7600017
Introdução à Física Computacional

Projeto 6: Sistemas aleatórios

Prof: José A. Hoyos

Victor Foscarini Almeida
nUsp: 10728101

São Carlos, 2019

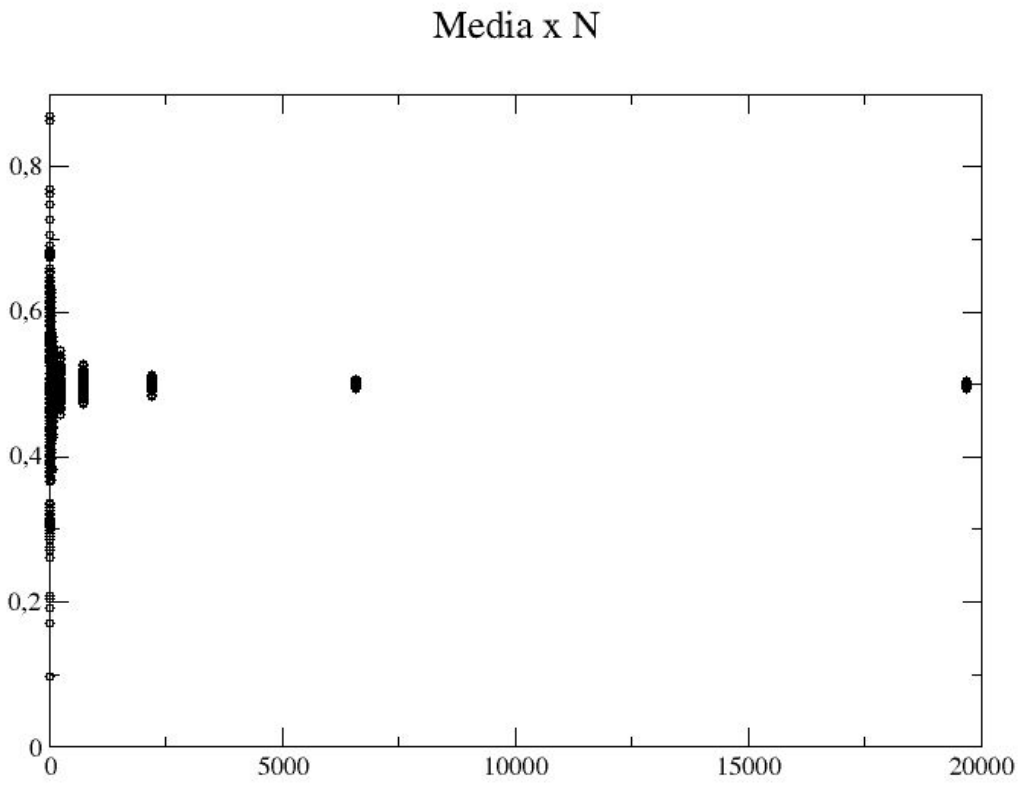
Introdução

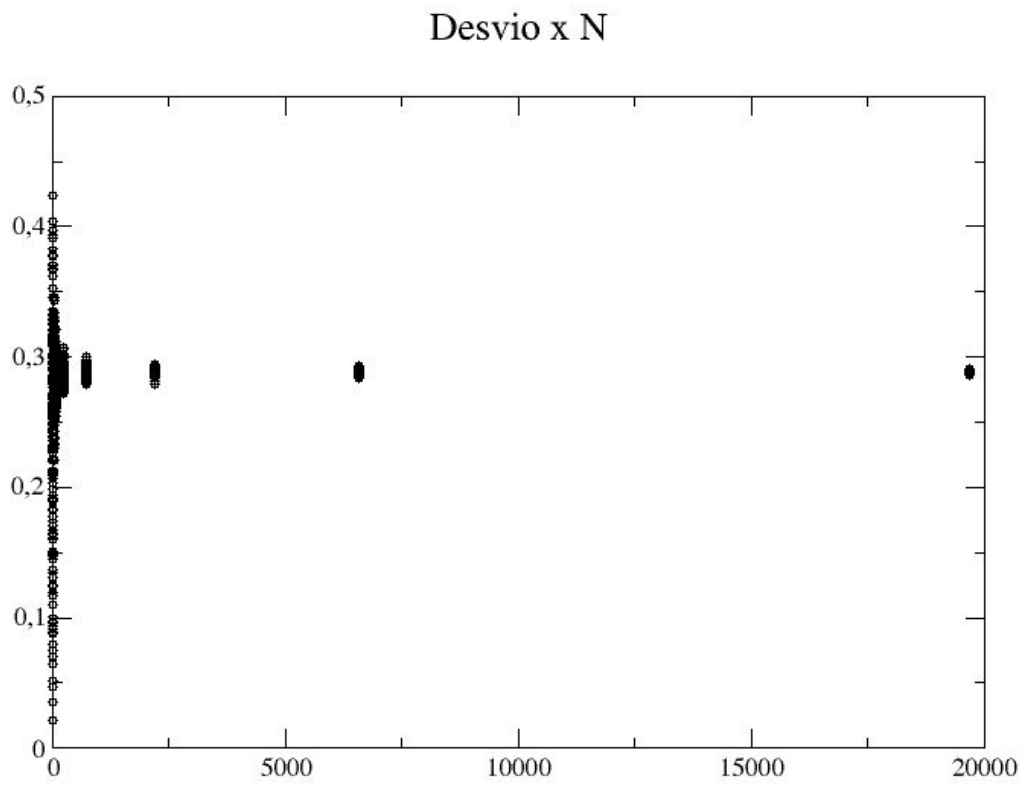
Desenvolvimento e Resultados

1- Números aleatórios, integrais e amostragem

1.a - Média e Desvio padrão

Para se estudar os números aleatórios, serão geradas 100 sequências destes de tamanhos diferentes(N), a partir da função subrotina intrínseca do Fortran call Random_Number(r). Nos gráficos abaixos, foram comparados os desvios e as médias para estas sequências, onde para cada valor de N, estão inseridos os 100 valores obtidos de média e desvio.

