Projeto 6 - Dinâmica molecular

Jefter Santiago (12559016)

6 de julho de 2024

1 Introdução

Potencial de Lennard-Jonnes

$$\mathcal{U}(\vec{r}) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^1 2 - \left(\frac{\sigma}{r}\right)^6 \right] \tag{1}$$

Utilizando unidades genericas tal que $m = 1$, então, pela segunda lei de Newton temos que as componentes da velocidade são

$$a_j^x = \sum_{j=1}^{N} \sum_{\ell \neq j}^{N} F_{j,\ell} \cos(\theta_{j,\ell}) a_j^y = \sum_{j=1}^{N} \sum_{\ell \neq j}^{N} F_{j,\ell} \sin(\theta_{j,\ell}) \tag{2}$$

porém é mais interessante encontrar uma relação que não envolva as funções trigonométricas. De fato, podemos escrever os senos e cossenos de $\theta_{j,\ell}$

$$\sin(\theta_{j,\ell}) = \frac{dy_{j,\ell}}{d_{j,\ell}}$$

$$\cos(\theta_{j,\ell}) = \frac{dx_{j,\ell}}{d_{j,\ell}}$$

1.0.1 Algoritmo de Verlet

$$t = n\Delta t \tag{3}$$
$$x_i(n+1) = 2x_i(n) - x_i(n-1) + a_i^x(n)(\Delta t)^2, n \geq 1 \tag{4}$$
$$y_i(n+1) = 2y_i(n) - y_i(n-1) + a_i^y(n)(\Delta t)^2, n \geq 1 \tag{5}$$
$$x_i(1) = x(0) + v_i^x(0)\Delta t \tag{6}$$
$$y_i(1) = y(0) + v_i^y(0)\Delta t \tag{7}$$
$$v_i^x(n) = \left(x_i(n+1) - x_i(n-1)\right)/(2\Delta t) \tag{8}$$
$$v_i^y(n) = \left(y_i(n+1) - y_i(n-1)\right)/(2\Delta t) \tag{9}$$

1.1 Detalhes de implementação

Módulo para simulações de dinâmica molecular

```
! Submodules for molecular dynamic simulations

! Velocity delta
function delta_pbc(r_next, r_prev,L)
      implicit real*8(a-h, o-y)
      delta_pbc = r_next - r_prev
      delta_pbc = delta_pbc - L * nint(delta_pbc / L)
end function delta_pbc

subroutine initialize_particles(N, L, r_curr,r_prev, v, v0)
      implicit real*8(a-h, o-y)
```

```
      dimension r_prev(20, 2)
      dimension r_curr(20, 2)
      dimension v(20, 2)

      ! Defining # rows/columns
      n_cols = ceiling(sqrt(N*1d0))
      n_rows = ceiling((N*1d0)/(n_cols*1d0))

      ! Spacing 1/4
      x_spacing = L/(1d0*n_cols)
      y_spacing = L/(1d0*n_rows)
      spacing = min(x_spacing, y_spacing)/4.0

      ! Centering in the grid
      x_offset = x_spacing / 2.0
      y_offset = y_spacing / 2.0
      call srand(562369)

      k = 1
      do j = 1, n_rows
            do i = 1, n_cols
                  r_curr(k, 1) = (i-1)*x_spacing+x_offset
                  r_curr(k, 2) = (j-1)*y_spacing+y_offset

                  r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
                  r_curr(k, 2) = r_curr(k,2)+(rand())*spacing

                  theta = 2*pi*rand()
                  v(k, 1) = v0*cos(theta)
                  v(k, 2) = v0*sin(theta)

                  r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
                  r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
                  k=k+1
            end do
      end do
end subroutine initialize_particles

! Updates acceleration a = ax, ay
! between particle i and all others
subroutine compute_acc(N,i,j,L,r_curr,acc, r)
      implicit real*8(a-h, o-y)
      dimension r_curr(20, 2)
      dimension acc(2)
      dimension r(20, 20)
      epsilon = 1e-3

      dx = r_curr(i, 1) - r_curr(j, 1)
      dy = r_curr(i, 2) - r_curr(j, 2)

      dx = dx - L * nint(dx / L)
      dy = dy - L * nint(dy / L)

      r_ij = sqrt(dx**2 + dy**2)

      r(i, j) = r_ij
      r(j, i) = r_ij

      if(r_ij > epsilon .and. r_ij <= 3d0) then
            F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
            acc(1) = acc(1) + F * dx / r_ij
            acc(2) = acc(2) + F * dy / r_ij
      end if
end subroutine compute_acc
```

```
76
77      subroutine compute_energy(N, L, v, r_curr, E, r)
78            implicit real*8(a-h, o-y)
79            dimension v(20, 2)
80            dimension r_curr(20, 2)
81            dimension r(20, 20)
82
83            epsilon = 1e-3
84            Tk = 0d0
85            do i = 1, N
86                Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
87            end do
88            U = 0d0
89            do i = 1, N
90              do j = i + 1, N
91                  r_ij = r(i, j)
92
93                  if (r_ij > epsilon .and. r_ij <= 3d0) then
94                      U = U + 4 * (r_ij**(-12) - r_ij**(-6))
95                  end if
96              end do
97            end do
98            E = Tk + U
99      end subroutine compute_energy
```

2 Tarefa A

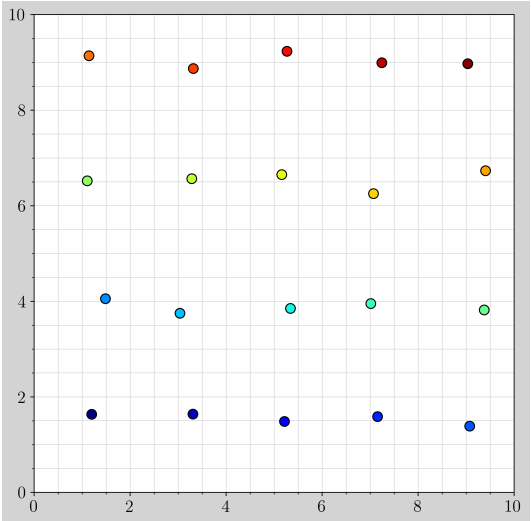

Figura 1: Posições iniciais das partículas.

