Física estatística computacional

MODELOS DE CRESCIMENTO

Aluno
Leonardo Vaz Ferreira n° 13862330

São Carlos
2025

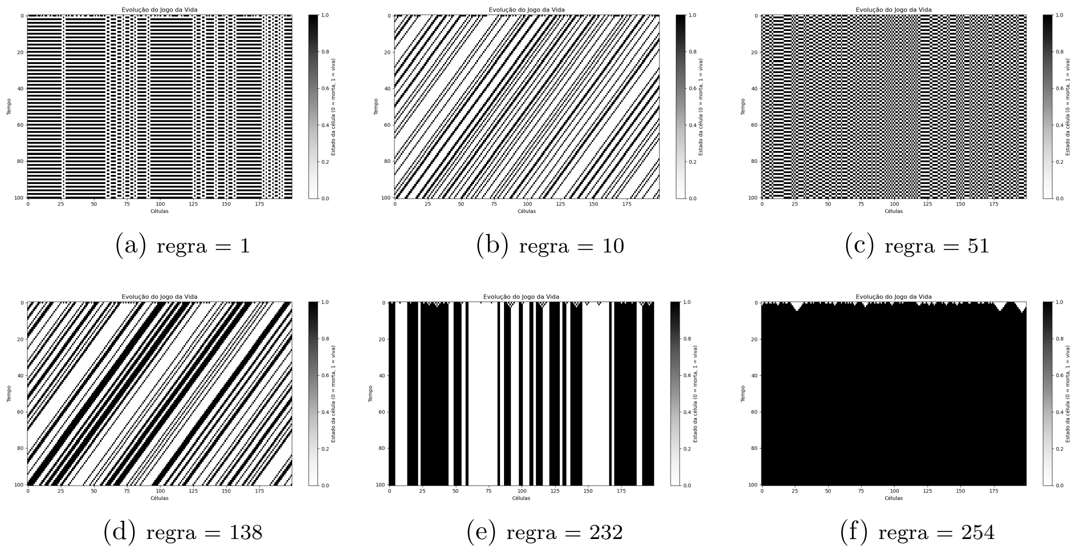
1 Tarefa 1

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:

```fortran
program ACD
    implicit none
    integer, parameter :: N = 200
    integer :: i, t, rule(8), L(N), new_L(N)

    open(unit=10, file='rule_138_13862330.out')

    L = InitPosition(N)
    rule = DecomposeRule(138)

    write(10,*)L

    do t = 1, 100
        new_L(1) = rule(L(N)*2**2 + L(1)*2**1 + L(2)*2**0 + 1)

        do i = 2, N-1
            new_L(i) = rule(L(i-1)*2**2 + L(i)*2**1 + L(i+1)*2**0 + 1)
        end do

        new_L(N) = rule(L(N-1)*2**2 + L(N)*2**1 + L(1)*2**0 + 1)

        L = new_L

        write(10,*)L
    end do

contains

    function InitPosition(N) result(L)
        implicit none
        integer, intent(in) :: N
        integer :: i
        real*8 :: L(N)

        call random_number(L)

        do i = 1, N
            L(i) = int(anint(L(i)))
        end do
    end function InitPosition

    function DecomposeRule(x) result(rule)
        integer, intent(in):: x
        integer:: i, new_x
        integer:: rule(8)

        new_x = x

        do i = 0, 7
            if (new_x - 2**(7-i) .GE. 0.0d0) then
                rule(8-i) = 1
                new_x = new_x - 2**(7-i)
            else
                rule(8-i) = 0
            end if
        end do
    end function DecomposeRule
end program ACD
```

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:



(a) regra = 1 (b) regra = 10 (c) regra = 51

(d) regra = 138 (e) regra = 232 (f) regra = 254

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:

```fortran
program DLA
    implicit none
    integer, parameter :: N = 200
    integer, parameter :: num_particles = 2000
    real*8, parameter :: pi = acos(-1.0d0)
    integer :: mesh(N,N), particles(2, num_particles), val(2)
    integer :: i, cord, val_pos
    real*8 :: r, k, max_radius, theta, spawn_radius

    open(unit=10, file="saida_1_13862330.out")
    open(unit=20, file ="saida_2_13862330.out")

    mesh = 0
    val = [-1, 1]

    particles(1, 1) = N / 2
    particles(2, 1) = N / 2

    max_radius = 0.0d0
    mesh(particles(1,1), particles(2,1)) = 1

    do i = 2, num_particles
        call ShowMesh(mesh, N)

        call random_number(r)
        spawn_radius = max(max_radius + 5.0d0, 5.0d0)

        theta = 2.0d0 * pi * r
        particles(1,i) = int(spawn_radius * cos(theta)) + N/2
        particles(2,i) = int(spawn_radius * sin(theta)) + N/2

        do while (.not. neighbor(particles(1,i), particles(2,i), mesh, N))
            call random_number(r)
            call random_number(k)

            cord = int(2 * r) + 1
            val_pos = int(2 * k) + 1

            particles(cord, i) = particles(cord, i) + val(val_pos)

            if (sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) > 2.0d0 * spawn_radius) then
                call random_number(r)
                theta = 2.0d0 * pi * r
                particles(1,i) = int(spawn_radius * cos(theta)) + N/2
                particles(2,i) = int(spawn_radius * sin(theta)) + N/2
            end if
        end do

        mesh(particles(1,i), particles(2,i)) = 1
        max_radius = max(max_radius, sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) + 5.0d0)
    end do

    call Fractal(mesh, N)

contains

    subroutine ShowMesh(mesh, N)
        implicit none
        integer :: i, j, N
        integer :: mesh(N,N)

        do i = 1, N
            do j = 1, N
                write(10,'(I4)', advance='no') mesh(i,j)
            end do
            write(10,*)
        end do
    end subroutine ShowMesh

    function neighbor(x, y, mesh, N) result(has_neighbor)
        implicit none
        integer :: x, y
        integer :: i, j, N
        integer :: mesh(N,N)
        logical :: has_neighbor

        has_neighbor = .false.

        do i = -1, 1
            do j = -1, 1
                if (x+i >= 1 .and. x+i <= N .and. y+j >= 1 .and. y+j <= N) then
                    if (mesh(x+i, y+j) == 1) then
                        has_neighbor = .true.
                        return
                    end if
                end if
            end do
        end do
    end function neighbor

    subroutine Fractal(mesh, N)
        implicit none
        integer, intent(in) :: N
        integer :: mesh(N,N)
        integer :: i, j, r, max_r, center_x, center_y
        real*8 :: dist
        integer, allocatable :: count(:)

        center_x = N / 2
        center_y = N / 2
        max_r = N / 2

        allocate(count(0:max_r))
        count = 0

        do i = 1, N
            do j = 1, N
                if (mesh(i,j) == 1) then
                    dist = sqrt(dble(i - center_x)**2 + dble(j - center_y)**2)
                    if (int(dist) <= max_r) then
                        count(int(dist)) = count(int(dist)) + 1
                    end if
                end if
            end do
        end do

        do r = 1, max_r
            count(r) = count(r) + count(r-1)
        end do

        do r = 1, max_r
            write(20,*) r, count(r)
        end do

        deallocate(count)
    end subroutine Fractal

end program DLA
```

Figura 3: Código da tarefa 2

Assim foram obtidos os seguintes resultados:

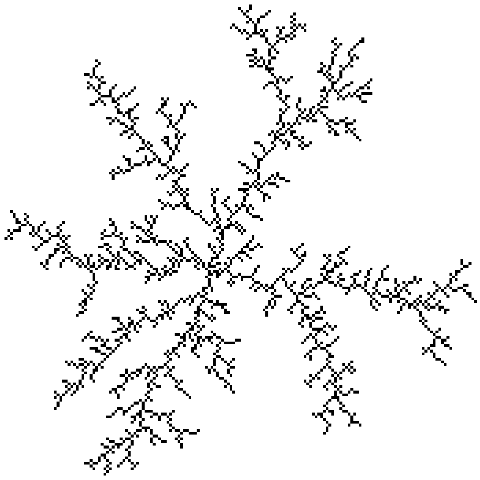


Figura 4: Estágio final da evolução temporal DLA

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 lnxln e calculamos a inclinação da reta.

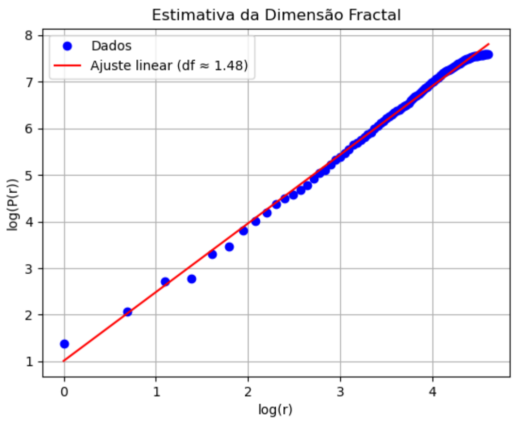


Figura 5: Dimensão fractal do crescimento DLA

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:

```fortran
program DLA
    implicit none
    integer, parameter :: N = 200
    integer, parameter :: num_particles = 4000
    real*8, parameter :: pi = acos(-1.0d0)
    integer :: mesh(N, N, N), particles(3, num_particles), val(2)
    integer :: i, cord, val_pos
    real*8 :: r, k, max_radius, theta, spawn_radius, phi

    open(unit = 10, file="saida_1_13862330.out")

    mesh = 0
    val = [-1, 1]

    particles(1, 1) = N / 2
    particles(2, 1) = N / 2
    particles(3, 1) = N / 2

    max_radius = 0.0d0
    mesh(particles(1,1), particles(2,1), particles(3,1)) = 1

    do i = 2, num_particles
        spawn_radius = max(max_radius + 5.0d0, 5.0d0)

        call random_number(r)
        call random_number(k)

        theta = 2.0d0 * pi * r
        phi = acos(2.0d0 * k - 1.0d0)

        particles(1,i) = int(spawn_radius * sin(phi) * cos(theta)) + N/2
        particles(2,i) = int(spawn_radius * sin(phi) * sin(theta)) + N/2
        particles(3,i) = int(spawn_radius * cos(phi)) + N/2

        do while (.not. neighbor(particles(1,i), particles(2,i), particles(3,i), mesh, N))
            call random_number(r)
            call random_number(k)

            cord = int(3 * r) + 1
            val_pos = int(2 * k) + 1

            particles(cord, i) = particles(cord, i) + val(val_pos)

            if (sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) > 2.0d0 * spawn_radius) then
                call random_number(r)
                call random_number(k)

                theta = 2.0d0 * pi * r
                phi = acos(2.0d0 * k - 1.0d0)

                particles(1,i) = int(spawn_radius * sin(phi) * cos(theta)) + N/2
                particles(2,i) = int(spawn_radius * sin(phi) * sin(theta)) + N/2
                particles(3,i) = int(spawn_radius * cos(phi)) + N/2
            end if
        end do

        mesh(particles(1,i), particles(2,i), particles(3,i)) = 1

        max_radius = max(max_radius, sqrt((particles(1,i)-N/2.0d0)**2 + (particles(2,i)-N/2.0d0)**2) + 5.0d0)
    end do

    call Fractal(mesh, N)

contains

    function neighbor(x, y, z, mesh, N) result(has_neighbor)
        implicit none
        integer :: x, y, z
        integer :: i, j, k, N
        integer :: mesh(N,N,N)
        logical :: has_neighbor

        has_neighbor = .false.

        do i = -1, 1
            do j = -1, 1
                do k = -1, 1
                    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
                        if (mesh(x+i, y+j, z+k) == 1) then
                            has_neighbor = .true.
                            return
                        end if
                    end if
                end do
            end do
        end do
    end function neighbor

    subroutine Fractal(mesh, N)
        implicit none
        integer, intent(in) :: N
        integer :: mesh(N,N,N)
        integer :: i, j, k, r, max_r, center_x, center_y, center_z
        real*8 :: dist
        integer, allocatable :: count(:)

        center_x = N / 2
        center_y = N / 2
        center_z = N / 2
        max_r = N / 2

        allocate(count(0:max_r))
        count = 0

        do i = 1, N
            do j = 1, N
                do k = 1, N
                    if (mesh(i,j,k) == 1) then
                        dist = sqrt(real((i - center_x)**2 + (j - center_y)**2 + (k - center_z)**2))
                        if (int(dist) <= max_r) then
                            count(int(dist)) = count(int(dist)) + 1
                        end if
                    end if
                end do
            end do
        end do

        do r = 1, max_r
            count(r) = count(r) + count(r-1)
        end do


        do r = 1, max_r
            write(10,*) r, count(r)
        end do

        deallocate(count)
    end subroutine Fractal

end program DLA
```