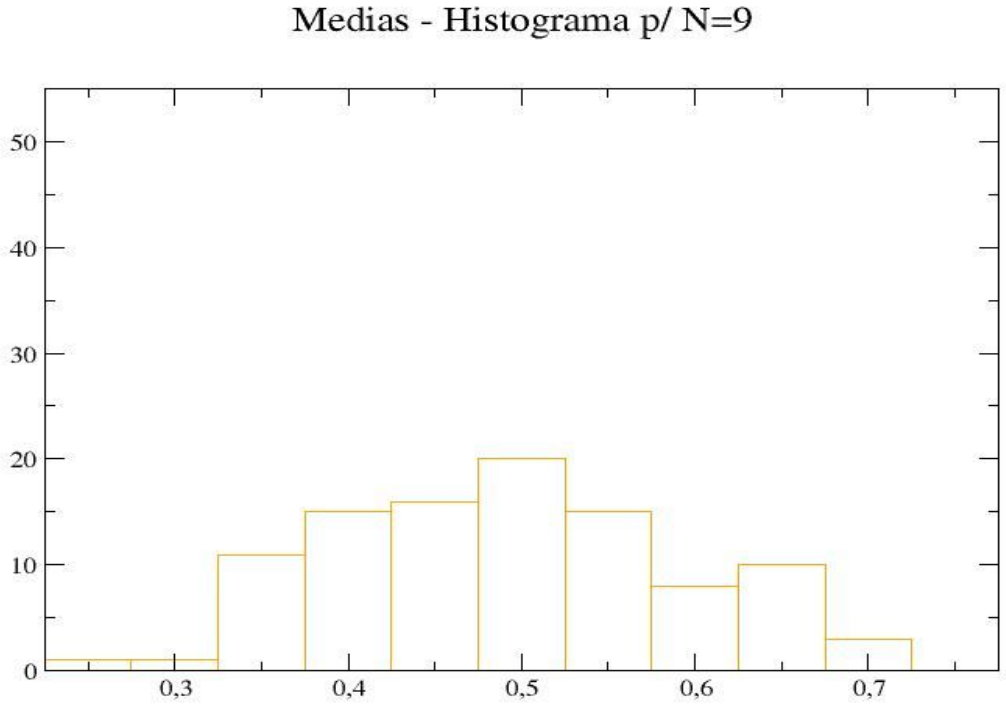




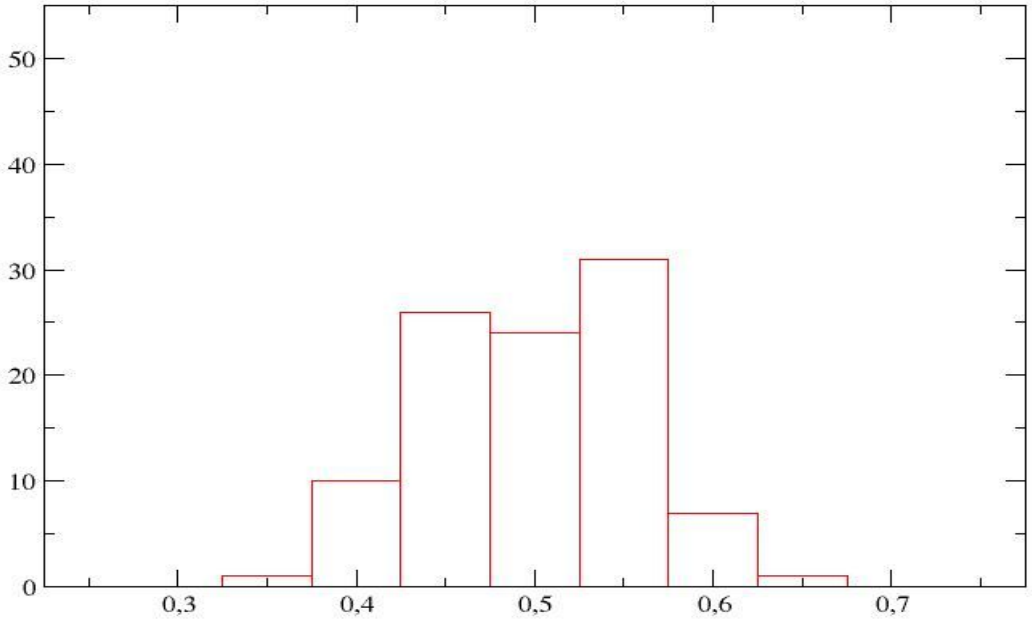
Nota-se que tanto no caso da média em função de N quanto no caso do desvio em função de N, quanto mais números aleatórios(N) são utilizados mais o valor se aproxima do esperado teoricamente. Para o caso com 3^9 números aleatórios, na última das 100 iterações, obteve-se o valor de **média = 0.49883695649922621 e desvio = 0.29020122370781642** , valores próximos dos esperados de média = 0.5 e desvio = 0.288675. Pelos gráficos, é possível notar claramente que para cada valor de N, as 100 iterações oscilam em torno desses valores esperados, porém com N menor, as oscilações são maiores e para N maior, elas são menores.

1.b - Histograma

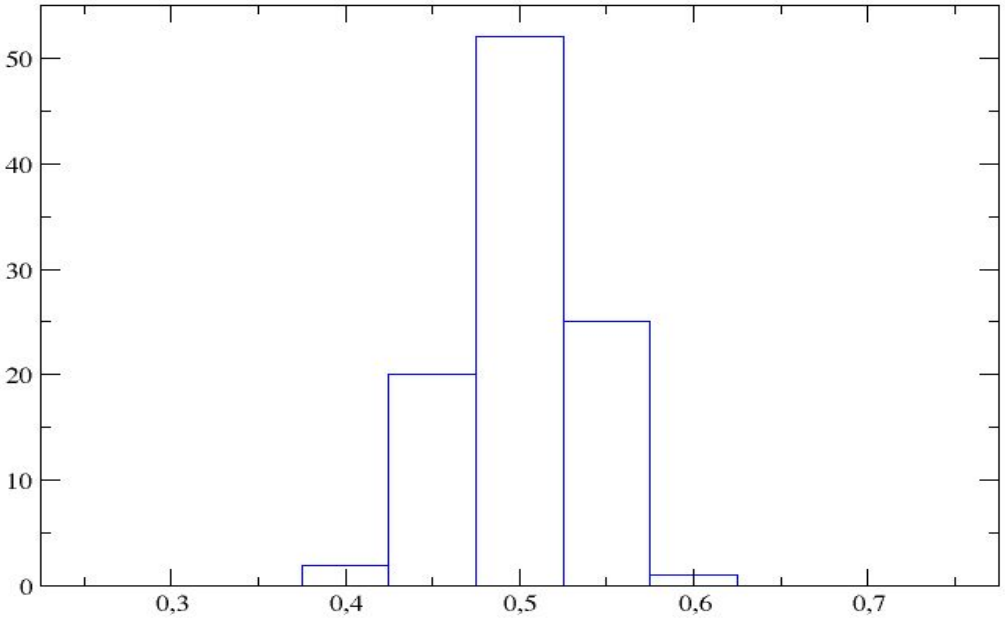
Utilizando gráficos na mesma escala, foram plotados os histogramas para as as médias dos N números aleatórios gerados pelo Fortran. É possível notar que, quanto maior o número de números aleatórios gerados nas sequências, mais o histograma fica concentrado no valor de 0.5, ou seja, menor é o desvio das médias.



