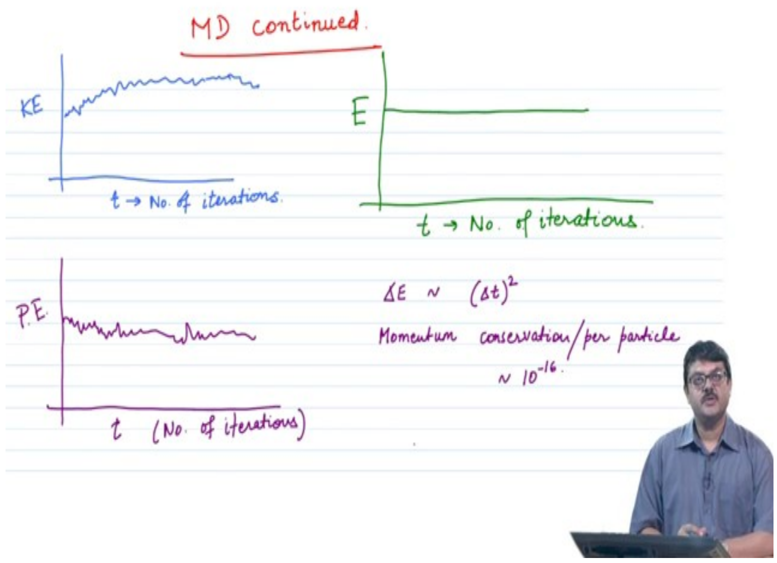
Computational Physics

Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture – 51

Molecular Dynamics Analysis
Part 01

(Refer Slide Time: 00:17)


MD continued.
KE
$t \to$ No. of iterations.
E
$t \to$ No. of iterations.
P.E.
t (No. of iterations)
$\Delta E \sim (\Delta t)^2$
Momentum conservation/per particle
$\sim 10^{-16}$.

So, welcome back. So, we shall continue discussing Molecular Dynamics. So, whatever we discussed from last class. So, last class I had asked you to basically implement a simple molecular dynamics say with 1000 particles in a 20 cross 20 cross 20 box and you have to basically initialize the positions, initialize the velocities calculate the force and keep on doing updating the position force and velocity in that order.

And you can look visualize the trajectory in space x y and z as a function of t and, but before that you have to check that your code or whatever you have implemented it is working correctly. Now, what is the simplest way or the simplest tests of checking with you that your code is working correctly, what you could do is well you have the initial position of all the particles the initial velocities and what you could monitor is the kinetic energy of the system, the potential energy of the system and the total energy of the system.

So, what do you expect? So, basically at the initial position at time t equal to 0 you have certain particles setting at certain position in space within a simulation box with periodic boundary conditions. Now, as you update position they are changing positions with respect to each other as well. If they are changing positions with respect to each other as well, then the total potential energy of that configuration is going to change.

In the next iteration when they move further after 2 delta t the position is changed again and as a consequence the total potential energy of the system would again changed right. And if there is a potential energy between the particles which is changing we have a basically system which is thermally isolated at the moment what I have told you force position whatever we have discussed up till now is essentially a system which is isolated right.

So, the total energy of the system is to remain conserved. So, if potential energy of the system is changing suppose as in here, so, I have drawn schematically the potential energy versus time or number of iterations. So, number of iterations into dt the integration constant would give you the time as. So, you can look at the position and velocity as a function of time.

And as the particles move around what you would see essentially is that the potential energy is changing, it is changing and correspondingly the kinetic energy also has to change with the number of iterations right. If this is decreasing that has to increase. If the potential energy increases due to change of positions of the particles, then the kinetic energy has to decrease to keep the total energy constant conserved energy conservation as basic as it gets right.

And so, whether the potential energy will increase or decrease will depend upon the initial position of the particles moreover it will depend upon the initial position of particles with respect to each other moreover it will depend upon the nature of the interaction density so as to say. And so those are the basic tests that you should do. I should show the data later in this lecture.

Energy will remain conserved as I said it should be constant, it should not change, but remember when we try to derive overlay algorithm I think two classes before what we did was a Taylor expansion right and we kept terms in the position and the velocity basically the position up to the second order; second order and delta t whole square we neglect it higher order terms. So, when we look at energy conservation remember that energy will be accurate up to delta t whole square and not more, we change delta t the fluctuation in energy will change right, but the momentum conservation.

So, along with energy conservation you shall also have momentum conservation right. If some particle is moving in this direction and it collides with some other particle total momentum will be conserved in the process here you have a potential instead of discrete collisions.

And momentum conservation since whatever force is acting on one particle equal and opposite force is acting on the other particle, the momentum conservation will be the per particle will be order of 10 to the power minus 16. Why 10 to the minus 16 because we have taken real star 8 variables. So, real star 8 variables double float variables assures us the accuracy of the calculation till the 16 place of decimal.

So, if you look at the total momentum conservation and suppose you have 1000 particles it will they will fluctuate all at the last digit after the decimal the 16 place. So, when you add it up to say about 1000 or 10,000 particles momentum conservation you should be able to see up to 10 to the minus 14 13, but when you do energy momentum conservation per particle it comes down to back to the accuracy of the variables, the real star 8 variables double flow two variables we want accurate changes.