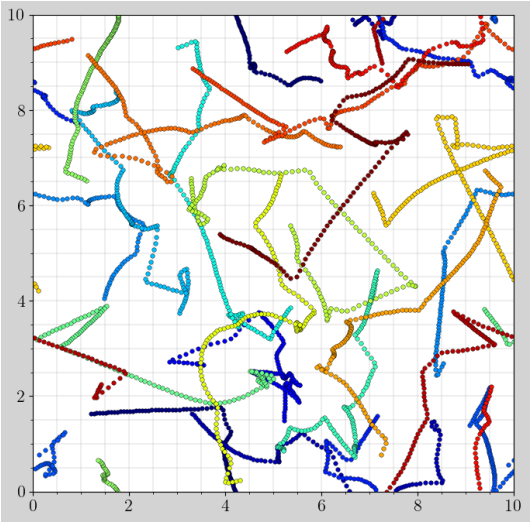

Figura 2: Coordenadas das partículas projetadas à cada $3\Delta t$.

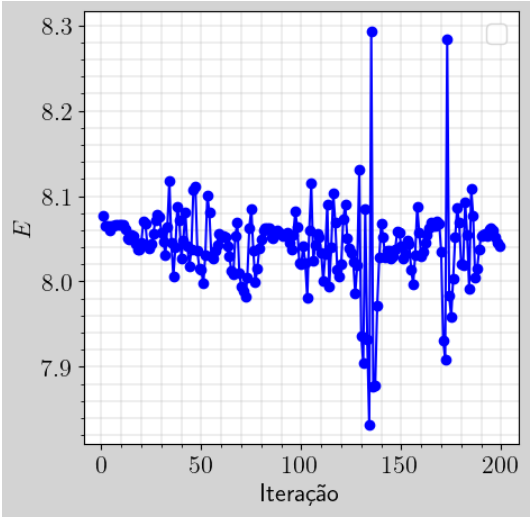

Figura 3: Energia do sistema à cada $3\Delta t$.