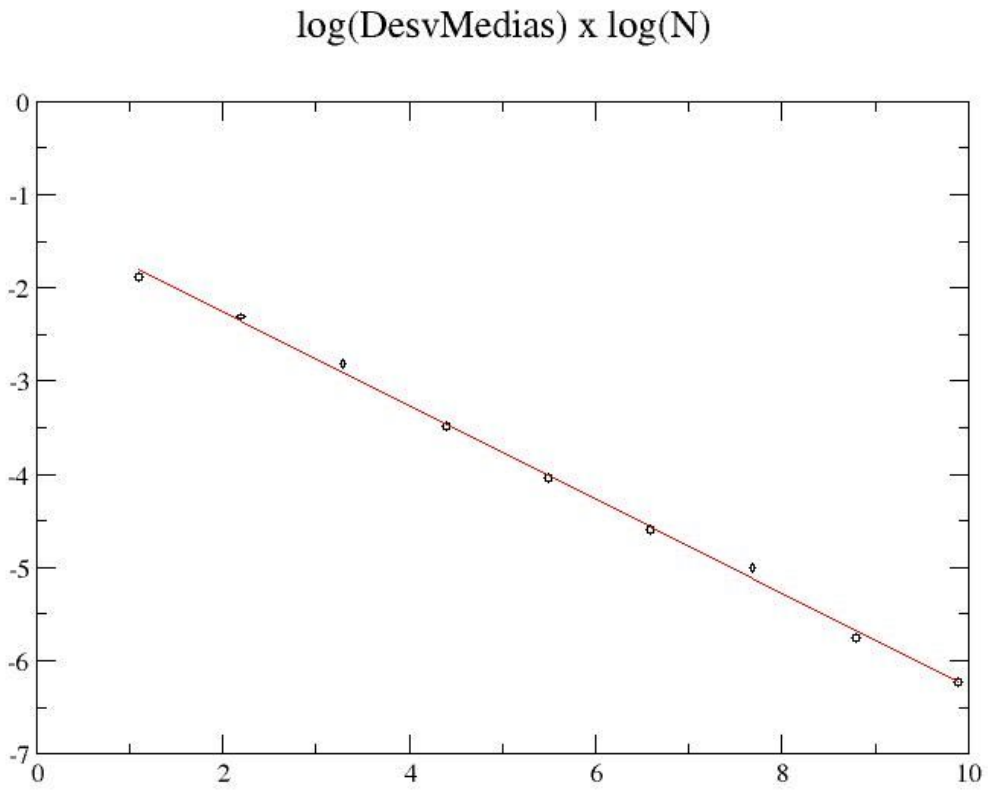
Medias - Histograma p/ N=27



Medias - Histograma p/ N=81



Então, notando essa relação de que quanto maior a quantidade de números aleatórios em uma sequência, menor o desvio das médias, calculou-se o desvio das médias, que foi plotado em função da quantidade de números aleatórios em uma sequência(N) em um gráfico logxlog.



Linearizando esse gráfico na grace, obteve-se a seguinte equação:

y = -1.249 -0.50319 * x

Conclui-se então, que o desvio das médias é proporcional a $N^{-0.5}$, o que nos leva ao teorema central do limite.

Código Fortran utilizado nos itens 1.a e 1.b:

```
program numerosaleatorios

implicit none

real*8 num,desvio,media,medmedias,Desvmedias
real*8,dimension(3**9) :: numeros
real*8,dimension(100) :: medias
```

```
real*8,external :: Desv
integer i,j,N,M
character :: c(1)

open(10,file="media.dat")
open(20,file="desvio.dat")

open(30,file="desvmediasxN.dat")

do M=1,9

        N = 3**M
        c(1) = char(48+M)

        open(40,file="medias"//c(1)//".dat")

        medmedias = 0

        do j=1,100

                media = 0

                do i=1,N

                        call Random_Number(num)
                        numeros(i) = num
                        media = media+num

                enddo

                media = media/N
                desvio = Desv(media,numeros,N)

                medias(j) = media

                medmedias = medmedias + media

                write(10,*)N,media
                write(20,*)N,desvio
                write(40,*)j,media
```

```
        enddo

        medmedias = medmedias/100
        Desvmedias = Desv(medmedias,medias,100)

        write(30,*)log(real(N)),log(Desvmedias)

        close(40)


enddo

end program

function Desv(media,numeros,N)
        implicit none
        real*8 :: media
        real*8,dimension(100) :: numeros
        integer :: N
        real*8 Desv
        integer i


        Desv = 0

        do i=1,N
      Desv = Desv + ( numeros(i) - media )**2
   enddo

   Desv = ( Desv/(N) ) ** 0.5

        return
end function
```

1.c - Área de um círculo por números aleatórios

Neste item, será mostrado que partir da geração de números aleatórios, é possível calcular o valor de pi. Gerando dois números aleatórios entre 0 e 1, que serão os valores das coordenadas (x,y), checa-se se a coordenada aleatória está dentro do círculo e assim conta-se o número de coordenadas que estão dentro do círculo e, a partir disso, encontra-se a relação entre as áreas do círculo e do quadrado no qual ele está contida(variando as coordenadas 0 e 1 em ambos os eixos X e Y obtém-se um quadrado). Note que aqui estamos verificando ¼ de círculo ao invés de 1 círculo inteiro(as coordenadas vão de 0 a 1 ao invés de -1 a +1), porém pela simetria do círculo, a razão entre as áreas é a mesma.

Assim, a interpretação geométrica deste algoritmo é a possibilidade de encontrar a razão entre as áreas/volume de duas superfícies/sólidos a partir de números aleatórios e da função que gera essa superfície/sólido.

