统计力学及其应用作业

——二维分子动力学模拟

小组成员：黄裕函 杨阳 刘乔 周文丰 翁翕

1 问题描述

三个粒子在外势场$U(r)=\frac{1}{2}r^2$下的二维分子动力学模拟。三粒子间的相互作用势为$U_{ij}=4[(\frac{1}{r_{ij}})^{12}-(\frac{1}{r_{ij}})^{6}]$，其中$r_{ij}=\left|\vec{r_i}-\vec{r_j}\right|$，问：

（1） 当三个粒子初始位置在 r=1 的圆上时，画出粒子的轨迹，并设法表征系统的相轨迹。

（2） 求总能量，以及每个粒子的能量随时间的变化。

（3） 画出对某一粒子 x 方向上（x，Px）的分布。

2 理论基础

2.1 系统能量守恒证明

系统动能为

$$T\left(\vec{p}_i\right)=\sum_i\frac{{p_i}^2}{2m_i} \tag{2.1.1}$$

系统势能为

$$\begin{aligned}V\left(q_i\right)&=V_{exter}+V_{inter}\\&=\sum_i\frac{1}{2}{q_i}^2+\sum_i\sum_{j,j\neq i}V_{inter}\left(\left|q_i-q_j\right|\right)\end{aligned} \tag{2.1.2}$$

系统哈密顿量为

$$H=T+V \tag{2.1.3}$$

利用哈密顿方程可以得到

$$\frac{dH}{dt}=\frac{\partial H}{\partial t} \tag{2.1.4}$$

由于H不是时间的显式函数，从而

$$\frac{dH}{dt}=\frac{\partial H}{\partial t}=0 \tag{2.1.5}$$

即系统的能量在给出的不显含时间的外势场以及粒子间势下守恒。

2.2 模拟方法

利用保证系统能量守恒的 Verlet 算法进行分子动力学模拟。其包括 Verlet 算法及速度 Verlet 算法。

（1）Verlet 算法：

粒子的空间位置用以下公式给出

$$r\left(t+\delta t\right)=2r\left(t\right)-r\left(t-\delta t\right)+\delta t^{2}a\left(t\right) \tag{2.2.1}$$

该方法关于时间是 4 阶精度。对于其中的加速度，对于保守系统可以利用粒子场势的负梯度给出

$$\vec{a}_{i}\left(t\right)=-\frac{1}{m_{i}}\sum_{j=1}^{n}\nabla U(r_{ij}) \tag{2.2.2}$$

其中 $r=\left|r_{i}-r_{j}\right|=\sqrt{\left(x_{i}-x_{j}\right)^{2}}$，$U=U_{exter}(\vec{r}_{i})+U_{\mathrm{int}er}\left(r_{ij}\right)$。

外势场由 $U_{exter}(\vec{r}_{i})=\frac{1}{2}r_{i}^{2}$ 给出。对应的粒子间势函数由 Lenard-Jones 势给出

$$U_{\mathrm{int}er}=4\left[\left(\frac{1}{r}\right)^{12}-\left(\frac{1}{r}\right)^{6}\right] \tag{2.2.3}$$

带入加速度公式中可以导出

$$\vec{f}_{ij}=-\nabla U_{\mathrm{int}er}\left(r_{ij}\right)==-24\frac{\vec{r}_{i}-\vec{r}_{j}}{\left|\vec{r}_{i}-\vec{r}_{j}\right|}\left[2\left(\frac{1}{\left|\vec{r}_{i}-\vec{r}_{j}\right|}\right)^{13}-\left(\frac{1}{\left|\vec{r}_{i}-\vec{r}_{j}\right|}\right)^{7}\right] \tag{2.2.4}$$

$$\vec{a}_{i}=\frac{1}{m_{i}}\vec{f}_{i}=\sum_{j=1,j\neq i}^{N}\vec{f}_{ij} \tag{2.2.5}$$

（2）速度 Verlet 算法：

粒子的空间位置用以下公式给出

$$\vec{r}_{i}\left(t+\delta t\right)=\vec{r}_{i}\left(t\right)+\delta t\vec{v}_{i}\left(t\right)+\frac{1}{2}\delta t^{2}\vec{a}_{i}\left(t\right) \tag{2.2.6}$$

利用新的空间位置后，速度以及加速度通过（2.7）、（2.8）、（2.9）进行更新：

$$\vec{v}_{i}\left(t+\frac{1}{2}\delta t\right)=\vec{v}_{i}\left(t\right)+\frac{1}{2}\delta t\vec{a}_{i}\left(t\right) \tag{2.2.7}$$

$$\vec{a}_{i}\left(t+\delta t\right)=-\frac{1}{m_{i}}\sum_{j=1}^{n}\nabla U(r_{ij}(t+\delta t)) \tag{2.2.8}$$

$$\vec{v}_i\left(t+\frac{1}{2}\delta t\right)=\vec{v}_i\left(t+\frac{1}{2}\delta t\right)+\frac{1}{2}\delta t\vec{a}_i\left(t+\delta t\right) \qquad (2.2.9)$$