Código

O código abaixo está no diretório `tarefa-a/` e contém as simulações referentes às tarefas A, B e parte da D.

```
! Tarefas A, B e parte da D
implicit real*8(a-h, o-y)
parameter (pi = acos(-1.e0))
dimension r_prev(20, 2)
dimension r_curr(20, 2)
dimension r_next(20, 2)
dimension v(20, 2)
dimension r(20, 20)
dimension acc(2)
L = 10
rL = 10d0
N = 20

! Tarefa A
open(unit = 99, file="saidas/tarefa-A/parametros.dat")
open(unit = 1,  file="saidas/tarefa-A/posicoes-iniciais.dat")
open(unit = 3,  file="saidas/tarefa-A/evolucao-posicoes.dat")
! Tarefa B
open(unit = 5,  file="saidas/tarefa-B/velocidades.dat")
open(unit = 6,  file="saidas/tarefa-B/evolucao-posicoes.dat")
! Tarefa D
open(unit = 9,  file="saidas/tarefa-D/temperatura-b.dat")

dt = 0.02
v0 = 1.0
write(99, *) N, L, v0, dt
close(99)
! Defining # rows/columns
n_cols = ceiling(sqrt(N*1d0))
n_rows = ceiling((N*1d0)/(n_cols*1d0))

! Spacing 1/4
x_spacing = L/(1d0*n_cols)
y_spacing = L/(1d0*n_rows)
spacing = min(x_spacing, y_spacing)/4.0

! Centering in the grid
x_offset = x_spacing / 2.0
y_offset = y_spacing / 2.0
call srand(562369)

k = 1
do j = 1, n_rows
      do i = 1, n_cols
            r_curr(k, 1) = (i-1)*x_spacing+x_offset
            r_curr(k, 2) = (j-1)*y_spacing+y_offset

            r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
            r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
            theta = 2*pi*rand()
            v(k, 1) = v0*cos(theta)
            v(k, 2) = v0*sin(theta)

            r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
            r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
            k=k+1
      end do
end do

do i = 1, N
      write(1, *) r_curr(i, 1), r_curr(i, 2)
```

```
      write(3, *) 0d0, r_curr(i,1), r_curr(i, 2)
   end do
   close(1)

   DB = 1.0
   ! Dynamics
   do k = 1, 5000

      t = k * dt

      acc(1) = 0d0
      acc(2) = 0d0

      do i = 1, N
         acc(1) = 0d0
         acc(2) = 0d0
         do j = 1, N
            if(i /= j) then
               call compute_acc(N,i,j,L,r_curr,acc,r)
            end if
         end do
         ! UPDATE POSITIONS
         r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
         r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)

         ! APPLY PBC
         r_next(i,1) = mod(r_next(i,1)+rL, rL)
         r_next(i,2) = mod(r_next(i,2)+rL, rL)

         delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
         delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)

         ! UPDATE VELOCITIES using adjusted displacements
         v(i, 1) = delta_r_x / (2 * dt)
         v(i, 2) = delta_r_y / (2 * dt)
      end do

      r_prev(:, 1) = r_curr(:, 1)
      r_prev(:, 2) = r_curr(:, 2)

      r_curr(:, 1) = r_next(:, 1)
      r_curr(:, 2) = r_next(:, 2)

      if(k < 200) then
         E = 0d0
         call compute_energy(N, L, v, r_curr, E, r)
         write(19,*) k, E
      end if

      ! TAREFA A
      if(mod(k, 3) == 0 .and. k < 400) then
         do i = 1, N
            write(3,*) k, r_curr(i,1),r_curr(i, 2)
         end do
      end if

      ! Tarefa B & D
      if(mod(k, 20) == 0) then
         do i = 1, N
            v_mag = sqrt(v(i,1)**2+v(i,2)**2)
            write(5,*) k, v_mag, v(i,1), v(i,2)
            write(9,*) .5d0 * v_mag**2
            write(6,*) k, r_curr(i,1),r_curr(i, 2)
         end do
```

```
      end if
    end do
    close(3)
    close(5)
