

Física estatística computacional



# MODELOS DE CRESCIMENTO

**Aluno**

Leonardo Vaz Ferreira n° 13862330

São Carlos  
2025

# 1 Tarefa 1

Nessa primeira parte do projeto, foi proposto a criação de um programa em fortran para analisar a evolução temporal de um sistema uni-dimensional utilizando as regras binárias com condições de contorno periódicas. Dessa maneira foi elaborado o seguinte código:

```
1 program ACD
2   implicit none
3   integer, parameter :: N = 200
4   integer :: i, t, rule(8), L(N), new_L(N)
5
6   open(unit=10, file='rule_138_13862338.out')
7
8   L = InitPosition(N)
9   rule = DecomposeRule(138)
10
11  write(10,*)L
12
13  do t = 1, 100
14    new_L(1) = rule(L(N)*2**2 + L(1)*2**1 + L(2)*2**0 + 1)
15
16    do i = 2, N-1
17      new_L(i) = rule(L(i-1)*2**2 + L(i)*2**1 + L(i+1)*2**0 + 1)
18    end do
19
20    new_L(N) = rule(L(N-1)*2**2 + L(N)*2**1 + L(1)*2**0 + 1)
21
22    L = new_L
23
24    write(10,*)L
25  end do
26
27 contains
28
29  function InitPosition(N) result(L)
30    implicit none
31    integer, intent(in) :: N
32    integer :: i
33    real*8 :: L(N)
34
35    call random_number(L)
36
37    do i = 1, N
38      L(i) = int(anint(L(i)))
39    end do
40  end function InitPosition
41
42  function DecomposeRule(x) result(rule)
43    integer, intent(in) :: x
44    integer :: i, new_x
45    integer :: rule(8)
46
47    new_x = x
48
49    do i = 0, 7
50      if (new_x - 2**(7-i) .GE. 0.0d0) then
51        rule(8-i) = 1
52        new_x = new_x - 2**(7-i)
53      else
54        rule(8-i) = 0
55      end if
56    end do
57  end function DecomposeRule
58 end program ACD
59
```

Figura 1: Código da tarefa 1

Com o código, foram testadas as regras (1, 10, 51, 138, 232 e 254), dessa maneira foram obtidos os seguintes resultados:

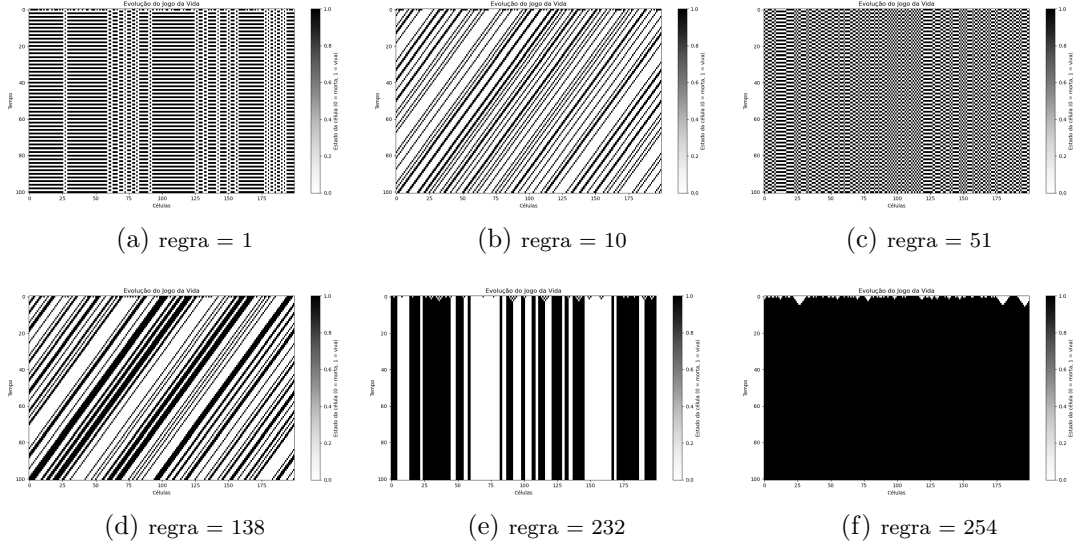


Figura 2: Evolução temporal das regras binárias

Podemos analisar que nas primeira linhas de cada uma das imagens é o estado inicial do sistema e quando vamos abaixando temos a evolução temporal, dessa maneira, todos eles chegam em um certo estado final, onde os padrões se tornam repetitivos, podendo ser estáticos como as regras 232 e 254, padrões repetitivos como as regras 51 e 1, e alguns casos como as regras 10 e 138 onde aparece uma evolução de deslocamento do sistema, mas se mantendo de certa forma parecido com o estágio anterior.

## 2 Tarefa 2

Nessa parte do trabalho foi proposto a criação de um código para analisar a evolução temporal de um modelo DLA, onde uma partícula semente está do meio do sistema e são liberadas partículas ao redor dela e quando encostam elas se juntam formando um aglomerado. Dessa maneira foi criado o seguinte código:

```

1 program DLA
2   implicit none
3   integer, parameter :: N = 200
4   integer, parameter :: num_particles = 2000
5   real*8, parameter :: pi = acos(-1.000)
6   integer :: mesh(N,N), particles(2, num_particles), val(2)
7   integer :: i, cord, val_pos
8   real*8 :: r, w, max_radius, theta, spawn_radius
9
10  open(unit=10, file="saida_1_13862338.out")
11  open(unit=20, file="saida_2_13862338.out")
12
13  mesh = 0
14  val = [-1, 1]
15
16  particles(1, 1) = N / 2
17  particles(2, 1) = N / 2
18
19  max_radius = 0.000
20  mesh(particles(1,1), particles(2,1)) = 1
21
22  do i = 2, num_particles
23    call ShowMesh(mesh, N)
24
25    call random_number(r)
26    spawn_radius = max(max_radius + 5.000, 5.000)
27
28    theta = 2.000 * pi * r
29    particles(1,i) = int(spawn_radius + cos(theta)) * N/2
30    particles(2,i) = int(spawn_radius + sin(theta)) * N/2
31
32    do while (.not. neighbor(particles(1,i), particles(2,i), mesh, N))
33      call random_number(r)
34      call random_number(x)
35
36      cord = int(2 * r) + 1
37      val_pos = int(2 * x) + 1
38
39      particles(cord, i) = particles(cord, i) + val(val_pos)
40
41      if (sqrt((particles(1,i)-N/2.000)**2 + (particles(2,i)-N/2.000)**2) > 2.000 + spawn_radius) then
42        call random_number(r)
43        theta = 2.000 * pi * r
44        particles(1,i) = int(spawn_radius + cos(theta)) * N/2
45        particles(2,i) = int(spawn_radius + sin(theta)) * N/2
46      end if
47    end do
48
49    mesh(particles(1,i), particles(2,i)) = 1
50    max_radius = max(max_radius, sqrt((particles(1,i)-N/2.000)**2 + (particles(2,i)-N/2.000)**2) + 5.000)
51  end do
52
53  call Fractal(mesh, N)
54
55  contains
56
57  subroutine ShowMesh(mesh, N)
58    implicit none
59    integer :: i, j, N
60    integer :: mesh(N,N)
61
62    do i = 1, N
63      do j = 1, N
64        write(10, '(I4)', advance='no') mesh(i,j)
65      end do
66      write(10, *)
67    end do
68  end subroutine ShowMesh
69
70  function neighbor(x, y, mesh, N) result(has_neighbor)
71    implicit none
72    integer :: x, y
73    integer :: i, j, N
74    integer :: mesh(N,N)
75    logical :: has_neighbor
76
77    has_neighbor = .false.
78
79    do i = -1, 1
80      do j = -1, 1
81        if (x+i > 1 .and. x+i < N .and. y+j > 1 .and. y+j < N) then
82          if (mesh(x+i, y+j) == 1) then
83            has_neighbor = .true.
84            return
85          end if
86        end if
87      end do
88    end do
89  end function neighbor
90
91  subroutine Fractal(mesh, N)
92    implicit none
93    integer, intent(in) :: N
94    integer :: mesh(N,N)
95    integer :: i, j, r, max_r, center_x, center_y
96    real*8 :: dist
97    integer, allocatable :: count(:)
98
99    center_x = N / 2
100    center_y = N / 2
101    max_r = N / 2
102
103    allocate(count(8*max_r))
104    count = 0
105
106    do i = 1, N
107      do j = 1, N
108        if (mesh(i,j) == 1) then
109          dist = sqrt(dble(1 - center_x)**2 + dble(j - center_y)**2)
110          if (int(dist) < max_r) then
111            count(int(dist)) = count(int(dist)) + 1
112          end if
113        end if
114      end do
115    end do
116
117    do r = 1, max_r
118      count(r) = count(r) + count(r-1)
119    end do
120
121    do r = 1, max_r
122      write(20, '(I4)', advance='no') r, count(r)
123    end do
124
125    deallocate(count)
126  end subroutine Fractal
127
128 end program DLA
129