其中的势函数与 Verlet 算法相同。

3 求解过程

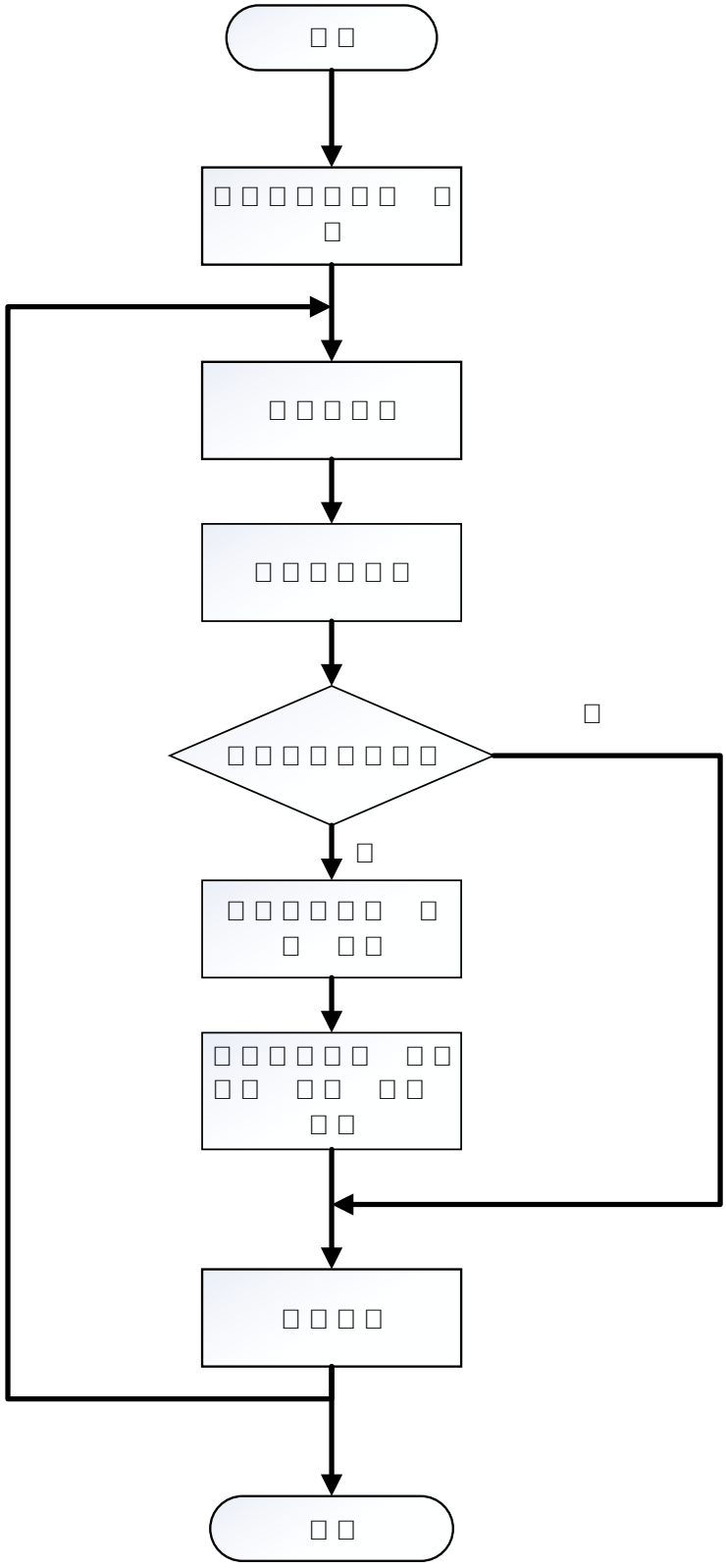

图 1 计算流程图

4 结果解释

4.1 取三粒子的初始位置分别为（0，1）、（-1，0）、（$\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}$），用 C++编程。（1）粒子的轨迹如图 2。

