MODELOS DE CRESCIMENTO

Edgard Macena Cabral Nº 11820833

Maio 2023

Introdução

Nessa tarefa, buscamos estudar modelos de crescimento. Dinâmicas de crescimento ocorrem por toda natureza, desde do crescimento de cristais à reprodução de sistemas biológicos e sociais.

Os modelos que estudamos podem ser divididos em dois tipos: Autômatas Celulares Determinísticos (ACD) e modelos de crescimento com aleatoriedade.

No primeiro, o crescimento é dado por uma regra, que no nosso caso podia ser dada pela representação binária de um número. Sabendo essa regra, pode-se determinar o modelo para qualquer interação.

No segundo, o crescimento acontece de maneira aleatória, sendo o modelo de crescimento mais associado a fenômenos como cristais e descargas elétricas.

ACD

Aqui, buscamos reproduzir 3 regras que dependem apenas dos vizinhos imediatos da célula em $t-1$. Isso é

$$c_i^{t+1} = f(c_{i-1}^t, c_i^t, c_{i+1}^t) \tag{1}$$

- A regra da maioria, ou regra 232, onde o valor de c_i é o da maioria dos vizinhos em $t-1$,
- A regra epidemia, ou regra 254, onde basta um vizinho ser 1 para a célula positivar.
- A regra do contra, ou regra 54, onde a célula simplesmente assume o valor oposto ao que tinha anteriormente.

  Testamos essas regras para 3 condições: Quando todos os valores iniciais são 1, quando todos são 0, ou quando eles assumem valores aleatórios=

Programa

Usamos um programa que permitisse rodar um ACD para qualquer que seja nossa regra. Fizemos

Listing 1: Programa da tarefa 1

```
module AcdModules
  implicit none

  public :: bernoulli, vetorBinario, &
    propagaRegra, dancaDaCadeira, imprimeConfig
  contains

  ! Função de bernoulli
  ! Para p = 0.5, se torna um gerador
  ! de 0 ou 1 com probabilidade identica
  integer function bernoulli(p)
    real(8) :: p, numAleatorio
    call random_number(numAleatorio)
    if ( numAleatorio < p ) then
      bernoulli = 1
    else
      bernoulli = 0
    end if
  end function bernoulli

  ! Converte o valor decimal em um vetor binário
  subroutine vetorBinario(vetorRegra, regra)
    integer, intent(out) ::  vetorRegra(0:7)
    integer :: regra, index, N

    N = regra

    do index = 0, 7
      vetorRegra(index) = mod(N,2)
      N = N/2
    end do

  end subroutine vetorBinario


  subroutine &
    propagaRegra(configAtual, configSeguinte, vetorRegra, L)
    integer, intent(in) :: configAtual(:), &
      vetorRegra(0:7), L
    integer, intent(out) ::  configSeguinte(:)

    ! Se a maioria dos vizinhos
    configSeguinte(2:L-1) = &
      vetorRegra((configAtual(1:L-2)*2**2 &
      + configAtual(2:L-1)*2 + configAtual(3:L)))
```

```
    ! Ajusta as bordas
    configSeguinte(1) = &
       vetorRegra(configAtual(L)*2**2 &
       + configAtual(1)*2 + configAtual(2))

    configSeguinte(L) = &
       vetorRegra(configAtual(L-1)*2**2 &
       +  configAtual(L-2)*2 + configAtual(1))
end subroutine propagaRegra


subroutine dancaDaCadeira(configAtual, configSeguinte)
    integer, intent(inout) :: configAtual(:), configSeguinte(:)
    configatual(:) = configSeguinte(:)
end subroutine dancaDaCadeira


subroutine imprimeConfig(configAtual, file, L)
    integer, intent(in) :: configAtual(:)
    integer :: L, file
    integer :: i
    character :: ascii(0:1)

    ascii(0) = "."
    ascii(1) = '$'
    write(file, *) (ascii(configAtual(i)), i=1, L)
end subroutine imprimeConfig

subroutine setConfig(configAtual, config,  L)
    integer, intent(in) :: config, L
    integer, intent(out) :: configAtual(:)
    integer :: i

    if ( config == 0 ) then
       configAtual = 0
    else if ( config == 1 ) then
       configAtual = 1
    else if ( config == -1 ) then
       do i = 1, L
          configAtual(i) = bernoulli(0.5d0)
       end do
    end if

end subroutine setConfig
```

```
    subroutine acdRegra(regra, config)
        integer, intent(in) :: regra, config
        integer, parameter :: L = 200
        integer :: vetorRegra(0:7), configAtual(L), configSeguinte(L)
        integer :: i
        character(len=26) :: filename


        write (filename, "(A6, I3, A5, I2)")&
            "saida-", regra, "_con_", config


        call vetorBinario(vetorRegra, regra)

        open(1, file = trim(filename))

        call setConfig(configAtual, config, L)

        do i = 0, 100
            call imprimeConfig(configAtual, 1, L)
            call propagaRegra(configAtual, configSeguinte, vetorRegra, L)
            call dancaDaCadeira(configAtual, configSeguinte)
        end do
        call imprimeConfig(configAtual, 1, L)
        close(1)


    end subroutine acdRegra

end module

program ACD
    use AcdModules

    ! Regra do Contra
    call acdRegra(51, 0)
    call acdRegra(51, 1)
    call acdRegra(51,-1)

    ! Regra da Maioria
    call acdRegra(232, 0)
    call acdRegra(232, 1)
    call acdRegra(232,-1)

    ! Regra da Infecção
    call acdRegra(254, 0)
```

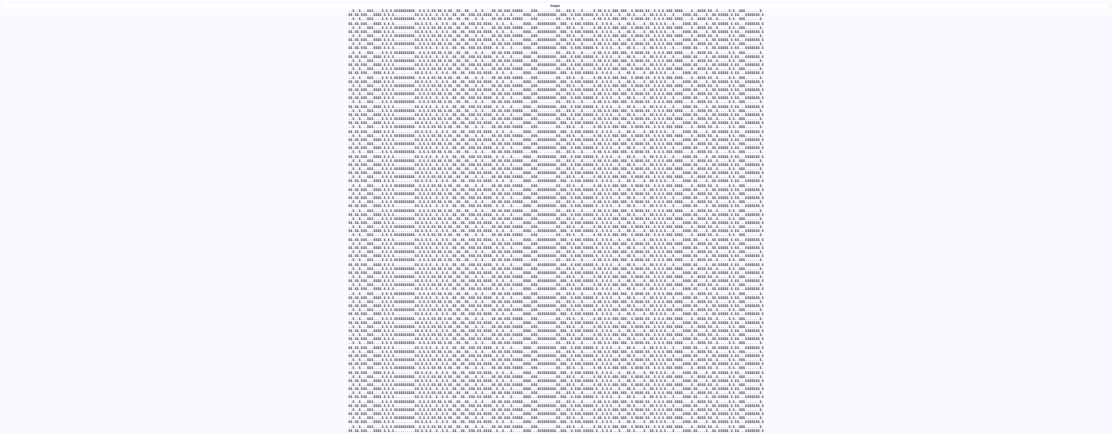
```
    call acdRegra(254, 1)
    call acdRegra(254,-1)
end program ACD
```