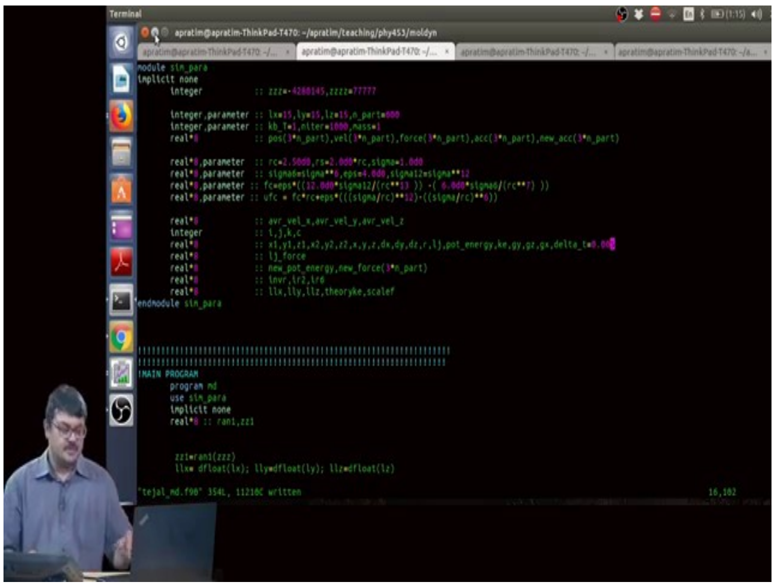
We want to follow the position and velocity up till 16 places of decimal with real star for or just real variables often errors add up and that leads to problems. That discussion if you need it you can read it up from the books, but it is outside the scope of this discussion ok.

With this background let us look at the code and the results what you expect when you have some number of particles. What I want to insist is compared to the when I discuss the Ising model where I really took you through every step of the code. Here I will my aim is to give a broad background. It was the end of the course you should be able to write your or develop your own algorithm.

So, I am going to very quickly show you the flash show you the code right without discussing things in detail because I expect you to become to come up with algorithm and implement it on your own you shall make errors there is no way to avoid errors, but you should be able to debug them.

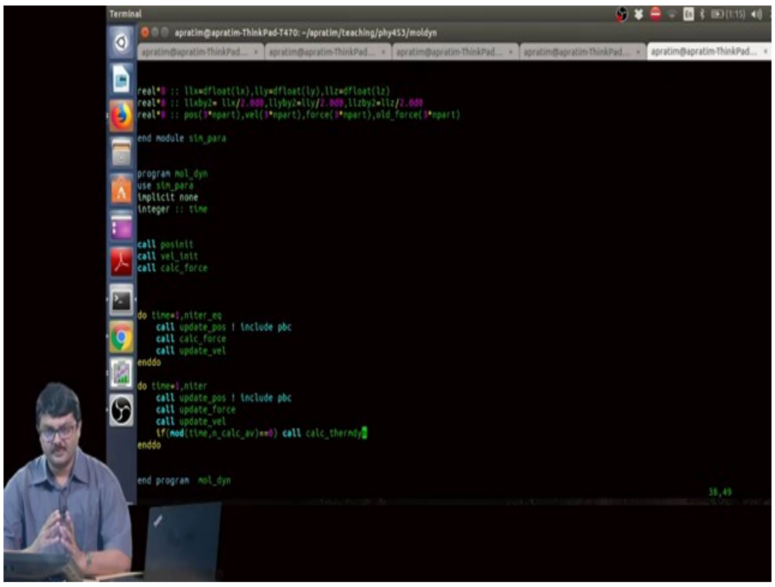
Because when you do a new problem wherever for physics or wherever you are in ISRO or BARC or does not matter where you choose your job later, you will have an unknown problem, you will have to implement your own algorithm on the computer. And debugging skills is as important as you should have be knowing by the towards the end of the course ok.

(Refer Slide Time: 08:18)


Terminal
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn

```
module sim_para
implicit none
        integer          :: zzz=-4280145,zzzz=77777

        integer,parameter :: lx=15,ly=15,lz=15,n_part=800
        integer,parameter :: kb_T=1,niter=1000,mass=1
        real*8            :: pos(3*n_part),vel(3*n_part),force(3*n_part),acc(3*n_part),new_acc(3*n_part)

        real*8,parameter  :: rc=2.50d0,rs=2.0d0*rc,sigma=1.0d0
        real*8,parameter  :: sigma6=sigma**6,eps=4.0d0,sigma12=sigma**12
        real*8,parameter :: fc=eps*((12.0d0*sigma12/(rc**13 )) -( 6.0d0*sigma6/(rc**7) ))
        real*8,parameter :: ufc = fc*rc+eps*(((sigma/rc)**12)-((sigma/rc)**6))

        real*8            :: avr_vel_x,avr_vel_y,avr_vel_z
        integer           :: i,j,k,c
        real*8            :: x1,y1,z1,x2,y2,z2,x,y,z,dx,dy,dz,r,lj,pot_energy,ke,gy,gz,gx,delta_t=0.00
        real*8            :: lj_force
        real*8            :: new_pot_energy,new_force(3*n_part)
        real*8            :: invr,ir2,ir6
        real*8            :: llx,lly,llz,theoryke,scalef
endmodule sim_para



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!MAIN PROGRAM
        program md
        use sim_para
        implicit none
        real*8 :: ran1,zz1


         zz1=ran1(zzz)
         llx= dfloat(lx); lly=dfloat(ly); llz=dfloat(lz)

"tejal_md.f90" 354L, 11210C written                                                   16,102
```

(Refer Slide Time: 08:30)


Terminal
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn

```
real*8 :: llx=dfloat(lx),lly=dfloat(ly),llz=dfloat(lz)
real*8 :: llxby2= llx/2.0d0,llyby2=lly/2.0d0,llzby2=llz/2.0d0
real*8 :: pos(3*npart),vel(3*npart),force(3*npart),old_force(3*npart)

end module sim_para


program mol_dyn
use sim_para
implicit none
integer :: time


call posinit
call vel_init
call calc_force



do time=1,niter_eq
    call update_pos ! include pbc
    call calc_force
    call update_vel
enddo

do time=1,niter
    call update_pos ! include pbc
    call update_force
    call update_vel
    if(mod(time,n_calc_av)==0) call calc_thermdy
enddo


end program  mol_dyn
                                                                                      38,49
```

With that I shall move to the laptop and what I show here is basically, so, this is just a schematic of how your code should be structured right now right at the beginning you could define the total number of particles n part you can choose it to be 1000, 600 whatever. Then the box size lx here I have chosen to be 30, 30, 30, mass equal to 1, then the box size lx here I have chosen to be 30, 30, 30.