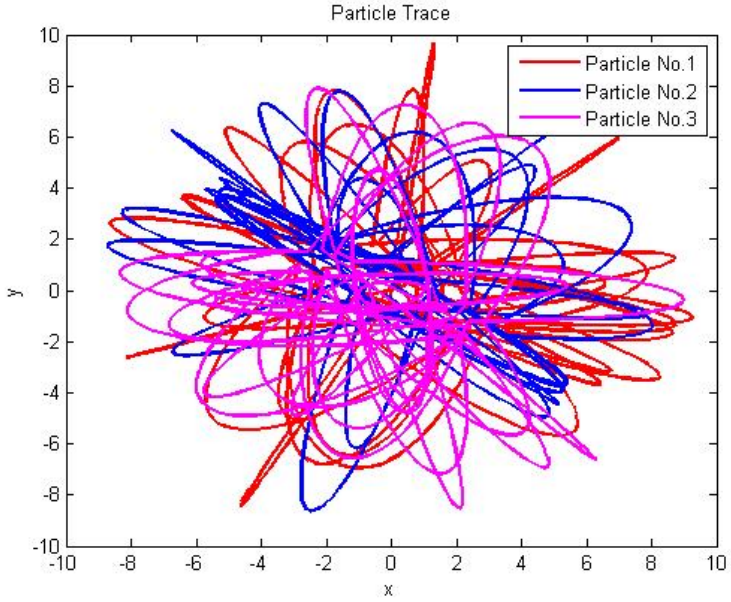

图 2 粒子的轨迹（100 步）

（2）此二维三粒子系统相空间为 12 维，为直观表示粒子相轨迹，将相空间投影到各粒子的各维度，得到六个二维投影相轨迹，如图 3。

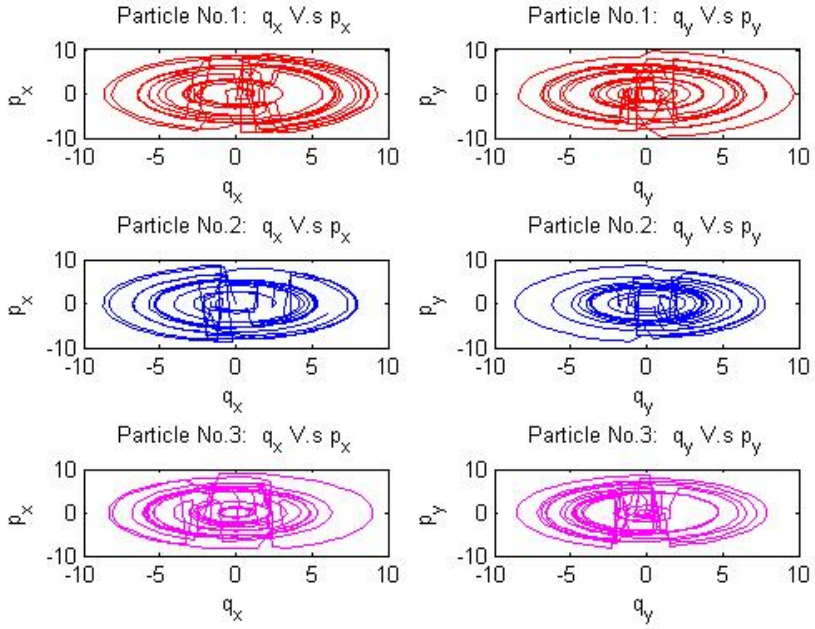

图 3 系统的相轨迹（100 步）

（3）每个粒子的能量为动能与势能之和，各粒子能量随时间变化如图 4 所示。

由模拟结果可见系统的能量守恒。

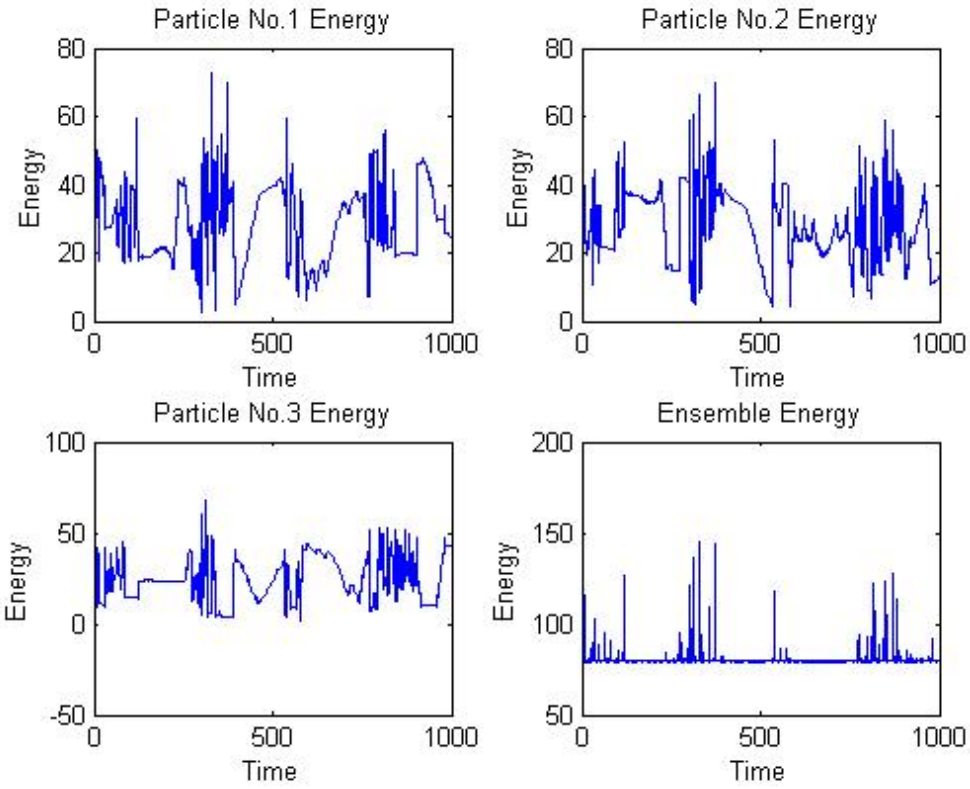

图 4 粒子及系综能量变化（1000 步）

4.2 另取三粒子的初始位置分别为为（0，1）、（$-\frac{\sqrt{3}}{2},-\frac{1}{2}$）、（$\frac{\sqrt{3}}{2},-\frac{1}{2}$），分别在半径为 1 的圆的内切正三角形的顶点。用编程 Fortran 编程，其他参数一致，结果分别如图 5，6，7。