No caso trabalhado, foi obtido um valor de pi, com precisão na quarta casa decimal, de **3.1416212200000002** sendo necessário gerar **10^9** números aleatórios. Abaixo estão os valores de pi gerados para a geração de uma quantidade de números aleatórios múltiplos de 10, que foi parada quando obteve-se a precisão buscada.

```
        10   2.7999999999999998
       100   3.1600000000000001
      1000   3.1160000000000001
     10000   3.1383999999999999
    100000   3.1410399999999998
   1000000   3.1405280000000002
  10000000   3.1422867999999999
 100000000   3.1413151199999998
1000000000   3.1416212200000002
 3.1416212200000002       1000000000
```

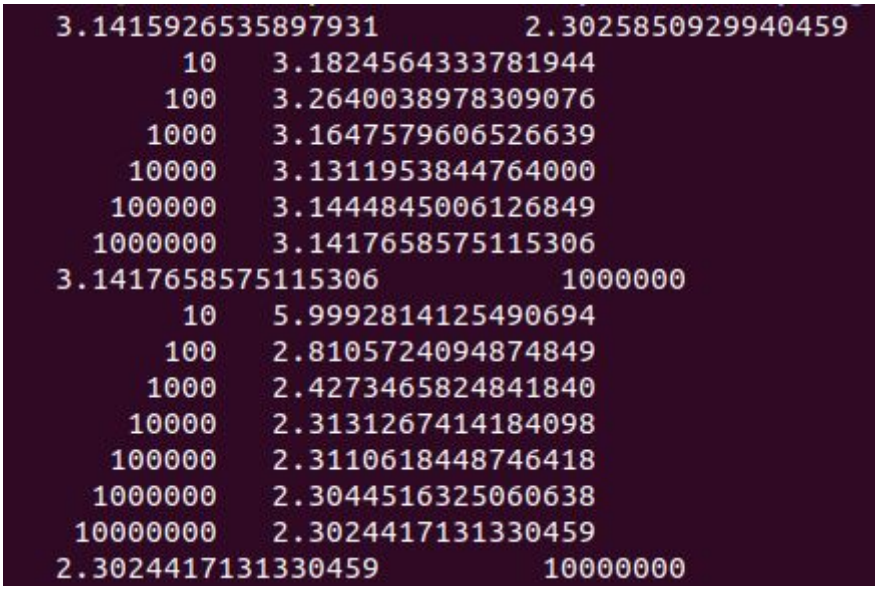
```
program circulo

implicit none

real*8 x,y,eps,pi,Ai
real*8 Ni !Ni se refere ao número de pontos que satisfazem a condição
integer i,N

N = 1
eps = 1.d0 !definição para poder iniciar o loop

do while(eps>0.0001) !busca-se precisão de 4 casas decimais
```

```
        Ni = 0
        N = N*10
        do i=1,N

                call Random_Number(x)
                call Random_Number(y)

                if(x**2 + y**2 < 1) then !a condição é estar dentro do círculo
                        Ni = Ni + 1
                endif

        enddo

        pi = 4 * Ni/N !relação obtida das areas do circulo e do quadrado
        eps = abs(3.1416 - pi)

        write(*,*)N,pi

enddo

write(*,*) pi,N

end program
```

1.d - Método de Monte Carlo

O método Monte-Carlo foi utilizado neste item para calcular duas integrais definidas. Uma das principais vantagens do método é quando deseja-se realizar integração para dimensões maiores e ele é baseado no teorema do valor médio.

Notou-se que, até mesmo devido à natureza aleatória do método, cada vez que ele era rodado, eram obtidos valores diferentes de números aleatórios “N” necessários para se chegar na precisão pedida, porém sempre variando entre 10**5 e 10**7. Notou-se também que, na segunda integral, que foi feita no intervalo 1:10, foram necessários ser gerados mais números aleatórios para que o método seja eficiente. Por fim, tomou-se abaixo os valores obtidos, na primeira linha os valores esperados das integrais, após os obtidos para cada valor de números aleatórios gerados, até por fim chegar no resultado buscado.




```
program montecarlo

implicit none

real(8),external :: Fmc !função que realiza a integração pelo
método Monte-Carlo
real(8) eps
integer N,j
```

```
real(8),dimension(2) :: Inte,Esperado

!abaixo estão os valores das integrais obtidas analiticamente
Esperado(1) = 4 * ( atan(1.d0) - atan(0.d0) )
Esperado(2) = log(10.d0) - log(1.d0)

write(*,*)Esperado(1),Esperado(2)


!foi utilizado um "j" para mais na frente selecionar qual a função
que será integrada
eps = 1.d0
N = 1.d0
j=1
do while(eps>0.001d0) !realiza-se a integração com a geração de
uma quantidade de números aleatórios N maioir até que se encontre
o valor desejado

     N = N*10

     Inte(j) = Fmc(0.d0,1.d0,N,j)

     eps = abs(Esperado(j)-Inte(j))

enddo

write(*,*)Inte(j),N

eps = 1.d0
N = 1.d0
j=2
do while(eps>0.001d0) !processo de integração semelhante ao
anterior porém com "j" diferente

     N = N*10

     Inte(j) = Fmc(1.d0,10.d0,N,j)

     eps = abs(Esperado(j)-Inte(j))

enddo

write(*,*)Inte(j),N
```

```
end program

function Fmc(a,b,N,j) !realiza-se a integração por Monte-Carlo

      implicit none

      real*8 :: a,b
      integer :: N,j
      real*8,external :: F
      real*8 Fmc
      real*8 rand,fmed
      integer i

      do i=1,N
            call Random_Number(rand)
            rand = a + rand*(b-a)
            fmed = fmed + F(rand,j)
      enddo

      Fmc = (b-a)/N * fmed

      write(*,*)N,Fmc

      return
end function



function F(x,j) !função utilizada para chamar a função que está
sendo integrada, apesar de o "if" selecionando qual das integrais
será utilizada ser menos eficiente para o computador, ele facilita
o trabalho do programador de realizar a integração duas vezes

      implicit none

      integer :: j
      real*8 :: x
      real*8 F

      if (j==1) then
            F = 4/(x**2 + 1)
      elseif (j==2) then
            F = 1/x
      endif
```

```
     return
end function
```

2- Decaimento Exponencial

2.a - Código para o decaimento

Abaixo está o programa utilizado para simular o decaimento de N0 núcleos radioativos a partir da vida média(chamada de tau).

```
program decaimento
     implicit none
!N é o número de átomos que será atualizado a cada iteração do
programa e Tmax é o tempo máximo
!lambda é a constante de decaimento do átomo,dT é o intervalo
de tempo considerado para o decaimento e random o número
aleatório que será chamado
!note que os parametros tau(vida média),N0(número inicial de
átomos) e dT(intervalo de tempo) e Tmax(tempo da simulação)
devem ser inseridos para cada caso que for ser iterado
     integer :: N,i,j
     real*8 :: lambda,random
     real*8,parameter :: tau = ,N0 = ,dT = ,Tmax =

     N = N0
     lambda = 1/tau

     open(10,file="decaimento.dat")

!loop para checar realizar as iterações da simulação e checar,
para cada átomo, se ele irá decair nessa iteração
     do i =1, int(Tmax/dT)
          do j = 1, N
               call random_number(random) !gera um número
pseudo-aleatório entre 0 e 1
               if (random <= (lambda*dT)) then
```