```

Figura 3: Código da tarefa 2

Assim foram obtidos os seguintes resultados:

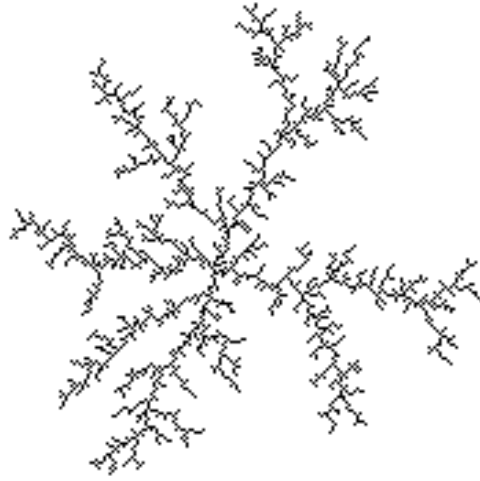


Figura 4: Estágio final da evolução temporal DLA

Podemos analisar o crescimento interessante do aglomerado, criando muitas ramificações a partir do centro onde ele foi iniciado. Podemos além de ver o estágio final, analisar a dimensão fractal do autômato, para isso calculamos a quantidade de partículas ao longo do raio e fazemos um gráfico  $\ln x / \ln$  e calculamos a inclinação da reta.

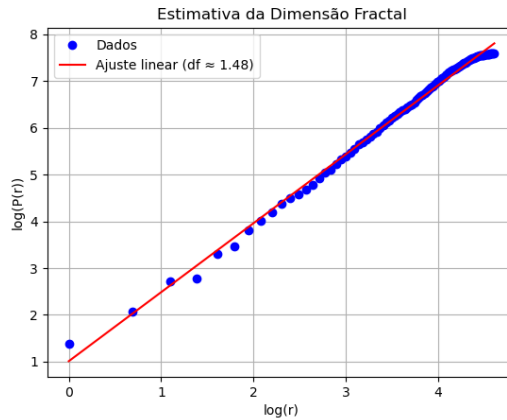


Figura 5: Dimensão fractal do crescimento DLA

Dessa maneira, podemos ver que a dimensão fractal nesse caso é de 1.48, condizente com o esperado, já que a dimensão fractal para um sistema não totalmente povoado deve ser menor que 2.