Resultados

Regra 51

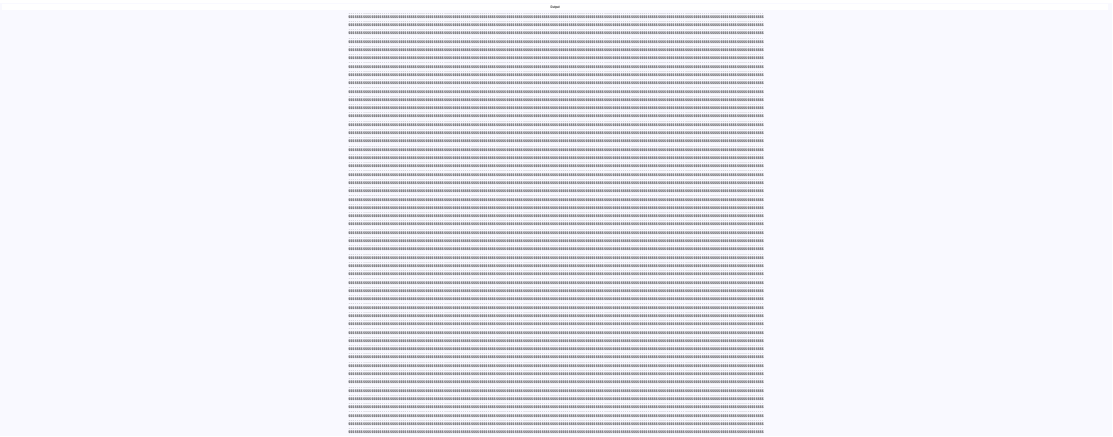
- Para valores iniciais aleatório

```
cat saidas/tarefa-1/regra-51/saida_-1
```



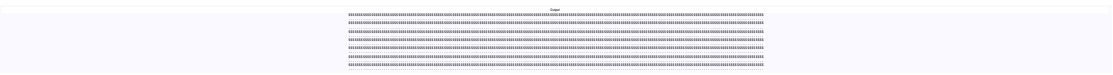
- Para valores iniciais 0

```
cat saidas/tarefa-1/regra-51/saida_0
```



- Para valores iniciais 1

```
cat saidas/tarefa-1/regra-51/saida_1
```

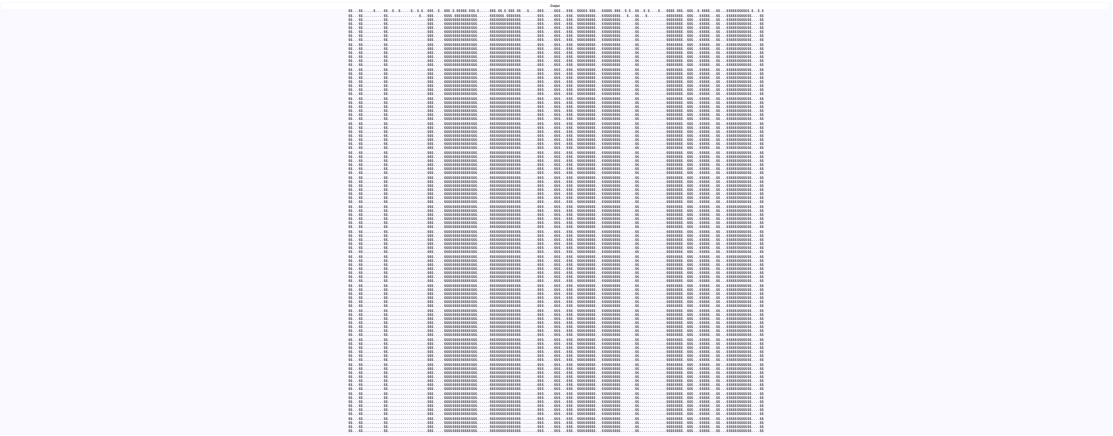


Regra 232

Para regra 232, vimos

- Para valores iniciais aleatórios

```
cat saidas/tarefa-1/regra-232/saida_-1
```

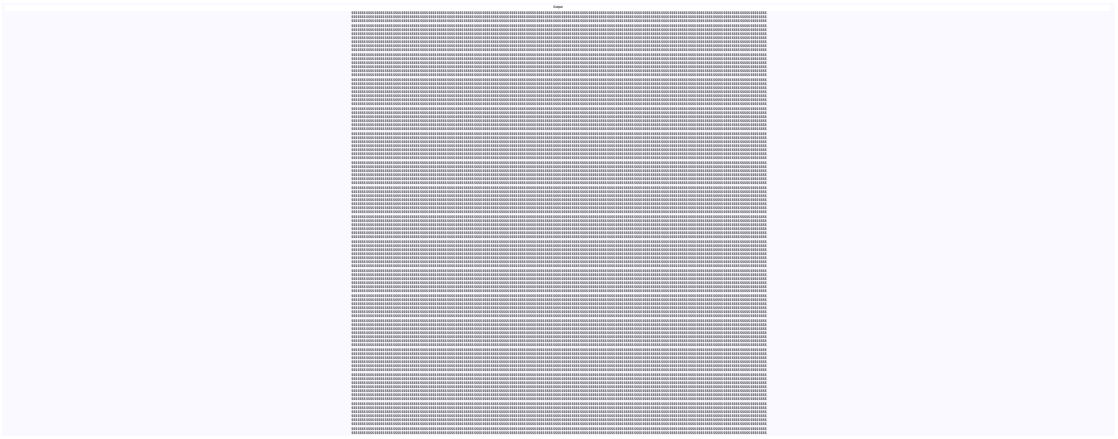


- Para valores iniciais 0

```
cat saidas/tarefa-1/regra-232/saida_0
```

- Para valores iniciais 1

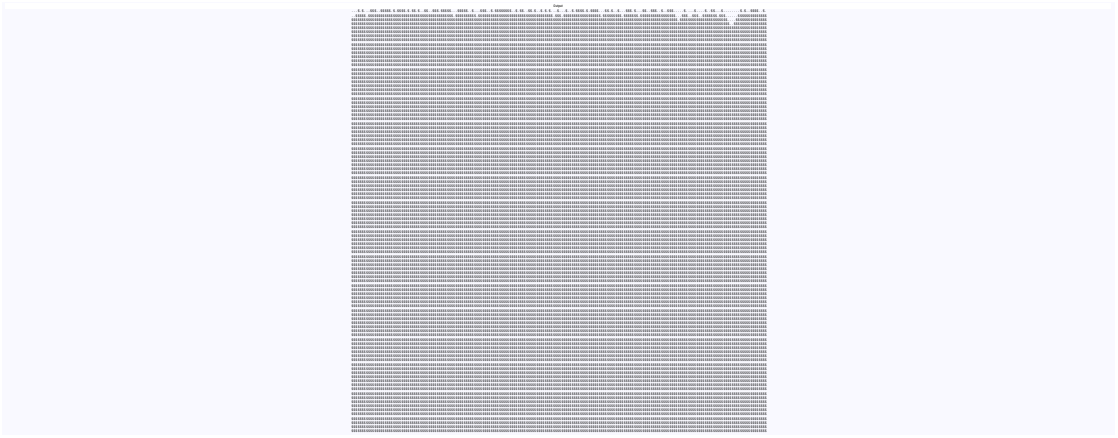
```
cat saidas/tarefa-1/regra-232/saida_1
```



