

Projeto 6 - Dinâmica molecular

Jefter Santiago (12559016)

6 de julho de 2024

1 Introdução

Potencial de Lennard-Jonnes

$$\mathcal{U}(\vec{r}) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad (1)$$

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

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

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

$$t = n\Delta t \quad (3)$$

$$x_i(n+1) = 2x_i(n) - x_i(n-1) + a_i^x(n)(\Delta t)^2, n \geq 1 \quad (4)$$

$$y_i(n+1) = 2y_i(n) - y_i(n-1) + a_i^y(n)(\Delta t)^2, n \geq 1 \quad (5)$$

$$x_i(1) = x(0) + v_i^x(0)\Delta t \quad (6)$$

$$y_i(1) = y(0) + v_i^y(0)\Delta t \quad (7)$$

$$v_i^x(n) = (x_i(n+1) - x_i(n-1)) / (2\Delta t) \quad (8)$$

$$v_i^y(n) = (y_i(n+1) - y_i(n-1)) / (2\Delta t) \quad (9)$$

1.1 Detalhes de implementação

Módulo para simulações de dinâmica molecular

```
1  ! Submodules for molecular dynamic simulations
2
3  ! Velocity delta
4  function delta_pbc(r_next, r_prev, L)
5      implicit real*8(a-h, o-y)
6      delta_pbc = r_next - r_prev
7      delta_pbc = delta_pbc - L * nint(delta_pbc / L)
8  end function delta_pbc
9
10 subroutine initialize_particles(N, L, r_curr, r_prev, v, v0)
11     implicit real*8(a-h, o-y)
```

```

12     dimension r_prev(20, 2)
13     dimension r_curr(20, 2)
14     dimension v(20, 2)
15
16     ! Defining # rows/columns
17     n_cols = ceiling(sqrt(N*1d0))
18     n_rows = ceiling((N*1d0)/(n_cols*1d0))
19
20     ! Spacing 1/4
21     x_spacing = L/(1d0*n_cols)
22     y_spacing = L/(1d0*n_rows)
23     spacing = min(x_spacing, y_spacing)/4.0
24
25     ! Centering in the grid
26     x_offset = x_spacing / 2.0
27     y_offset = y_spacing / 2.0
28     call srand(562369)
29
30     k = 1
31     do j = 1, n_rows
32         do i = 1, n_cols
33             r_curr(k, 1) = (i-1)*x_spacing+x_offset
34             r_curr(k, 2) = (j-1)*y_spacing+y_offset
35
36             r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
37             r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
38
39             theta = 2*pi*rand()
40             v(k, 1) = v0*cos(theta)
41             v(k, 2) = v0*sin(theta)
42
43             r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
44             r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
45             k=k+1
46         end do
47     end do
48     end subroutine initialize_particles
49
50     ! Updates acceleration a = ax, ay
51     ! between particle i and all others
52     subroutine compute_acc(N,i,j,L,r_curr,acc, r)
53         implicit real*8(a-h, o-y)
54         dimension r_curr(20, 2)
55         dimension acc(2)
56         dimension r(20, 20)
57         epsilon = 1e-3
58
59         dx = r_curr(i, 1) - r_curr(j, 1)
60         dy = r_curr(i, 2) - r_curr(j, 2)
61
62         dx = dx - L * nint(dx / L)
63         dy = dy - L * nint(dy / L)
64
65         r_ij = sqrt(dx**2 + dy**2)
66
67         r(i, j) = r_ij
68         r(j, i) = r_ij
69
70         if(r_ij > epsilon .and. r_ij <= 3d0) then
71             F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
72             acc(1) = acc(1) + F * dx / r_ij
73             acc(2) = acc(2) + F * dy / r_ij
74         end if
75     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

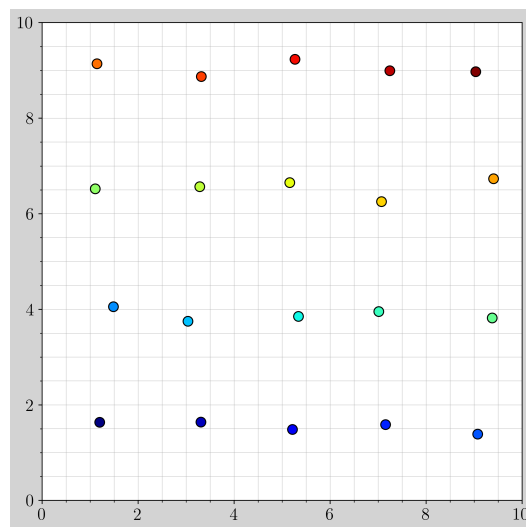


Figura 1: Posições iniciais das partículas.

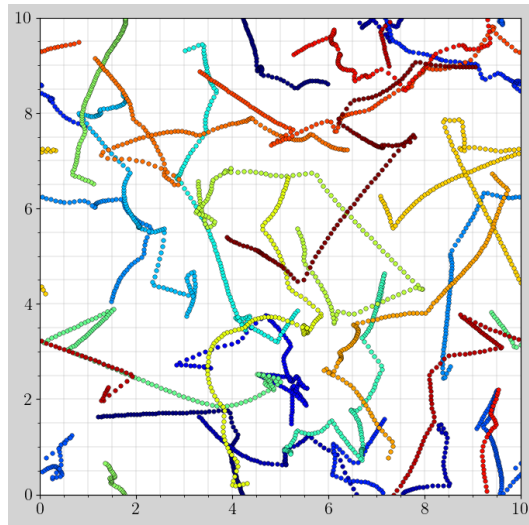


Figura 2: Coordenadas das partículas projetadas à cada $3\Delta t$.

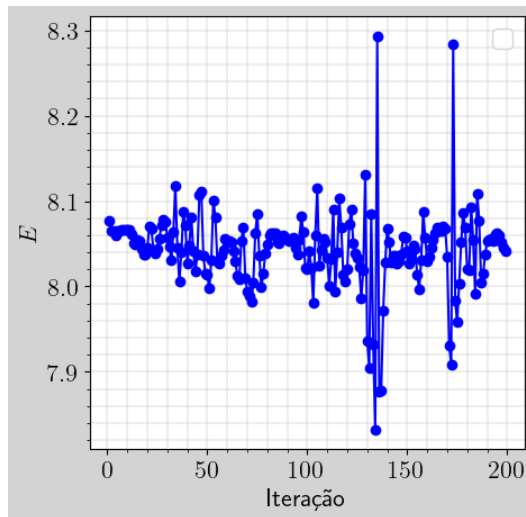


Figura 3: Energia do sistema à cada $3\Delta t$.

Código

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