### 3 Tarefa 3

Na terceira parte do trabalho foi proposto analisar a dimensão fractal para o caso 3D da tarefa anterior. Dessa maneira, com algumas poucas alterações no código anterior, temos:

```

1 program DLA
2 implicit none
3 integer, parameter :: N = 256
4 integer, parameter :: num_particles = 4800
5 real*8, parameter :: pi = acos(-1.0d0)
6 integer :: mesh(N, N, N), particles(3, num_particles), val(2)
7 integer :: i, coord, val_pos
8 real*8 :: r, k, max_radius, theta, spawn_radius, phi
9
10 open(unit = 10, file="saida_1.13862338.out")
11
12 mesh = 0
13 val = [-1, 1]
14
15 particles(1, 1) = N / 2
16 particles(2, 1) = N / 2
17 particles(3, 1) = N / 2
18
19 max_radius = 0.0d0
20 mesh(particles(1,1), particles(2,1), particles(3,1)) = 1
21
22 do i = 2, num_particles
23   spawn_radius = max(max_radius + 5.0d0, 5.0d0)
24
25   call random_number(r)
26   call random_number(k)
27
28   theta = 2.0d0 * pi * r
29   phi = acos(2.0d0 * k - 1.0d0)
30
31   particles(1,i) = int(spawn_radius * sin(phi) * cos(theta)) + N/2
32   particles(2,i) = int(spawn_radius * sin(phi) * sin(theta)) + N/2
33   particles(3,i) = int(spawn_radius * cos(phi)) + N/2
34
35   do while (.not. neighbor(particles(1,i), particles(2,i), particles(3,i), mesh, N))
36     call random_number(r)
37     call random_number(k)
38
39     coord = int(3 * r) + 1
40     val_pos = int(2 * k) + 1
41
42     particles(coord, i) = particles(coord, i) + val(val_pos)
43
44     if (sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) > 2.0d0 * spawn_radius) then
45       call random_number(r)
46       call random_number(k)
47
48       theta = 2.0d0 * pi * r
49       phi = acos(2.0d0 * k - 1.0d0)
50
51       particles(1,i) = int(spawn_radius * sin(phi) * cos(theta)) + N/2
52       particles(2,i) = int(spawn_radius * sin(phi) * sin(theta)) + N/2
53       particles(3,i) = int(spawn_radius * cos(phi)) + N/2
54     end if
55   end do
56
57   mesh(particles(1,i), particles(2,i), particles(3,i)) = 1
58
59   max_radius = max(max_radius, sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) * 5.0d0)
60 end do
61
62 call Fractal(mesh, N)
63
64 contains
65
66 function neighbor(x, y, z, mesh, N) result(has_neighbor)
67 implicit none
68 integer :: x, y, z
69 integer :: i, j, k, N
70 integer :: mesh(N,N,N)
71 logical :: has_neighbor
72
73 has_neighbor = .false.
74
75 do i = -1, 1
76   do j = -1, 1
77     do k = -1, 1
78       if (x+i >= 1 .and. x+i <= N .and. y+j >= 1 .and. y+j <= N .and. z+k >= 1 .and. z+k <= N) then
79         if (mesh(x+i, y+j, z+k) == 1) then
80           has_neighbor = .true.
81           return
82         end if
83       end do
84     end do
85   end do
86 end do
87 end function neighbor
88
89 subroutine Fractal(mesh, N)
90 implicit none
91 integer, intent(in) :: N
92 integer :: mesh(N,N,N)
93 integer :: i, j, k, r, max_r, center_x, center_y, center_z
94 real*8 :: dist
95 integer, allocatable :: count(:)
96
97 center_x = N / 2
98 center_y = N / 2
99 center_z = N / 2
100 max_r = N / 2
101
102 allocate(count(0:max_r))
103 count = 0
104
105 do i = 1, N
106   do j = 1, N
107     do k = 1, N
108       if (mesh(i,j,k) == 1) then
109         dist = sqrt(real((i - center_x)**2 + (j - center_y)**2 + (k - center_z)**2))
110         if (int(dist) < max_r) then
111           count(int(dist)) = count(int(dist)) + 1
112         end if
113       end do
114     end do
115   end do
116 end do
117
118 do r = 1, max_r
119   count(r) = count(r) + count(r-1)
120 end do
121
122 do r = 1, max_r
123   write(10,*) r, count(r)
124 end do
125
126 deallocate(count)
127 end subroutine Fractal
128
129 end program DLA
130

```

Figura 6: Código da tarefa 3

Dessa maneira, podemos calcular a dimensão fractal da mesma forma que anterior.

