# HW7 Molecular Dynamics

Fei Ding

December 21, 2009

(i) The total energy $E$ was calculated as $E = PE + KE$, where

$$KE = \sum_{i=1}^{N} \frac{1}{2}(v_{i,x}^2 + v_{i,y}^2),$$

$$PE = \sum_{i<j} 4\left[\left(\frac{1}{r_{i,j}}\right)^{12} - \left(\frac{1}{r_{i,j}}\right)^{6}\right]$$

The plot of total energy vs. time was shown in figure 1, the energy oscillates about an constant average value. And the energy is relatively low because of the low temperature start.

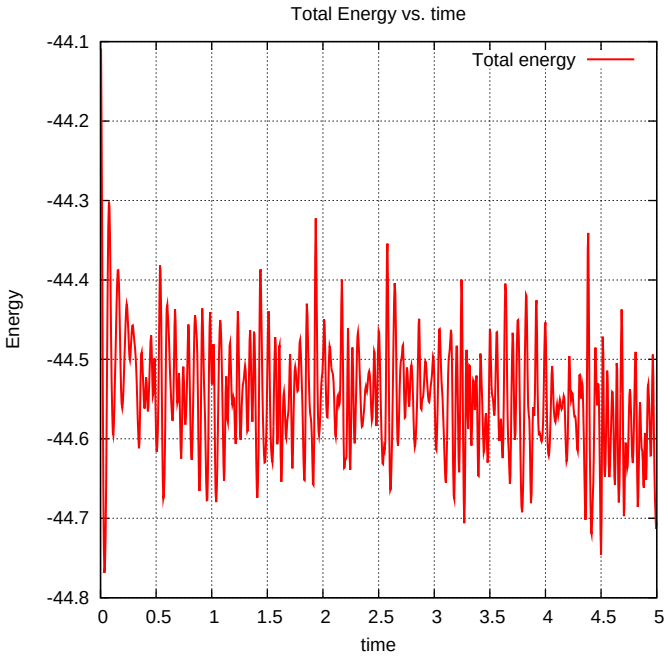


Figure 1: total energy vs. time

(ii) The CM velocity $v_{CM}$ was calculated as

$$\vec{v}_{CM} = \frac{1}{N}\sum_{i=1}^{N} \vec{v}_i$$

$v_{CM,x}$, $v_{CM,y}$ and $v_{CM}$ are all shown in the plot of CM velocity vs. time, figure 2. The CM velocity is around the order of $10^{-13}$, which is very close to zero.

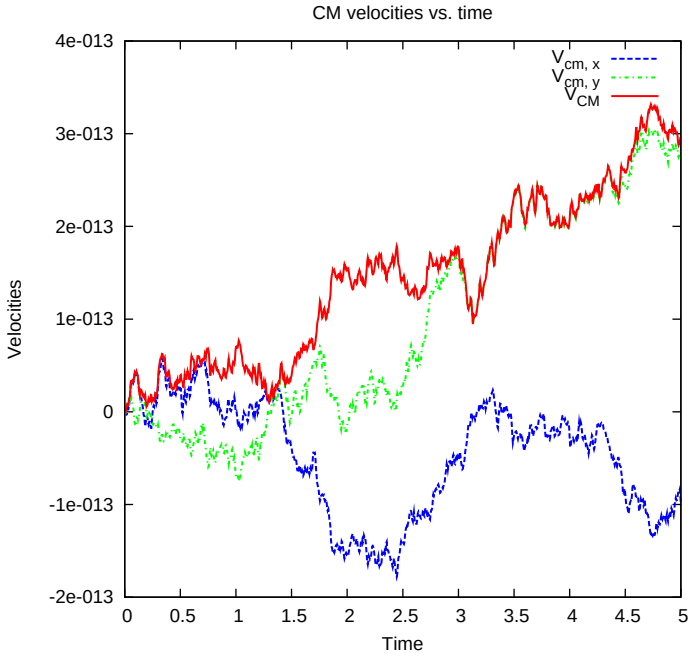


Figure 2: CM velocity vs. time

(iii) The temperature T was calculated using 2D equipartition theorem,

$$kT = \frac{1}{2} m \left\langle (v_x^2 + v_y^2) \right\rangle$$

Temperature vs. time plot was shown on figure 3, the mean value after 400 steps was 1.197.