```
1      ! Tarefas A, B e parte da D
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 r(20, 20)
9      dimension acc(2)
10     L = 10
11     rL = 10d0
12     N = 20
13
14     ! Tarefa A
15     open(unit = 99, file="saidas/tarefa-A/parametros.dat")
16     open(unit = 1, file="saidas/tarefa-A/posicoes-iniciais.dat")
17     open(unit = 3, file="saidas/tarefa-A/evolucao-posicoes.dat")
18     ! Tarefa B
19     open(unit = 5, file="saidas/tarefa-B/velocidades.dat")
20     open(unit = 6, file="saidas/tarefa-B/evolucao-posicoes.dat")
21     ! Tarefa D
22     open(unit = 9, file="saidas/tarefa-D/temperatura-b.dat")
23
24     dt = 0.02
25     v0 = 1.0
26     write(99, *) N, L, v0, dt
27     close(99)
28     ! Defining # rows/columns
29     n_cols = ceiling(sqrt(N*1d0))
30     n_rows = ceiling((N*1d0)/(n_cols*1d0))
31
32     ! Spacing 1/4
33     x_spacing = L/(1d0*n_cols)
34     y_spacing = L/(1d0*n_rows)
35     spacing = min(x_spacing, y_spacing)/4.0
36
37     ! Centering in the grid
38     x_offset = x_spacing / 2.0
39     y_offset = y_spacing / 2.0
40     call srand(562369)
41
42     k = 1
43     do j = 1, n_rows
44         do i = 1, n_cols
45             r_curr(k, 1) = (i-1)*x_spacing+x_offset
46             r_curr(k, 2) = (j-1)*y_spacing+y_offset
47
48             r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
49             r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
50             theta = 2*pi*rand()
51             v(k, 1) = v0*cos(theta)
52             v(k, 2) = v0*sin(theta)
53
54             r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
55             r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
56             k=k+1
57         end do
58     end do
59
60     do i = 1, N
61         write(1, *) r_curr(i, 1), r_curr(i, 2)
```