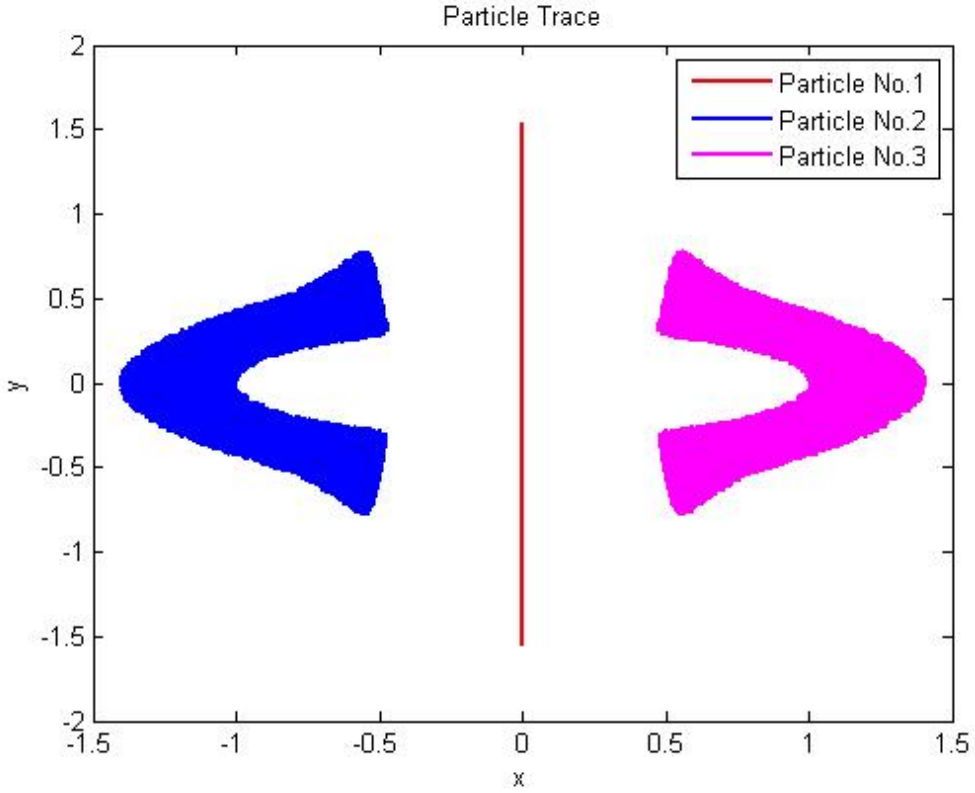

图 5 粒子的轨迹（1000 步）

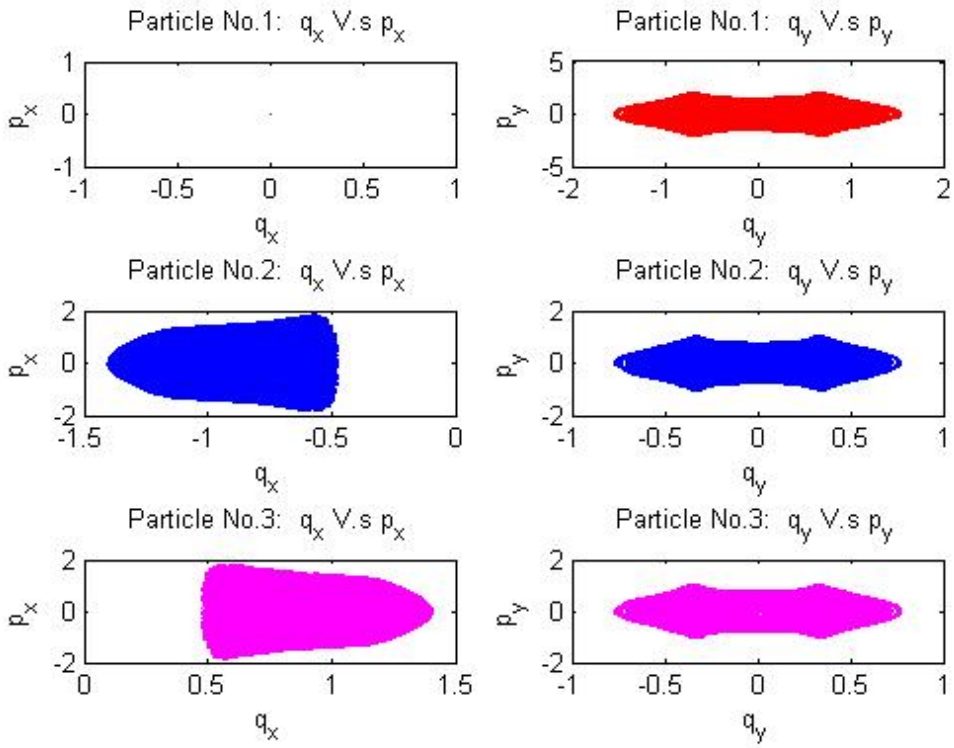

图 6 系统的相轨迹（1000 步）

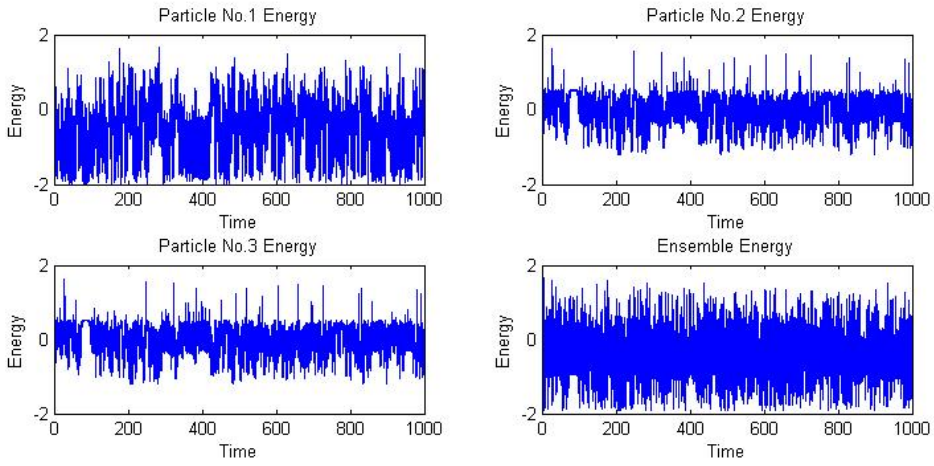

图 7 粒子及系综能量变化（1000 步）

5 参考文献

1. Rapaport D C. The Art of Molecular Dynamics Simulation[M]// The art of molecular dynamics simulation /. Cambridge University Press, 2004.