    close(6)
    close(9)
    close(15)

    end
    ! Submodules for molecular dynamic simulations
    ! Velocity delta
    function delta_pbc(r_next, r_prev,L)
          implicit real*8(a-h, o-y)
          delta_pbc = r_next - r_prev
          delta_pbc = delta_pbc - L * nint(delta_pbc / L)
    end function delta_pbc

    subroutine initialize_particles(N, L, r_curr,r_prev, v, v0)
          implicit real*8(a-h, o-y)
          dimension r_prev(20, 2)
          dimension r_curr(20, 2)
          dimension v(20, 2)

          ! Defining # rows/columns
          n_cols = ceiling(sqrt(N*1d0))
          n_rows = ceiling((N*1d0)/(n_cols*1d0))

          ! Spacing 1/4
          x_spacing = L/(1d0*n_cols)
          y_spacing = L/(1d0*n_rows)
          spacing = min(x_spacing, y_spacing)/4.0

          ! Centering in the grid
          x_offset = x_spacing / 2.0
          y_offset = y_spacing / 2.0
          call srand(562369)

          k = 1
          do j = 1, n_rows
                do i = 1, n_cols
                      r_curr(k, 1) = (i-1)*x_spacing+x_offset
                      r_curr(k, 2) = (j-1)*y_spacing+y_offset

                      r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
                      r_curr(k, 2) = r_curr(k,2)+(rand())*spacing

                      theta = 2*pi*rand()
                      v(k, 1) = v0*cos(theta)
                      v(k, 2) = v0*sin(theta)

                      r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
                      r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
                      k=k+1
                end do
          end do
    end subroutine initialize_particles

    ! Updates acceleration a = ax, ay
    ! between particle i and all others
    subroutine compute_acc(N,i,j,L,r_curr,acc, r)
          implicit real*8(a-h, o-y)
          dimension r_curr(20, 2)
          dimension acc(2)
          dimension r(20, 20)
```

```
      epsilon = 1e-3

      dx = r_curr(i, 1) - r_curr(j, 1)
      dy = r_curr(i, 2) - r_curr(j, 2)

      dx = dx - L * nint(dx / L)
      dy = dy - L * nint(dy / L)

      r_ij = sqrt(dx**2 + dy**2)

      r(i, j) = r_ij
      r(j, i) = r_ij

      if(r_ij > epsilon .and. r_ij <= 3d0) then
            F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
            acc(1) = acc(1) + F * dx / r_ij
            acc(2) = acc(2) + F * dy / r_ij
      end if
  end subroutine compute_acc

  subroutine compute_energy(N, L, v, r_curr, E, r)
      implicit real*8(a-h, o-y)
      dimension v(20, 2)
      dimension r_curr(20, 2)
      dimension r(20, 20)

      epsilon = 1e-3
      Tk = 0d0
      do i = 1, N
          Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
      end do
      U = 0d0
      do i = 1, N
        do j = i + 1, N
            r_ij = r(i, j)

            if (r_ij > epsilon .and. r_ij <= 3d0) then
                U = U + 4 * (r_ij**(-12) - r_ij**(-6))
            end if
        end do
      end do
      E = Tk + U
  end subroutine
```

3 Tarefa B

$$P(v) \sim \frac{v^2}{K_B T} \exp\left(-\frac{mv^2}{2K_B T}\right) \tag{10}$$

$$P(v_x) \sim \frac{1}{\sqrt{K_B T}} \exp\left(-\frac{mv_x^2}{2K_B T}\right) \tag{11}$$

$$P(v_y) \sim \frac{1}{\sqrt{K_B T}} \exp\left(-\frac{mv_y^2}{2K_B T}\right) \tag{12}$$

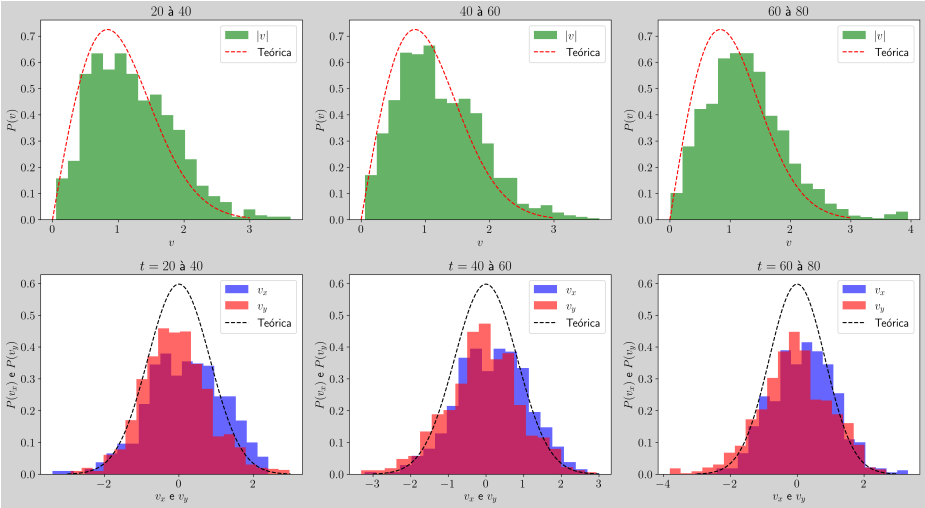

Figura 4: Distribuição da velocidade, magnitude e componentes em intervalos $t = 20-40$, $t = 40-60$ e $t = 60-80$.