```

62     write(3, *) 0d0, r_curr(i,1), r_curr(i, 2)
63 end do
64 close(1)
65
66 DB = 1.0
67 ! Dynamics
68 do k = 1, 5000
69
70     t = k * dt
71
72     acc(1) = 0d0
73     acc(2) = 0d0
74
75     do i = 1, N
76         acc(1) = 0d0
77         acc(2) = 0d0
78         do j = 1, N
79             if(i /= j) then
80                 call compute_acc(N,i,j,L,r_curr,acc,r)
81             end if
82         end do
83         ! UPDATE POSITIONS
84         r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
85         r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)
86
87         ! APPLY PBC
88         r_next(i,1) = mod(r_next(i,1)+rL, rL)
89         r_next(i,2) = mod(r_next(i,2)+rL, rL)
90
91         delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
92         delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)
93
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     r_prev(:, 1) = r_curr(:, 1)
100    r_prev(:, 2) = r_curr(:, 2)
101
102    r_curr(:, 1) = r_next(:, 1)
103    r_curr(:, 2) = r_next(:, 2)
104
105    if(k < 200) then
106        E = 0d0
107        call compute_energy(N, L, v, r_curr, E, r)
108        write(19,*) k, E
109    end if
110
111    ! TAREFA A
112    if(mod(k, 3) == 0 .and. k < 400) then
113        do i = 1, N
114            write(3,*) k, r_curr(i,1),r_curr(i, 2)
115        end do
116    end if
117
118    ! Tarefa B & D
119    if(mod(k, 20) == 0) then
120        do i = 1, N
121            v_mag = sqrt(v(i,1)**2+v(i,2)**2)
122            write(5,*) k, v_mag, v(i,1), v(i,2)
123            write(9,*) .5d0 * v_mag**2
124            write(6,*) k, r_curr(i,1),r_curr(i, 2)
125        end do

```

```

126     end if
127 end do
128 close(3)
129 close(5)
130 close(6)
131 close(9)
132 close(15)
133
134 end
135 ! Submodules for molecular dynamic simulations
136 ! Velocity delta
137 function delta_pbc(r_next, r_prev,L)
138     implicit real*8(a-h, o-y)
139     delta_pbc = r_next - r_prev
140     delta_pbc = delta_pbc - L * nint(delta_pbc / L)
141 end function delta_pbc
142
143 subroutine initialize_particles(N, L, r_curr,r_prev, v, v0)
144     implicit real*8(a-h, o-y)
145     dimension r_prev(20, 2)
146     dimension r_curr(20, 2)
147     dimension v(20, 2)
148
149     ! Defining # rows/columns
150     n_cols = ceiling(sqrt(N*1d0))
151     n_rows = ceiling((N*1d0)/(n_cols*1d0))
152
153     ! Spacing 1/4
154     x_spacing = L/(1d0*n_cols)
155     y_spacing = L/(1d0*n_rows)
156     spacing = min(x_spacing, y_spacing)/4.0
157
158     ! Centering in the grid
159     x_offset = x_spacing / 2.0
160     y_offset = y_spacing / 2.0
161     call srand(562369)
162
163     k = 1
164     do j = 1, n_rows
165         do i = 1, n_cols
166             r_curr(k, 1) = (i-1)*x_spacing+x_offset
167             r_curr(k, 2) = (j-1)*y_spacing+y_offset
168
169             r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
170             r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
171
172             theta = 2*pi*rand()
173             v(k, 1) = v0*cos(theta)
174             v(k, 2) = v0*sin(theta)
175
176             r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
177             r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
178             k=k+1
179         end do
180     end do
181 end subroutine initialize_particles
182
183 ! Updates acceleration a = ax, ay
184 ! between particle i and all others
185 subroutine compute_acc(N,i,j,L,r_curr,acc, r)
186     implicit real*8(a-h, o-y)
187     dimension r_curr(20, 2)
188     dimension acc(2)
189     dimension r(20, 20)

```