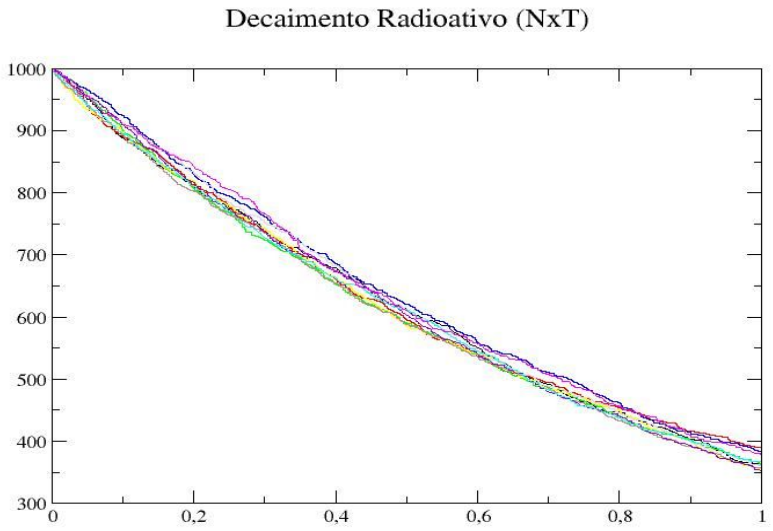
```
                    N = N - 1
                end if
            end do
            write (10,*) i*dt, N
        end do


end program
```

2.b - A simulação

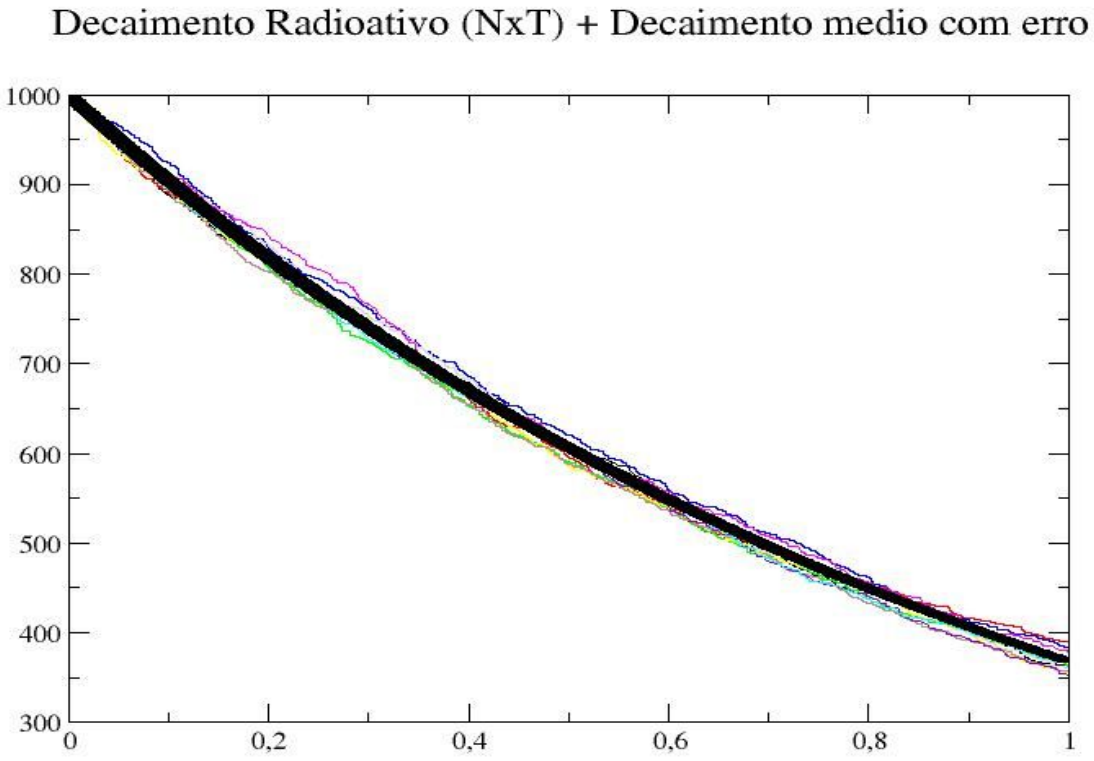
O programa do item anterior foi expandido para que se possa realizar o processo de simulação k=10^3 vezes, salvando os processos de decaimento em um vetor para que se possa calcular a média e o desvio padrão e erro, resultando assim em um processo de decaimento médio obtido a partir das diversas simulações feitas. Foram utilizados os valores dados no projeto. Note que o código foi feito de forma que seja simples modificar os valores da condição inicial e da precisão/tempo de simulação mudando apenas os parâmetros de algumas variáveis.

Foi utilizado um valor de Tmax de 1 e de Dt de 10^{-3} para obter um decaimento bom(os valores de núcleos caem de cerca de 100 para algo próximo a 300), sem sobrecarregar demais o computador, visto que foi necessário repetir essa simulação 10^3 vezes.

Para as primeiras 10 simulações feitas, obteve-se o seguinte gráfico de decaimento(Número de núcleos/átomos em função do tempo):



Então, plotou-se junto com os gráficos dos 10 primeiros decaimentos o decaimento médio junto com as barras de erro.



Por alguma razão, no gráfico parece que os erros são maiores no início da simulação(quando o número de átomos é maior), porém o que acontece é exatamente o contrário, como podemos ver nos dados abaixo, que mostram **Tempo**, **Número de átomos** e **Erro**, respectivamente, nas tabelas.

Nota-se que os erros são menores no início da simulação e vão aumentando conforme o programa é rodado, chegando ao erro no final ser mais de 10 vezes maior que o erro inicial. Isso ocorre pois uma diferença aleatória gerada no início afeta todo o loop que depende das condições anteriores, assim o erro vai aumentando conforme o código é rodado.

Esses resultados são interessantes pois nos mostram como é possível confiar em simulações que predizem eventos próximos, porém quanto mais distante mais difícil de a previsão estar correta.