Além disso há um gif para essa simulação.

4 Tarefa C

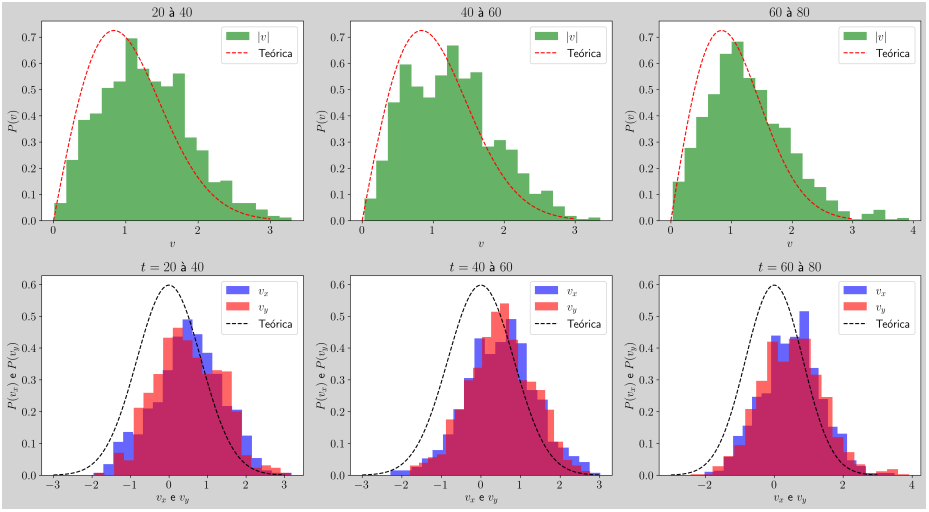

Figura 5: Distribuição da velocidade, magnitude e componentes em intervalos $t = 20-40$, $t = 40-60$ e $t = 60-80$.

Código

Nesse código temos a simulação desse item, considerandos as mudanças do perfil inicial de velocidades das partículas. E além disso ela também gera os dados necessários para o cálculo da tarefa posterior, D, assim como parte do código da tarefa A também fazia o mesmo.

```
1
2       ! Tarefa C
3       implicit real*8(a-h, o-y)
4       parameter (pi = acos(-1.e0))
5       dimension r_prev(20, 2)
6       dimension r_curr(20, 2)
7       dimension r_next(20, 2)
8       dimension v(20, 2)
9       dimension acc(2)
10      dimension r(20, 20)
11      L = 10
12      rL = 10d0
13      N = 20
14      dt = 0.02
15      v0 = 1.0
16
17      open(unit = 7,  file="saidas/tarefa-C/velocidades.dat")
18      open(unit = 4,  file="saidas/tarefa-C/evolucao-posicoes.dat")
19      open(unit = 66, file="saidas/tarefa-C/velocidades-iniciais.dat")
20
21      ! Tarefa D
22      open(unit = 8,  file="saidas/tarefa-D/temperatura-c.dat")
23      ! Modificacoes para tarefa C
24
25      r_prev = 0
26      r_curr = 0
27      r_next = 0
28      acc = 0
29
30      ! Setting velocities
31      v = 0
32      do i = 1, N/2
33            v(i, 1) = v0
34            v(i+N/2, 2) = v0
35      end do
```

```
36
37      ! Initialize particles
38
39      n_cols = ceiling(sqrt(N*1d0))
40      n_rows = ceiling((N*1d0)/(n_cols*1d0))
41
42      ! Spacing 1/4
43      x_spacing = L/(1d0*n_cols)
44      y_spacing = L/(1d0*n_rows)
45
46      spacing = min(x_spacing, y_spacing)/4.0
47
48      ! Centering in the grid
49      x_offset = x_spacing / 2.0
50      y_offset = y_spacing / 2.0
51
52      call srand(124689)
53
54      k = 1
55      do j = 1, n_rows
56            do i = 1, n_cols
57                  r_curr(k, 1) = (i-1)*x_spacing+x_offset
58                  r_curr(k, 2) = (j-1)*y_spacing+y_offset
59
60                  r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
61                  r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
62
63                  r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
64                  r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
65                  k=k+1
66            end do
67      end do
68
69      ! Dynamics
70      do k = 1, 5000
71
72            t = k * dt
73            acc(1) = 0d0
74            acc(2) = 0d0
75            do i = 1, N
76                  acc(1) = 0d0
77                  acc(2) = 0d0
78
79                  do j = 1, N
80                        if(i /= j) then
81                              call compute_acc(N,i,j,L,r_curr,acc, r)
82                        end if
83                  end do
84
85                  ! UPDATE POSITIONS
86                  r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
87                  r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)
88
89                  r_next(i,1) = mod(r_next(i,1)+rL, rL)
90                  r_next(i,2) = mod(r_next(i,2)+rL, rL)
91
92                  delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
93                  delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)
94                  ! UPDATE VELOCITIES using adjusted displacements
95                  v(i, 1) = delta_r_x / (2 * dt)
96                  v(i, 2) = delta_r_y / (2 * dt)
97            end do
98
99            ! SWAP VECTOR POSITIONS.
100           r_prev(:, 1) = r_curr(:, 1)
```

```
      r_prev(:, 2) = r_curr(:, 2)

      r_curr(:, 1) = r_next(:, 1)
      r_curr(:, 2) = r_next(:, 2)

      if(mod(k, 20) == 0) then
            do i = 1, N
                  v_mag = sqrt(v(i,1)**2+v(i,2)**2)
                  write(7,*) k,  v_mag, v(i,1), v(i,2)
                  write(8,*) .5d0 * v_mag**2
                  write(4,*) k, r_curr(i,1),r_curr(i, 2)
            end do
      end if
end do
close(4)
close(8)
end

function delta_pbc(r_next, r_prev,L)
      implicit real*8(a-h, o-y)
      delta_pbc = r_next - r_prev
      delta_pbc = delta_pbc - L * nint(delta_pbc / L)
end function delta_pbc



! Updates acceleration a = ax, ay
! between particle i and all others
subroutine compute_acc(N,i,j,L,r_curr,acc, r)
      implicit real*8(a-h, o-y)
      dimension r_curr(20, 2)
      dimension acc(2)
      dimension r(20, 20)
      epsilon = 1e-3

      dx = r_curr(i, 1) - r_curr(j, 1)
      dy = r_curr(i, 2) - r_curr(j, 2)

      dx = dx - L * nint(dx / L)
      dy = dy - L * nint(dy / L)

      r_ij = sqrt(dx**2 + dy**2)

      r(i, j) = r_ij
      r(j, i) = r_ij

      if(r_ij > epsilon .and. r_ij <= 3d0) then
            F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
            acc(1) = acc(1) + F * dx / r_ij
            acc(2) = acc(2) + F * dy / r_ij
      end if
end subroutine compute_acc

subroutine compute_energy(N, L, v, r_curr, E, r)
      implicit real*8(a-h, o-y)
      dimension v(20, 2)
      dimension r_curr(20, 2)
      dimension r(20, 20)

      epsilon = 1e-3
      Tk = 0d0
      do i = 1, N
          Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
      end do
```

```
165           U = 0d0
166           do i = 1, N
167             do j = i + 1, N
168                 r_ij = r(i, j)
169
170                 if (r_ij > epsilon .and. r_ij <= 3d0) then
171                     U = U + 4 * (r_ij**(-12) - r_ij**(-6))
172                 end if
173             end do
174           end do
175           E = Tk + U
176     end subroutine
```

