Lecture 13

The Dynamics of Many Particle Systems

Hai-Qing Lin

Beijing Computational Science Research Center

This PowerPoint Notes Is Based on the Textbook '***An Introduction to Computer Simulation Methods : Applications to Physical Systems***', 2nd Edition, Harvey Gould and Jan Tobochnik, Addison-Wesley(1996); "A First Course in Computational Physics"; "Numerical Recipes"; "Elementary Numerical Analysis"; "Computational Methods in Physics and Engineering".

Introduction

- Gasses, liquids, solids , etc.
- System of $\approx 10^{24}$ molecules, how to deal with it?
- Molecular Dynamics (**MD**): ***Computer simulation of dynamics of many particle system***.
- Interaction between particles are assumed.
- Typical size of simulated system: $N \sim 10^4$ - 10^6. ***(parallel computing)***
- **Familiarize with basic MD procedures.**

Objectives

- **Concepts and techniques in molecular dynamics simulation: The Lennard-Jones potential, Verlet algorithm, periodic boundary, minimum image**, etc.
- **Molecular dynamics program, "md.f90".**
- **MD calculations of thermodynamics quantities, P, T, $n(t)$, $N(v)$, etc.**
- Correlation functions, $g(r)$.
- Hard disks and diffusion equations.

Questions to be Noted:

- Characteristic scales?
  (Classical or quantum?)
- What is the sensible model to start with?
- Parameters to be used?
- Symmetries and conservation laws?
- Quantities to be measured?

The Intermolecular Potential

- Specify the **model system** before simulation (1st step).
- Assumptions: *classical, spherical, chemical inert, and pairwise interaction, N*(*N* – 1)/2

$$U = u\left(r_{12}\right) + u\left(r_{13}\right) + \cdots + u\left(r_{23}\right) + \cdots$$

$$= \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u\left(r_{ij}\right), \; r_{ij} = \left|\mathbf{r}_i - \mathbf{r}_j\right|.$$

- *u*(*r*) is usually chosen *empirically*.

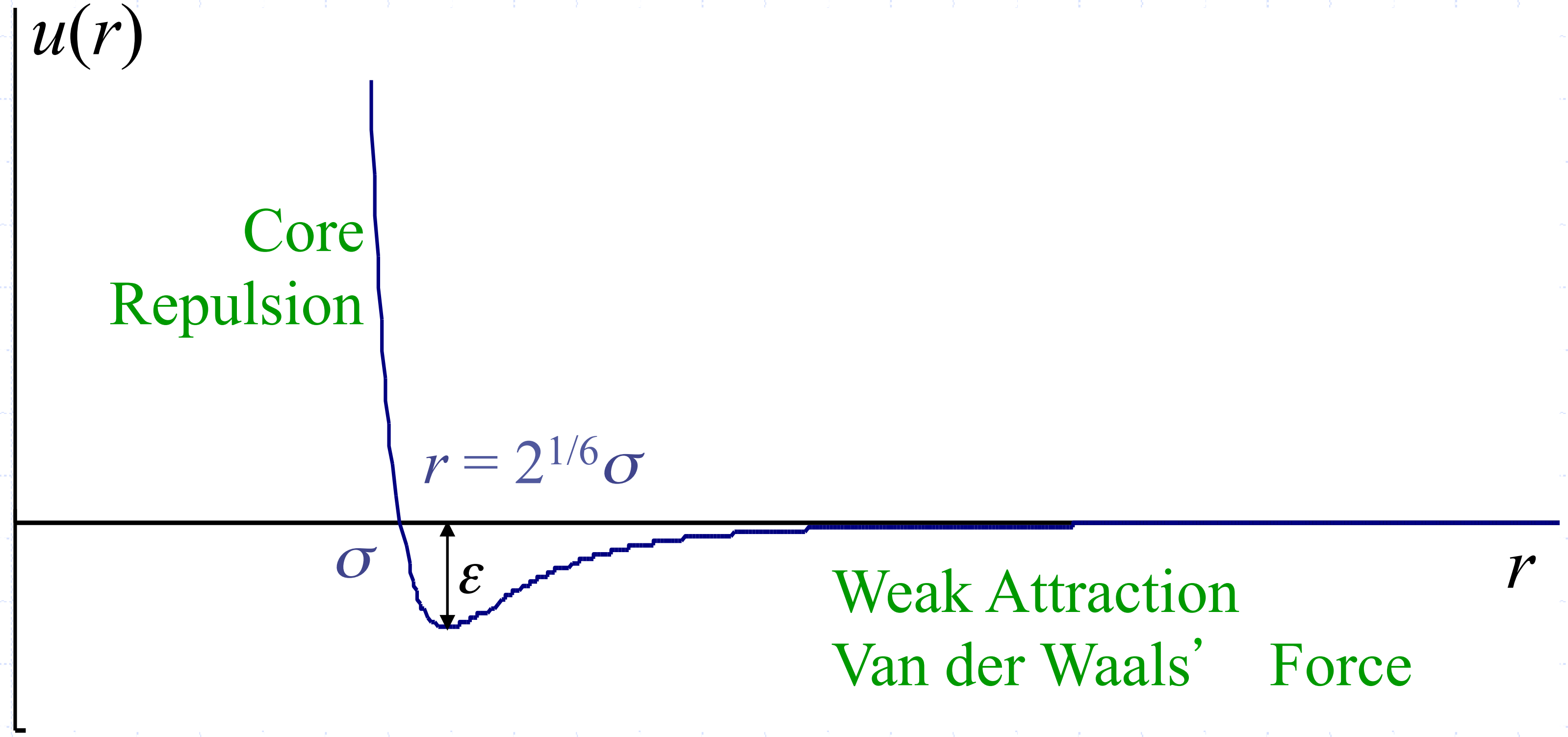
The Lennard-Jones Potential :

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \quad \mathbf{f}(r) = -\nabla u(r)$$

- Parameters:
  length scale σ, energy scale ε.
- For liquid argon:
  $\sigma = 3.4$ Å, $\varepsilon = 1.65 \times 10^{-21}$ J.