Mass equal to 1, mass of each particle equal to 1 we shall be discussing units in that later part of today's class. Temperature equal to 1 half set temperature equal to 1 and sigma the diameter of the particle is 1 and eps the energy the Lennard Jones epsilon 4 epsilon that has been set equal to 1 basically, but you can change epsilon to 2, 3 etcetera basically what you will be doing is increasing the attractive well of the Lennard Jones potential right.

And so basically this is written in some module. So, please check up what a module is in FORTRAN you should be knowing by now and this is the main program; program mol dyn call pos, init means position initialize call velocity initialize, call calculate force that is even before the beginning of the program this should be the basic structure, then time equal to.

So, number of iterations goes from 1 to niter eq n iter n iter is n number of iterations n iter eq; eq for equilibration. So, even when you start from some initial condition you have to allow the system to equilibrate as we did in Ising model. When the system is reached equilibrium and especially if it is simple systems like we are considering then basically the kinetic energy the potential energy shall fluctuate about in average an average value. Whereas, from the beginning of the code, it could change about some to some average kinetic energy, it could change from whatever is the initial potential energy to some other value where the mean is now maintained.

So, once you reach equilibrium the mean will be maintained and the potential energy shall fluctuate about an average as the particles interchange kinetic and potential energy amongst each other right. So, this is in saying depending is upon what the density is you should set it out set out some iterations for it to equilibrate and how many iterations you want to leave aside is can be controlled by niter equilibrium suppose who gave niter equilibrium to be 1 lakh, well it really depends upon the size of the system, the density of the system and so on so forth.

And so this is the loop and you run it your md update pos calc force and update vel, well update velocity for a certain number of iterations say 1 lakh 10 to the power 5 iterations say and after you are convinced of course, you have to check whether your system has actually reached equilibrium accordingly you can change your niter eq n iter eq.

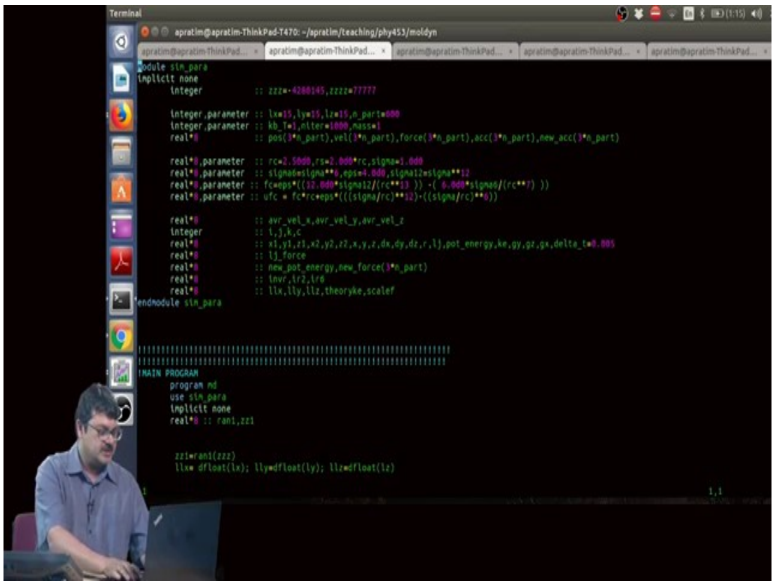
And after you are convinced that the system has reached equilibrium, you again start your md step till say n iter a new number of iterations which could be 10; 10 to the power 6 10 to the power 7 depends upon what statistical quantity you want to extract from your system and then keep on doing the md changing the phase space coordinates of the particles update position, update force and update velocity.

And every some step some 100 iterations 1000 iterations is it right it really depends upon dt the value of dt as well. You can call subroutine which will calculate all calc thermdyn means calculate the thermodynamic quantities. You do not need to call this calc thermdyn the subroutine which will calculate an average the thermodynamic quantities each iteration because in one iteration as I said before the particles are going to move relatively little right.

So, they are just going to move small amount, we have kbt equal to 1 your velocities or speeds will be around 1 and if your dt is 0.005 right, then basically in one iteration the average displacement of each particle will be 0.005; 0.005 times the diameter.

So, basically the particles have not really moved much and so it does not make sense to calculate statistical quantities each iteration. So, depending upon the value of dt that you choose you would call it every 100 or 1000 iterations so that you have a statistically independent microstate of the system and then you take your quantities of interest and average over it right. Now, looking at it in a slightly greater detail, ok.

(Refer Slide Time: 14:37)

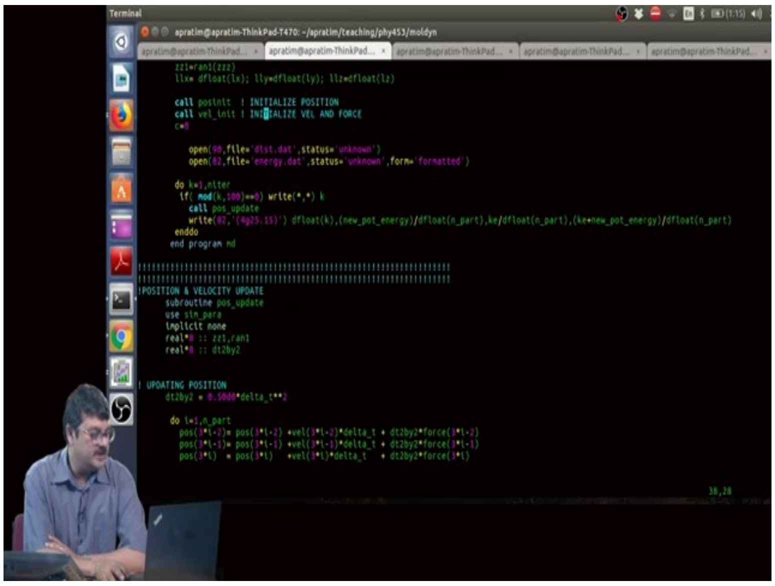


```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn
module sim_para
implicit none
        integer              :: zzz=-4280145,zzzz=77777

        integer,parameter :: lx=15,ly=15,lz=15,n_part=600
        integer,parameter :: kb_T=1,niter=1000,mass=1
        real*8               :: pos(3*n_part),vel(3*n_part),force(3*n_part),acc(3*n_part),new_acc(3*n_part)

        real*8,parameter  :: rc=2.50d0,rs=2.0d0*rc,sigma=1.0d0
        real*8,parameter  :: sigma6=sigma**6,eps=4.0d0,sigma12=sigma**12
        real*8,parameter :: fc=eps*((12.0d0*sigma12/(rc**13 )) -( 6.0d0*sigma6/(rc**7) ))
        real*8,parameter :: ufc = fc*rc+eps*(((sigma/rc)**12)-((sigma/rc)**6))

        real*8               :: avr_vel_x,avr_vel_y,avr_vel_z
        integer              :: i,j,k,c
        real*8               :: x1,y1,z1,x2,y2,z2,x,y,z,dx,dy,dz,r,lj,pot_energy,ke,gy,gz,gx,delta_t=0.005
        real*8               :: lj_force
        real*8               :: new_pot_energy,new_force(3*n_part)
        real*8               :: invr,ir2,ir6
        real*8               :: llx,lly,llz,theoryke,scalef
endmodule sim_para


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!MAIN PROGRAM
        program md
        use sim_para
        implicit none
        real*8 :: ran1,zz1


         zz1=ran1(zzz)
         llx= dfloat(lx); lly=dfloat(ly); llz=dfloat(lz)

1                                                                      1,1
```