附录 1　分子动力学模拟代码（C++）

编译环境：VS2012

```
#include "stdafx.h"
#include <iostream>
#include <cmath>
#include <cstdio>
#include <fstream>

using namespace std;

#define N 3                 //粒子个数
#define DIM 2               //问题维数
#define MAX_STEP 10000000   //最大迭代步数

double* pos;                //存储位置
double* pre_pos;            //存储上一步位置
double* velocity;           //存储速度（暂未使用）
double* force;              //存储受力
double* potential;
double* kinetic_energy;
double max_force;           //存储最大受力

void initialize(){
```

```
    //此处设置初始位置
    pos[0] = sqrt(2)/2.0;       //粒子1的x坐标
    pos[1] = sqrt(2)/2.0;       //粒子1的y坐标
    pos[2] = 0.0;               //粒子2的x坐标
    pos[3] = 1.0;               //粒子2的y坐标
    pos[4] = -1.0;              //粒子3的x坐标
    pos[5] = 0.0;               //粒子3的y坐标

    for(int i=0;i<DIM*N; i++)
        pre_pos[i] = pos[i];

    for(int i=0;i<DIM*N; i++)
        *(velocity+i) = 0.0;
}

void compute_force(){
    /*
      totalForce 两个粒子ij之间的相互作用力合力,吸引为负
      dist 两个粒子之间的距离
      */
    for(int i=0; i<N*DIM; i++){
        force[i] = 0.0;             //将每个粒子的受力清零
        max_force = 0.0;            //存储最大合力（用于变步长计算）
    }
    double  r, x, y, dist, dx, dy, totalForce;
    for(int i=0;i<N;i++){
        x = pos[i*DIM + 0];
        y = pos[i*DIM + 1];
        r = x*x + y*y;
        force[i*DIM + 0] += -x;         //外势力作用
        force[i*DIM + 1] += -y;         //外势力作用
        for(int j=i+1;j<N;j++){
            dx = pos[i*DIM + 0] - pos[j*DIM + 0];
            dy = pos[i*DIM + 1] - pos[j*DIM + 1];
            dist = sqrt(dx*dx+dy*dy);
            totalForce = (48*pow(dist,-13.0)-24*pow(dist,-7.0));
            //printf("i=%d,j=%d, dist ij = %f, force = %f\n",i,j,dist,
totalForce);
            force[i*DIM + 0] += totalForce*dx/dist;
            force[i*DIM + 1] += totalForce*dy/dist;
            force[j*DIM + 0] -= totalForce*dx/dist;
            force[j*DIM + 1] -= totalForce*dy/dist;
        }
        if(abs(force[i*DIM + 0]>max_force)) max_force = abs(force[i*DIM + 0]);
```

```
        if(abs(force[i*DIM + 1]>max_force)) max_force = abs(force[i*DIM + 1]);
    }
    /*
    for(int i=0; i<N; i++){
        printf("i=%d,fx=%f,fy=%f\n",i,force[i*DIM+0],force[i*DIM+1]);
    }*/
}

// Verlet 算法
void compute_pos(double step = 1e-5){
    for(int i=0;i<N*DIM;i++){
        pre_pos[i] = 2*pos[i]-pre_pos[i]+step*step*force[i];  // Verlet 算法
        velocity[i] = (pos[i]-pre_pos[i]) / step;
    }
    swap(pre_pos,pos);
}

void compute_energy(){
    double  dist, dx, dy;
    for(int i=0;i<N;i++){
        kinetic_energy[i] = 0.5*velocity[i*DIM+0]*velocity[i*DIM+0]
                           +0.5*velocity[i*DIM+1]*velocity[i*DIM+1];
        potential[i] =
0.5*(pos[i*DIM+0]*pos[i*DIM+0]+pos[i*DIM+1]*pos[i*DIM+1]);
        for(int j=0;j<N;j++){
            if (i==j) continue;
            dx = pos[i*DIM + 0] - pos[j*DIM + 0];
            dy = pos[i*DIM + 1] - pos[j*DIM + 1];
            dist = sqrt(dx*dx+dy*dy);
            potential[i] += 4*(pow(dist,-12)-pow(dist,-6));
        }
    }
}

//打印所有粒子的位置信息
void print_pos(int step = 0){
    for(int i=0; i<N; i++){

    printf("step:%d,i=%d,x=%f,y=%f\n",step,i,pos[i*DIM+0],pos[i*DIM+1]);
    }
    cout << endl;
}

int main(){
```

```
    pos         = new double[N*DIM];
    pre_pos     = new double[N*DIM];
    velocity    = new double[N*DIM];
    force       = new double[N*DIM];
    potential   = new double[N];
    kinetic_energy = new double[N];
    initialize();
    double time_step;  //时间步长
    double time=0.0;   //统计当前物理时间
    //print_pos(0);
    ofstream out("result.txt");

    for(int i=0;i<MAX_STEP;i++){
        compute_force();                    //计算所有粒子受力
        time_step = 1e-5;1.0/max_force;       //计算应该使用的时间步长（取为
固定步长）
        compute_pos(time_step);               //向前推进一步
        compute_energy();
        time+=time_step;                    //计算此步物理时间
        if(i%10000 == 0)                    //每隔10000步存储一次数据
        {
            out << time << " " ;
            for(int in=0; in<N; in++){
                out << pos[in*DIM +0] << " " << pos[in*DIM +1] << " "
                    << velocity[in*DIM +0] << " " << velocity[in*DIM +1] << "
"
//                  << sqrt(0.5*velocity[in*DIM +0]*velocity[in*DIM +0]
//                          +0.5*velocity[in*DIM +1]*velocity[in*DIM +1]) << "
"
                    << kinetic_energy[in] << " " << potential[in] << " ";
            }
            out << endl;
            if(i%1000000 == 0)              //每隔1000000显示一次进度
                print_pos(i);
        }
    }
    return 0;
}
```