The Lennard-Jones Potential: $u(r)$ vs r

Note that the potential is characterized by the length σ and the energy ε .



The Numerical Algorithm

- We use the ***Verlet algorithm***:

$$x_{n+1} = x_n + v_n \Delta t + a_n (\Delta t)^2/2,$$
$$v_{n+1} = v_n + (a_{n+1} + a_n)\, \Delta t/2.$$

- 3rd order in position, 2nd order in velocity.
- Higher order may not necessary?

Stability

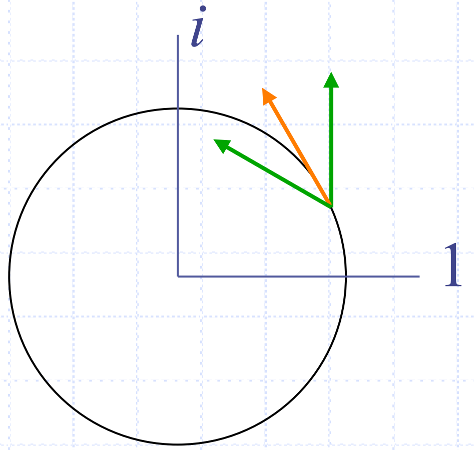
The stability properties of a particular numerical method will, in general, depend upon the type of differential equation as well as the integration step size.

$$\frac{d^2x}{dt^2} = -\omega_0^2 x, \quad x(t=0) = 1, \quad \left.\frac{dx}{dt}\right|_{t=0} = -i\omega_0$$

$$x(t) = e^{i\omega_0 t}$$

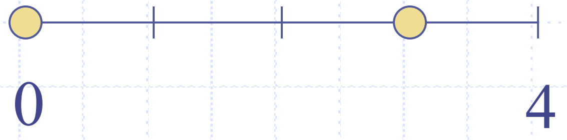
← amplitude true (on circle)

← non-amplitude true

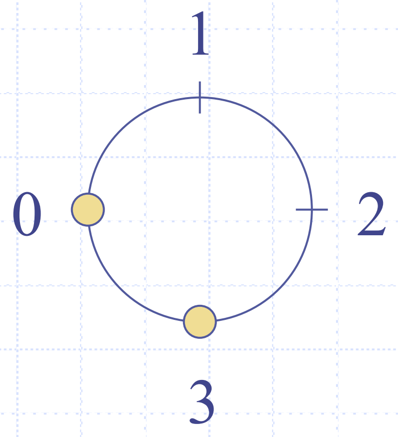


Boundary Conditions

- Plays an important role in all kinds of simulations. We use ***periodic boundary conditions (PBC)***.
- **Example in 1D**
  - Two particles at $x = 0$ and $x = 3$ on a line of length $L = 4$; the distance between the particles is **3**.

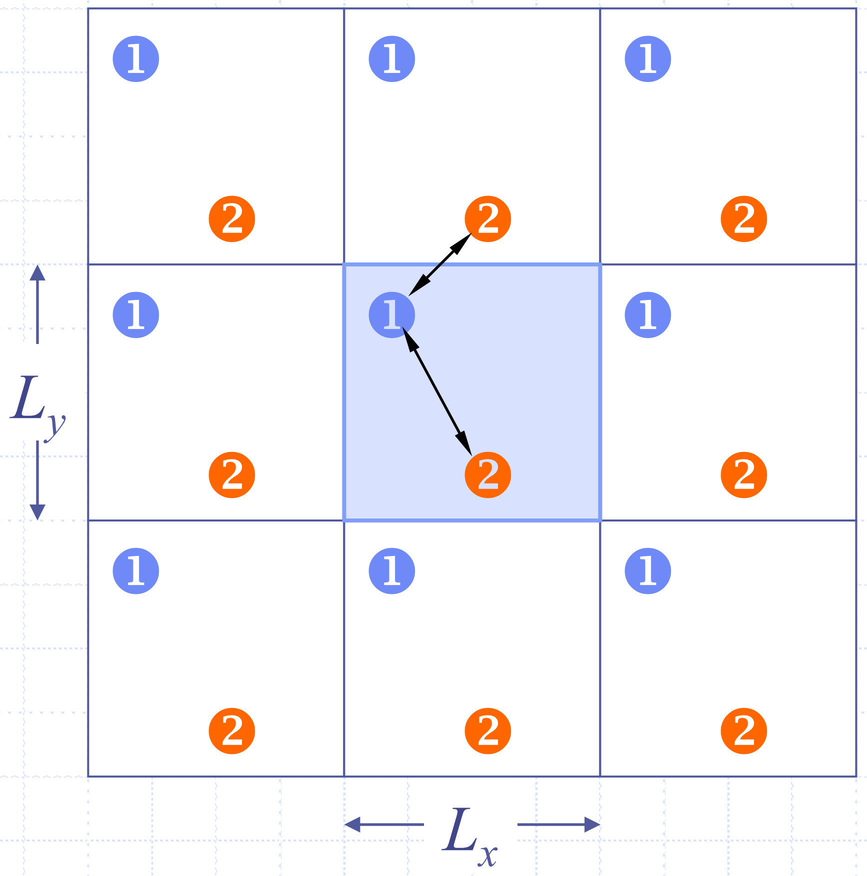


  - The application of PBC for short range interactions is equivalent to thinking of the line as forming a circle of circumference L. In this case the **minimum distance** between the two particles is **1**.



The Minimum Image Approximation

The minimum image distance convention implies that the separation between particles ❶ and ❷ is given by the **shorter** one of the two distances shown.