(iv) For the $v_x$, I picked up the data at the last step. The plot roughly looks like Gaussian distribution, but not really. Although if I picked up data at different time points would yield better results, I think it does not make sense to get distribution for $v_x$ using data from different time points. The distribution and the fit was shown in figure 4.

$v_x$ obeys the Maxwell distribution

$$P(v_x) = \frac{1}{\sqrt{2\pi k_B T}} \exp\left(\frac{-v_x^2}{2k_B T}\right)$$

where $\frac{1}{\sqrt{2\pi}}$ is just a constant, let gnuplot fit $k_B T$, got 1.207, which agrees the result I got in part (iii)

(v) The pair correlation function $g(r)/r$ vs. r is shown on figure 5. Assuming triangle lattice, the direction vector from one particle to another can only be the linear combination of the following three vectors,

$$\begin{aligned} \vec{a_1} &= a\,(1,0) \\ \vec{a_2} &= a\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right) \\ \vec{a_3} &= a\left(0, \sqrt{3}\right), \end{aligned} \tag{1}$$

$a$ is the lattice size. Notice that, $\vec{a_2} = \frac{1}{2}\left(\vec{a_1} + \vec{a_3}\right)$ We can write direction vector as,

$$\begin{aligned} \vec{R} &= n_1\vec{a_1} + n_2\vec{a_2} + n_3\vec{a_3} \\ &= \left(n_1 + \frac{1}{2}n_2\right)\vec{a_1} + \left(n_3 + \frac{1}{2}n_2\right)\vec{a_3}. \end{aligned} \tag{2}$$

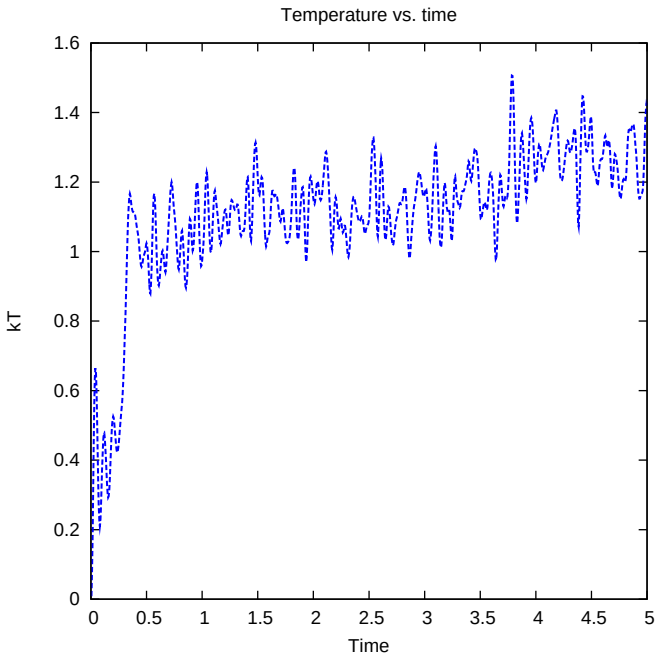


Figure 3: temperature vs. time

where, $n1$,$n2$,$n3$=1,2,3$\cdots$ So

$$r = |\vec{R}| = a\sqrt{\left(n_1 + \frac{1}{2}n_2\right)^2 + \left[\left(n_3 + \frac{1}{2}n_2\right)\sqrt{3}\right]^2}$$

The arrows on the plot were gained by set a=1, they are are 1, 1.732, 2, 2.646, 3, 3.464, 3.606, 4, 4.359, 4.583, 5, 5.196, 5.292, 5.568, 6, 6.083, 6.245, 6.558, and 7. But actually I don't know what exactly a is. So the positions of the arrows and the peaks of the $g(r)/r$ are off by a factor.