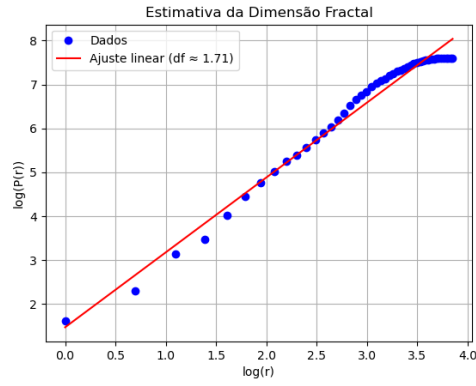


Figura 7: Dimensão fractal para o caso 3D do crescimento DLA

Podemos notar que para o caso 3D a dimensão se mostra um pouco maior que a anterior, com dimensão de 1.71, mas mesmo assim se mantendo menor que 2, como o esperado.

## 4 Tarefa 4

Na quarta tarefa do trabalho foi proposto um crescimento parecido com o da tarefa 2, porem dessa vez, ao invés de crescer a partir de uma semente localizada no centro do sistema, as sementes estarão na borda do plano e agora, as partículas serão lançadas a uma distancia  $h$  das semente de maneira aleatória ao longo do eixo  $x$ . Dessa maneiram foi criado o seguinte código.

```

1 program DLA_X
2   implicit none
3   integer, parameter :: N = 400
4   integer, parameter :: num_particles = 5000
5   integer :: mesh(N, N), particles(2, num_particles), val(2)
6   integer :: i, h, cord, val_pos
7   real*8 :: r, k
8
9   open(unit = 10, file="saida_1.13862338.out")
10
11   mesh = 0
12   val = [-1, 1]
13   h = 40
14
15   do i = 1, N
16     mesh(1, i) = 1
17   end do
18
19   do i = 1, num_particles
20     call ShowMesh(mesh, N)
21
22     call random_number(k)
23
24     particles(2, i) = int(N*k)+1
25     particles(1, i) = h
26
27
28     do while (.not. neighbor(particles(1,i), particles(2,i), mesh, N))
29       call random_number(r)
30       call random_number(k)
31
32       cord = int(2 + r) + 1
33       val_pos = int(2 + k) + 1
34
35       particles(cord, i) = particles(cord, i) + val(val_pos)
36
37       write(*,*)i, particles(1,i), particles(2,i), cord, val_pos
38
39       if (particles(2,i)>N .or. particles(2,i)<1 .or. particles(1,i)>2*h) then
40         particles(2, i) = int(N*k)+1
41         particles(1, i) = h
42       end if
43     end do
44     mesh(particles(1,i), particles(2,i)) = 1
45     if (particles(1,i) > h) h = particles(1,i)
46   end do
47
48   contains
49
50   subroutine ShowMesh(mesh, N)
51     implicit none
52     integer :: i, j, N
53     integer :: mesh(N,N)
54
55     do i = N, 1, -1
56       do j = 1, N
57         write(10, '(I4)', advance='no') mesh(i,j)
58       end do
59       write(10,*)
60     end do
61   end subroutine ShowMesh
62
63   function neighbor(x, y, mesh, N) result(has_neighbor)
64     implicit none
65     integer :: x, y
66     integer :: i, j, N
67     integer :: mesh(N,N)
68     logical :: has_neighbor
69
70     has_neighbor = .false.
71
72     do i = -1, 1
73       do j = -1, 1
74         if (x+i ≥ 1 .and. x+i ≤ N .and. y+j ≥ 1 .and. y+j ≤ N) then
75           if (mesh(x+i, y+j) ≥ 1) then
76             has_neighbor = .true.
77             return
78           end if
79         end if
80       end do
81     end do
82   end function neighbor
83 end program

```

Figura 8: Código da tarefa 4

Assim foi possível obter o seguinte resultado:



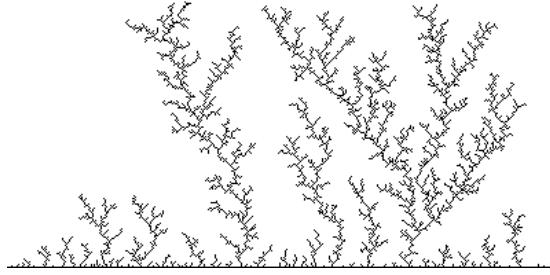


Figura 9: Estágio final da evolução do crescimento DLA vertical

Podemos observar que no estágio final da evolução temos um comportamento muito semelhante com descargas elétricas, onde existem diversas ramificações, dentre elas algumas maiores que apresentam a partir delas mais ramificações, gerando um padrão muito interessante de ser analisado.