Units

- Choose units so that the computed quantities are neither too small nor too large. (e.g., astronomical units.)
- For molecular dynamics here, we use
  - Length: $\boldsymbol{\sigma}$ (***L***)
  - Energy: $\boldsymbol{\varepsilon}$ (***E***)
  - Mass: ***m***(***M***) (mass of one atom)
  - Time: $\boldsymbol{t = \sigma (m/\varepsilon)^{1/2}}$(***T***)

Quantity	Unit	Value for argon
Length	σ	3.4×10^{-10} m
Energy	ε	1.65×10^{-21} J
Mass	m	6.69×10^{-26} kg
Time	$\sigma(m/\varepsilon)^{1/2}$	2.17×10^{-12} s
Velocity	$(\varepsilon/m)^{1/2}$	1.57×10^{2} m/s
Force	ε/σ	4.85×10^{-12} N
Pressure	ε/σ^2	1.43×10^{-2} N · m^{-1}
Temperature	ε/k	120 K

The system of units used in the molecular dynamics simulation of particles interacting via the Lennard-Jones potential. The numerical values of $\boldsymbol{\sigma}$, $\boldsymbol{\varepsilon}$ and $\boldsymbol{m}$ are for argon. The quantity $\boldsymbol{k}$ is Boltzmann's constant and has the value $\boldsymbol{k} = 1.38 \times 10^{-23}$ J/K. The unit of pressure is for a 2D system.

MD Program

- Module(2): periodic, common
- Function(2): separation, pbc,
- Subroutine(8): initial, allocate, Verlet, accel, force, check_momentum, save_config, output

A Molecular Dynamics Simulation Program of a 2D Lennard-Jones System

```
1    program md ! ends on line 21
2    ! program adapted from Gould & Tobochnik, Chapter 8
3    ! simple N(N–1)/2 calculation of forces
4    use periodic
5    use common
6    real (kind = double) :: E
7    real (kind = double) :: &
         ke,kecum,pecum,vcum,area,pe,virial
8    integer :: ncum
9    call initial(ke,kecum,pecum,vcum,area)
10   call accel(pe,virial)
```

```
11  E = ke + pe                 ! total energy
12  ncum = 0                    ! number of times data accumulated
13    do
14        if (t > 2.0) then     ! modify
15            exit
16        end if
17        call Verlet(ke,pe,virial)
18        call output(ke,pe,virial,kecum,vcum,ncum,area)
19    end do
20    call save_config()
21  end program md ! line 1
```

Note the "PUBLIC" statement in the program

```
22
23 module periodic ! ends on line 52
24 public :: separation,pbc
25 integer, public, parameter :: double = 8  ! could be different
26 Contains
27
28 function separation(ds,L) result (separation_result)
29   real (kind = double), intent (in) :: ds,L
30   real (kind = double) :: separation_result
31   if (ds > 0.5*L) then
32      separation_result = ds – L
33   else if (ds < –0.5*L) then
34      separation_result = ds + L
35   else
36      separation_result = ds
37   end if
38 end function separation
```

One may use a “table” instead of a function

```
39
40  function pbc(pos,L) result (f_pbc) ! periodic boundary
41    real (kind = double), intent (in) :: pos,L
42    real (kind = double) :: f_pbc
43    if (pos < 0.0) then
44       f_pbc = pos + L
45    else if (pos > L) then
46       f_pbc = pos – L
47    else
48       f_pbc = pos
49    end if
50  end function pbc
51
52  end module periodic  ! line 23
53
```

```
54  module common ! ends on line 281
55
56  use periodic
57  private
58
59  public :: initial,allocate_arrays,Verlet,accel,force
60  public :: check_momentum,save_config,output
61
62  integer, public :: N
63  real (kind = double), public :: Lx,Ly,t,dt,dt2
64  real (kind = double), public, dimension (:), &
          allocatable :: x,y,vx,vy,ax,ay
65  integer, dimension(2), public :: seed
66
67  Contains all subroutines here
68
```

```
69 subroutine initial(ke,kecum,pecum,vcum,area) !ends on line 154
70    real (kind = double), intent (out) :: &
          ke,kecum,pecum,vcum,area
71    character(len = 20) :: start,file_name
72    character(len = 100) :: dum
73    integer :: n1,i,row,col
74    real :: a_x,a_y,vmax,rnd
75    dt = 0.005
76    dt2 = dt*dt
77    seed(1) = 1239
78    seed(2) = 1111
79    call random_seed(put=seed)
80    print *, "read data (d), read file (f), or lattice start (l) ="
81    read *, start
```