(vi) For the high temperature start, I let the directions of velocity to be random, calculate the CM velocity then subtract it from each particles velocity. The $g(r)/r$ vs. $r$ plot was shown on figure 6. For high temperature, the equilibrium state should be random distribution. So the plot is basically a flat line. Because we use periodic boundary condition, the longest distance between two particles are $5\sqrt{2}$, and much less particle since $r = 5$, so it can be seen in the plot beyond about 5, the correlation function decayed rapidly.

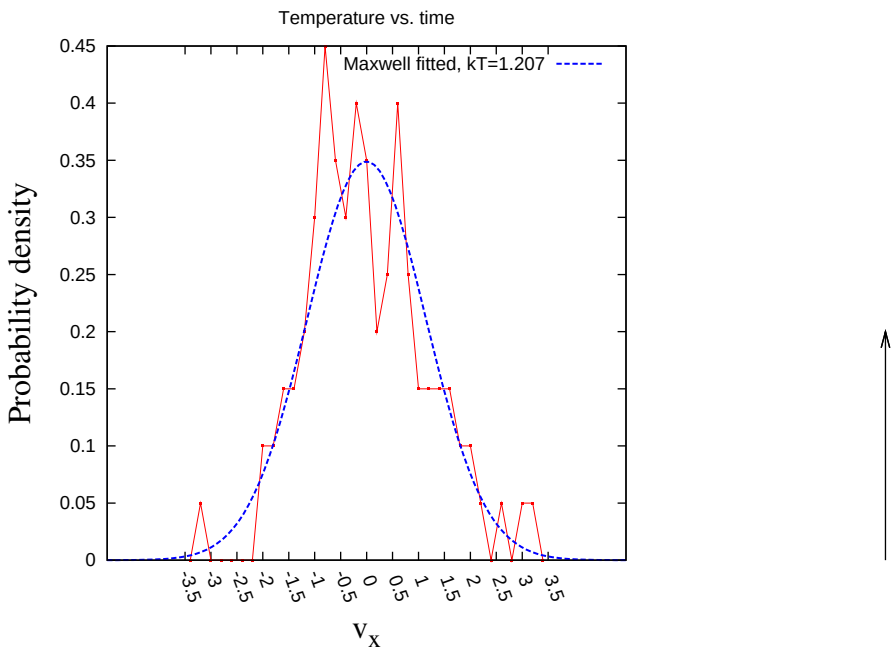


Figure 4: $v_x$ vs. time

# A Codes

```
PROGRAM main
  IMPLICIT NONE
  INTEGER :: seed
  DOUBLEPRECISION :: RAND
  INTEGER, PARAMETER :: N = 100
  INTEGER, PARAMETER :: maxT = 1000
  INTEGER :: step
  DOUBLEPRECISION, PARAMETER :: deltaT = 0.005D0
  INTEGER :: i,z
  DOUBLEPRECISION :: L = 10D0
  DOUBLEPRECISION, DIMENSION(N) :: xnew,xcurr,xprev !x-component of position of the i_th particle
  DOUBLEPRECISION, DIMENSION(N) :: ynew,ycurr,yprev !y-component
  DOUBLEPRECISION, DIMENSION(N) :: vx,vy !x,y-component of velocity of the i_th particle
  DOUBLEPRECISION :: v0 = 0D0
  DOUBLEPRECISION:: ax,ay !x,y-component of acceleration of the i_th particle
  DOUBLEPRECISION, DIMENSION(N,N) :: r,forcex, forcey
  DOUBLEPRECISION :: PE, KE,E  !energy
  DOUBLEPRECISION :: Vcmx,Vcmy !x,y CM velo
  DOUBLEPRECISION :: KT !Temperature
  double precision::deltaR=0.05d0


  seed = 918172
  CALL SRAND(seed)


!set initial position and velocities
  DO i = 1, N
     xcurr(i) =  0.5d0 +dble(mod(i-1,10)) + deltar*(2D0*RAND()-1D0)
     ycurr(i) =  0.5d0 +dble(INT(i-0.01)/10) + deltar*(2D0*RAND()-1D0)
```