So, the code which, so that was just for schematics, the code which for which I should show you data there the box size is 15, the number of particles is just 600. I would like you to notice that rc the cut off of the force here has been set to 2.530, 2.5 sigma; sigma equal to 1 has been chosen as before and then sigma has been chosen to be 1 as here then I this sigma to the power 6 the Lennard Jones sigma to the power 6 and sigma to the power 12 that does not have to be calculated every time in the four subroutine.

So, I have just calculated it right in the beginning sigma here is sigma 6, here is sigma 12, then I calculate fc, when you cut off the force you need the value of fc this is ufc basically the so, from cut off of the force you calculate v dash the modified potential from that you need the modified value of vc what I called vc in the lectures here I have called it ufc which is basically the modified potential as it goes to 0 how much is the difference the value of the potential at the cut off that again has been calculated just once right at the beginning of the course, beginning of the code right and after that you enter the main program. So, from in the main program I have essentially basically called sim para ok.

(Refer Slide Time: 16:15)

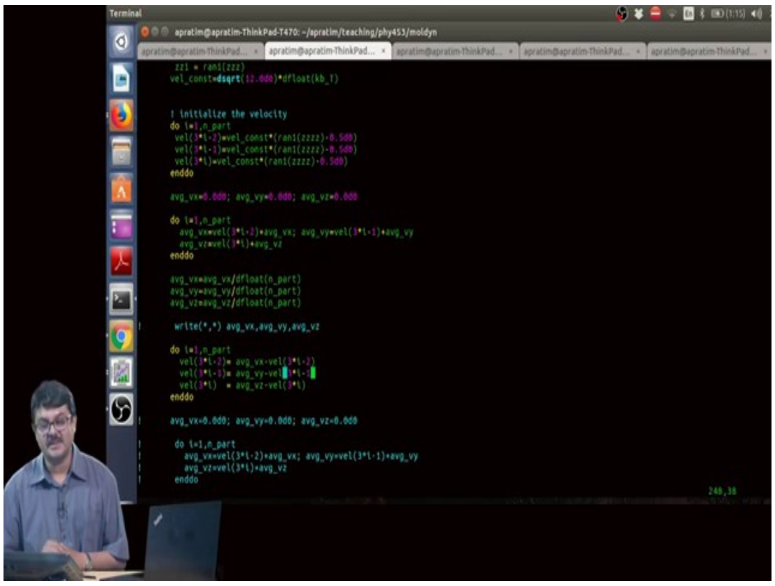


```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn
        zz1=ran1(zzz)
        llx= dfloat(lx); lly=dfloat(ly); llz=dfloat(lz)

        call posinit  ! INITIALIZE POSITION
        call vel_init ! INITIALIZE VEL AND FORCE
        c=0

          open(90,file='dist.dat',status='unknown')
          open(82,file='energy.dat',status='unknown',form='formatted')

        do k=1,niter
         if( mod(k,100)==0) write(*,*) k
          call pos_update
          write(82,'(4g25.15)') dfloat(k),(new_pot_energy)/dfloat(n_part),ke/dfloat(n_part),(ke+new_pot_energy)/dfloat(n_part)
        enddo
       end program md

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!POSITION & VELOCITY UPDATE
      subroutine pos_update
      use sim_para
      implicit none
      real*8 :: zz1,ran1
      real*8 :: dt2by2


! UPDATING POSITION
      dt2by2 = 0.50d0*delta_t**2

       do l=1,n_part
         pos(3*l-2)= pos(3*l-2) +vel(3*l-2)*delta_t + dt2by2*force(3*l-2)
         pos(3*l-1)= pos(3*l-1) +vel(3*l-1)*delta_t + dt2by2*force(3*l-1)
         pos(3*l)  = pos(3*l)   +vel(3*l)*delta_t   + dt2by2*force(3*l)

                                                                                                    38,28
```

I have called sim para and then what am I doing? I am introducing initializing position into initializing velocity within velocity I am also introducing calculating the force in the same subroutine right. But ideally you should do it separately in the way I showed you in the schematics I showed you just before this right. You can introduce particles on and lattice. Here I have introduced, so that two power I have introduced the particles randomly in such a manner so that they do not sit on top of each other right.

You ensure it and so that is what I have done here, but it really does not matter you can just do it the lattice; anyway each particle will have a different velocity after. So, let me show you a bit of vel init.