```

190     epsilon = 1e-3
191
192     dx = r_curr(i, 1) - r_curr(j, 1)
193     dy = r_curr(i, 2) - r_curr(j, 2)
194
195     dx = dx - L * nint(dx / L)
196     dy = dy - L * nint(dy / L)
197
198     r_ij = sqrt(dx**2 + dy**2)
199
200     r(i, j) = r_ij
201     r(j, i) = r_ij
202
203     if(r_ij > epsilon .and. r_ij <= 3d0) then
204         F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
205         acc(1) = acc(1) + F * dx / r_ij
206         acc(2) = acc(2) + F * dy / r_ij
207     end if
208 end subroutine compute_acc
209
210 subroutine compute_energy(N, L, v, r_curr, E, r)
211     implicit real*8(a-h, o-y)
212     dimension v(20, 2)
213     dimension r_curr(20, 2)
214     dimension r(20, 20)
215
216     epsilon = 1e-3
217     Tk = 0d0
218     do i = 1, N
219         Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
220     end do
221     U = 0d0
222     do i = 1, N
223         do j = i + 1, N
224             r_ij = r(i, j)
225
226             if (r_ij > epsilon .and. r_ij <= 3d0) then
227                 U = U + 4 * (r_ij**(-12) - r_ij**(-6))
228             end if
229         end do
230     end do
231     E = Tk + U
232 end subroutine

```


3 Tarefa B

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

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

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

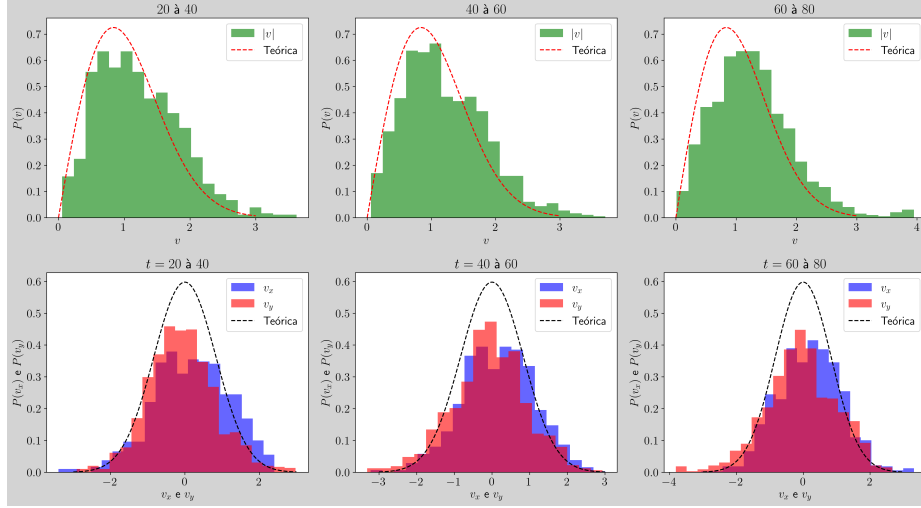


Figura 4: Distribuição da velocidade, magnitude e componentes em intervalos $t = 20 - 40$, $t = 40 - 60$ e $t = 60 - 80$.

Além disso há um gif para essa simulação.

4 Tarefa C

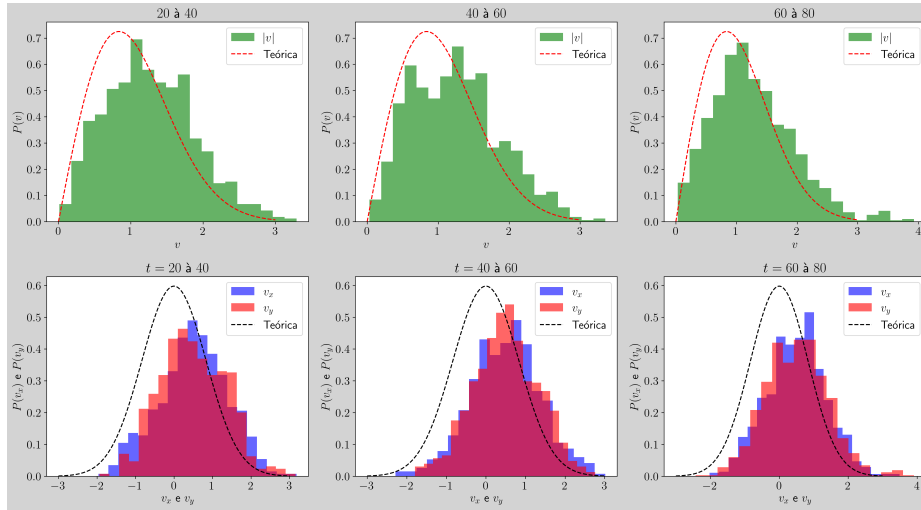


Figura 5: Distribuição da velocidade, magnitude e componentes em intervalos $t = 20 - 40$, $t = 40 - 60$ e $t = 60 - 80$.

Código

Nesse código temos a simulação desse item, considerando 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)