5 Tarefa D

$$K_B T = \langle \frac{m}{2} \left(v_x^2 + v_y^2 \right) \rangle \tag{13}$$

6 Tarefa E

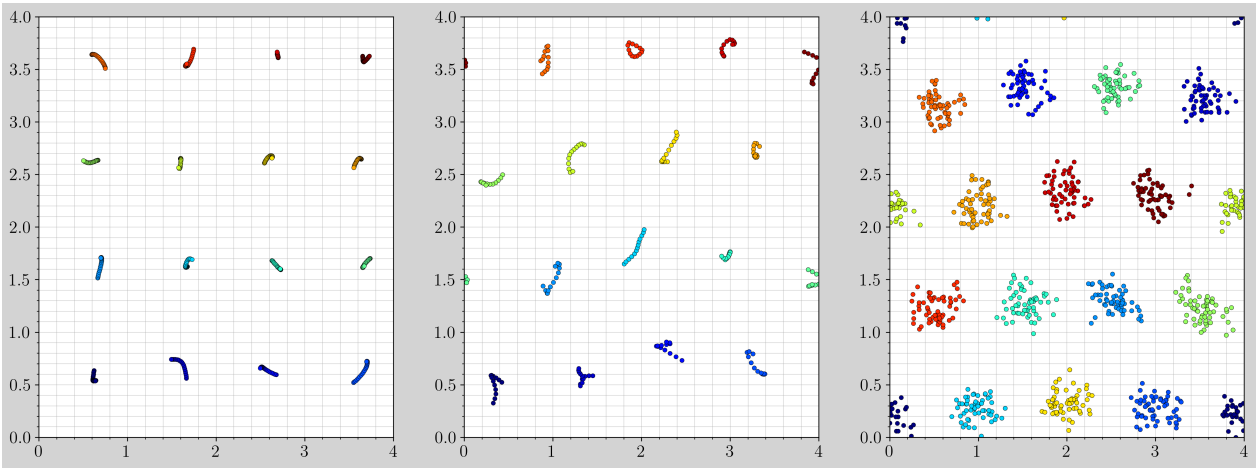

Figura 6

Código

```
1
2       ! TAREFA E
3       implicit real*8(a-h, o-y)
4       parameter (pi = acos(-1.e0))
5       dimension r_prev(20, 2)
6       dimension r_curr(20, 2)
7       dimension r_next(20, 2)
8       dimension v(20, 2)
9       dimension acc(2)
10      dimension r(20, 20)
11
12      open(unit=75, file="saidas/tarefa-E/parametros.dat")
13      open(unit=76, file="saidas/tarefa-E/posicoes-iniciais.dat")
14      open(unit=77, file="saidas/tarefa-E/evolucao-posicoes-1.dat")
15      open(unit=78, file="saidas/tarefa-E/evolucao-posicoes-2.dat")
16      open(unit=79, file="saidas/tarefa-E/evolucao-posicoes-3.dat")
17
18      ! Reset variables:
19      r_prev = 0
20      r_curr = 0
21      r_next = 0
22      v = 0
23
24      L = 4
25      rL = 4d0
26      N = 16
27      dt = 5e-3
28
29      v0 = 0.2
30
31      write(75, *) N, L, v0, dt
32      close(75)
33
34      ! Initialize particles
```

```
      n_cols = ceiling(sqrt(N*1d0))
      n_rows = ceiling((N*1d0)/(n_cols*1d0))

      ! Spacing 1/4
      x_spacing = L/(1d0*n_cols)
      y_spacing = L/(1d0*n_rows)
      spacing = min(x_spacing, y_spacing)/4.0

      ! Centering in the grid
      x_offset = x_spacing / 2.0
      y_offset = y_spacing / 2.0

      call srand(3512341)

      k = 1
      do j = 1, n_rows
            do i = 1, n_cols
                  r_curr(k, 1) = (i-1)*x_spacing+x_offset
                  r_curr(k, 2) = (j-1)*y_spacing+y_offset

                  r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
                  r_curr(k, 2) = r_curr(k,2)+(rand())*spacing

                  theta = 2*pi*rand()

                  v(k, 1) = v0*cos(theta)
                  v(k, 2) = v0*sin(theta)

                  r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
                  r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
                  k=k+1
            end do
      end do

      do i = 1, N
            write(76, *) r_curr(i, 1), r_curr(i, 2)
      end do
      close(76)

      ! Dynamics
      do k = 1, 3200
            t = k * dt
            acc(1) = 0d0
            acc(2) = 0d0
            do i = 1, N
                  acc(1) = 0d0
                  acc(2) = 0d0
                  do j = 1, N
                        if(i /= j) then
                        call compute_acc(N,i,j,L,r_curr,acc, r)
                        end if
                  end do
                  ! UPDATE POSITIONS
                  r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
                  r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)

                  ! APPLY PBC
                  r_next(i,1) = mod(r_next(i,1)+rL, rL)
                  r_next(i,2) = mod(r_next(i,2)+rL, rL)

                  delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
                  delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)

                  ! UPDATE VELOCITIES using adjusted displacements
```

```
                v(i, 1) = delta_r_x / (2 * dt)
                v(i, 2) = delta_r_y / (2 * dt)
            end do
            r_prev(:, 1) = r_curr(:, 1)
            r_prev(:, 2) = r_curr(:, 2)

            r_curr(:, 1) = r_next(:, 1)
            r_curr(:, 2) = r_next(:, 2)

            if(k < 21) then
                do i = 1, N
                    write(77,*) k, r_curr(i,1),r_curr(i,2)
                end do
            else if (k > 40 .and. k < 81 .and. mod(k,3)==0) then
                do i = 1, N
                    write(78,*) k, r_curr(i,1),r_curr(i,2)
                end do
            else if (k > 2600 .and. k < 3200 .and. mod(k,10)==0) then
                do i = 1, N
                    write(79,*) k, r_curr(i,1),r_curr(i,2)
                end do
            end if
        end do
        close(77)
        close(78)
        close(79)
        end
        function delta_pbc(r_next, r_prev,L)
            implicit real*8(a-h, o-y)
            delta_pbc = r_next - r_prev
            delta_pbc = delta_pbc - L * nint(delta_pbc / L)
        end function delta_pbc
        ! Updates acceleration a = ax, ay
        ! between particle i and all others
        subroutine compute_acc(N,i,j,L,r_curr,acc, r)
            implicit real*8(a-h, o-y)
            dimension r_curr(20, 2)
            dimension acc(2)
            dimension r(20, 20)
            epsilon = 1e-3

            dx = r_curr(i, 1) - r_curr(j, 1)
            dy = r_curr(i, 2) - r_curr(j, 2)

            dx = dx - L * nint(dx / L)
            dy = dy - L * nint(dy / L)

            r_ij = sqrt(dx**2 + dy**2)

            r(i, j) = r_ij
            r(j, i) = r_ij

            if(r_ij > epsilon .and. r_ij <= 3d0) then
                F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
                acc(1) = acc(1) + F * dx / r_ij
                acc(2) = acc(2) + F * dy / r_ij
            end if
        end subroutine compute_acc

        subroutine compute_energy(N, L, v, r_curr, E, r)
            implicit real*8(a-h, o-y)
            dimension v(20, 2)
            dimension r_curr(20, 2)
            dimension r(20, 20)
```

```
164
165             epsilon = 1e-3
166             Tk = 0d0
167             do i = 1, N
168                 Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
169             end do
170             U = 0d0
171             do i = 1, N
172               do j = i + 1, N
173                   r_ij = r(i, j)
174
175                   if (r_ij > epsilon .and. r_ij <= 3d0) then
176                       U = U + 4 * (r_ij**(-12) - r_ij**(-6))
177                   end if
178               end do
179             end do
180             E = Tk + U
181       end subroutine
```