## 5 Tarefa 5

Na quinta tarefa foi proposto um crescimento parecido com a tarefa 2, porém, dessa vez, ao invés de lançarmos as partículas no aglomerado as partículas estarão fixas ao longo da malha com uma probabilidade  $p$  e o aglomerado como um todo irá se mover aleatoriamente a medida que vai se juntando com as partículas da malha. Dessa maneira, foi criado o seguinte código:

```

1  program DLA
2      implicit none
3      integer, parameter :: N = 250
4      integer, parameter :: pass = 2000
5      real*8, parameter :: p = 0.1
6      integer, parameter :: max_part = N**2
7      integer :: mesh(N,N)
8      integer :: pos_x(max_part), pos_y(max_part)
9      integer :: i, j, dx, dy, val(2)
10     integer :: num_part, x, y
11
12     real*8 :: rand1, rand2
13
14     open(unit=10, file="saida_1_13862330.out")
15     open(unit=20, file="saida_2_13862330.out")
16
17     val = [-1, 1]
18     mesh = 0
19
20     do i = 1, N
21         do j = 1, N
22             call random_number(rand1)
23             if (rand1 <= p) then
24                 mesh(i,j) = 1
25             end if
26         end do
27     end do
28
29     num_part = 1
30     pos_x(1) = N / 2
31     pos_y(1) = N / 2
32     mesh(pos_x(1), pos_y(1)) = 2
33
34     do i = 1, pass
35         call random_number(rand1)
36         call random_number(rand2)
37         dx = val(int(2*rand1)+1)
38         dy = val(int(2*rand2)+1)
39
40         do j = 1, num_part
41             x = pos_x(j) + dx
42             y = pos_y(j) + dy
43             if (x < 1 .or. x > N .or. y < 1 .or. y > N) cycle
44             if (mesh(x, y) == 2) cycle
45         end do
46
47         do j = 1, num_part
48             mesh(pos_x(j), pos_y(j)) = 0
49             pos_x(j) = pos_x(j) + dx
50             pos_y(j) = pos_y(j) + dy
51         end do
52         do j = 1, num_part
53             mesh(pos_x(j), pos_y(j)) = 2
54         end do
55
56         do j = 1, num_part
57             do dx = -1, 1
58                 do dy = -1, 1
59                     if (dx == 0 .and. dy == 0) cycle
60                     x = pos_x(j) + dx
61                     y = pos_y(j) + dy
62                     if (x >= 1 .and. x <= N .and. y >= 1 .and. y <= N) then
63                         if (mesh(x, y) == 1) then
64                             if (num_part < max_part) then
65                                 num_part = num_part + 1
66                                 pos_x(num_part) = x
67                                 pos_y(num_part) = y
68                                 mesh(x, y) = 2
69                             end if
70                         end if
71                     end if
72                 end do
73             end do
74         end do
75
76         call ShowMesh(mesh, N)
77         write(*,*) "Passo:", i, " Aglomerado:", num_part, " particulas"
78     end do
79
80     call Fractal(pos_x, pos_y, num_part, N)
81
82 contains
83
84 subroutine ShowMesh(mesh, N)
85     implicit none
86     integer :: i, j, N
87     integer :: mesh(N,N)
88
89     do i = 1, N
90         do j = 1, N
91             write(10,'(I4)', advance='no') mesh(i,j)
92         end do
93         write(10,*)
94     end do
95 end subroutine ShowMesh
96
97 subroutine Fractal(pos_x, pos_y, num_part, N)
98     implicit none
99     integer, intent(in) :: pos_x(:), pos_y(:), num_part, N
100    integer :: i, r, max_r
101    real*8 :: dist, cx, cy
102    integer :: count(0:N/2)
103
104    cx = 0.0d0
105    cy = 0.0d0
106    do i = 1, num_part
107        cx = cx + dble(pos_x(i))
108        cy = cy + dble(pos_y(i))
109    end do
110    cx = cx / dble(num_part)
111    cy = cy / dble(num_part)
112
113    count = 0
114    max_r = N / 2
115
116    do i = 1, num_part