```

```

101     r_prev(:, 2) = r_curr(:, 2)
102
103     r_curr(:, 1) = r_next(:, 1)
104     r_curr(:, 2) = r_next(:, 2)
105
106     if(mod(k, 20) == 0) then
107         do i = 1, N
108             v_mag = sqrt(v(i,1)**2+v(i,2)**2)
109             write(7,*) k, v_mag, v(i,1), v(i,2)
110             write(8,*) .5d0 * v_mag**2
111             write(4,*) k, r_curr(i,1),r_curr(i, 2)
112         end do
113     end if
114 end do
115 close(4)
116 close(8)
117 end
118
119 function delta_pbc(r_next, r_prev,L)
120     implicit real*8(a-h, o-y)
121     delta_pbc = r_next - r_prev
122     delta_pbc = delta_pbc - L * nint(delta_pbc / L)
123 end function delta_pbc
124
125
126
127 ! Updates acceleration a = ax, ay
128 ! between particle i and all others
129 subroutine compute_acc(N,i,j,L,r_curr,acc, r)
130     implicit real*8(a-h, o-y)
131     dimension r_curr(20, 2)
132     dimension acc(2)
133     dimension r(20, 20)
134     epsilon = 1e-3
135
136     dx = r_curr(i, 1) - r_curr(j, 1)
137     dy = r_curr(i, 2) - r_curr(j, 2)
138
139     dx = dx - L * nint(dx / L)
140     dy = dy - L * nint(dy / L)
141
142     r_ij = sqrt(dx**2 + dy**2)
143
144     r(i, j) = r_ij
145     r(j, i) = r_ij
146
147     if(r_ij > epsilon .and. r_ij <= 3d0) then
148         F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
149         acc(1) = acc(1) + F * dx / r_ij
150         acc(2) = acc(2) + F * dy / r_ij
151     end if
152 end subroutine compute_acc
153
154 subroutine compute_energy(N, L, v, r_curr, E, r)
155     implicit real*8(a-h, o-y)
156     dimension v(20, 2)
157     dimension r_curr(20, 2)
158     dimension r(20, 20)
159
160     epsilon = 1e-3
161     Tk = 0d0
162     do i = 1, N
163         Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
164     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

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

6 Tarefa E

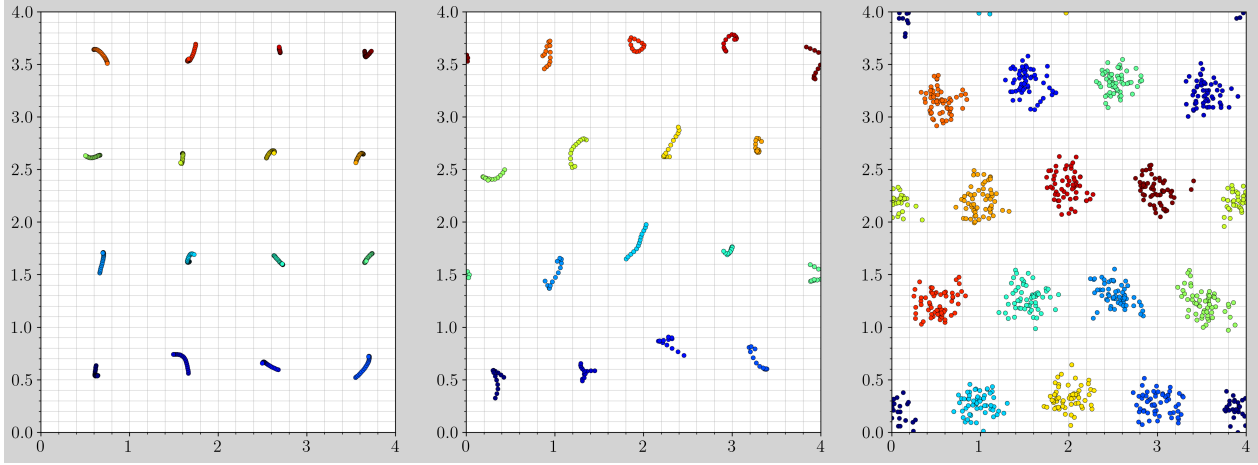


Figura 6

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
```

```

35
36 n_cols = ceiling(sqrt(N*1d0))
37 n_rows = ceiling((N*1d0)/(n_cols*1d0))
38
39 ! Spacing 1/4
40 x_spacing = L/(1d0*n_cols)
41 y_spacing = L/(1d0*n_rows)
42 spacing = min(x_spacing, y_spacing)/4.0
43
44 ! Centering in the grid
45 x_offset = x_spacing / 2.0
46 y_offset = y_spacing / 2.0
47
48 call srand(3512341)
49
50 k = 1
51 do j = 1, n_rows
52     do i = 1, n_cols
53         r_curr(k, 1) = (i-1)*x_spacing+x_offset
54         r_curr(k, 2) = (j-1)*y_spacing+y_offset
55
56         r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
57         r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
58
59         theta = 2*pi*rand()
60
61         v(k, 1) = v0*cos(theta)
62         v(k, 2) = v0*sin(theta)
63
64         r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
65         r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
66         k=k+1
67     end do
68 end do
69
70 do i = 1, N
71     write(76, *) r_curr(i, 1), r_curr(i, 2)
72 end do
73 close(76)
74
75 ! Dynamics
76 do k = 1, 3200
77     t = k * dt
78     acc(1) = 0d0
79     acc(2) = 0d0
80     do i = 1, N
81         acc(1) = 0d0
82         acc(2) = 0d0
83         do j = 1, N
84             if(i /= j) then
85                 call compute_acc(N,i,j,L,r_curr,acc, r)
86             end if
87         end do
88         ! UPDATE POSITIONS
89         r_next(i,1) = 2*r_curr(i,1)-r_prev(i,1)+acc(1)*(dt**2)
90         r_next(i,2) = 2*r_curr(i,2)-r_prev(i,2)+acc(2)*(dt**2)
91
92         ! APPLY PBC
93         r_next(i,1) = mod(r_next(i,1)+rL, rL)
94         r_next(i,2) = mod(r_next(i,2)+rL, rL)
95
96         delta_r_x = delta_pbc(r_next(i,1),r_prev(i,1),L)
97         delta_r_y = delta_pbc(r_next(i,2),r_prev(i,2),L)
98
99         ! UPDATE VELOCITIES using adjusted displacements