Figura 6: Código da tarefa 3

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

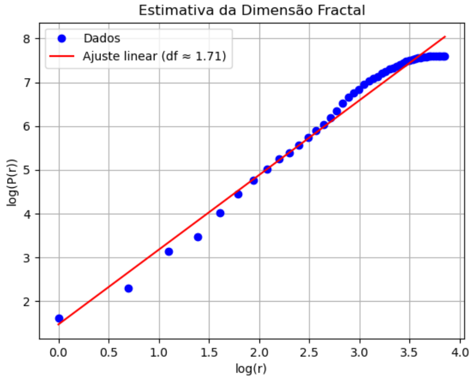


Figura 7: Dimensão fractal para o caso 3D do crescimento DLA

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.

```fortran
program DLA_X
    implicit none
    integer, parameter :: N = 400
    integer, parameter :: num_particles = 5000
    integer :: mesh(N, N), particles(2, num_particles), val(2)
    integer :: i, h, cord, val_pos
    real*8 :: r, k

    open(unit = 10, file="saida_1_13862330.out")

    mesh = 0
    val = [-1, 1]
    h = 40

    do i = 1, N
        mesh(1, i) = 1
    end do

    do i = 1, num_particles
        call ShowMesh(mesh, N)

        call random_number(k)

        particles(2, i) = int(N*k)+1
        particles(1, i) = h


        do while (.not. neighbor(particles(1,i), particles(2,i), mesh, N))
            call random_number(r)
            call random_number(k)

            cord = int(2 * r) + 1
            val_pos = int(2 * k) + 1

            particles(cord, i) = particles(cord, i) + val(val_pos)

            write(*,*)i, particles(1,i), particles(2,i), cord, val_pos

            if (particles(2,i)>N .or. particles(2,i)<1 .or. particles(1,i)>2*h) then
                particles(2, i) = int(N*k)+1
                particles(1, i) = h
            end if
        end do
        mesh(particles(1,i), particles(2,i)) = 1
        if (particles(1,i) > h) h = particles(1,i)
    end do

    contains

    subroutine ShowMesh(mesh, N)
        implicit none
        integer :: i, j, N
        integer :: mesh(N,N)

        do i = N, 1, -1
            do j = 1, N
                write(10,'(I4)', advance='no') mesh(i,j)
            end do
            write(10,*)
        end do
    end subroutine ShowMesh

    function neighbor(x, y, mesh, N) result(has_neighbor)
        implicit none
        integer :: x, y
        integer :: i, j, N
        integer :: mesh(N,N)
        logical :: has_neighbor

        has_neighbor = .false.

        do i = -1, 1
            do j = -1, 1
                if (x+i >= 1 .and. x+i <= N .and. y+j >= 1 .and. y+j <= N) then
                    if (mesh(x+i, y+j) >= 1) then
                        has_neighbor = .true.
                        return
                    end if
                end if
            end do
        end do
    end function neighbor
end program
```