Regra 254

- Para valores iniciais aleatórios

```
cat saidas/tarefa-1/regra-254/saida_-1
```

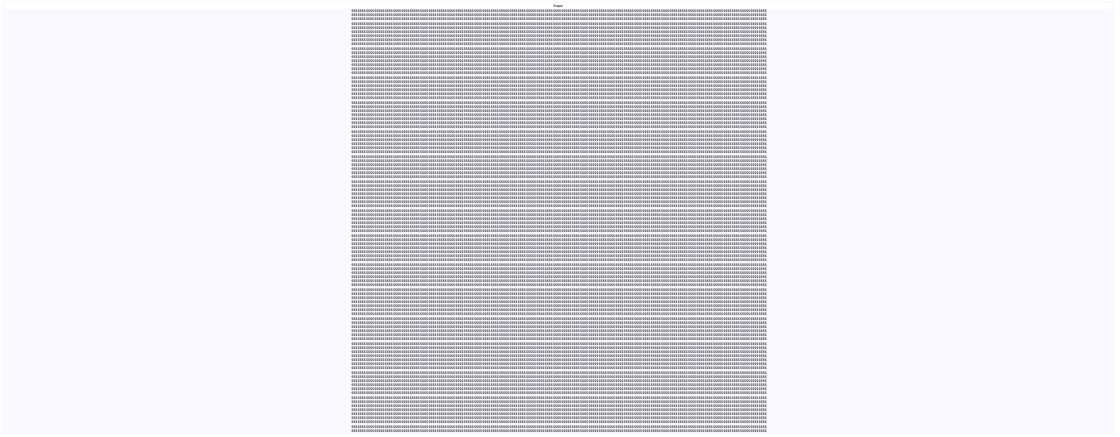


- Para valores iniciais 0

```
cat saidas/tarefa-1/regra-254/saida_0
```

- Para valores iniciais 1

```
cat saidas/tarefa-1/regra-254/saida_1
```



Aqui defendemos o ascii.

DLA

No estudo de modelos aleatórios, geramos moléculas dentro do intervalor de $r = r_a + 5$ até $1,5r$ para um aglomerado de raio r_a.

Essas moléculas então apresentavam um andar de bêbado ou até alcançar uma vizinhança próxima do aglomerado, quado se depositavam, ou até sair superiormente de r, onde elas ficam muito custosas computacionalmente

Programa

Para o programa do DLA, o foco do código foi em ter um algoritmo capaz de criar posições aleatórias dentro do intervalo comentado, num tabuleiro 201x201.

Para isso, é criada uma posição em coordenadas polares que depois é convertido em posições **x** e **y**.

Ademais, computacionalmente, escolhemos que nosso programa parace ao atingir 3000 pontos aglutinados ou $r > 98$.

Na prática, ambos os limites foram muito próximos, com alguns programas acabando antes disso, e outro

Listing 2: Programa da tarefa 2

```
module DlaModule
   implicit none
   real(8), parameter :: pi =  acos(-1.d0)
   integer, parameter :: idx(0:8) = (/-1, -1, -1,  0, 0, 0,  1, 1, 1/),&
                         idy(0:8) = (/-1,  0,  1, -1, 0, 1, -1, 0, 1/)
contains

   subroutine andarDoBebado(x,  y)
      integer, intent(out) :: x, y
      real(8) ::  numAleatorio
      integer ::  idr

      call random_number(numAleatorio)
      idr = nint(8*numAleatorio)

      x = x + idx(idr)
      y = y + idy(idr)
   end subroutine andarDoBebado


   subroutine escrevePonto(x,  y, numPontos, r)
      integer, intent(in) :: x, y, numPontos
      real(8) :: r
      write(1, *) x, y, numPontos, r
   end subroutine escrevePonto


   integer function numVizinhos(x, y,tabuleiro)
      integer :: x, y, tabuleiro(-100:100,-100:100)
      numVizinhos = tabuleiro(x+1,y) &
         + tabuleiro(x-1,y) + tabuleiro(x,y+1) &
         + tabuleiro(x, y-1)
   end function numVizinhos


   integer function raioQuad(x, y)
      integer, intent(in) :: x, y
      raioQuad = x**2 + y**2
   end function raioQuad


   subroutine geraPonto(r, tabuleiro, numPontos)
      integer :: x, y, tabuleiro(-100:100,-100:100), numPontos
      real(8), intent(out) :: r
      real(8) :: teta, numAleatorio
      ! rho pertence [rhoMin, rhoMin + drho]
```