附录 2　分子动力学模拟代码（Fortran）

```
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! Program name: ThreeBodyMD.f90
!
! Program Purpose:
!                   Solve three-bodies probelm in molecule simulation.
!  Version message:
!        Date          Programmer           Description of change
!      ========        ============          =========================
!  1.   13/3/2016      Yang Yang :-)              Source Code

! A brief algorithm introduction:
! First time step is caculated using velocity-verlet algorithm and other time
step is
! caculated using verlet algorithm. Velocity is caculated using centered
differential scheme.
! $r(0 + \delta t) = 2r(0) - \delta t*v(0) + \frac{1}{2} \delta \t^{2} a(0)$
!!$v(0 + \delta t) = v(t) + \frac{1}{2} \delta t [a(0) + a(0 + \delta t)]$
! $r(t+\delta t) = 2r(t) - r(t - \delta t)+ \delta \t^{2}a(t)$
! $v(t) = [r(t + \delta t) - r(t - \delta t)] / 2\delta t$
! $v(N) = [r(N) - r(N - \delta t)] / \delta t$
! external field : $V(r) = \frac{1}{2} \r^{2}
! internal reaction : Leonard-Jones 6-12 potential  $V(r_{ij}) =
4[(\frac{1}{r_{ij}})^{12} - (\frac{1}{r_{ij}})^{6}]
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
!-----------------------------------------------------------------------
----------------------
! Setup computing parameters
MODULE ParametersBlock
implicit none

integer, parameter:: N_particles = 3;                                    !
Number of particle
integer, parameter:: N_dimension = 2;                                    !
Problem dimension

! real*8, parameter:: Mass = 1.d0                                         !
Particles' mass (this term is not necessary!)

integer, parameter:: N_time = 1000000000;                                !
Number of time step
real*8, parameter::  delta_t = 1.d-6;                                    !
Time step size  (really important)

integer, parameter:: Output_interval = 100000                            !
Output interval

character(len=20), parameter:: OutputFile = 'result.txt'                 !
Output file name

real*8, parameter:: x1_ini = 0.d0, y1_ini = 1.d0                         !
Particle No.1's position
real*8, parameter:: x2_ini = -sqrt(3.d0) / 2.d0, y2_ini = -1.d0 / 2.d0   !
```

```
Particle No.2's position
real*8, parameter:: x3_ini = sqrt(3.d0) / 2.d0, y3_ini = -1.d0 / 2.d0       !
Particle No.3's position

!real*8, parameter:: x1_ini = sqrt(2.d0)/2.d0, y1_ini = sqrt(2.d0)/2.d0       !
Particle No.1's position
!real*8, parameter:: x2_ini = 0.d0, y2_ini = 1.d0                             !
Particle No.2's position
!real*8, parameter:: x3_ini = -1.d0, y3_ini = 0.d0                            !
Particle No.3's position

END MODULE ParametersBlock
!-------------------------------------------------------------------------
----------------------
!-------------------------------------------------------------------------
----------------------



!-------------------------------------------------------------------------
----------------------
! The main program
PROGRAM Main
use ParametersBlock
implicit none
! Variables
real*8 x_new(1:N_particles,1:N_dimension),
x_current(1:N_particles,1:N_dimension),
x_old(1:N_particles,1:N_dimension)   ! Particles' direction
real*8
u_current(1:N_particles,1:N_dimension)
```

```
                                        ! Particles' velocity
real*8
acceleration(1:N_particles,1:N_dimension)
                                        ! Particles' acceleration
real*8 potential_energy(1:N_particles),
kinetic_energy(1:N_particles)
     ! particle's Potential energy and Kinetic energy

! Index
integer i,j,ierror
integer:: t = 0

! Initial all varibales
do i = 1, N_particles
  do j = 1, N_dimension
       x_new(i,j) = 0.d0
       x_current(i,j) = 0.d0
       x_old(i,j) = 0.d0
       u_current(i,j) = 0.d0
       acceleration(i,j) = 0.d0
  end do
end do

do i = 1, N_particles
    potential_energy(i) = 0.d0
    kinetic_energy(i) = 0.d0
end do

! Initial particles' state
! particles' initial position
```

```
x_current(1,1) =  x1_ini
x_current(1,2) =  y1_ini

x_current(2,1) = x2_ini
x_current(2,2) = y2_ini

x_current(3,1) = x3_ini
x_current(3,2) = y3_ini

call compute_acceleration(x_current,acceleration)          ! Compute
acceleration at first time step

! time step evolution
do i = 1,N_particles
   do j = 1,N_dimension
      x_new(i,j) = x_current(i,j) + delta_t*u_current(i,j) +
0.5d0*(delta_t**2)*acceleration(i,j)
   end do
end do

! compute potential energy
call compute_potnetialEnergy(x_current,potential_energy)    ! Compute
potential energy

! open the output file
open(unit = 10, file = Trim(OutputFile), IOSTAT = ierror)
if (ierror.EQ.0) then
  print*, 'File result.txt successfully opened!...'
endif
```

```
! output result in first time step
Write(10,100) (t-1)*delta_t,&
& x_current(1,1), x_current(1,2), u_current(1,1), u_current(1,2),
kinetic_energy(1), potential_energy(1),&
& x_current(2,1), x_current(2,2), u_current(2,1), u_current(2,2),
kinetic_energy(2), potential_energy(2),&
& x_current(3,1), x_current(3,2), u_current(3,1), u_current(3,2),
kinetic_energy(3), potential_energy(3)
100 format(1x,19ES24.15,/)

! Output simulation setup parameters
call OutputParameters(x_current,u_current,kinetic_energy,potential_energy)
Write(*,*) 'Are these parameters property?... Programing Paused, Continue?...'
pause

! new to old
x_old = x_current
x_current = x_new



! Time evolution
do t = 2,N_time

   ! Compute acceleration
     call compute_acceleration(x_current,acceleration)
   ! particles' new pisition
     call compute_position(x_current,x_old,acceleration,x_new)

   ! Output
```

```
       if(mod(t,Output_interval).EQ.0) then
          ! compute particle's celocity
          call compute_velocity(x_new,x_old,u_current)
          ! compute particle's potential energy
          call compute_potnetialEnergy(x_current,potential_energy)
          ! compute particles' kinetic energy