(Refer Slide Time: 17:25)



```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn

  zz1 = ran1(zzz)
 vel_const=dsqrt(12.0d0)*dfloat(kb_T)


 ! initialize the velocity
 do i=1,n_part
  vel(3*i-2)=vel_const*(ran1(zzzz)-0.5d0)
  vel(3*i-1)=vel_const*(ran1(zzzz)-0.5d0)
  vel(3*i)=vel_const*(ran1(zzzz)-0.5d0)
 enddo

 avg_vx=0.0d0; avg_vy=0.0d0; avg_vz=0.0d0

 do i=1,n_part
   avg_vx=vel(3*i-2)+avg_vx; avg_vy=vel(3*i-1)+avg_vy
   avg_vz=vel(3*i)+avg_vz
 enddo

 avg_vx=avg_vx/dfloat(n_part)
 avg_vy=avg_vy/dfloat(n_part)
 avg_vz=avg_vz/dfloat(n_part)

!       write(*,*) avg_vx,avg_vy,avg_vz

 do i=1,n_part
   vel(3*i-2)= avg_vx-vel(3*i-2)
   vel(3*i-1)= avg_vy-vel(3*i-1)
   vel(3*i)  = avg_vz-vel(3*i)
 enddo

!      avg_vx=0.0d0; avg_vy=0.0d0; avg_vz=0.0d0

!       do i=1,n_part
!         avg_vx=vel(3*i-2)+avg_vx; avg_vy=vel(3*i-1)+avg_vy
!         avg_vz=vel(3*i)+avg_vz
!       enddo
```

So, here I have used the random number minus 0.5, so that the vel vx say so, this is vel 2 i minus 2 is for the ith particle this is the x component of the ith particle it can be in the positive direction or in the negative direction ran 1 gives me a random number from 0 to 1. When I subtract minus 0.5 I get a random number from minus 0.5 to plus 0.5 any number between minus 0.5 and plus 0.5 and I have multiplied it by some vel constant appropriate amplitude.

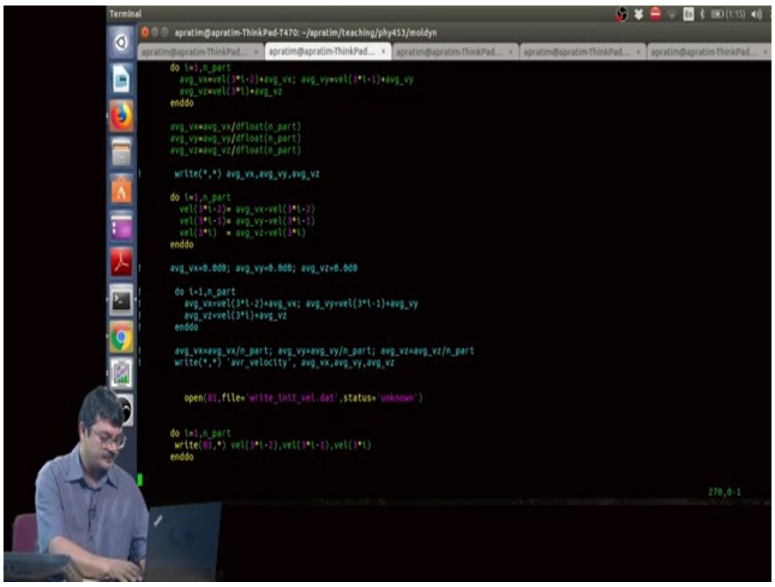
So, that says that 1 by root 12 factor. So, that the average initial kinetic energy remains close to 1 kbt; the kinetic energy the temperature is maintained right. So, this is what I have done for the y component this is what I have done for the z component, but if you give a random velocities to say 600 particles or in our case we are giving 600 particles we have 600 particles in a 15 cube box.

Then when you sum it up the total velocity of all the particles in each direction what you are going to have is the total velocity or the total momentum actually, the total velocity and multiply it by mass momentum is not going to be 0, it is going to be some number close to 0, but not exactly 0.

Which means that the centre of mass. So, the box will have a centre of mass velocity right. Now, you want to study a system if it is in equilibrium in a frame in an initial frame where the centre of mass velocity is 0. To that end what we do is calculate, so we define some quantity called average vx, the average value of the vx, average value of the vy and average value of the vz which we set to 0. Go to each particle add the velocity of each particle the x y and z component to average vx, vy and v z. So, that once you have completed the loop you know what is the total velocity of all the particles average value in the x direction, y direction and z direction divided by the total number of particles so that you know what is the average excess momentum per particle in the box in the x, y and z direction.

Now, if we subtract this average vx from each particle velocity right, the average momentum per particle from each velocity some are going to the x some are going to the y does not matter, when you subtract that out, then essentially you have a system where the total momentum of all the particles in the box is exactly equal to 0, when I say exactly equal to 0 it means accuracy of 10 to the power minus 16.

(Refer Slide Time: 21:04)