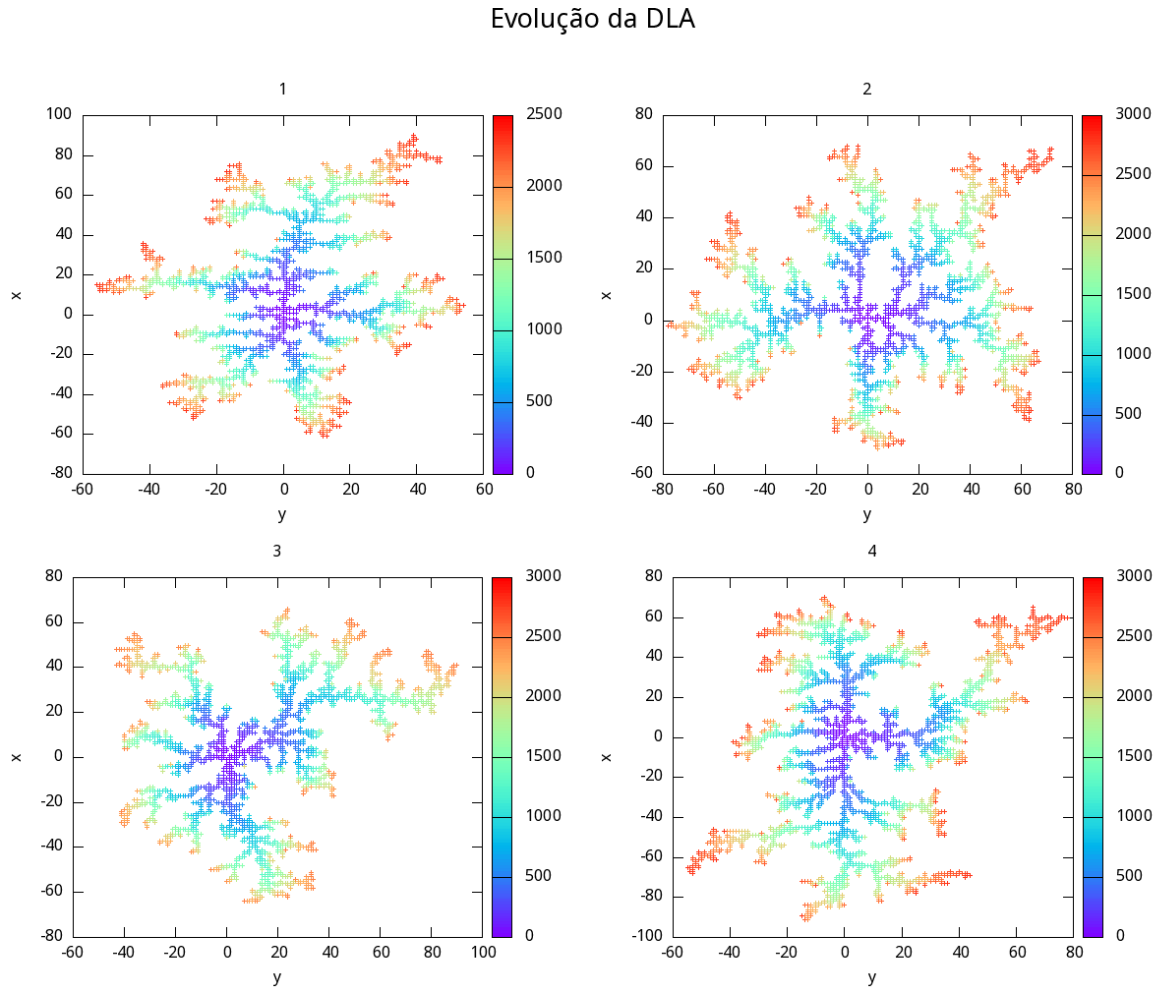
```
        real(8) :: rhoMin, drho, rho

        ! gera teta
        call random_number(numAleatorio)
        teta =  2*pi*numAleatorio

        ! gera p
        call random_number(numAleatorio)
        rhoMin = r + 5
        drho = 0.5d0*rhoMin
        rho = rhoMin + drho*numAleatorio
        x = floor(rho*cos(teta))
        y = floor(rho*sin(teta))

        do while ( raioQuad(x, y) .le. (1.5d0*rhoMin)**2 &
            .and. abs(x) < 99 .and. abs(y) < 99 )
           call andarDoBebado(x,y)

           ! Achou local para acoplar, atualiza tabuleiro
           if ( numVizinhos(x,y,tabuleiro) > 0 ) then
              numPontos = numPontos + 1
              tabuleiro(x,y) = numPontos
              if ( raioQuad(x,y) > r**2  ) then
                 r = raioQuad(x,y)**(1.d0/2.d0)
              end if
              call escrevePonto(x, y, numPontos,r)

              exit
           end if

        end do

    end subroutine geraPonto

end module DlaModule

program DLA
    use DlaModule
    integer, save :: tabuleiro(-100:100,-100:100)
    integer :: numPontos
    real(8) :: r

    ! Iniciamos nosso tabuleiro
    tabuleiro = 0
    tabuleiro(0,0) = 1
    numPontos = 1
```

```
    open(1, file="saida-1-11820833")

    r = 0.d0

    do while (numPontos .le. 3000 .and. r < 98)
       call geraPonto(r, tabuleiro, numPontos)
    end do
    close(1)

end program DLA
```

Resultados

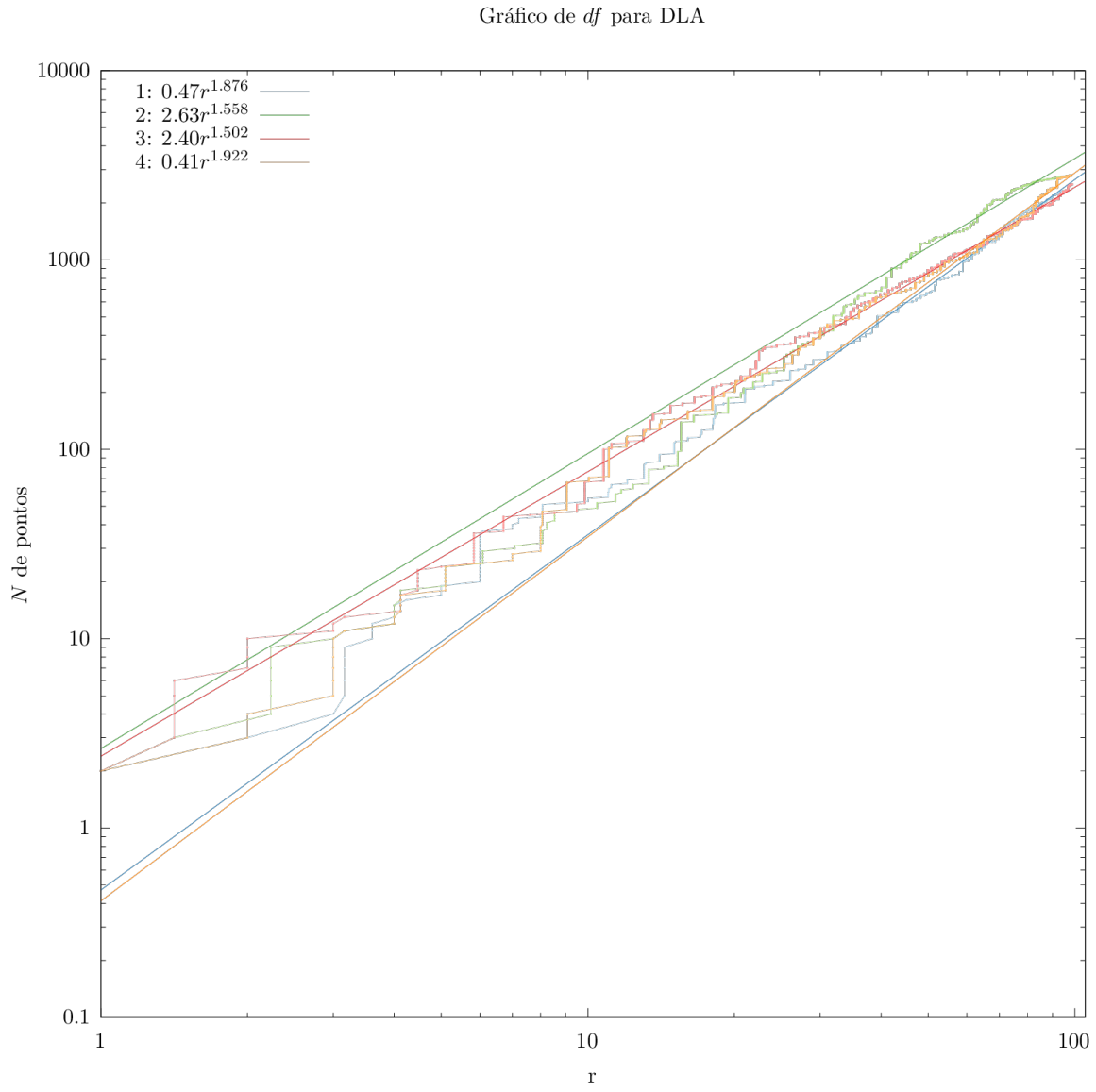
Rodamos 50 interações do programa da tarefa 2 e obtemos um valor médio de $df = 1,64$, com desvio padrão $\sigma = 0,10$, em concordância com o valor esperado de $1,65$. Apesar dos resultados concordantes, nota-se a incerteza relativamente grande.

Embora seja inviável mostrar os gráficos de cada interação, os de quatro delas estão a seguir.



Aqui, as cores representam em que interação determinado ponto se aglutinou ao aglomerado.