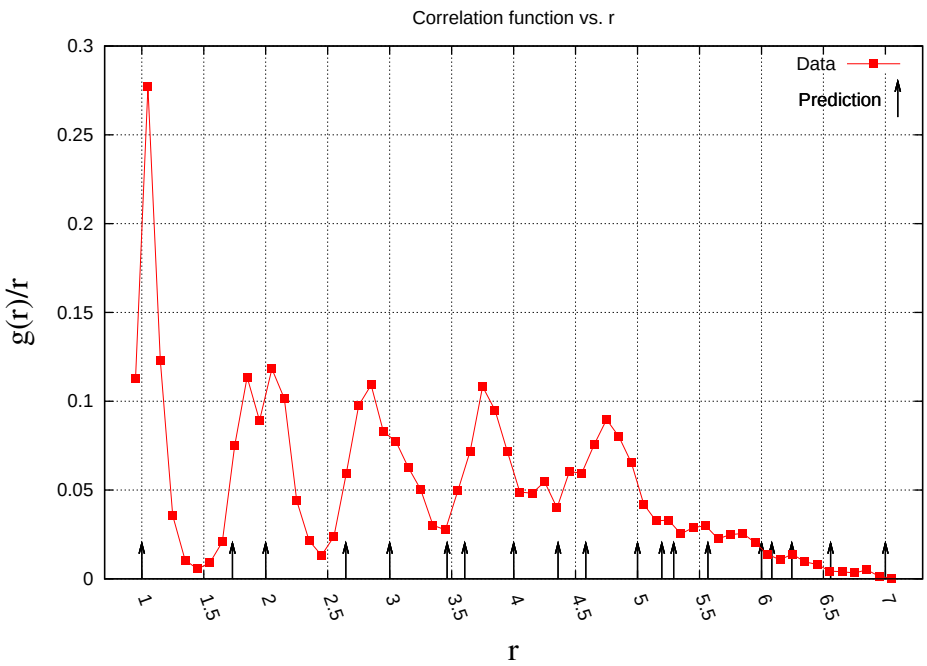


Figure 5: $g(r)/r$ vs. $r$ of Low Temperature

```
 vx(i) = v0
 vy(i) = v0
 WRITE(11,*) i, xcurr(i), ycurr(i)
  ENDDO

!Euler to get xprev
 DO i = 1, N
   xprev(i) = xcurr(i) - vx(i)*deltaT
   yprev(i) = ycurr(i) - vy(i)*deltaT
 ENDDO

DO step = 1, maxT

CALL force !calculate force before updating

DO i = 1, N
  CALL accel(i,ax,ay)
  xnew(i) = 2D0*xcurr(i) - xprev(i) + ax*deltaT**2
      IF ( xnew(i) .GT. L ) THEN
    xnew(i) = xnew(i) - L
        xprev(i)= xprev(i)-L
xcurr(i)=xcurr(i)-L
  ELSEIF ( xnew(i) .LT. 0D0 ) THEN
    xnew(i) = xnew(i) + L
xprev(i)= xprev(i)+L
xcurr(i)=xcurr(i)+L
 ! ELSE
 !   xnew(i) = xnew(i)
! xprev(i)= xprev(i)
      ENDIF

  ynew(i) = 2D0*ycurr(i) - yprev(i) + ay*deltaT**2
```