1.0000000000000000E-003	998.99599999999998	3.1811695962334088E-002
2.0000000000000000E-003	997.97500000000002	4.4185687728041059E-002
3.0000000000000001E-003	996.98199999999997	5.5332413646975998E-002
4.0000000000000001E-003	995.99500000000000	6.3426926458720917E-002
5.0000000000000001E-003	994.95500000000004	7.0646832908488819E-002
6.0000000000000001E-003	993.98500000000001	7.7037490872950615E-002
7.0000000000000001E-003	992.96900000000005	8.3306896473220912E-002
8.0000000000000002E-003	991.97000000000003	8.9180154743081108E-002
9.0000000000000011E-003	990.99199999999996	9.4052836214544869E-002
1.0000000000000000E-002	989.96600000000001	9.8513166632688998E-002

0.98999999999999999	371.59300000000002	0.47963251662079781
0.99099999999999999	371.22699999999998	0.47808521311582142
0.99199999999999999	370.83100000000002	0.47795861640941251
0.99299999999999999	370.45999999999998	0.47816566166967756
0.99399999999999999	370.10899999999998	0.47787772390016281
0.99500000000000000	369.72699999999998	0.47746043919889292
0.99600000000000000	369.35300000000001	0.47786231385201317
0.99700000000000000	368.97399999999999	0.47747599311378996
0.99800000000000000	368.61799999999999	0.47715833430843507
0.99900000000000000	368.28100000000001	0.47708074683432800
1.0000000000000000	367.90400000000000	0.47703331539841115

```
program decaimento
     implicit none
!N é o número de átomos que será atualizado a cada iteração do
programa e Tmax é o tempo máximo
!lambda é a constante de decaimento do átomo,dT é o intervalo de
tempo considerado para o decaimento e random o número aleatório
que será chamado
```

```
!note que os parametros tau(vida média),N0(número inicial de
átomos) e dT(intervalo de tempo) e Tmax(tempo da simulação) devem
ser inseridos para cada caso que for ser iterado
!a função Desv realiza o cálculo do desvio padrão
!Niteracoes guarda o numero de iteracoes de cada simulacao

      integer :: N,i,j,l
      real*8 :: lambda,random
    real*8,parameter :: tau = 1.d0,N0 = 10.d0**3,dT =
10.d0**(-3),Tmax = 1.d0
    integer,parameter :: k=10**3,Niteracoes=int(Tmax/dT)
      real*8,dimension(k,  Niteracoes) :: Ns
      real*8,dimension(Niteracoes) :: Nmed,erro
      character :: c(1)
      real*8,external :: Desv

!os valores acima foram escolhidos visando ter um Tmax>>dT e uma
precisão razoável de forma que uma quantidade razoável de átomos
decaia, porém sem que isso aconteça muito devagar ou muito
rapidamente
!foi adicionado uma variável Nmed para guardar os valores
calculados no primeiro loop abaixo
!foi adicionada uma variável "c" para salvar os arquivos do
primeiro loop abaixo
!foi adicionado um parâmetro "k" acima e os loops abaixo para
realizar a simulação k vezes

      lambda = 1/tau

!o primeiro loop é utilizado para plotar os gráficos N(t) e as 10
primeiras simulações
!o segundo loop é utilizado para fazer as k-10 outras iterações em
que não é necessário plotar o gráfico

do l=1,10
    c(1) = char(48+l)
      open(10,file="decaimento"//c(1)//".dat")

      N = N0 !retorna ao numero inicial de átomos

!loop para realizar as iterações da simulação e checar, para cada
átomo, se ele irá decair nessa iteração
      do i =1, Niteracoes
           do j = 1, N
                call random_number(random) !gera um número
pseudo-aleatório entre 0 e 1
```

```
                if (random <= (lambda*dT)) then
                     N = N - 1
                end if
           end do
           Ns(l,i) = N !armazena N
          write(10,*)i*dT,N
     end do

     close(10)

end do

do l=11,k

     N = N0 !retorna ao numero inicial de átomos

!loop para realizar as iterações da simulação e checar, para cada
átomo, se ele irá decair nessa iteração
     do i =1, Niteracoes
          do j = 1, N
                call random_number(random) !gera um número
pseudo-aleatório entre 0 e 1
                if (random <= (lambda*dT)) then
                     N = N - 1
                end if
           end do
           Ns(l,i) = N !armazena N
     end do

end do


!utiliza-se um vetor para guardar a média de cada iteração para as
k simulações
do i=1,Niteracoes
     Nmed(i) = 0.d0
enddo

!calcula-se a média, primeiro somando todos os Ns de cada iteração
de cada simulação e depois dividindo pelo número de simulações

!somando todos os Ns
do i=1,k
     do j=1,Niteracoes
          Nmed(j) = Nmed(j) + Ns(i,j)
     enddo
```

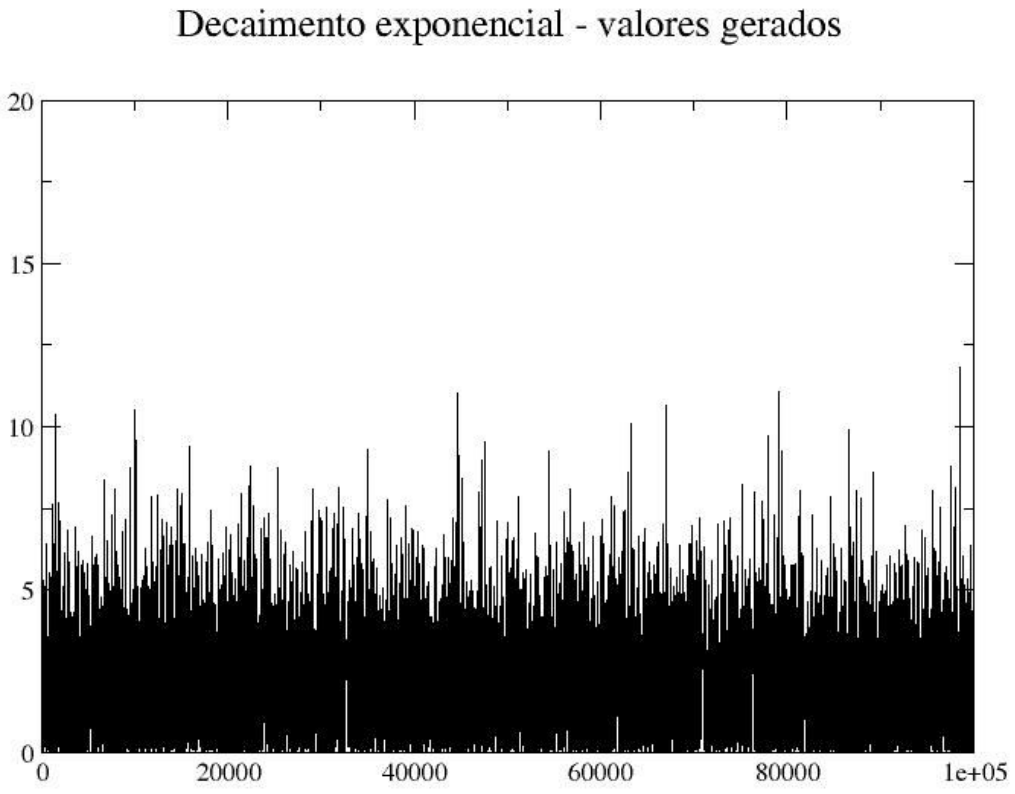
```
enddo

!dividindo pelo número de simulações e calculando o erro
do i=1,Niteracoes
      Nmed(i) = Nmed(i)/k
      erro(i) = Desv(Nmed(i),Ns,Niteracoes,i,k)/sqrt(real(k,8))
enddo

!salva-se as médias de cada iteração das "k" simulações em um
arquivo

open(20,file="mediadecaimento.dat")


do i=1,Niteracoes
      write(20,*) dT*i,Nmed(i),erro(i)
enddo


end program


!função para calcular o desvio de forma semelhante ao exercicio 1
do projeto, porém alterada para receber os valores de N
armazenados no vetor para uma iteração específica(para diversas
simulações)
!foram recebidos os dados do vetor Ns(que é recebido pela variável
números) para realizar o cálculo do desvio padrão
!note que no loop "i" se refere à simulação e toma-se uma iteração
fixa para várias simulações

function Desv(media,numeros,N,iteracao,simulacoes)
      implicit none
      real*8 :: media
      integer :: N,simulacoes,iteracao
      real*8,dimension(simulacoes,N) :: numeros
      real*8 Desv
      integer i


      Desv = 0

      do i=1,N
         Desv = Desv + ( numeros(i,iteracao) - media )**2
     enddo
```

```
    Desv = ( Desv/(N) ) ** 0.5

     return
end function
```