Podemos ver o gráfico de df para essas configurações na imagem a seguir, em escala logarítima:



A divergência do fitting em torno de $r = 0$ pode assustar, mas, embora ocupe muito do gráfico (ele é logaritmo, afinal), a concordância ao final, onde estão concentrados quase todos os pontos, é muito boa.

Note que o alto desvio padrão se repete na amostra. os valores pipocam em várias regiões. Isso não surpreende olhando o gráfico, é possível ver vários momentos onde o número de pontos cresce sem grande influência no r. Apesar disso, para um número de pontos consideravelmente maior, ou para resultados estátisticos (com fizemos), df deve convergir para o valor esperado.

DLA em 3 dimensões

Podemos fazer o mesmo processo para três dimensões com poucas alterações ao programa, seguindo as mesmas regras.

Programa

O programa está a seguir:

Listing 3: Programa da tarefa 3

```
module DlaModule
   implicit none
   real(8), parameter :: pi =  acos(-1.d0)

contains

   subroutine andarDoBebado(x,  y, z)
      integer, intent(out) :: x, y, z
      real(8) ::  numAleatorio
      integer ::  dx, dy, dz

      call random_number(numAleatorio)
      dx = nint(2*numAleatorio - 1)
      x = x + dx

      call random_number(numAleatorio)
      dy = nint(2*numAleatorio - 1)
      y = y + dy

      call random_number(numAleatorio)
      dz = nint(2*numAleatorio - 1)
      z = z + dz
   end subroutine andarDoBebado


   subroutine escrevePonto(numPontos, r)
      integer, intent(in) :: numPontos
      real(8) :: r
      write(1, *) numPontos, r
   end subroutine escrevePonto


   integer function raioQuad(x, y, z)
      integer, intent(in) :: x, y, z
      raioQuad = x**2 + y**2 + z**2
```

```
end function raioQuad



integer function numVizinhos(x, y, z, tabuleiro)
   integer :: x, y, z, tabuleiro(-75:75,-75:75,-75:75)
   numVizinhos = &
      tabuleiro(x+1, y, z) + tabuleiro(x-1,y,z) + &
      tabuleiro(x, y+1, z) + tabuleiro(x,y-1,z) + &
      tabuleiro(x, y, z+1) + tabuleiro(x,y,z-1)
end function numVizinhos


subroutine geraPonto(r, tabuleiro, numPontos)
   integer :: x, y, z, tabuleiro(-75:75,-75:75,-75:75), numPontos
   real(8), intent(out) :: r
   real(8) :: teta, fi, numAleatorio
   ! rho pertence [rhoMin, rhoMin + drho]
   real(8) :: rhoMin, drho, rho

   ! gera teta
   call random_number(numAleatorio)
   teta =  pi*numAleatorio

   ! gera fi
   call random_number(numAleatorio)
   fi = 2*pi*numAleatorio

   ! gera p
   call random_number(numAleatorio)
   rhoMin = r + 5
   drho = 0.5d0*rhoMin
   rho = rhoMin + drho*numAleatorio
   x = floor(rho*sin(teta)*cos(fi))
   y = floor(rho*sin(teta)*sin(fi))
   z = floor(rho*cos(teta))

   do while ( raioQuad(x,y,z) .le. (1.5d0*rhoMin)**2 &
       .and. abs(x) < 74 .and. abs(y) < 74 .and. abs(z) < 74 )
      call andarDoBebado(x,y,z)
      ! Achou local para acoplar, atualiza tabuleiro
      if ( numVizinhos(x,y, z, tabuleiro) > 0 ) then
         numPontos = numPontos + 1
         tabuleiro(x,y,z) = 1
         if ( raioQuad(x,y,z) > r**2  ) then
            r = raioQuad(x,y,z)**(1.d0/2.d0)
```

```
            end if
            call escrevePonto(numPontos,r)
            exit
          end if

        end do

    end subroutine geraPonto

end module DlaModule

program DLA_3D
    use DlaModule
    implicit none
    integer, save :: tabuleiro(-75:75,-75:75,-75:75)
    integer :: numPontos
    real(8) :: r

    ! Iniciamos nosso tabuleiro
    tabuleiro = 0
    tabuleiro(0,0,0) = 1
    numPontos = 1

    open(1, file="saida-3-11820833")

    r = 0.d0

    do while (numPontos .le. 10000 .and. r < 70)
        call geraPonto(r, tabuleiro, numPontos)
    end do
    close(1)


end program DLA_3D
```

A principal diferença está em na geração do movimento da partícula.

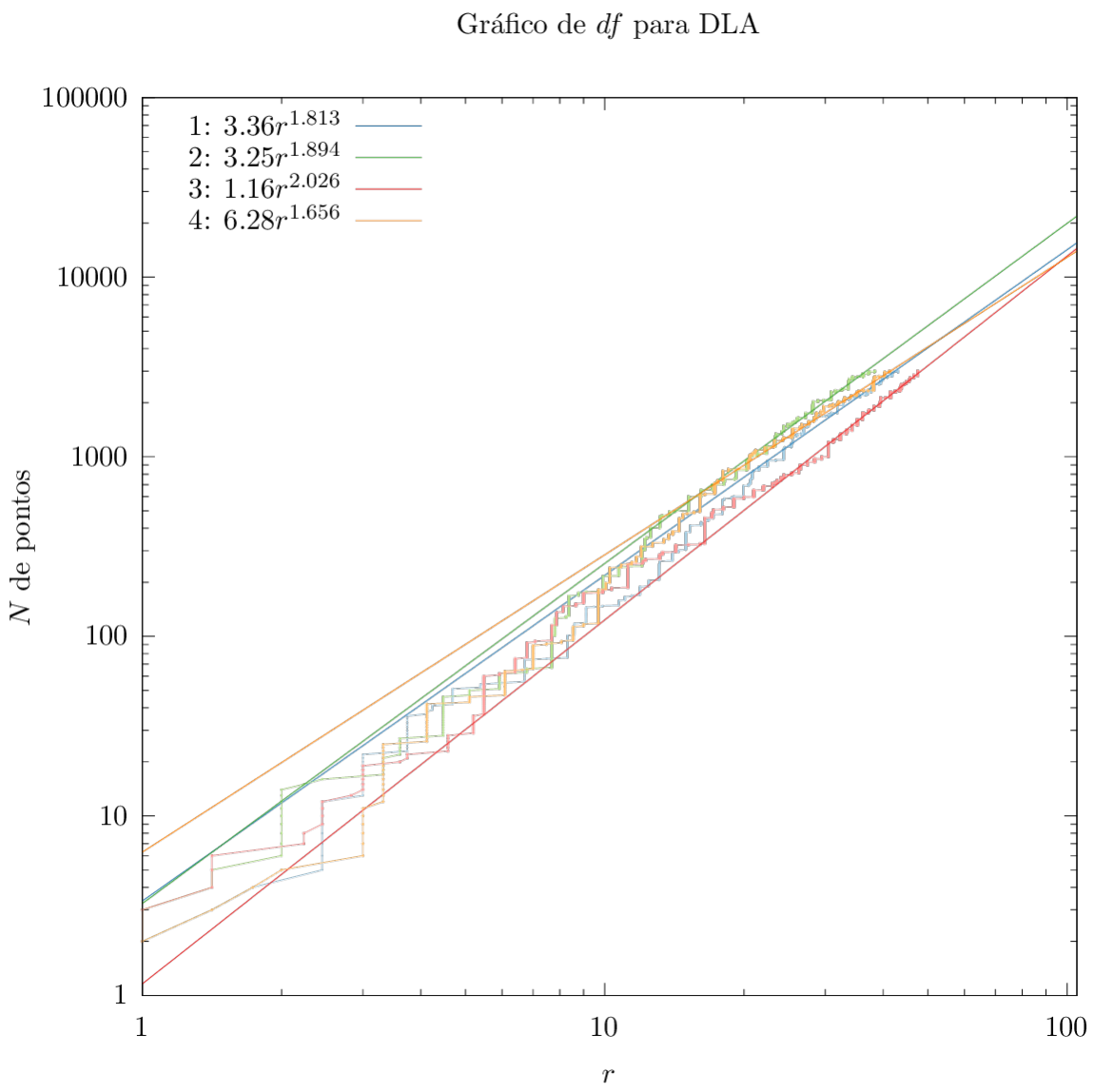
Embora seria possível (e ideal) adaptar o algoritmo anterior para o caso 3D, prefirimos usar a geração de um número aleatório para cada direção, que é bem mais legível do que 3 arrays de 27 ints no início.