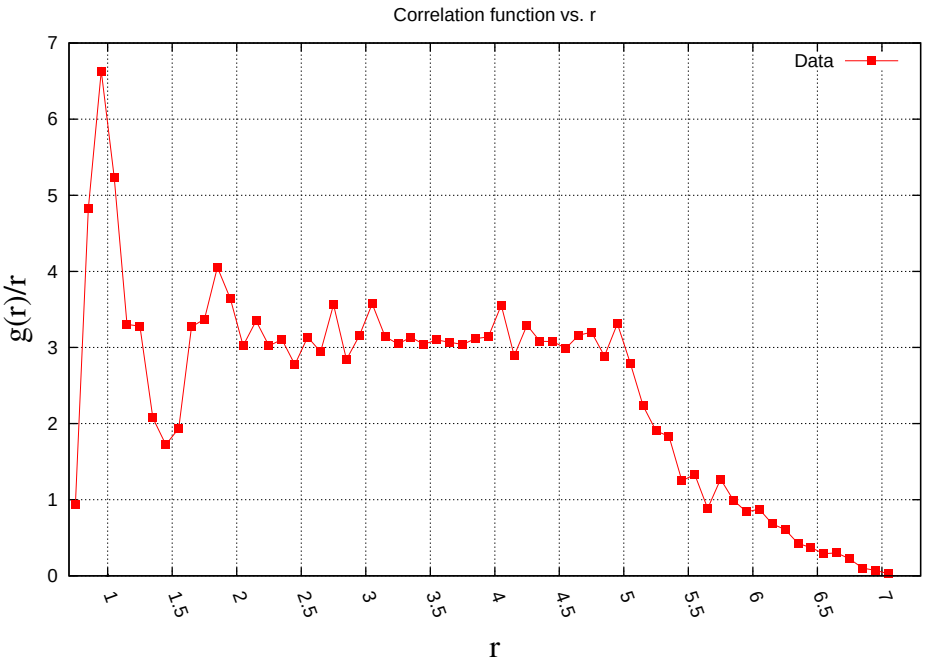


Figure 6: $g(r)/r$ vs. $r$ of High Temperature

```
  IF ( ynew(i) .GT. L ) THEN
    ynew(i) = ynew(i) - L
        yprev(i)= yprev(i)-L
ycurr(i)=ycurr(i)-L
  ELSEIF ( ynew(i) .LT. 0D0 ) THEN
    ynew(i) = ynew(i) + L
yprev(i)= yprev(i)+L
ycurr(i)=ycurr(i)+L
 ! ELSE
 !   ynew(i) = ynew(i)
! yprev(i)= yprev(i)
      ENDIF
  vx(i) = ( xnew(i) - xprev(i) )/(2d0* deltaT )
  vy(i) = ( ynew(i) - yprev(i) )/(2d0* deltaT )

      xprev(i) = xcurr(i)
  yprev(i) = ycurr(i)
  xcurr(i) = xnew(i)
  ycurr(i) = ynew(i)

    ENDDO !i

    !calculating energy, vcm, temperature
KE = 0D0
Vcmx = 0D0
Vcmy = 0D0
DO i = 1, N
      KE = KE + ( (vx(i))**2 + (vy(i))**2 )
  vcmx = vcmx + vx(i)
  vcmy = vcmy + vy(i)
ENDDO
KE=KE*0.5d0
```

```
    Vcmx=Vcmx/N
Vcmy=Vcmy/N
KT = KE/DBLE(N)
E=PE+KE
WRITE(14,'(4ES20.10)') step*deltaT, KE,PE,E
WRITE(15,'(4ES20.10)') step*deltaT, Vcmx, Vcmy, DSQRT(vcmx**2+vcmy**2)
WRITE(16,'(2ES20.10)') step*deltaT, KT

  ENDDO

!record position
  DO i = 1, N
    WRITE(21,*) i, xnew(i), ynew(i)
WRITE(22,*) i, vx(i)
  ENDDO

   DO i = 1, N
     DO z = 1, i-1
          write (23,*) r(i, z)
 ENDDO
   ENDDO

 CONTAINS

 SUBROUTINE force     !calculate Fij, the force of i ON j
   IMPLICIT NONE
   INTEGER :: ifor, jfor
   DOUBLEPRECISION, PARAMETER :: rcut = 3D0
   DOUBLEPRECISION :: xtemp, ytemp
   DOUBLEPRECISION :: costhe, sinthe  !cos and sin of theta

   PE = 0D0
   DO ifor = 1, N
     DO jfor = 1, ifor      !j <= i
       IF ( ifor == jfor ) THEN
           forcex(ifor,jfor) = 0D0
   forcey(ifor,jfor) = 0D0
       ELSE
         IF ( (xcurr(jfor) - xcurr(ifor)) > (L/2D0) ) THEN       !right -> left
           xtemp = xcurr(jfor) - L
         ELSEIF ( (xcurr(jfor) - xcurr(ifor)) < (-L/2D0) ) THEN  !left -> right
           xtemp = xcurr(jfor) + L
         ELSE                                !no telegraph
           xtemp = xcurr(jfor)
         ENDIF