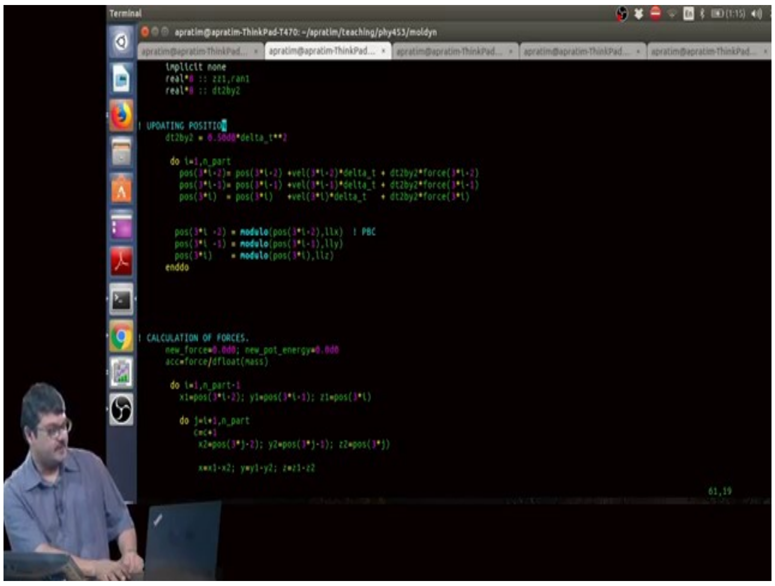

Terminal
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn

```
      do i=1,n_part
        avg_vx=vel(3*i-2)+avg_vx; avg_vy=vel(3*i-1)+avg_vy
        avg_vz=vel(3*i)+avg_vz
      enddo

      avg_vx=avg_vx/dfloat(n_part)
      avg_vy=avg_vy/dfloat(n_part)
      avg_vz=avg_vz/dfloat(n_part)

!      write(*,*) avg_vx,avg_vy,avg_vz

      do i=1,n_part
        vel(3*i-2)= avg_vx-vel(3*i-2)
        vel(3*i-1)= avg_vy-vel(3*i-1)
        vel(3*i)  = avg_vz-vel(3*i)
      enddo

!     avg_vx=0.0d0; avg_vy=0.0d0; avg_vz=0.0d0

!      do i=1,n_part
!        avg_vx=vel(3*i-2)+avg_vx; avg_vy=vel(3*i-1)+avg_vy
!        avg_vz=vel(3*i)+avg_vz
!      enddo

!      avg_vx=avg_vx/n_part; avg_vy=avg_vy/n_part; avg_vz=avg_vz/n_part
!      write(*,*) 'avr_velocity', avg_vx,avg_vy,avg_vz


        open(81,file='write_init_vel.dat',status='unknown')


      do i=1,n_part
       write(81,*) vel(3*i-2),vel(3*i-1),vel(3*i)
      enddo

                                                              270,0-1
```

So, what you have done is set the total momentum of the box to a number close to 0 to exactly 0 within the computer accuracy 10 to the power minus 16. So, you have set the momentum equal to 0, right. So, that is all that you have done. And then what else has to be done? Well after that you basically start the simulation.

So, everything is being done, so do k equal to one to n iter and call pos update actually this subroutine does both the position update force update and velocity update I will flash you that.

(Refer Slide Time: 21:54)

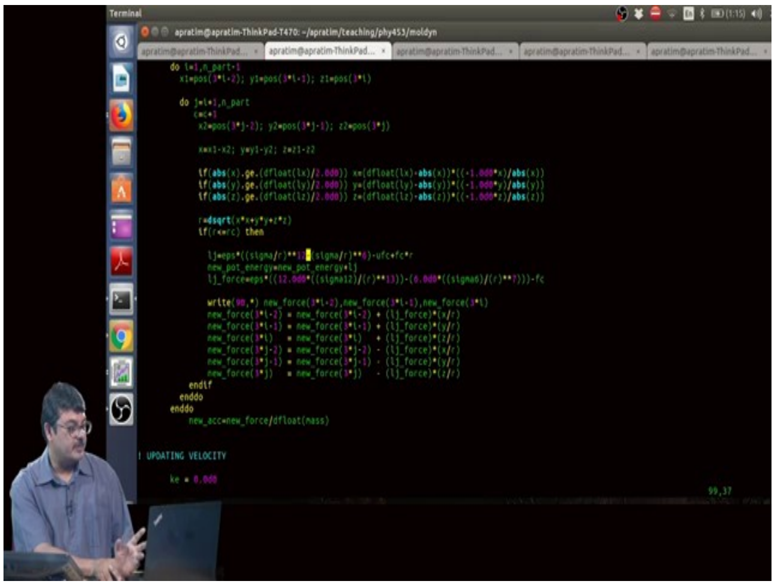


```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn
      implicit none
      real*8 :: zz1,ran1
      real*8 :: dt2by2


! UPDATING POSITION
      dt2by2 = 0.50d0*delta_t**2

       do i=1,n_part
         pos(3*i-2)= pos(3*i-2) +vel(3*i-2)*delta_t + dt2by2*force(3*i-2)
         pos(3*i-1)= pos(3*i-1) +vel(3*i-1)*delta_t + dt2by2*force(3*i-1)
         pos(3*i)  = pos(3*i)   +vel(3*i)*delta_t   + dt2by2*force(3*i)


        pos(3*i -2) = modulo(pos(3*i-2),llx)  ! PBC
        pos(3*i -1) = modulo(pos(3*i-1),lly)
        pos(3*i)    = modulo(pos(3*i),llz)
      enddo




! CALCULATION OF FORCES.
      new_force=0.0d0; new_pot_energy=0.0d0
      acc=force/dfloat(mass)

       do i=1,n_part-1
         x1=pos(3*i-2); y1=pos(3*i-1); z1=pos(3*i)

         do j=i+1,n_part
            c=c+1
             x2=pos(3*j-2); y2=pos(3*j-1); z2=pos(3*j))

             x=x1-x2; y=y1-y2; z=z1-z2
```

So, as you see first its updating position do i equal to 1 to n part; n part number of particles and this is exactly the formula which I had shown you pos plus vel into delta t delta t is the dt, the time over which you are integrating which we have chosen to be 0.005 in our simulation and then dt square by 2 basically dt square by 2 into force; force it ideally should be acceleration, but since mass is chosen to be 1 force is equal to acceleration right. Now, I would like you to note this part that dt square by 2 instead of calculating it in each iteration I have defined a certain variable before the beginning of the loop which is 0.5 into delta t whole square so that you save computing time in this step right.

This is PBC you can do it also the way we discussed, but this simple modulo operation pos x modulo pos x minus ll x modulo pos position in the y direction lly and modulo pos in the z direction dot llz also works; try it out what does modulo do figure it out. If you have problems with that you can always do what I discussed in one of the lectures.