Note the three initial configurations below

```
! Preset initial positions and velocities
if (start == "D" .or. start == "d") then
   N = 16
   call allocate_arrays()
   Lx = 6.0
   Ly = 6.0
   x(1:8) = (/ 1.09,3.12,0.08,0.54,2.52,3.03,4.25,0.89 /)
   x(9:16) = (/ 2.76,3.14,0.23,1.91,4.77,5.10,4.97,3.90 /)
   y(1:8) = (/ 0.98,5.25,2.38,4.08,4.39,2.94,3.01,3.11 /)
   y(9:16) = (/ 0.31,1.91,5.71,2.46,0.96,4.63,5.88,0.20 /)
   vx(1:8) = (/ –0.33,0.12,–0.08,–1.94,0.75,1.70,0.84,–1.04 /)
   vx(9:16) = (/ 1.64,0.38,–1.58,–1.55,–0.23,–0.31,1.18,0.46 /)
   vy(1:8) = (/ –0.78,–1.19,–0.10,–0.56,0.34,–1.08,0.47, 0.06 /)
   vy(9:16) = (/ 1.36,–1.24,0.55,–0.16,–0.83,0.65,1.48,–0.51 /)
```

```
else if (start == "l" .or. start == "L") then
   ! For triangular lattice
   print *, "N = "
   read *, N        ! assume that sqr(N) is an integer
   call allocate_arrays()
   print *, "Lx = "
   read *, Lx
   Ly = 0.5*sqrt(3.0)*Lx
   n1 = sqrt(real(N))
   a_x = Lx/n1      ! lattice spacing
   a_y = 0.5*sqrt(3.0)*a_x
   vmax = 1.0
   i = 0
```

```
! triangular lattice
do row = 1,n1
   do col = 1,n1
      i = i + 1
      x(i) = (col + 0.5*modulo(row,2) – 1)*a_x
      y(i) = (row – 0.5)*a_y
      ! choose random velocities
      call random_number(rnd)
      vx(i) = (2*rnd – 1)*vmax
      call random_number(rnd)
      vy(i) = (2*rnd – 1)*vmax
   end do
end do
do i = 1,N
   x(i) = pbc(x(i),Lx)
   y(i) = pbc(y(i),Ly)
end do
```

```
else if (start == "f" .or. start == "f") then
! Initial positions and velocities from a data file
   print *, "file name = "
   read *, file_name
   open (unit=5,file=file_name,status="old",action="read")
   read (unit=5,fmt = *) N
   read (unit=5,fmt = *) Lx,Ly
   call allocate_arrays()
   read (unit=5,fmt = *) dum
   do i = 1,N
      read (unit=5,fmt = *) x(i),y(i)
   end do
   read (unit=5,fmt = *) dum
   do i = 1, N
      read (unit=5,fmt = *) vx(i),vy(i)
   end do
   close (unit=5)
end if
```

```
141     call check_momentum()
142     ke = 0.0                   ! kinetic energy
143     do i = 1,N
144        ke = ke + vx(i)*vx(i) + vy(i)*vy(i)
145     end do
146     ke = 0.5*ke
147     ! initialize sums
148     kecum = 0.0
149     pecum = 0.0
150     vcum = 0.0
151     area = Lx*Ly
152     ! print heading for data
153     print "(t6,a,t17,a,t27,a,t37,a)", "time","E","T","P"
154   end subroutine initial ! starts from line 69
155
```

```
156  subroutine allocate_arrays()
157    allocate(x(N))
158    allocate(y(N))
159    allocate(vx(N))
160    allocate(vy(N))
161    allocate(ax(N))
162    allocate(ay(N))
163  end subroutine allocate_arrays
164
165  subroutine Verlet(ke,pe,virial) ! ends on line 190
166    real (kind = double), intent (out) :: ke
167    real (kind = double), intent (inout) :: pe,virial
168    integer :: i
169    real (kind = double) :: xnew,ynew
170
```

```
171      do i = 1, N
172          xnew = x(i) + vx(i)*dt + 0.5*ax(i)*dt2
173          ynew = y(i) + vy(i)*dt + 0.5*ay(i)*dt2
174          x(i) = pbc(xnew,Lx)
175          y(i) = pbc(ynew,Ly)
176          ! partially update velocity using old acceleration
177          vx(i) = vx(i) + 0.5*ax(i)*dt
178          vy(i) = vy(i) + 0.5*ay(i)*dt
179      end do
180      call accel(pe,virial)        ! new acceleration
181      ke = 0.0
182      do i = 1, N
183   ! complete the update of the velocity using new acceleration
184          vx(i) = vx(i) + 0.5*ax(i)*dt
185          vy(i) = vy(i) + 0.5*ay(i)*dt
186          ke = ke + vx(i)*vx(i) + vy(i)*vy(i)
187      end do
188      ke = 0.5*ke
189      t = t + dt
190   end subroutine Verlet ! line 165
```

```
191
192  subroutine accel(pe,virial) ! ends on line 216
193     real (kind = double), intent (inout) :: pe,virial
194     real (kind = double) :: dx,dy,fxij,fyij,pot
195     integer :: i,j
196     do i = 1, N
197        ax(i) = 0.0
198        ay(i) = 0.0
199     end do
200     pe = 0.0
201     virial = 0.0
```

```
202    do i = 1, N – 1              ! compute total force on particle i
203      do j = i + 1,N             ! due to particles j > i
204         dx = separation(x(i) – x(j),Lx)
205         dy = separation(y(i) – y(j),Ly)
206      ! acceleration = force because mass = 1 in reduced units
207         call force(dx,dy,fxij,fyij,pot)
208         ax(i) = ax(i) + fxij
209         ay(i) = ay(i) + fyij
210         ax(j) = ax(j) – fxij   ! Newton's third law
211         ay(j) = ay(j) – fyij
212         pe = pe + pot
213         virial = virial + dx*fxij + dy*fyij
214      end do
215    end do
216  end subroutine accel ! line 192
217
```

```
subroutine force(dx,dy,fx,fy,pot)      ! Lennard-Jones
  real (kind = double), intent (in) :: dx,dy
  real (kind = double), intent (out) :: fx,fy,pot

  real (kind = double) :: r2,rm2,rm6,f_over_r

  r2 = dx*dx + dy*dy
  rm2 = 1.0/r2
  rm6 = rm2*rm2*rm2
  f_over_r = 24*rm6*(2*rm6 – 1)*rm2   ! epsilon=1
  fx = f_over_r*dx
  fy = f_over_r*dy
  pot = 4.0*(rm6*rm6 – rm6)
end subroutine force

```

Note vector manipulations here

```
subroutine check_momentum()
  real (kind = double) :: vxsum, vysum,vxcm,vycm
  ! compute total center of mass velocity (momentum)
  vxsum = sum(vx)
  vysum = sum(vy)
  vxcm = vxsum/N
  vycm = vysum/N
  vx = vx – vxcm           ! center mass
  vy = vy – vycm
end subroutine check_momentum