2.c - Distribuição Exponencial

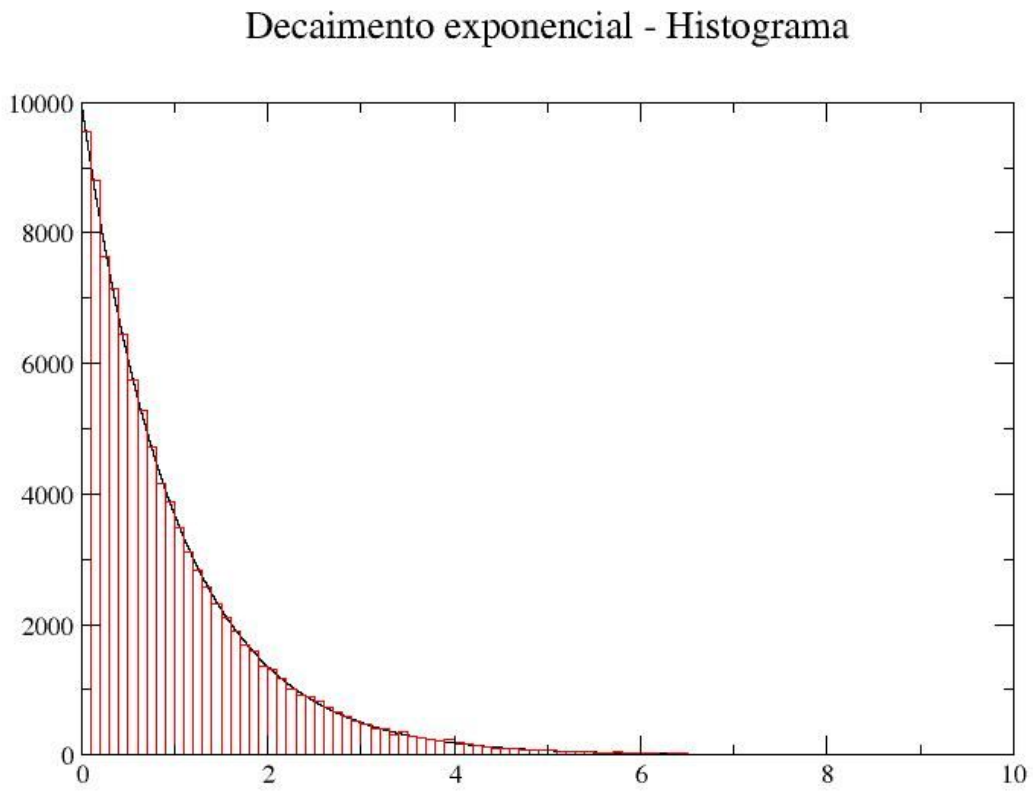
Foram gerados 10^5 números aleatórios a partir da equação abaixo e colocados em um gráfico:

$$x = -\frac{1}{\lambda}\ln(1-r)$$



Então, fazendo um histograma a partir desses números aleatórios gerados a partir da equação acima, nota-se a distribuição exponencial. Para confirmar que ela segue a equação abaixo, plotou-se a equação abaixo junto com o histograma.

$$p(x) = \lambda e^{-\lambda x},\ x \geq 0$$



Foi possível demonstrar então uma forma de se gerar números aleatórios de acordo com uma distribuição não-uniforme, ao utilizar uma função do número aleatórios.