(Refer Slide Time: 23:27)



```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn

      do i=1,n_part-1
        x1=pos(3*i-2); y1=pos(3*i-1); z1=pos(3*i)

        do j=i+1,n_part
          c=c+1
           x2=pos(3*j-2); y2=pos(3*j-1); z2=pos(3*j)

           x=x1-x2; y=y1-y2; z=z1-z2

           if(abs(x).ge.(dfloat(lx)/2.0d0)) x=(dfloat(lx)-abs(x))*((-1.0d0*x)/abs(x))
           if(abs(y).ge.(dfloat(ly)/2.0d0)) y=(dfloat(ly)-abs(y))*((-1.0d0*y)/abs(y))
           if(abs(z).ge.(dfloat(lz)/2.0d0)) z=(dfloat(lz)-abs(z))*((-1.0d0*z)/abs(z))

           r=dsqrt(x*x+y*y+z*z)
           if(r<=rc) then

             lj=eps*((sigma/r)**12-(sigma/r)**6)-ufc+fc*r
             new_pot_energy=new_pot_energy+lj
             lj_force=eps*((12.0d0*((sigma12)/(r)**13))-(6.0d0*((sigma6)/(r)**7)))-fc

             write(90,*) new_force(3*i-2),new_force(3*i-1),new_force(3*i)
             new_force(3*i-2) = new_force(3*i-2) + (lj_force)*(x/r)
             new_force(3*i-1) = new_force(3*i-1) + (lj_force)*(y/r)
             new_force(3*i)   = new_force(3*i)   + (lj_force)*(z/r)
             new_force(3*j-2) = new_force(3*j-2) - (lj_force)*(x/r)
             new_force(3*j-1) = new_force(3*j-1) - (lj_force)*(y/r)
             new_force(3*j)   = new_force(3*j)   - (lj_force)*(z/r)
         endif
       enddo
     enddo
         new_acc=new_force/dfloat(mass)

! UPDATING VELOCITY

     ke = 0.0d0
                                                                      99,37
```

This is nothing but calculation of forces. You are going from 1 to number of particles minus 1 because you want to consider every pair of particles you do not want to over count j equal to i plus 1 means if i equal to 1, then j will be 2 3 4 5 6 7 8 so on up till the number of particles 600 in our case.

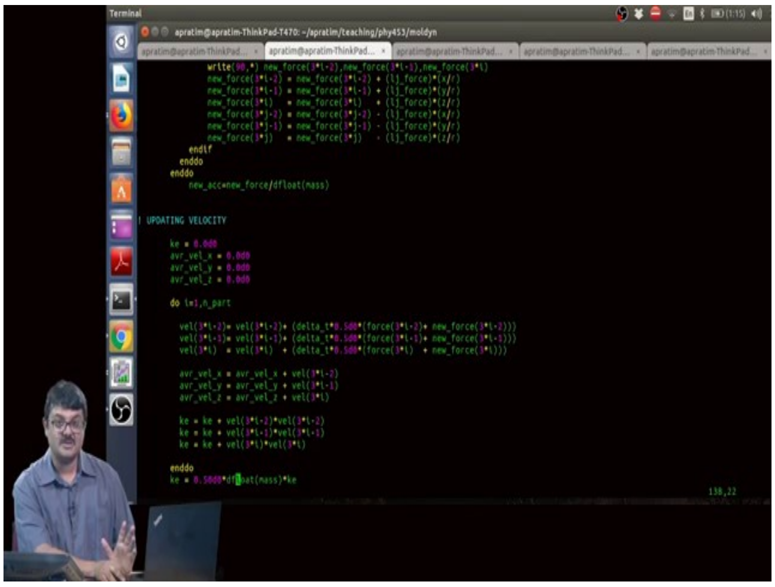
So, basically by this we are taking every pair of particles. You are storing the position of your ith particle in x 1, y 1 and z 1, the x y and z coordinate of the jth particle force ij between i and j in x 2, y 2 and z 2 in this step and then you calculate essentially basically the difference in position in x y and z. Apply minimum image convention as we discussed exactly that is what is done here.

Then you calculate the force here, the potential energy here, the Lennard Jones equal to eps I have multiplied it by 4 in all probability right at the beginning of the code I have 2. And sigma by r to the power 12 minus sigma by r to the power 6 minus u fc plus fc into r which has been calculated right at the beginning just once because if once you set your r cut you are done you know the value of u fc and fc.

And here you calculate the Lennard Jones force where you have 12 into sigma to the power 12 sigma to the power 6 by r and so on and so forth minus fc, then you take the appropriate component in the x direction the y direction and the z direction. The force acting on i and j right have the opposite sign plus plus plus for x y z on x y z component of the ith particle minus minus minus component on the force on the jth particle.

Moreover any one particle can be acted upon by many by force emanating from many particles more than 1 right. So, basically we have to sum up the forces in each direction acting on from different particles. Which particles have actually giving the force; which particles are actually giving the force? Only those particles where the distance between these two particles which is saved in r; r is x square plus y square plus z square. That thing is less than r c r cut off right only then is some force expression being calculated else you do not need to calculate the force because else the force is 0 right.

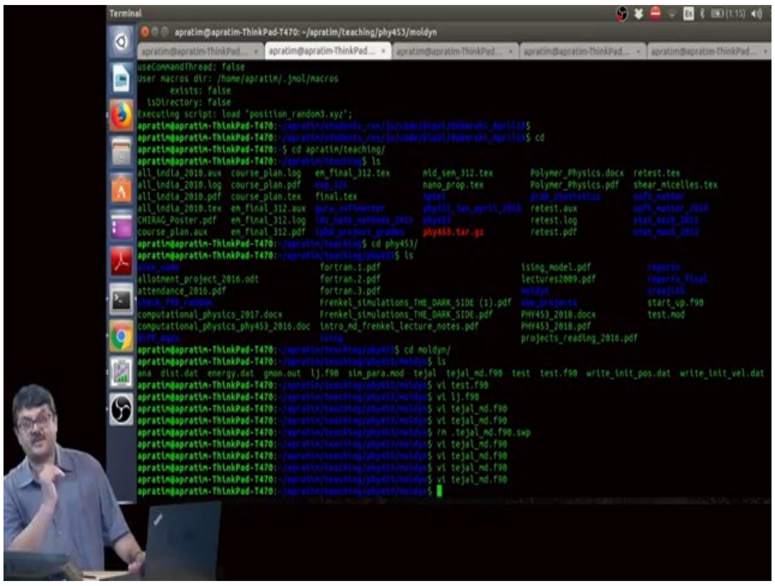
(Refer Slide Time: 29:41)


