
    end do
end subroutine rmsd
end module rmsdisplacement
```