          call compute_kineticEnergy(u_current,kinetic_energy)
          print 200, (t-1)
          Write(10,100) (t-1)*delta_t,&
&            x_current(1,1), x_current(1,2), u_current(1,1), u_current(1,2),
kinetic_energy(1), potential_energy(1),&
&            x_current(2,1), x_current(2,2), u_current(2,1), u_current(2,2),
kinetic_energy(2), potential_energy(2),&
&            x_current(3,1), x_current(3,2), u_current(3,1), u_current(3,2),
kinetic_energy(3), potential_energy(3)
       end if
    ! time update
    x_old = x_current
    x_current = x_new

end do
200 format(1x,'Current Time Step = ',I10)
 close(10)

! Output computing log file
call logfile(t,x_current,u_current,kinetic_energy,potential_energy)

END PROGRAM Main
!-------------------------------------------------------------------------
----------------------
```

```
!-------------------------------------------------------------------------
----------------------



!-------------------------------------------------------------------------
----------------------
SUBROUTINE
OutputParameters(x_current,u_current,kinetic_energy,potential_energy)
use ParametersBlock
implicit none

! decline variables
real*8, INTENT(IN)::  x_current(1:N_particles,1:N_dimension),
u_current(1:N_particles,1:N_dimension)          ! Particles' position and
velocity
real*8, INTENT(IN)::  kinetic_energy(1:N_particles),
potential_energy(1:N_particles)                              ! Particles'
kinetic energy and potential energy

! file name
character(len=20), parameter:: FileName1='SimulationParameters.txt'

! Print parameters in screen
Write(*,100) N_particles,N_dimension
100 format(1x,'Particles = 'I10,', Problem Dimension = ',I10,/)
Write(*,101) N_time, delta_t, Output_interval
101 format(1x,'Total time step = 'I10,', Time step size = ',ES15.5,', Data output
interval = ',I10,/)
Write(*,102)
102 format(1x,'Position     Velocity      Kinetic Energy    Potential
```

```
Energy',/)
Write(*,103) x_current(1,1), x_current(1,2), u_current(1,1), u_current(1,2),
kinetic_energy(1), potential_energy(1)
103 format(1x,', x1 = ',ES15.5,', y1 = ',ES15.5,', u1 = ',ES15.5,', v1 =
',ES15.5,', k1 = ',ES15.5,', p1 = ',ES15.5,/)
Write(*,104) x_current(2,1), x_current(2,2), u_current(2,1), u_current(2,2),
kinetic_energy(2), potential_energy(2)
104 format(1x,', x2 = ',ES15.5,', y2 = ',ES15.5,', u2 = ',ES15.5,', v2 =
',ES15.5,', k2 = ',ES15.5,', p2 = ',ES15.5,/)
Write(*,105) x_current(3,1), x_current(3,2), u_current(3,1), u_current(3,2),
kinetic_energy(3), potential_energy(3)
105 format(1x,', x3 = ',ES15.5,', y3 = ',ES15.5,', u3 = ',ES15.5,', v3 =
',ES15.5,', k3 = ',ES15.5,', p3 = ',ES15.5,/)



! Output to file
Open(unit=20,file=Trim(FileName1))
Write(20,100) N_particles,N_dimension
Write(20,101) N_time, delta_t, Output_interval
Write(20,102)
Write(20,103) x_current(1,1), x_current(1,2), u_current(1,1), u_current(1,2),
kinetic_energy(1), potential_energy(1)
Write(20,104) x_current(2,1), x_current(2,2), u_current(2,1), u_current(2,2),
kinetic_energy(2), potential_energy(2)
Write(20,105) x_current(3,1), x_current(3,2), u_current(3,1), u_current(3,2),
kinetic_energy(3), potential_energy(3)
 close(20)



END SUBROUTINE OutputParameters
```

```
!-----------------------------------------------------------------------
----------------------
!-----------------------------------------------------------------------
----------------------




!-----------------------------------------------------------------------
----------------------
! Compute force
SUBROUTINE compute_acceleration(x_current,acceleration)
use ParametersBlock
implicit none

! decline variables
real*8, INTENT(IN)::
x_current(1:N_particles,1:N_dimension)
! Particles' current position
real*8, INTENT(OUT)::
acceleration(1:N_particles,1:N_dimension)
! Particles' acceleration

! Temporary varibles
real*8 x_r,y_r,r,f
! real*8 x_r,y_r,z_r,r,f     ! use this for 3-D

! index
integer i,j,k

! recation with external potential field
```

```
do i = 1, N_particles
  do k = 1, N_dimension
      acceleration(i,k) = -x_current(i,k)
  end do
end do

! reaction between particles

do i = 1, N_particles - 1
     do j = i+1, N_particles
        x_r = x_current(i,1) - x_current(j,1)                    ! compute
vector x_ij
        y_r = x_current(i,2) - x_current(j,2)                    ! compute
vector y_ij
!       z_r = x_current(i,3) - x_current(j,3)                    ! compute
vector z_ij
        r = sqrt( x_r**2 + y_r**2 )                              ! compute
norm of vector r_ij
!       r = sqrt( x_r**2 + y_r**2 + z_r**2)                      ! Three
dimension
        f = 24.d0 * ( 2.d0*(1.d0/r)**13 - (1.d0/r)**7 )          ! compute
force scalar due to LJ 6-12 Potential
        acceleration(i,1) = acceleration(i,1) + x_r / r * f      ! compute
acceleration in x direction
        acceleration(i,2) = acceleration(i,2) + y_r / r * f      ! compute
acceleration in y direction
        acceleration(j,1) = acceleration(j,1) - x_r / r * f      ! compute
acceleration in x direction  (inverse force)
        acceleration(j,2) = acceleration(j,2) - y_r / r * f      ! compute
acceleration in y direction  (inverse force)
```

```
     end do
end do



END SUBROUTINE compute_acceleration
!------------------------------------------------------------------------
----------------------


!------------------------------------------------------------------------
----------------------
! Update particle's Direction
SUBROUTINE compute_position(x_current,x_old,acceleration,x_new)
use ParametersBlock
implicit none

! decline variables
real*8, INTENT(IN):: x_current(1:N_particles,1:N_dimension),
x_old(1:N_particles,1:N_dimension)  ! particles' current position and old
position
real*8, INTENT(IN)::
acceleration(1:N_particles,1:N_dimension)
 ! Particles' current acceleration
real*8, INTENT(OUT)::
x_new(1:N_particles,1:N_dimension)
! Particles' new direction

! Index
integer i,j

! Verlet Scheme