```

```
subroutine save_config()          ! Pay attention to format
  character(len = 32) :: config
  integer :: i
  print *, "file name of configuration?"
  read *, config
  print *, config
  open (unit=1,file=config,status="new",action="write")
  write (unit=1, fmt="(i4)") N
  write (unit=1, fmt="(2f13.6)") Lx,Ly
  write (unit=1,fmt="(t3,a,t20,a)") "x","y"
  do i=1,N
     write (unit=1, fmt="(2f13.6)") x(i),y(i)
  end do
  write(unit=1, fmt="(t3,a,t20,a)") "vx","vy"
  do i = 1,N
     write(unit=1,fmt="(2f13.6)") vx(i),vy(i)
  end do
  close(unit=1)
end subroutine save_config

```

```
264  subroutine output(ke,pe,virial,kecum,vcum,ncum,area)
265    integer, intent (inout) :: ncum
266    real (kind = double), intent(in) :: ke,pe,virial,area
267    real (kind = double), intent(inout) :: kecum,vcum
268    real (kind = double) :: E,mean_ke,P
269
270    ncum = ncum + 1
271    E = ke + pe                    ! total energy
272    kecum = kecum + ke
273    vcum = vcum + virial
274    mean_ke = kecum/ncum ! still need to divide by N
275    P = mean_ke + (0.5*vcum)/ncum  ! mean pressure * area
276    P = P/area
277    ! mean_ke/N = mean kinetic temperature
278    print "(4f10.4)", t,E,mean_ke/N,P
279  end subroutine output
280
281  end module common ! starts on line 54
```

Thermodynamic Quantities

- The trajectories generated by molecular dynamics give pictorial descriptions of the system. Such ***microscopic*** view could be too complex to see what is really going on.
- The system can be described more simply by specifying its ***macroscopic state***.
- Examples :
  - Temperature (T)
  - Pressure (P)
  - Particle Density $n(t)$

General Properties of Macroscopic Variables

- After the removal of an internal constraint, an isolated system changes in time from a “less random” to a “more random” state.
- The equilibrium macroscopic state is characterised by relatively small fluctuations about a mean that is independent of time. A many particles system whose macroscopic state is independent of time is said to be in ***equilibrium***.

Temperature *T*(*t*): (via Equipartition Theorem)

$$kT(t) = \frac{2}{d}\frac{K(t)}{N} = \frac{1}{dN}\sum_{i=1}^{N} m_i \mathbf{v}_i(t)\cdot\mathbf{v}_i(t)$$

- The mean temperature can be expressed as the time (t) average of $T(t)$ over many configurations.
- More precisely, (the CM has zero momentum)

$$kT(t) = \frac{1}{dN-d}\sum_{i=1}^{N} m_i \mathbf{v}_i(t)\cdot\mathbf{v}_i(t)$$

Pressure $P(t)$: (Force per Unit Area per Unit Time)

$$P(t) = \frac{N}{V} kT(t) + \frac{1}{dV} \sum_{i<j} \mathbf{r}_{ij}(t) \cdot \mathbf{F}_{ij}(t)$$

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, and

$\mathbf{F}_{ij}$ is the force acts on particle *i* due to particle *j*.

The computed quantity of interest is called *virial,*

$$\frac{PV}{NkT} - 1 = \frac{1}{dNkT} \overline{\sum_{i<j} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij}} \quad \text{(Configuration average)}$$

The correction to the ideal gas equation of state due to interactions between particles.