```

```

100         v(i, 1) = delta_r_x / (2 * dt)
101         v(i, 2) = delta_r_y / (2 * dt)
102     end do
103     r_prev(:, 1) = r_curr(:, 1)
104     r_prev(:, 2) = r_curr(:, 2)
105
106     r_curr(:, 1) = r_next(:, 1)
107     r_curr(:, 2) = r_next(:, 2)
108
109     if(k < 21) then
110         do i = 1, N
111             write(77,*) k, r_curr(i,1),r_curr(i,2)
112         end do
113     else if (k > 40 .and. k < 81 .and. mod(k,3)==0) then
114         do i = 1, N
115             write(78,*) k, r_curr(i,1),r_curr(i,2)
116         end do
117     else if (k > 2600 .and. k < 3200 .and. mod(k,10)==0) then
118         do i = 1, N
119             write(79,*) k, r_curr(i,1),r_curr(i,2)
120         end do
121     end if
122 end do
123 close(77)
124 close(78)
125 close(79)
126 end
127 function delta_pbc(r_next, r_prev,L)
128     implicit real*8(a-h, o-y)
129     delta_pbc = r_next - r_prev
130     delta_pbc = delta_pbc - L * nint(delta_pbc / L)
131 end function delta_pbc
132 ! Updates acceleration a = ax, ay
133 ! between particle i and all others
134 subroutine compute_acc(N,i,j,L,r_curr,acc, r)
135     implicit real*8(a-h, o-y)
136     dimension r_curr(20, 2)
137     dimension acc(2)
138     dimension r(20, 20)
139     epsilon = 1e-3
140
141     dx = r_curr(i, 1) - r_curr(j, 1)
142     dy = r_curr(i, 2) - r_curr(j, 2)
143
144     dx = dx - L * nint(dx / L)
145     dy = dy - L * nint(dy / L)
146
147     r_ij = sqrt(dx**2 + dy**2)
148
149     r(i, j) = r_ij
150     r(j, i) = r_ij
151
152     if(r_ij > epsilon .and. r_ij <= 3d0) then
153         F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
154         acc(1) = acc(1) + F * dx / r_ij
155         acc(2) = acc(2) + F * dy / r_ij
156     end if
157 end subroutine compute_acc
158
159 subroutine compute_energy(N, L, v, r_curr, E, r)
160     implicit real*8(a-h, o-y)
161     dimension v(20, 2)
162     dimension r_curr(20, 2)
163     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

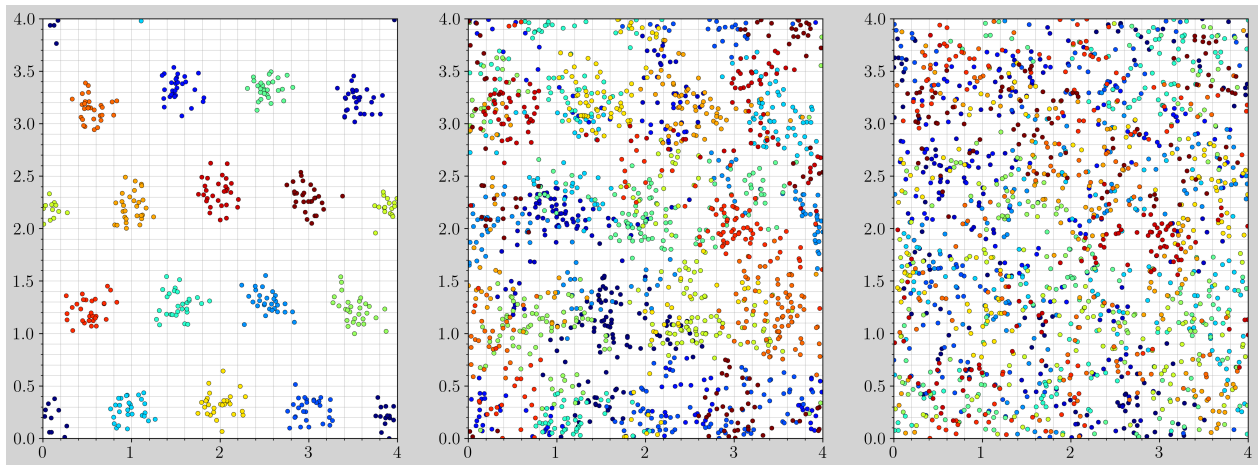


Figura 7

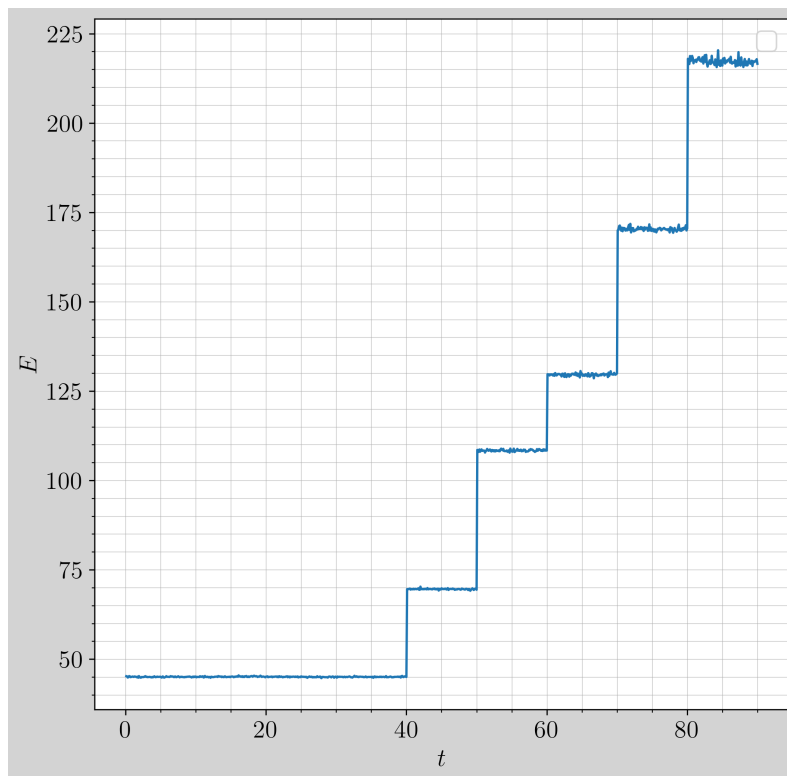


Figura 8: Energia total a cada tempo.

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            ! Liquid 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    end if
135
136    ! Increase velocity
137    if(mod(k, 2000) == 0 .and. k > 7000) then
138        do i = 1, N
139            r_prev(i,1)=r_curr(i,1)-(r_curr(i,1)-r_prev(i,1))*1.5