Aqui, o limite imposto ao número de pontos propenderou sobre o imposto ao raio, que ficou em torno de 45 para maioria das interações.

Resultados

Obtivemos, para 30 interações $df = 2,00 \pm 0.14$.

Dessa vez, é impeditivo (e não muito útil) enxergar o aglomerado, mas podemos ver as curvas de df para 4 iterações a seguir



Vemos outra vez um gráfico bastante distribuído. Novamente, para N grande o suficiente, esses valores devem convergir.

Efeito Corona

Ao mudarmos a geração dos novos pontos de acontecer em torno de uma origem para acontecer acima de uma linha, temos efeitos parecidos com o do efeito corona. Buscamos reproduzir esse efeito com código a seguir

Programa

Aqui, apenas alteramos levemente o programa da DLA.

Primeiramente, fizemos de crescimento da semente todo eixo $y = 0$, depois fizemos o programa gerar novas moléculas apenas em $y > 0$. Por fim, essa moléculas tinham que estar inicialmente ao menos 5 blocos acima do nosso aglomerado.

Listing 4: Programa usado na tarefa 4

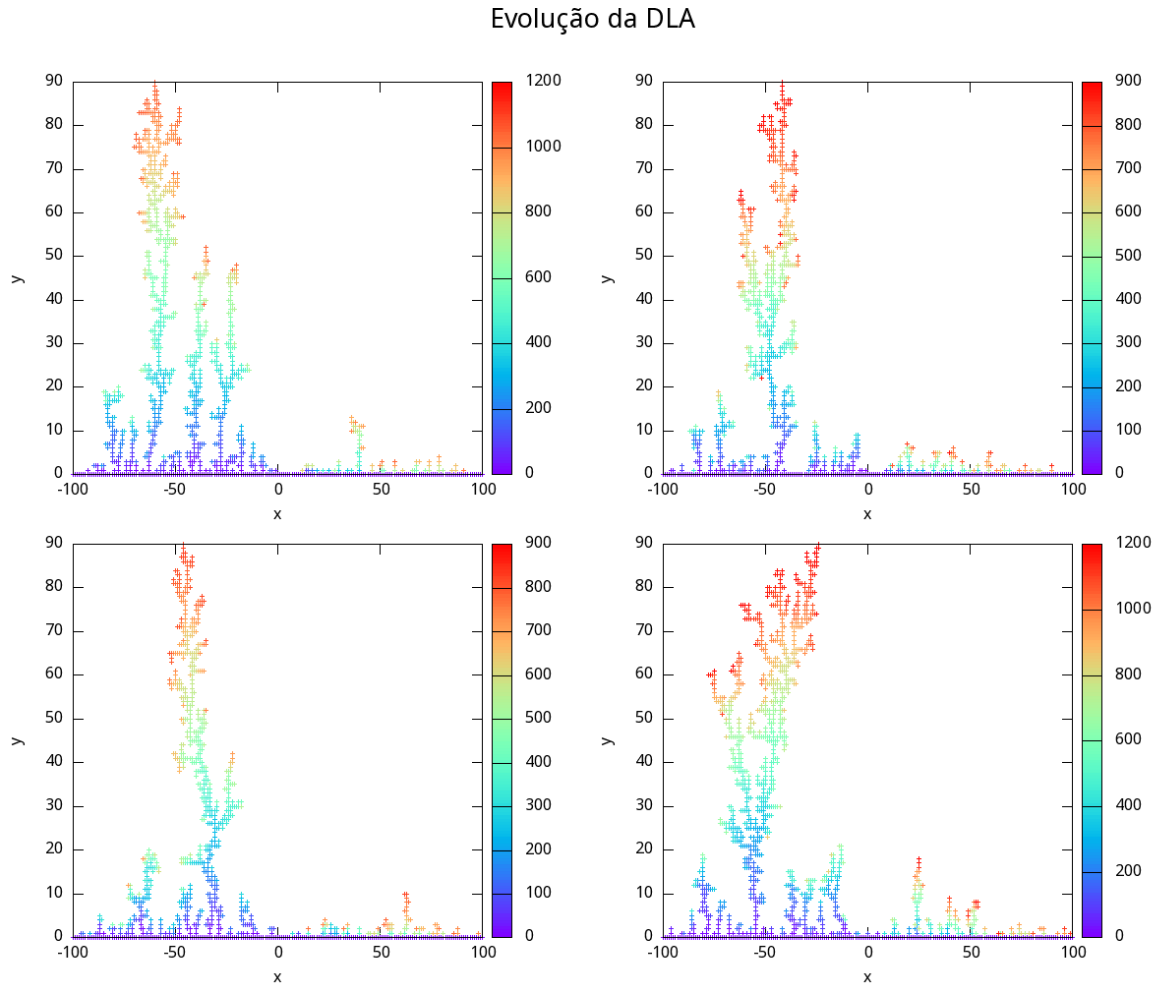
```
module DlaModules
  implicit none
  real(8), parameter :: pi =  acos(-1.d0)
  integer, parameter :: idx(0:8) = (/-1, -1, -1,  0, 0, 0,  1, 1, 1/),&
                        idy(0:8) = (/-1,  0,  1, -1, 0, 1, -1, 0, 1/)
contains

  integer function bernoulli(p)
    real(8) :: p, numAleatorio
    call random_number(numAleatorio)
    if ( numAleatorio < p ) then
      bernoulli = 1
    else
      bernoulli = 0
    end if
  end function bernoulli

  subroutine andarDoBebado(x,  y)
    integer, intent(out) :: x, y
    real(8) ::  numAleatorio
    integer ::  idr

    call random_number(numAleatorio)
    idr = nint(8*numAleatorio)

    x = x + idx(idr)
    y = y + idy(idr)
  end subroutine andarDoBebado
```

```
subroutine escrevePonto(x,  y, numPontos, z)
   integer, intent(in) :: x, y, numPontos
   integer :: z
   write(1, *) x, y, numPontos, z
end subroutine escrevePonto


integer function numVizinhos(x, y, tabuleiro)
   integer :: x, y, tabuleiro(-100:100,0:100)
   numVizinhos = tabuleiro(x+1,y) &
      + tabuleiro(x-1,y) + tabuleiro(x,y+1) &
      + tabuleiro(x, y-1)
end function numVizinhos


subroutine geraPonto(z, tabuleiro, numPontos)
   integer :: x, y, tabuleiro(-100:100,0:100), numPontos
   integer, intent(out) :: z
   real(8) :: numAleatorio
   ! y0 pertence [yMin - dy, yMin + dy]
   integer :: yMin, dy

   ! gera x
   call random_number(numAleatorio)
   yMin = z + 5
   dy = nint(0.5d0*yMin)
   y = (yMin + nint(dy*numAleatorio))

   call random_number(numAleatorio)
   x = nint(200*numAleatorio-100)

   do while ( x /= 0 .and. x < 1.5d0*yMin .and. &
       abs(x) < 99 .and. abs(y) < 99 )
      call andarDoBebado(x,y)
      ! Achou local para acoplar, atualiza tabuleiro
      if ( numVizinhos(x,y,tabuleiro) > 0 ) then
         numPontos = numPontos + 1
         tabuleiro(x,y) = numPontos
         if ( y > z  ) then
            z = y
         end if
         call escrevePonto(x, y, numPontos, z)

         exit
      end if
```

```
        end do

    end subroutine geraPonto


end module DlaModules

program DLA
    use DlaModules
    implicit none
    integer :: tabuleiro(-100:100,0:100)
    integer :: numPontos, z = 0, i = 0
    ! Iniciamos nosso tabuleiro
    tabuleiro = 0

    open(1, file="saida-4-11820833")
    do i = -100,100
        tabuleiro(i,0) = 1
        call escrevePonto(i, 0, 0, 0)
    end do

    numPontos = 1

    do while (numPontos .le. 3000 .and. z < 90)
        call geraPonto(z, tabuleiro, numPontos)
    end do
    close(1)


end program DLA
```