Figura 8: Código da tarefa 4

Assim foi possível obter o seguinte resultado:

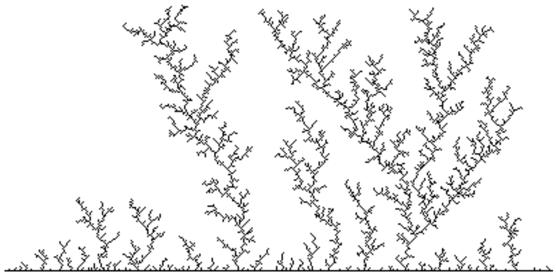


Figura 9: Estágio final da evolução do crescimento DLA vertical

Podemos observar que no estágio final da evolução tempos um comportamento muito semelhante com descargas elétricas, onde existem diversas ramificações, dentre ela 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:

```fortran
program DLA
    implicit none
    integer, parameter :: N = 250
    integer, parameter :: pass = 2000
    real*8, parameter :: p = 0.1
    integer, parameter :: max_part = N**2
    integer :: mesh(N,N)
    integer :: pos_x(max_part), pos_y(max_part)
    integer :: i, j, dx, dy, val(2)
    integer :: num_part, x, y

    real*8 :: rand1, rand2

    open(unit=10, file="saida_1_13862330.out")
    open(unit=20, file="saida_2_13862330.out")

    val = [-1, 1]
    mesh = 0

    do i = 1, N
        do j = 1, N
            call random_number(rand1)
            if (rand1 <= p) then
                mesh(i,j) = 1
            end if
        end do
    end do

    num_part = 1
    pos_x(1) = N / 2
    pos_y(1) = N / 2
    mesh(pos_x(1), pos_y(1)) = 2

    do i = 1, pass
        call random_number(rand1)
        call random_number(rand2)
        dx = val(int(2*rand1)+1)
        dy = val(int(2*rand2)+1)

        do j = 1, num_part
            x = pos_x(j) + dx
            y = pos_y(j) + dy
            if (x < 1 .or. x > N .or. y < 1 .or. y > N) cycle
            if (mesh(x, y) == 2) cycle
        end do

        do j = 1, num_part
            mesh(pos_x(j), pos_y(j)) = 0
            pos_x(j) = pos_x(j) + dx
            pos_y(j) = pos_y(j) + dy
        end do
        do j = 1, num_part
            mesh(pos_x(j), pos_y(j)) = 2
        end do

        do j = 1, num_part
            do dx = -1, 1
                do dy = -1, 1
                    if (dx == 0 .and. dy == 0) cycle
                    x = pos_x(j) + dx
                    y = pos_y(j) + dy
                    if (x >= 1 .and. x <= N .and. y >= 1 .and. y <= N) then
                        if (mesh(x, y) == 1) then
                            if (num_part < max_part) then
                                num_part = num_part + 1
                                pos_x(num_part) = x
                                pos_y(num_part) = y
                                mesh(x, y) = 2
                            end if
                        end if
                    end if
                end do
            end do
        end do

        call ShowMesh(mesh, N)
        write(*,*) "Passo:", i, " Aglomerado:", num_part, " partículas"
    end do

    call Fractal(pos_x, pos_y, num_part, N)

contains

subroutine ShowMesh(mesh, N)
    implicit none
    integer :: i, j, N
    integer :: mesh(N,N)

    do i = 1, N
        do j = 1, N
            write(10,'(I4)', advance='no') mesh(i,j)
        end do
        write(10,*)
    end do
end subroutine ShowMesh

subroutine Fractal(pos_x, pos_y, num_part, N)
    implicit none
    integer, intent(in) :: pos_x(:), pos_y(:), num_part, N
    integer :: i, r, max_r
    real*8 :: dist, cx, cy
    integer :: count(0:N/2)

    cx = 0.0d0
    cy = 0.0d0
    do i = 1, num_part
        cx = cx + dble(pos_x(i))
        cy = cy + dble(pos_y(i))
    end do
    cx = cx / dble(num_part)
    cy = cy / dble(num_part)

    count = 0
    max_r = N / 2

    do i = 1, num_part
```