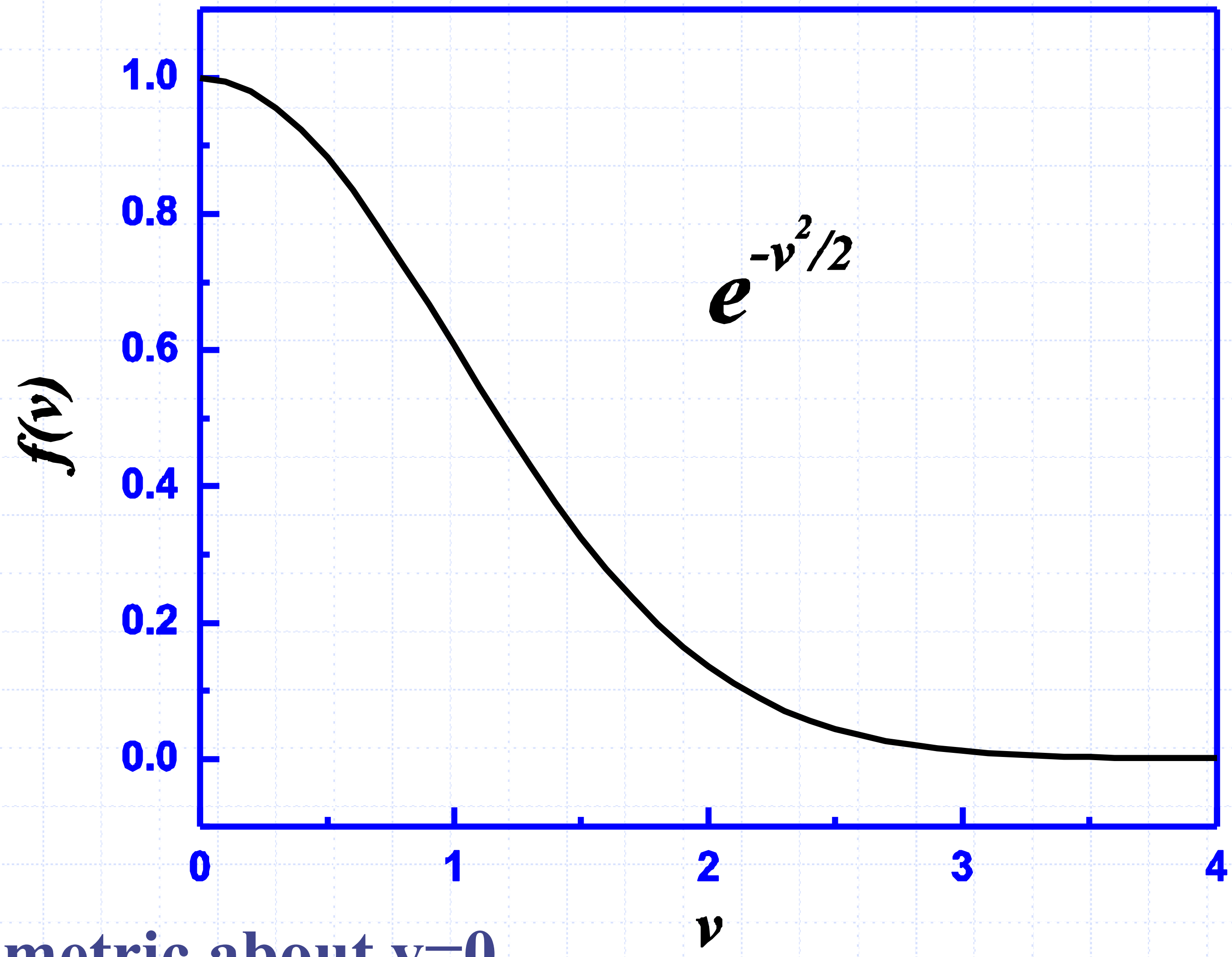
Notes on Molecular Dynamics

- Molecular dynamics allows us to compute various **time averages** of the phase space trajectory over finite time intervals.
- It is conceptually different from the **ensemble average** used in equilibrium statistical mechanics. But quasi-ergodic hypothesis asserts their equivalence (May exist non-ergotic systems.)
- Our time intervals must be sufficiently long to allow the system to explore phase space and yield meaningful averages, and our system must be sufficiently big to represent true materials.
- The condition $(F/m)\,(\Delta t)^2 \ll \sigma$ must be satisfied for a finite difference method to be applicable.

Notes on Molecular Dynamics

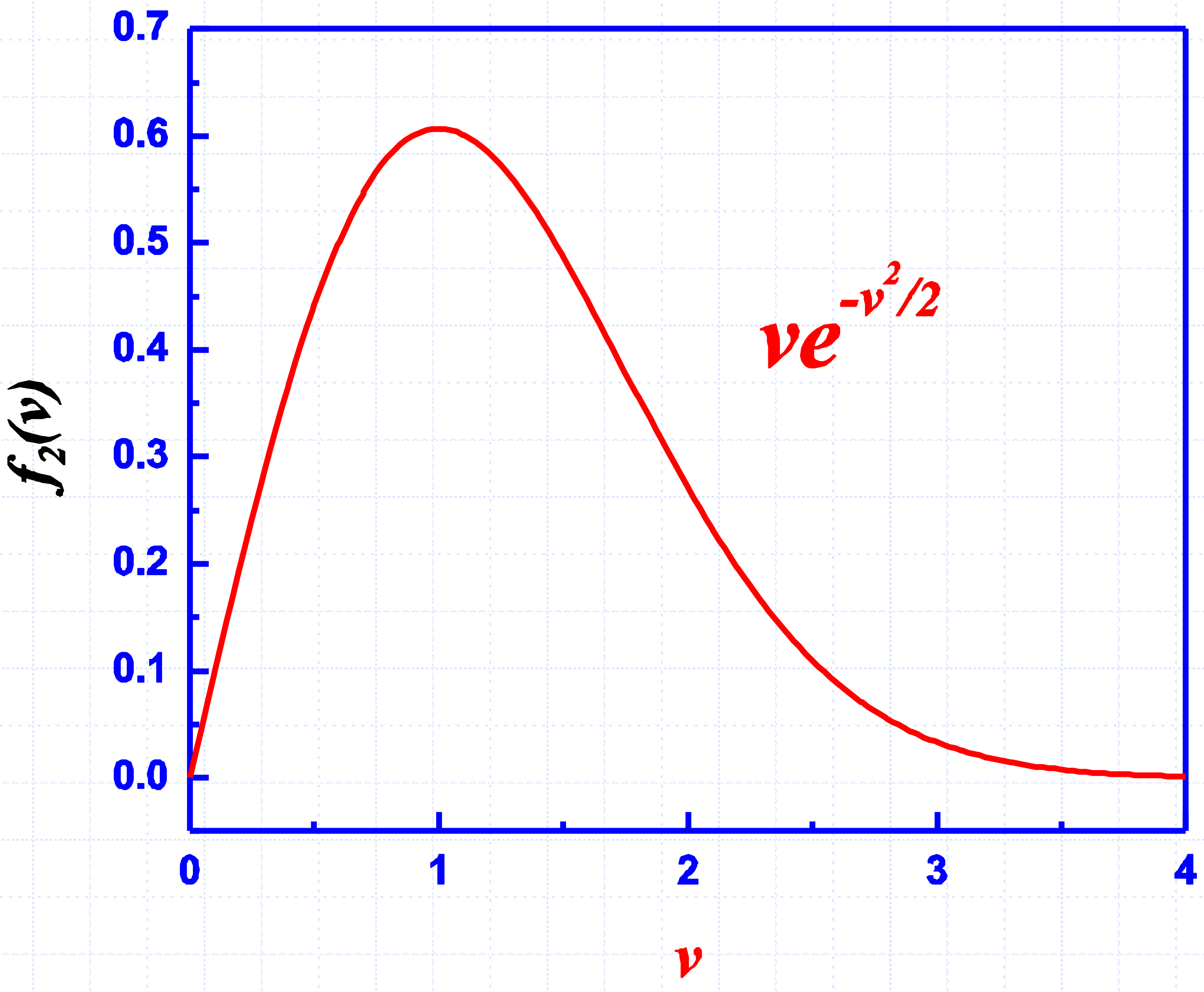
- Usually, we place the particle on the sites of a regular lattice with desired density. The initial velocities are chosen at random according to the Maxell-Boltzmann distribution. Temperature sets the scale of velocities.
- To simulate a solid we need to choose the shape of the central cell to be consistent with the symmetry of the solid phase of the system.
- It is possible that the system is trapped into a state with energy being local minimum but not global minimum, called the meta-stable state. Such possibility always exist and cares should be taken when performing simulation.