Resultados

Temos a seguir 4 interações



Onde novamente as cores indicam o número de pontos quando a molécula se agregou ao aglomerado

Modelo de Revoluções populares

> An specter is haunting the lattice, the specter of communism – John Lennon

Buscamos fazer um experimento diferente: Se ao invés de moléculas andando como bêbadas, tivermos nosso aglomerado andando e agragando as moléculas onde esbarram, teremos um novo modelo de crescimento.

Programa

Dessa vez, ao invés de termos uma grade, fizemos duas arrays, uma guardando a posição de todas as moléculas. e outra contendo os nossos revolucionários.

Em cada interação, moviamos os revolucionários viamos quais células eram vizinhas a eles. Essas células são então removidas do vetor de todas as moléculas (No caso, jogadas para um ponto bem longe do limite do tabuleiro) e inseridas às massas mobilizidas.

Por fim, a configuração de "todas as moléculas", foi dada por uma inicialização que, com uma chance p, adicionava uma nova molécula, representada por sua posição, no vetor de todas moléculas.

Para guardar a posição das moléculas, criamos uma struct vértice.

Listing 5: Programa usado na tarefa 5

```
module RevolutionModule
  implicit none
  real(8), parameter :: pi =  acos(-1.d0)
  integer, parameter :: idx(0:8) = (/-1, -1, -1,  0, 0, 0,  1, 1, 1/),&
                        idy(0:8) = (/-1,  0,  1, -1, 0, 1, -1, 0, 1/)


  type :: vertice
    integer :: x
    integer :: y
  contains
    procedure :: mudaVertice
  end type

contains

  subroutine mudaVertice(this, dx, dy)
    class(vertice) :: this
    integer :: dx, dy
      this%x = this%x + dx
      this%y = this%y + dy
  end subroutine mudaVertice


  subroutine andarDoBebado(x,  y)
    integer, intent(out) :: x, y
    real(8) ::  numAleatorio
    integer ::  idr

    call random_number(numAleatorio)
    idr = nint(8*numAleatorio)

    x = x + idx(idr)
```

```
    y = y + idy(idr)
end subroutine andarDoBebado


subroutine inciaTabuleiro(pontosIniciais,&
     tamanhoIniciais, tamanhoTabuleiro, p)
    type(vertice), intent(out) :: pontosIniciais(:)
    integer, intent(in) :: tamanhoTabuleiro
    real(8), intent(in) ::  p
    integer :: tamanhoIniciais, x, y, ponto, numPontos = 0

    percorre_tabuleiro: do x = -tamanhoTabuleiro/2, tamanhoTabuleiro/2
        do y = -tamanhoTabuleiro/2, tamanhoTabuleiro/2
            ponto = bernoulli(p)

            if ( ponto == 1 ) then
                numPontos = numPontos+1
                call insereVertice(pontosIniciais,&
                    tamanhoIniciais, x, y)
            end if

        end do
    end do percorre_tabuleiro

end subroutine inciaTabuleiro


integer function raioQuad(x, y)
    integer, intent(in) :: x, y
    raioQuad = x**2 + y**2
end function raioQuad


integer function bernoulli(p)
    real(8) :: p, numAleatorio
    call random_number(numAleatorio)
        if ( numAleatorio < p ) then
            bernoulli = 1
        else
            bernoulli = 0
    end if
end function bernoulli


subroutine insereVertice(pontos,  tamanhoPontos, x, y)
    type(vertice), intent(out) :: pontos(:)
```

```
    integer, intent(inout) ::  tamanhoPontos
    integer :: x, y

    tamanhoPontos = tamanhoPontos + 1
    pontos(tamanhoPontos) = vertice(x,y)
end subroutine insereVertice

subroutine removeVertice(pontos, i, tamanhoTabuleiro)
    type(vertice), intent(in) :: pontos(:)
    integer, intent(in) ::  i, tamanhoTabuleiro

    ! Joga os pontos pra bem longe para não ter risco de interferência
    ! no resto do jogo
    call pontos(i)%mudaVertice(-10*tamanhoTabuleiro,-10*tamanhoTabuleiro)
end subroutine removeVertice

subroutine moveRevolucao(pontosRevolucao, &
            tamanhoRevolucao)
    type(vertice), intent(in) :: pontosRevolucao(:)
    integer, intent(out) :: tamanhoRevolucao
    integer :: dx, dy, i

    dx = 0; dy = 0
    call andarDoBebado(dx, dy)
    do i = 1, tamanhoRevolucao
       call pontosRevolucao(i)%mudaVertice(dx,dy)
    end do

end subroutine moveRevolucao


subroutine aglutinaCamaradas(pontosRevolucao, pontosIniciais,&
     tamanhoRevolucao, tamanhoIniciais, tamanhoTabuleiro, rQuad)

    type(vertice) :: pontosRevolucao(:), pontosIniciais(:)
    integer :: tamanhoIniciais, tamanhoTabuleiro, tamanhoRevolucao
    integer :: i, j, x_i, y_i, x_r, y_r,&
          x_trotsky, y_trotsky, tamanhoRevolucaoAtual
    logical :: vizinho
    real(8) :: rQuad, rQuad_i

    tamanhoRevolucaoAtual = tamanhoRevolucao
    ! Ponto inicial do cluster
    x_trotsky = pontosRevolucao(1)%x
    y_trotsky = pontosRevolucao(1)%y
```

```
      do i = 1, tamanhoIniciais
         x_i = pontosIniciais(i)%x
         y_i = pontosIniciais(i)%y

         ! Devemos tomar cuidado de usar o tamanho da revolução
         ! no inicio da iteração, já que esse valor se altera no loop
         do j = 1, tamanhoRevolucaoAtual

            x_r = pontosRevolucao(j)%x
            y_r = pontosRevolucao(j)%y
            ! Checa se o ponto é vizinho de um revolucionario
            vizinho = &
               (x_i == x_r .and. (y_i == y_r+1 .or. y_i == y_r-1)) &
               .or. (y_i == y_r .and. (x_i == x_r+1 .or. x_i == x_r-1))

            if ( vizinho ) then
               call insereVertice(pontosRevolucao, tamanhoRevolucao, x_i, y_i)

               rQuad_i = raioQuad(x_trotsky-x_i, y_trotsky-y_i)

               if (rQuad < rQuad_i ) then
                  rQuad = rQuad_i
               end if
               write(1,*) tamanhoRevolucao, sqrt(rQuad)

               call removeVertice(pontosIniciais, i, tamanhoTabuleiro)
               exit
            end if

         end do

      end do

   end subroutine aglutinaCamaradas

end module RevolutionModule


program trotskyExe
   use RevolutionModule
   real(8), parameter :: p = 0.2d0

   integer, parameter :: tamanhoTabuleiro = 200, &
                         tamanhoListas = 1.5*tamanhoTabuleiro**2*p
   type(vertice) :: pontosIniciais(tamanhoListas),&
                    pontosRevolucao(tamanhoListas)
```

```
    integer :: tamanhoIniciais = 0, tamanhoRevolucao = 0, i = 0
    real(8) :: rQuad = 0.d0
    ! Cria sociedade capitalista fadada ao fracasso
    call inciaTabuleiro(pontosIniciais, tamanhoIniciais, &
                        tamanhoTabuleiro, p)

    ! Um espectro ronda o tabuleiro, o espectro do comunismo
    call insereVertice(pontosRevolucao, tamanhoRevolucao, 0, 0)
    open(1, file="saida-5-11820833")

    do while (i < 2000 .and. rQuad < 0.7*(tamanhoTabuleiro/2)**2. .and. tamanhoRevolucao < 5000)
       i = i + 1
       call moveRevolucao(pontosRevolucao, tamanhoRevolucao)
        call aglutinaCamaradas(pontosRevolucao, pontosIniciais,&
              tamanhoRevolucao, tamanhoIniciais, tamanhoTabuleiro, rQuad)
    end do
    close(1)

end program trotskyExe
```