```

```
do i = 1,N_particles
   do j = 1,N_dimension
      x_new(i,j) = 2.d0 * x_current(i,j) - x_old(i,j) + (delta_t**2) *
acceleration(i,j)
   end do
end do



END SUBROUTINE compute_position
!-----------------------------------------------------------------------
----------------------
!-----------------------------------------------------------------------
----------------------
!-----------------------------------------------------------------------
----------------------
SUBROUTINE compute_velocity(x_new,x_old,u_current)
use ParametersBlock
implicit none
! particles' current position and old position
real*8, INTENT(IN):: x_new(1:N_particles,1:N_dimension),
x_old(1:N_particles,1:N_dimension)
real*8, INTENT(OUT)::
u_current(1:N_particles,1:N_dimension)
  ! Particles' new direction


! index
integer i,j


! compute velocity using centered difference scheme
do i = 1, N_particles
```

```
  do j = 1, N_dimension
      u_current(i,j) = ( x_new(i,j) - x_old(i,j) ) / (2.d0*delta_t)
  end do
end do
END SUBROUTINE compute_velocity
!-------------------------------------------------------------------------
----------------------
!-------------------------------------------------------------------------
----------------------
!-------------------------------------------------------------------------
----------------------
SUBROUTINE compute_potnetialEnergy(x_current,potential_energy)
use ParametersBlock
implicit none
real*8, INTENT(IN)::
x_current(1:N_particles,1:N_dimension)
   ! Particles' current position
real*8, INTENT(OUT)::
potential_energy(1:N_particles)
  ! Particles' potential energy

! Index
integer i,j

! temporary varibales
real*8 x_r,y_r,r,phi
! real*8 x_r,y_r,z_r,r,phi      ! 3D

! external field potential
do i = 1, N_particles
```

```
      potential_energy(i) = 0.5d0*( x_current(i,1)**2 + x_current(i,2)**2 )
!     Potential_energy(i) = 0.5d0*( x_current(i,1)**2 + x_current(i,2)**2 +
x_current(i,3)**2 )      ! 3D
end do

! Compute potential energy
do i = 1, N_particles - 1
     do j = i+1, N_particles
        x_r = x_current(i,1) - x_current(j,1)                    ! compute
vector x_ij
        y_r = x_current(i,2) - x_current(j,2)                    ! compute
vector y_ij
!       z_r = x_current(i,3) - x_current(j,3)                    ! compute
vector z_ij
        r = sqrt( x_r**2 + y_r**2 )                              ! compute
norm of vector r_ij
!       r = sqrt( x_r**2 + y_r**2 + z_r**2)                      ! Three
dimension
        phi = 4.d0 * ( (1.d0/r)**12 - (1.d0/r)**6 )              ! compute LJ
6-12 Potential
        potential_energy(i) = potential_energy(i) + phi
        potential_energy(j) = potential_energy(j) + phi
     end do
end do



END SUBROUTINE compute_potnetialEnergy
!-------------------------------------------------------------------------
----------------------
!-------------------------------------------------------------------------
```

```
----------------------



!------------------------------------------------------------------------
----------------------
SUBROUTINE compute_kineticEnergy(u_current,kinetic_energy)
use ParametersBlock
implicit none
real*8, INTENT(IN)::
u_current(1:N_particles,1:N_dimension)
    ! Particles' current velocity
real*8, INTENT(OUT)::
kinetic_energy(1:N_particles)
   ! Particles' kinetic energy


! index
integer i,j


! initial
do i = 1,N_particles
    kinetic_energy(i) = 0.d0
end do


! Sum
do i = 1,N_particles
   do j = 1,N_dimension
      kinetic_energy(i) = kinetic_energy(i) + 0.5d0*(u_current(i,j)**2)
   end do
end do
```

```
END SUBROUTINE compute_kineticEnergy
!------------------------------------------------------------------------------
----------------------
!------------------------------------------------------------------------------
----------------------




!------------------------------------------------------------------------------
----------------------
SUBROUTINE logfile(t,x_current,u_current,kinetic_energy,potential_energy)
use ParametersBlock
implicit none

! decline variables
integer, INTENT(IN)::
t
              ! Final time step
real*8, INTENT(IN)::  x_current(1:N_particles,1:N_dimension),
u_current(1:N_particles,1:N_dimension)         ! Particles' position and
velocity
real*8, INTENT(IN)::  kinetic_energy(1:N_particles),
potential_energy(1:N_particles)                        ! Particles' kinetic
energy and potential energy

! file name
character(len=20), parameter:: FileName1='Report.log'

! Output to ReportFile
Open(unit=20,file=Trim(FileName1))
```

```
Write(20,100) N_particles,N_dimension
Write(20,101) N_time, delta_t, Output_interval
Write(20,102) t, real(t-1)*delta_t
Write(20,103)
Write(20,104) x_current(1,1), x_current(1,2), u_current(1,1), u_current(1,2),
kinetic_energy(1), potential_energy(1)
Write(20,105) x_current(2,1), x_current(2,2), u_current(2,1), u_current(2,2),
kinetic_energy(2), potential_energy(2)
Write(20,106) x_current(3,1), x_current(3,2), u_current(3,1), u_current(3,2),
kinetic_energy(3), potential_energy(3)
100 format(1x,'Particles = 'I10,', Problem Dimension = ',I10,/)
101 format(1x,'Total time step = 'I10,', Time step size = ',ES15.5,', Data output
interval = ',I10,/)
102 format(1x,'Final time step= ',I10,', Final time = ',ES15.5,/)
103 format(1x,'Position     Velocity      Kinetic Energy    Potential
Energy',/)
104 format(1x,', x1 = ',ES15.5,', y1 = ',ES15.5,', u1 = ',ES15.5,', v1 =
',ES15.5,', k1 = ',ES15.5,', p1 = ',ES15.5,/)
105 format(1x,', x2 = ',ES15.5,', y2 = ',ES15.5,', u2 = ',ES15.5,', v2 =
',ES15.5,', k2 = ',ES15.5,', p2 = ',ES15.5,/)
106 format(1x,', x3 = ',ES15.5,', y3 = ',ES15.5,', u3 = ',ES15.5,', v3 =
',ES15.5,', k3 = ',ES15.5,', p3 = ',ES15.5,/)
 close(20)



END SUBROUTINE logfile
!-----------------------------------------------------------------------
----------------------
!-----------------------------------------------------------------------
----------------------
```