7 Tarefa F

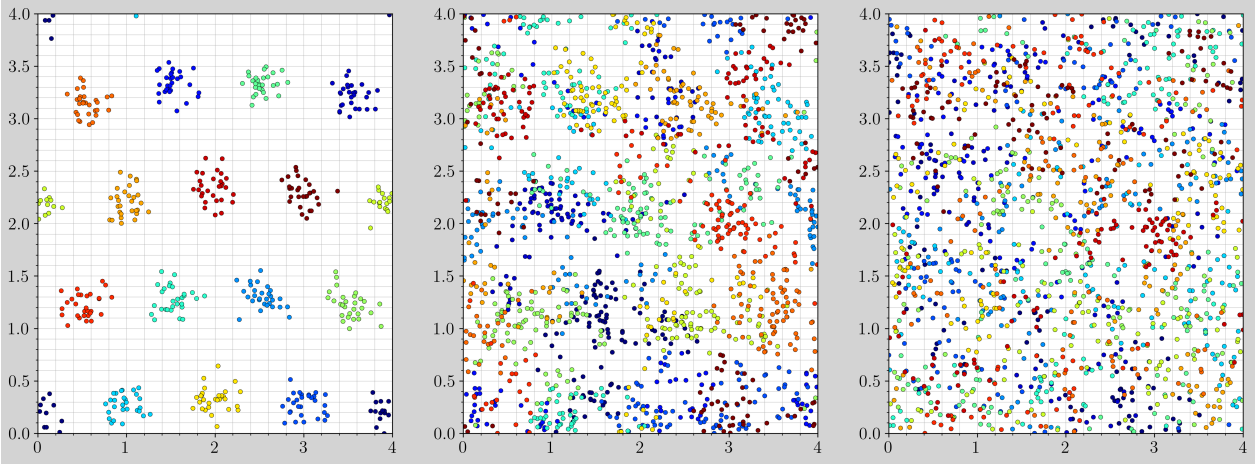

Figura 7

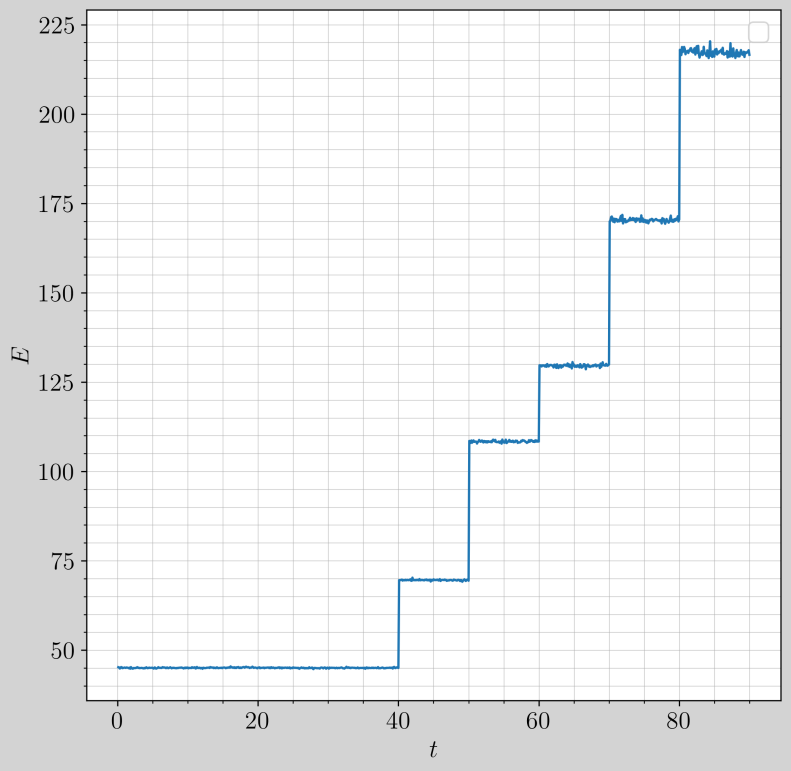

Figura 8: Energia total a cada tempo.