Velocity Distribution: 1D

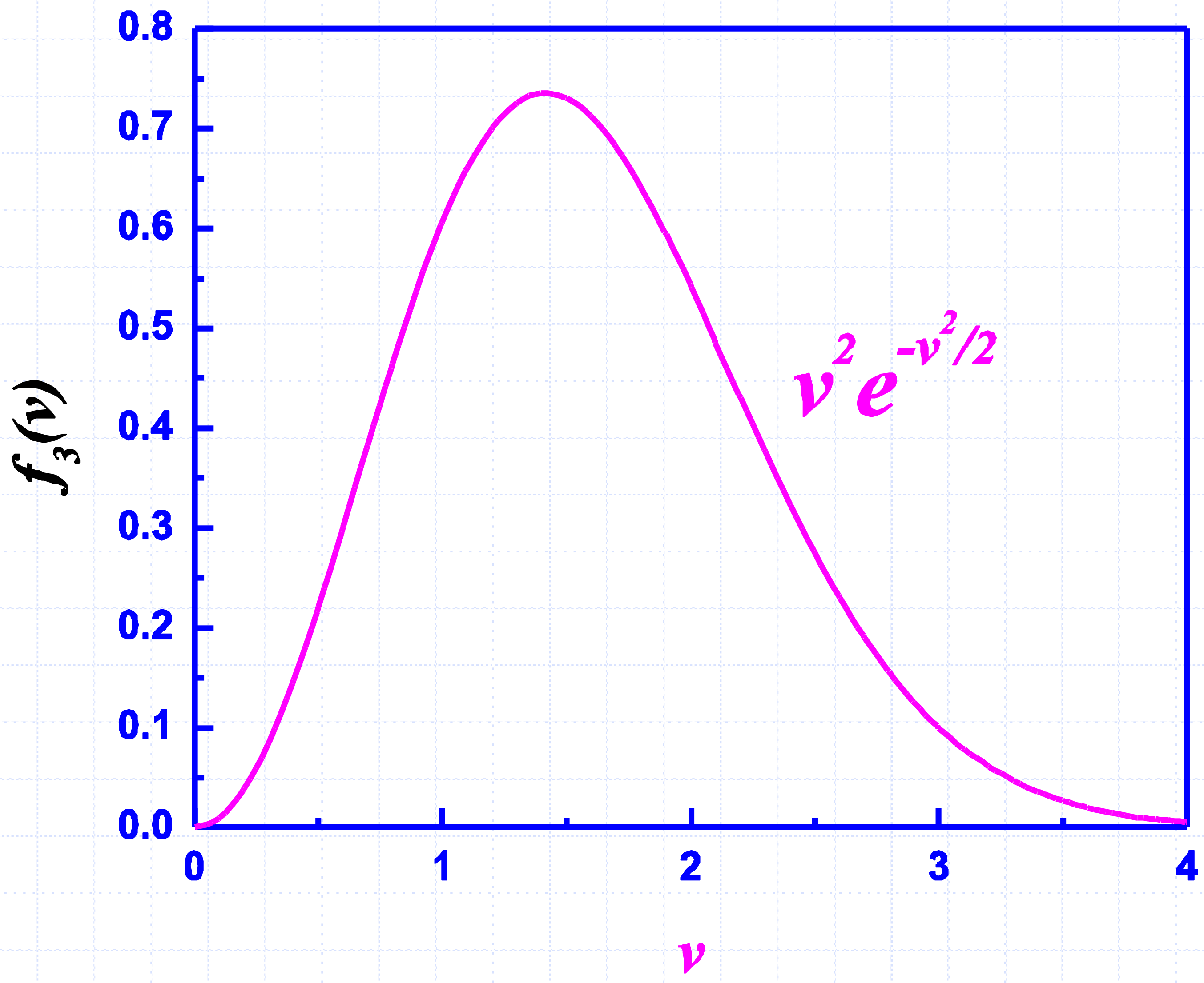


Symmetric about v=0

Velocity Distribution: 2D



Velocity Distribution: 3D



Notes on Molecular Dynamics

- Virial theorem
- Simulated Annealing
- Ewald Summations

Radial Distribution Function

- We use radial distribution function $g(r)$ to measure correlations between particle positions.
- Suppose one of the particle is at the origin, then the mean number of other particles in the shell between **r** and **r** + d**r** is given by

$$\rho\, g(\mathbf{r})\mathrm{d}\mathbf{r},\;\; \rho = N/V,\;\; d\mathbf{r} = \begin{cases} 4\pi r^2\, dr & d=3 \\ 2\pi r\, dr & d=2 \\ 2\, dr & d=1 \end{cases}$$

- with normalisation condition $\rho \int g(\mathbf{r}) d\mathbf{r} = N - 1 \approx N$

Radial Distribution Function

- For systems with spherical symmetry

  $g(\mathbf{r}) = g(r).$

- For ideal gas, there are no correlations and

  $g(r) = \text{constant} \to 1.$

- For Lennard-Jones interaction, we expect

  $g(r) \to 0$ as $r \to 0$; $g(r) \to 1$ as $r \to \infty$.