```

```

141         end do
142     end if
143
144     if (mod(k,20) == 0) then
145         E = 0d0
146         call compute_energy(N, L, v, r_curr, E, r)
147         write(90,*) k, E
148     end if
149 end do
150 close(85)
151 close(86)
152 close(87)
153 close(88)
154 close(89)
155 close(90)
156 end
157 ! Submodules for molecular dynamic simulations
158 ! Velocity delta
159 function delta_pbc(r_next, r_prev,L)
160     implicit real*8(a-h, o-y)
161     delta_pbc = r_next - r_prev
162     delta_pbc = delta_pbc - L * nint(delta_pbc / L)
163 end function delta_pbc
164
165 subroutine initialize_particles(N, L, r_curr,r_prev, v, v0)
166     implicit real*8(a-h, o-y)
167     dimension r_prev(20, 2)
168     dimension r_curr(20, 2)
169     dimension v(20, 2)
170
171     ! Defining # rows/columns
172     n_cols = ceiling(sqrt(N*1d0))
173     n_rows = ceiling((N*1d0)/(n_cols*1d0))
174
175     ! Spacing 1/4
176     x_spacing = L/(1d0*n_cols)
177     y_spacing = L/(1d0*n_rows)
178     spacing = min(x_spacing, y_spacing)/4.0
179
180     ! Centering in the grid
181     x_offset = x_spacing / 2.0
182     y_offset = y_spacing / 2.0
183     call srand(562369)
184
185     k = 1
186     do j = 1, n_rows
187         do i = 1, n_cols
188             r_curr(k, 1) = (i-1)*x_spacing+x_offset
189             r_curr(k, 2) = (j-1)*y_spacing+y_offset
190
191             r_curr(k, 1) = r_curr(k,1)+(rand())*spacing
192             r_curr(k, 2) = r_curr(k,2)+(rand())*spacing
193
194             theta = 2*pi*rand()
195             v(k, 1) = v0*cos(theta)
196             v(k, 2) = v0*sin(theta)
197
198             r_prev(k, 1) = r_curr(k, 1) - v(k, 1) * dt
199             r_prev(k, 2) = r_curr(k, 2) - v(k, 2) * dt
200             k=k+1
201         end do
202     end do
203 end subroutine initialize_particles
204