Terminal
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn
apratim@apratim-ThinkPad...
apratim@apratim-ThinkPad...
apratim@apratim-ThinkPad...
apratim@apratim-ThinkPad...
apratim@apratim-ThinkPad...

```
              write(90,*) new_force(3*l-2),new_force(3*l-1),new_force(3*l)
              new_force(3*l-2) = new_force(3*l-2) + (lj_force)*(x/r)
              new_force(3*l-1) = new_force(3*l-1) + (lj_force)*(y/r)
              new_force(3*l)   = new_force(3*l)   + (lj_force)*(z/r)
              new_force(3*j-2) = new_force(3*j-2) - (lj_force)*(x/r)
              new_force(3*j-1) = new_force(3*j-1) - (lj_force)*(y/r)
              new_force(3*j)   = new_force(3*j)   - (lj_force)*(z/r)
          endif
        enddo
      enddo
          new_acc=new_force/dfloat(mass)


! UPDATING VELOCITY

      ke = 0.0d0
      avr_vel_x = 0.0d0
      avr_vel_y = 0.0d0
      avr_vel_z = 0.0d0

      do i=1,n_part

        vel(3*l-2)= vel(3*l-2)+ (delta_t*0.5d0*(force(3*l-2)+ new_force(3*l-2)))
        vel(3*l-1)= vel(3*l-1)+ (delta_t*0.5d0*(force(3*l-1)+ new_force(3*l-1)))
        vel(3*l)  = vel(3*l)  + (delta_t*0.5d0*(force(3*l)  + new_force(3*l)))

        avr_vel_x = avr_vel_x + vel(3*l-2)
        avr_vel_y = avr_vel_y + vel(3*l-1)
        avr_vel_z = avr_vel_z + vel(3*l)

        ke = ke + vel(3*l-2)*vel(3*l-2)
        ke = ke + vel(3*l-1)*vel(3*l-1)
        ke = ke + vel(3*l)*vel(3*l)

      enddo
      ke = 0.5d0*dfloat(mass)*ke
                                                                      138,22
```

And, then you update the velocity as was discussed ok. Now, this is a quick snapshot I have not explained every step intentionally so that you your own algorithm implement it in your way.

(Refer Slide Time: 26:56)



```
apratim@apratim-ThinkPad-T470: ~/apratim/teaching/phy453/moldyn
useCommandThread: false
User macros dir: /home/apratim/.jmol/macros
       exists: false
  isDirectory: false
Executing script: load "position_random3.xyz";
apratim@apratim-ThinkPad-T470:[illegible]$
apratim@apratim-ThinkPad-T470:[illegible]$ cd
apratim@apratim-ThinkPad-T470:~$ cd apratim/teaching/
apratim@apratim-ThinkPad-T470:~/apratim/teaching$ ls
all_india_2010.aux  course_plan.log   em_final_312.tex      mid_sem_312.tex         Polymer_Physics.docx  retest.tex
all_india_2010.log  course_plan.pdf   [illegible]           nano_prop.tex           Polymer_Physics.pdf   shear_micelles.tex
all_india_2010.pdf  course_plan.tex   final.tex             [illegible]             [illegible]           soft_matter
all_india_2010.tex  em_final_312.aux  [illegible]           [illegible]             retest.aux            soft_matter_2018
CHIRAG_Poster.pdf   em_final_312.log  [illegible]           [illegible]             retest.log            [illegible]
course_plan.aux     em_final_312.pdf  [illegible]           phy453.tar.gz           retest.pdf            [illegible]
apratim@apratim-ThinkPad-T470:~/apratim/teaching$ cd phy453/
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453$ ls
[illegible]                            fortran.1.pdf                                  ising_model.pdf           [illegible]
allotment_project_2016.odt             fortran.2.pdf                                  lectures2009.pdf          [illegible]
attendance_2016.pdf                    fortran.3.pdf                                  moldyn                    [illegible]
[illegible]                            Frenkel_simulations_THE_DARK_SIDE (1).pdf      [illegible]               start_up.f90
computational_physics_2017.docx        Frenkel_simulations_THE_DARK_SIDE.pdf          PHY453_2018.docx          test.mod
computational_physics_phy453_2016.doc  intro_md_frenkel_lecture_notes.pdf             PHY453_2018.pdf
[illegible]                            ising                                          projects_reading_2016.pdf
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453$ cd moldyn/
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ ls
ana  dist.dat  energy.dat  gmon.out  lj.f90  sim_para.mod  tejal  tejal_md.f90  test  test.f90  write_init_pos.dat  write_init_vel.dat
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi test.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi lj.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ rm .tejal_md.f90.swp
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$ vi tejal_md.f90
apratim@apratim-ThinkPad-T470:~/apratim/teaching/phy453/moldyn$
```

Now, we have calculated in the update position once we have updated the position, we can also calculate kinetic energy half mv square sum, sum over all particles right. We can calculate the kinetic energy of all the particles, we can also calculate kinetic energy per particle just I have a total kinetic energy divided by the total number of particles. Moreover when you are calculating force, then for every pair of particles you can not only calculate force you are also calculating the potential energy between a pair of particles.

Now, you can calculate the potential energy of the system sby summing up this lj energy lj energy of the Lennard Jones interaction for every pair of particles. So, you need some variable suppose total energy and every time for a pair of particles you calculate the energy you add it to the total energy you keep adding the energy of every pair to this total energy.

And at the end what you can do is divide by the total number of particles and a plot and store the total energy potential energy per particle, the total kinetic energy per particle and the total energy per particle, multiply it by constant and you will get it for the total energy of the system as a function of time right.

So, the first column you can write it down as iterations; iteration 1 2 3 4 5 6 7 8, the second column could store the potential energy of the entire system or potential energy per particle in iteration 1 2 3 4 5 6 7 8. Next column in the file could be the total kinetic energy in iteration 1 2 3 4 5 6 and so on so forth and the last column the fourth column could be the total energy of the system as you go on in time.