Dessa maneira foi possível obter o seguinte resultado:

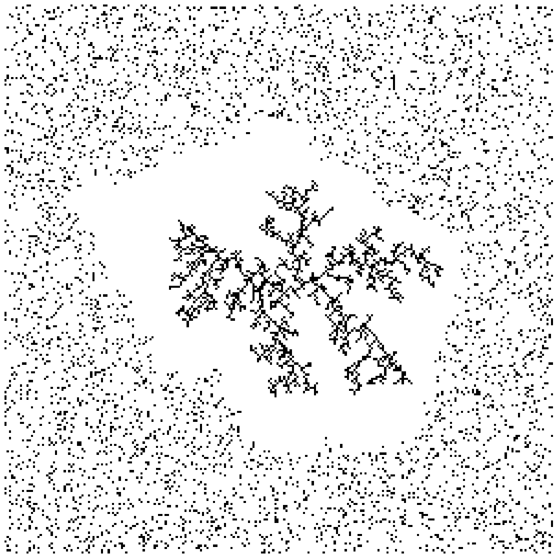


Figura 11: Evolução final com $p = 0.1$

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.

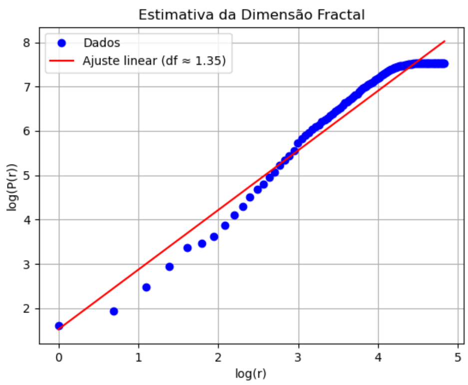
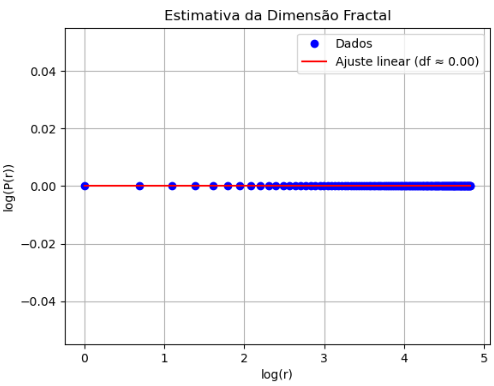
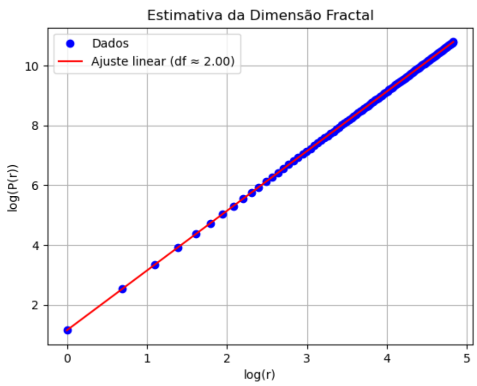


Figura 12: Dimensão fractal para o crescimento com p 0.1

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 maxímo esperado do sistema.