- The radial distribution function $g(r)$ can be measured indirectly by elastic radiation scattering experiments, e.g., X-rays. (model validity)

Thermodynamics Properties Obtained from $g(r)$

- The mean potential energy per particle is

$$\frac{U}{N} = \frac{\rho}{2}\int g(r)\,u(r)\,d\mathbf{r}$$

- The equation of state is

$$\frac{PV}{NkT} = 1 - \frac{\rho}{2dkT}\int g(r)\,r\,\frac{du(r)}{dr}\,d\mathbf{r}$$

Correlation Function

- *g(r)* is an example of correlation functions
- Correlation function can be used to explore if there exists any long-range order in the system
- Usually, one does the Fourier transform of the correlation function to observe long-range order

$$g(q) = \int g\left(r\right)\exp(iqr)\,d\mathbf{r}$$

- Examples: charge density wave (CDW) and spin density wave (SDW)

Calculation of $g(r)$:

1. Compute $n(r,\Delta r)$ = number of particles in a spherical shell of radius r and small width Δr.

```
SUB compute_g(ncorrel)
    DECLARE PUBLIC x(), y()
    DECLARE PUBLIC N, Lx, Ly
    DECLARE PUBLIC gcum(), nbin, dr
    DECLARE DEF separation
    !accumulate data for n(r)
    FOR i = 1 to N-1
        FOR j = i+1 to N
          LET dx = separation(x(i) - x(j), Lx)
          LET dy = separation(y(i) - y(j), Ly)
          LET r2 = dx*dx + dy*dy
          LET r = sqr(r2)
          LET ibin = truncate(r/dr,0) + 1
          IF ibin <= nbin then
             LET gcum(ibin) = gcum(ibin) + 1
          END IF
        NEXT j
    NEXT i
    LET ncorrel = ncorrel + 1
END SUB
```

Calculation of $g(r)$:

2. Accumulate $n(r,\Delta r)$ and normalise it, then obtain $g(r)$ from $n(r,\Delta r)$ by using the following relation,

$$\frac{1}{2}N\left(N-1\right)\frac{\Delta V}{V}g\left(r\right)=\overline{n\left(r,\Delta r\right)}$$

For $d=2$, $\Delta V=\pi\left(\left(r+\Delta r\right)^2-r^2\right)$

```
SUB normalize_g(ncorrel)
    DECLARE PUBLIC N, Lx, Ly
    DECLARE PUBLIC gcum(), dr
    LET density = N/(Lx*Ly)
    LET rmax = min(Lx/2,Ly/2)
    LET normalization = density*ncorrel*0.5*N
    LET bin = 1
    LET r= 0
    OPEN #2: name  "gdata" , access output, create new
    DO while r <= rmax
         LET area_shell = pi*((r+dr)^2 - r^2)
         LET g = gcum(bin)/(normalization*area_shell)
         PRINT r+dr/2, g
         PRINT #2: r+dr/2,g
         LET bin = bin + 1
         LET r = r + dr
    LOOP
    CLOSE #2
END SUB
```

Calculation of $g(r)$:

- The shell thickness Δr needs to be sufficiently small so that the important features of $g(r)$ are found, but large enough so that each bin has a reasonable number of contributions. (Try and error)

- Fast Fourier Transfer may be used, $G(\mathbf{q})$.

- Similar to Histogram.

Hard Disks

- A model system with the following interaction:

$$u(r) = \begin{cases} +\infty & r < \sigma \\ 0 & r \geq \sigma \end{cases}$$

- This is a sequence of two-body elastic collisions.
- We consider all pairs of particles ***i*** and ***j*** and to find the collision time $\boldsymbol{t_{ij}}$ for their next collision ignoring the presence of all other particles. We then find the minimum collision time and move all particles forward in time until the next collision occurs.

Dynamical Properties

The mean free time and mean free path concepts.

- Quantities to be considered are particle flux **J**(**r**,*t*), $\mathbf{J} = -D\nabla n(\mathbf{r},t)$, D: the self-diffusion coefficient.
- Conservation of particles:

$$\frac{\partial n(\mathbf{r},t)}{\partial t} + \nabla \cdot \mathbf{J}(r,t) = 0, \quad \frac{\partial n(\mathbf{r},t)}{\partial t} = D\nabla^2 n(\mathbf{r},t)$$

- The mean square displacement:

$$\overline{R^2(t)} = \frac{1}{N}\sum_i \overline{\left[\mathbf{r}_i(t) - \mathbf{r}_i(0)\right]^2} \rightarrow 2dDt, \quad t \rightarrow \infty$$

Extensions (More about Molecular Dynamics)

- Simulation of systems of hard disks and hard spheres were very successful.They are used as reference systems.
- There is no general way of determine system size and run time.
- The computer time for a simple MD program is $\propto t/N^2$. The most time consuming parts are generating an appropriate initial configurations (to reach equilibrium) and doing the bookkeeping. There are ways of reducing the equilibration time if the intermolecular force is sufficiently short ranged.

Extensions (More about Molecular Dynamics)

- It is possible to do MD at fixed T and/or P.
- MD can also be used to study the *transport properties* of the system such as *viscosity* and *thermal conductivity*.
- Current research on applications of MD are for non-equilibrium systems.
- However, there are fundamental limitations of MD.
    - Not all macroscopic properties of a many body system can be simply defined as some kind of time average.
    - The *multiple time scale problem*.
    - Quantum many body systems.

Lecture 13 Review & Required

- **The intermolecular Lennard-Jones potential.**
- **Periodic boundary conditions.**
- **Molecular dynamics program, "md.f90".**
- **MD calculations of thermodynamics quantities, P, T, $n(t)$, etc.**
- Correlation functions, $g(r)$.
- Hard disks and diffusion equations.