```

```

205 ! Updates acceleration a = ax, ay
206 ! between particle i and all others
207 subroutine compute_acc(N,i,j,L,r_curr,acc, r)
208     implicit real*8(a-h, o-y)
209     dimension r_curr(20, 2)
210     dimension acc(2)
211     dimension r(20, 20)
212     epsilon = 1e-3
213
214     dx = r_curr(i, 1) - r_curr(j, 1)
215     dy = r_curr(i, 2) - r_curr(j, 2)
216
217     dx = dx - L * nint(dx / L)
218     dy = dy - L * nint(dy / L)
219
220     r_ij = sqrt(dx**2 + dy**2)
221
222     r(i, j) = r_ij
223     r(j, i) = r_ij
224
225     if(r_ij > epsilon .and. r_ij <= 3d0) then
226         F = 24.0 * (2d0/r_ij**13 - 1d0/r_ij**7)
227         acc(1) = acc(1) + F * dx / r_ij
228         acc(2) = acc(2) + F * dy / r_ij
229     end if
230 end subroutine compute_acc
231
232 subroutine compute_energy(N, L, v, r_curr, E, r)
233     implicit real*8(a-h, o-y)
234     dimension v(20, 2)
235     dimension r_curr(20, 2)
236     dimension r(20, 20)
237
238     epsilon = 1e-3
239     Tk = 0d0
240     do i = 1, N
241         Tk = Tk + 0.5 * (v(i, 1)**2 + v(i, 2)**2)
242     end do
243     U = 0d0
244     do i = 1, N
245         do j = i + 1, N
246             r_ij = r(i, j)
247
248             if (r_ij > epsilon .and. r_ij <= 3d0) then
249                 U = U + 4 * (r_ij**(-12) - r_ij**(-6))
250             end if
251         end do
252     end do
253     E = Tk + U
254 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.