```

Dessa maneira foi possível obter o seguinte resultado:

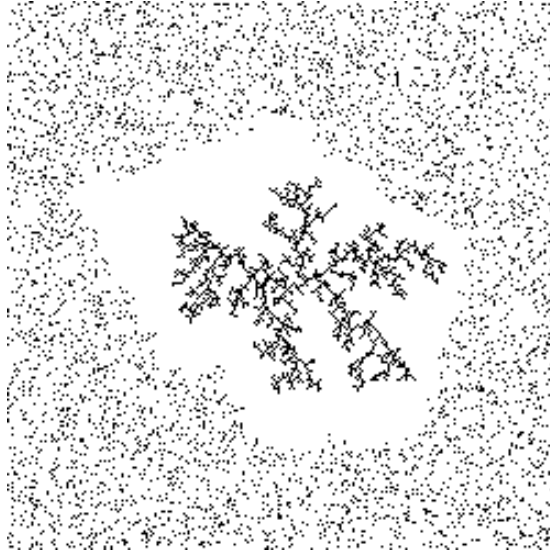


Figura 11: Evolução final com  $p = 0.1$

Podemos notar o crescimento parecido com a tarefa 5, pensando que as partículas distantes não estão juntas ao aglomerado. Além disso, podemos calcular a dimensão fractal dela também.

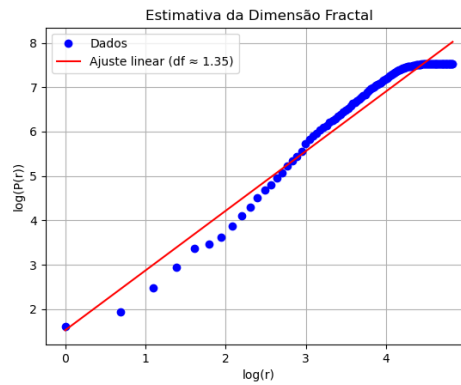
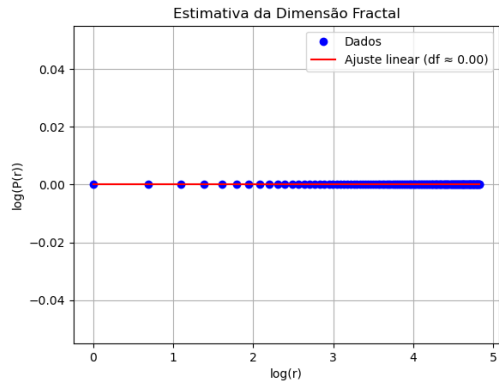
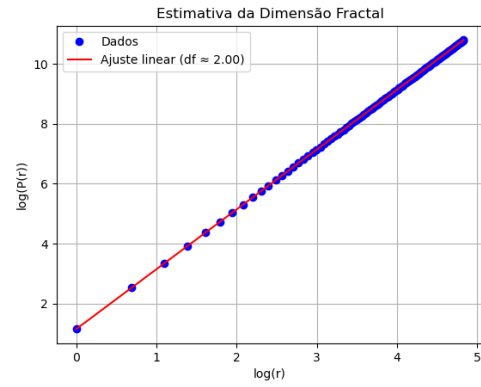


Figura 12: Dimensão fractal para o crescimento com  $p = 0.1$

Podemos ver que o valor se mantém consistente, já que o valor 1.35 da dimensão ainda é menor que 2 como o esperado. De maneira comparativa, podemos também analisar a dimensão fractal para os casos de  $p = 0$  e  $p = 1$ , dessa maneira podemos esperar valores diferentes dos vistos anteriormente.



(a) Dimensão fractal da evolução com  $p = 0$



(b) Dimensão fractal da evolução com  $p = 1$

Figura 13: Dimensão fractal da evolução com diferentes  $p$ 's

Dessa maneira, podemos ver que, conforme o esperado quando temos uma malha despovoada a dimensão fractal do sistema é 0, já em uma malha densamente povoada, ou seja, quando todos os pontos estão ocupados temos uma dimensão fractal de 2, sendo o máximo esperado do sistema.