         IF ( (ycurr(jfor) - ycurr(ifor)) > (L/2D0) ) THEN       !above -> under
           ytemp = ycurr(jfor) - L
         ELSEIF ( (ycurr(jfor) - ycurr(ifor)) < (-L/2D0) ) THEN  !under -> above
           ytemp = ycurr(jfor) + L
         ELSE                                !no telegraph
           ytemp = ycurr(jfor)
         ENDIF

         r(ifor,jfor) = distance(xcurr(ifor),ycurr(ifor),xtemp,ytemp)
 PE = PE + LJPotential(r(ifor,jfor))

         IF ( r(ifor,jfor) > rcut ) THEN
```

```
          forcex(ifor,jfor) = 0D0
    forcey(ifor,jfor) = 0D0
         ELSE
          costhe = (xtemp - xcurr(ifor))/r(ifor, jfor)   !the direction of r_ij pointing from i to j
          sinthe = (ytemp - ycurr(ifor))/r(ifor, jfor)
          forcex(ifor,jfor) = LJForce(r(ifor,jfor))*costhe
    forcey(ifor,jfor) = LJForce(r(ifor,jfor))*sinthe
  ENDIF

    ENDIF !(ifor ==jfor)

      ENDDO !jfor
    ENDDO !ifor

    DO ifor = 1, N
      DO jfor = ifor, N      !jfor >= ifor, Newton's 3rd Law
        forcex(ifor,jfor) = -forcex(jfor,ifor)
    forcey(ifor,jfor) = -forcey(jfor,ifor)
      ENDDO !jfor
    ENDDO !ifor
  END SUBROUTINE

  SUBROUTINE accel(i,ax,ay)  !acc of the i_th particle
    IMPLICIT NONE
    DOUBLEPRECISION, PARAMETER :: m = 1D0
    DOUBLEPRECISION :: ax, ay
    INTEGER :: i,j
    ax = 0D0
    ay = 0D0
    DO j = 1, N
      ax = ax + forcex(j,i)/m
  ay = ay + forcey(j,i)/m
    ENDDO
    RETURN
  END SUBROUTINE


  DOUBLEPRECISION FUNCTION distance(xi,yi,xj,yj)
    IMPLICIT NONE
    DOUBLEPRECISION :: xi,yi,xj,yj
    distance = DSQRT( (xi-xj)**2 + (yi-yj)**2 )
    RETURN
  END FUNCTION

  DOUBLEPRECISION FUNCTION LJForce(r)
    IMPLICIT NONE
    DOUBLEPRECISION :: r
    LJForce = 24D0 * ( 2D0 / r**13 - 1D0 / r**7)
    RETURN
  END FUNCTION

  DOUBLEPRECISION FUNCTION LJpotential(r)
    IMPLICIT NONE
    DOUBLEPRECISION :: r
    LJpotential = 4D0*( 1D0 / r**12 - 1D0 / r**6 )
    RETURN
  END FUNCTION
END PROGRAM
```