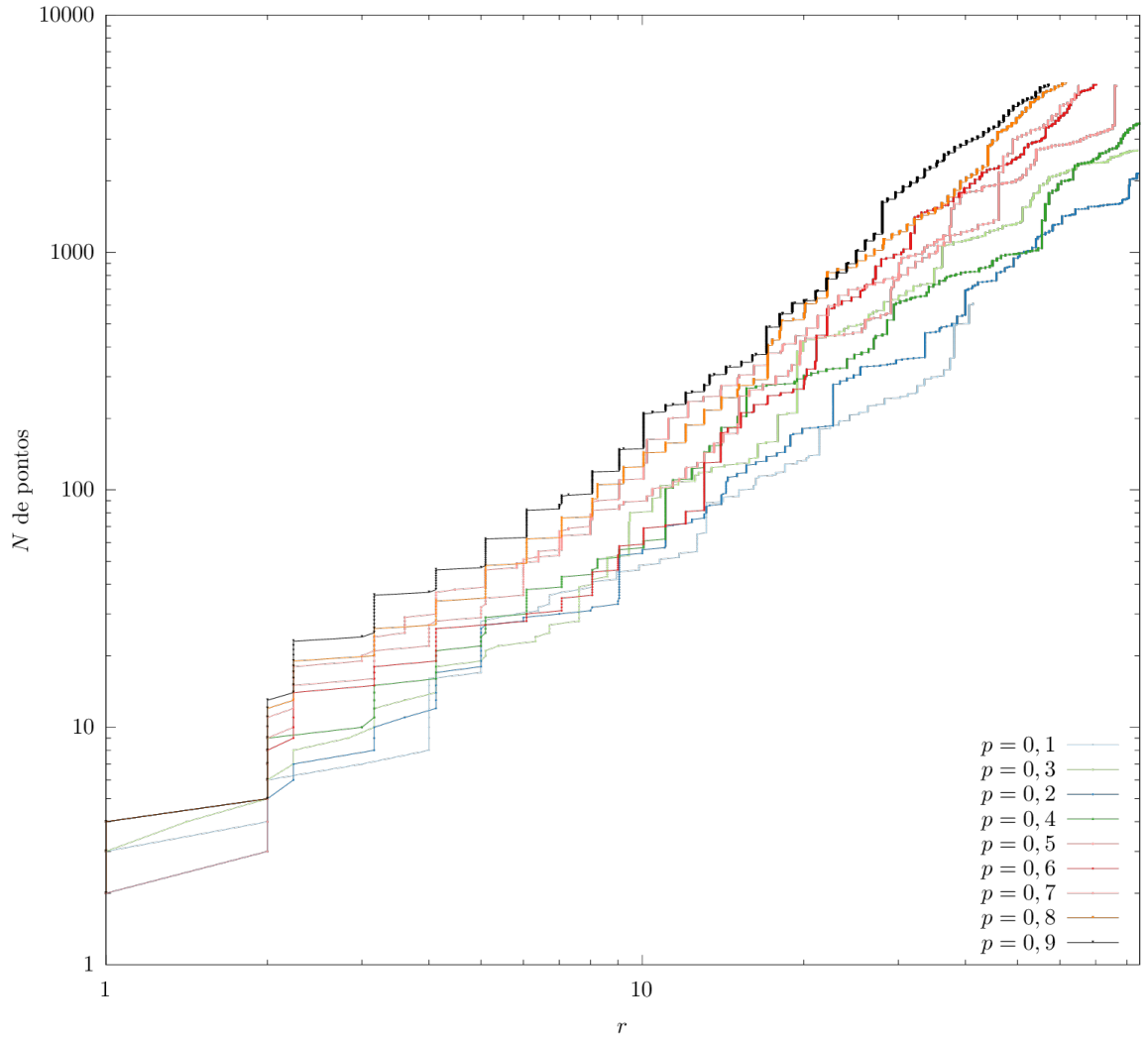
Infelizmente, esse código não permite visualizar a configuração final

Resultados

O gráfico de df para diferentes valores de p está a seguir:

Gráfico de df para diferentes p

Modelo de Revoluções



Observamos a tendência de convergência ds valores de df no extremo de r. Caso isso não tenha ficado claro, registrados o valor de df para cada p na tabela abaixo:

p	df
$0,1$	$1,54$
$0,2$	$1,68$
$0,3$	$1,74$

p	df
$0,4$	$1,72$
$0,5$	$1,77$
$0,6$	$2,05$
$0,7$	$1,90$
$0,8$	$2,00$
$0,9$	$1,94$