Código

```
1     ! Tarefa F
2     implicit real*8(a-h, o-y)
3     parameter (pi = acos(-1.e0))
4     dimension r_prev(20, 2)
5     dimension r_curr(20, 2)
6     dimension r_next(20, 2)
7     dimension v(20, 2)
8     dimension acc(2)
9     dimension r(20, 20)
10
```

```
11
12      ! TAREFA F
13      open(unit=85, file="saidas/tarefa-F/parametros.dat")
14      open(unit=86, file="saidas/tarefa-F/posicoes-iniciais.dat")
15      open(unit=87, file="saidas/tarefa-F/evolucao-posicoes-1.dat")
16      open(unit=88, file="saidas/tarefa-F/evolucao-posicoes-2.dat")
17      open(unit=89, file="saidas/tarefa-F/evolucao-posicoes-3.dat")
18      open(unit=90, file="saidas/tarefa-F/energia.dat")
19
20      ! Reset variables:
21      r_prev = 0
22      r_curr = 0
23      r_next = 0
24      v = 0
25
26      L = 4
27      rL = 4d0
28      N = 16
29
30      dt = 5e-3
31      v0 = 0.2
32
33      write(85, *) N, L, v0, dt
34      close(85)
35
36      ! Initialize particles
37
38      n_cols = ceiling(sqrt(N*1d0))
39      n_rows = ceiling((N*1d0)/(n_cols*1d0))
40
41      ! Spacing 1/4
42      x_spacing = L/(1d0*n_cols)
43      y_spacing = L/(1d0*n_rows)
44      spacing = min(x_spacing, y_spacing)/4.0
45
46      ! Centering in the grid
47      x_offset = x_spacing / 2.0
48      y_offset = y_spacing / 2.0
49
50      call srand(3512341)
51
52      k = 1
53      do j = 1, n_rows
54            do i = 1, n_cols
55                  r_curr(k, 1) = (i-1)*x_spacing+x_offset
56                  r_curr(k, 2) = (j-1)*y_spacing+y_offset
57
58                  r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
59                  r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
60
61                  theta = 2*pi*rand()
62
63                  v(k, 1) = v0*cos(theta)
64                  v(k, 2) = v0*sin(theta)
65
66                  r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
67                  r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
68                  k=k+1
69            end do
70      end do
71
72      do i = 1, N
73            write(76, *) r_curr(i, 1), r_curr(i, 2)
74      end do
75      close(76)
```

```
76
77      ! Dynamics
78      do k = 1, 18000
79            t = k * dt
80            acc(1) = 0d0
81            acc(2) = 0d0
82
83            E_k = 0d0
84            U = 0d0
85
86            do i = 1, N
87                  acc(1) = 0d0
88                  acc(2) = 0d0
89                  do j = 1, N
90                        if(i /= j) then
91                              call compute_acc(N,i,j,L,r_curr,acc, r)
92                        end if
93                  end do
94                  ! UPDATE POSITIONS
95                  r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
96                  r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)
97
98                  ! APPLY PBC
99                  r_next(i,1) = mod(r_next(i,1)+rL, rL)
100                 r_next(i,2) = mod(r_next(i,2)+rL, rL)
101
102                 delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
103                 delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)
104
105                 ! UPDATE VELOCITIES using adjusted displacements
106                 v(i, 1) = delta_r_x / (2 * dt)
107                 v(i, 2) = delta_r_y / (2 * dt)
108
109           end do
110
111           r_prev(:, 1) = r_curr(:, 1)
112           r_prev(:, 2) = r_curr(:, 2)
113
114           r_curr(:, 1) = r_next(:, 1)
115           r_curr(:, 2) = r_next(:, 2)
116
117           if(mod(k, 20) == 0) then
118                 if(k > 16000) then
119                       ! Liquid end
120                       do i = 1, N
121                             write(89,*) k, r_curr(i,1),r_curr(i,2)
122                       end do
123                 else if (k > 10000 .and. k < 12000) then
124                       !  Liquit initial
125                       do i = 1, N
126                             write(88,*) k, r_curr(i,1),r_curr(i,2)
127                       end do
128                 else if (k > 2600 .and. k < 3200) then
129                       ! Crystal
130                       do i = 1, N
131                             write(87,*) k, r_curr(i,1),r_curr(i,2)
132                       end do
133                 end if
134
135           end if
136
137           ! Increase velocity
138           if(mod(k, 2000) == 0 .and. k > 7000) then
139                 do i = 1, N
140                 r_prev(i,1)=r_curr(i,1)-(r_curr(i,1)-r_prev(i,1))*1.5
```

```
                end do
            end if

            if (mod(k,20) == 0) then
                E = 0d0
                call compute_energy(N, L, v, r_curr, E, r)
                write(90,*) k, E
            end if
        end do
        close(85)
        close(86)
        close(87)
        close(88)
        close(89)
        close(90)
        end
        ! Submodules for molecular dynamic simulations
        ! Velocity delta
        function delta_pbc(r_next, r_prev,L)
            implicit real*8(a-h, o-y)
            delta_pbc = r_next - r_prev
            delta_pbc = delta_pbc - L * nint(delta_pbc / L)
        end function delta_pbc

        subroutine initialize_particles(N, L, r_curr,r_prev, v, v0)
            implicit real*8(a-h, o-y)
            dimension r_prev(20, 2)
            dimension r_curr(20, 2)
            dimension v(20, 2)

            ! Defining # rows/columns
            n_cols = ceiling(sqrt(N*1d0))
            n_rows = ceiling((N*1d0)/(n_cols*1d0))

            ! Spacing 1/4
            x_spacing = L/(1d0*n_cols)
            y_spacing = L/(1d0*n_rows)
            spacing = min(x_spacing, y_spacing)/4.0

            ! Centering in the grid
            x_offset = x_spacing / 2.0
            y_offset = y_spacing / 2.0
            call srand(562369)

            k = 1
            do j = 1, n_rows
                do i = 1, n_cols
                    r_curr(k, 1) = (i-1)*x_spacing+x_offset
                    r_curr(k, 2) = (j-1)*y_spacing+y_offset

                    r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
                    r_curr(k, 2) = r_curr(k,2)+(rand())*spacing

                    theta = 2*pi*rand()
                    v(k, 1) = v0*cos(theta)
                    v(k, 2) = v0*sin(theta)

                    r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
                    r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
                    k=k+1
                end do
            end do
        end subroutine initialize_particles

```

```
    ! Updates acceleration a = ax, ay
    ! between particle i and all others
    subroutine compute_acc(N,i,j,L,r_curr,acc, r)
          implicit real*8(a-h, o-y)
          dimension r_curr(20, 2)
          dimension acc(2)
          dimension r(20, 20)
          epsilon = 1e-3

          dx = r_curr(i, 1) - r_curr(j, 1)
          dy = r_curr(i, 2) - r_curr(j, 2)

          dx = dx - L * nint(dx / L)
          dy = dy - L * nint(dy / L)

          r_ij = sqrt(dx**2 + dy**2)

          r(i, j) = r_ij
          r(j, i) = r_ij

          if(r_ij > epsilon .and. r_ij <= 3d0) then
                F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
                acc(1) = acc(1) + F * dx / r_ij
                acc(2) = acc(2) + F * dy / r_ij
          end if
    end subroutine compute_acc

    subroutine compute_energy(N, L, v, r_curr, E, r)
          implicit real*8(a-h, o-y)
          dimension v(20, 2)
          dimension r_curr(20, 2)
          dimension r(20, 20)

          epsilon = 1e-3
          Tk = 0d0
          do i = 1, N
              Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
          end do
          U = 0d0
          do i = 1, N
            do j = i + 1, N
                r_ij = r(i, j)

                if (r_ij > epsilon .and. r_ij <= 3d0) then
                    U = U + 4 * (r_ij**(-12) - r_ij**(-6))
                end if
            end do
          end do
          E = Tk + U
    end subroutine compute_energy
```

8 Conclusão?

Por fim, o arquivo `tests.f` é um compilado de todas simulações desenvolvidas nesse projeto e pode ser compilado para rodar todas elas de uma vez. Além disso o script `plots.py` cria todos os gráficos e gifs desse trabalho.

Referências

[1] N.J. Giordano. *Computational Physics*. Prentice Hall, 2006. ISBN: 9780133677232.

[2] Wikipedia contributors. *Periodic boundary conditions — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-June-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Periodic_boundary_conditions&oldid=1229053854.