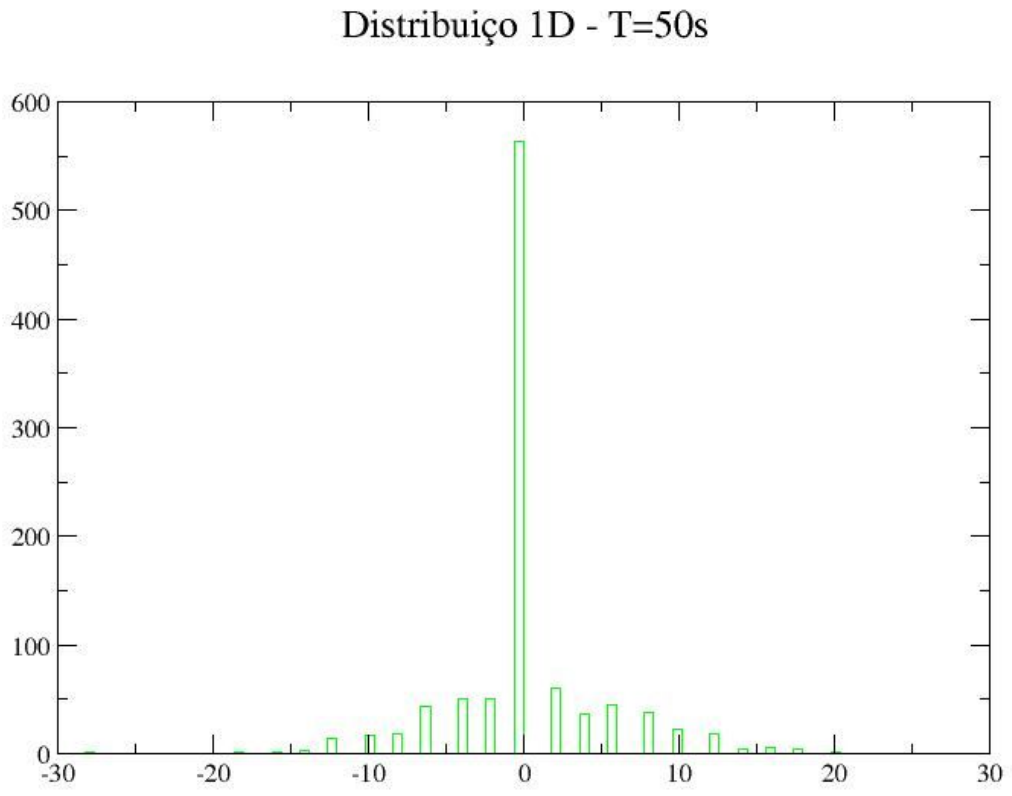
```
program distribuicaoexp

integer i
real*8 random
real*8,parameter :: lambda = 1.d0

open(10,file="distribuicaoexp.dat")
open(20,file="comparacao.dat")
```

```
!aqui são gerados os  números aleatórios a partir da função dada
no projeto, e são salvos em um arquivo para gerar o histograma no
xmgrace

do i=1,10**5

      call Random_Number(random)
      write(10,*) -1/lambda * log(1 - random)

enddo

!aqui é plotada a outra função que será comparada com o histograma
para mostrar a distribuição exponencial
!note que abaixo a função é multiplicada por 10**4, o que é apenas
uma correção quanto à escala do Grace
!visto que no Grace o que será gerado no histograma é, no eixo Y a
quantidade de valores e no eixo X o número aleatório gerado pela
função acima, da forma que o histograma foi feito notou-se que o
topo do gráfico era 10**4

do i=1,10**6


write(20,*)real(i,8)/10**5,lambda*exp(-lambda*(real(i,8)/10**5))*1
0**4

enddo


end program
```

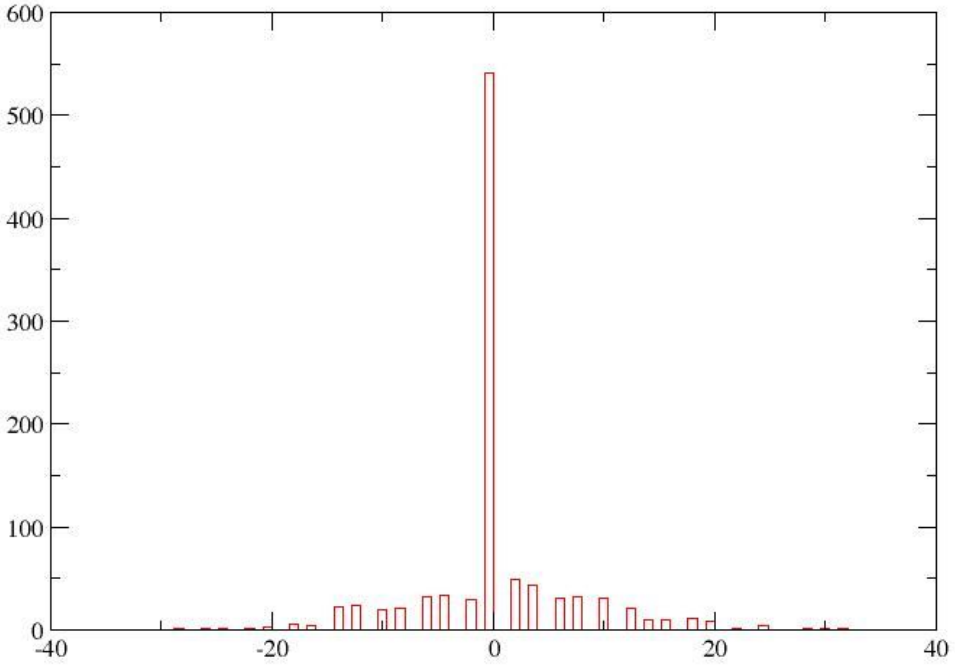
3- Passeio aleatório

3.1- Passeio unidimensional

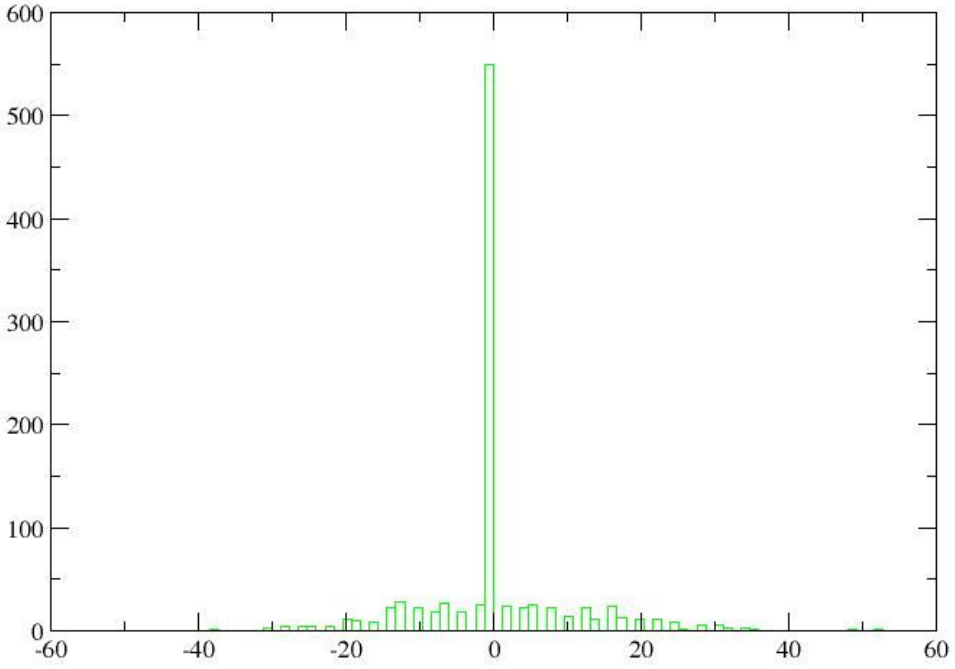
No passeio unidimensional, obtém-se uma distribuição semelhante a uma Gaussiana e muita parecida para os tempos T=50s,T = 100s e T=200s. Ao plotar o gráfico foi possível notar no xmgrace que as escalas mudaram para cada tempo,sendo em 50s o valor máximo chegando a 30, em 100s chegando a 40 e em 200s chegando a 60, o que mostra que para valores de tempo mais alto, foram obtidos valores mais afastados do mais obtido: zero, porém a distribuição não mudou.



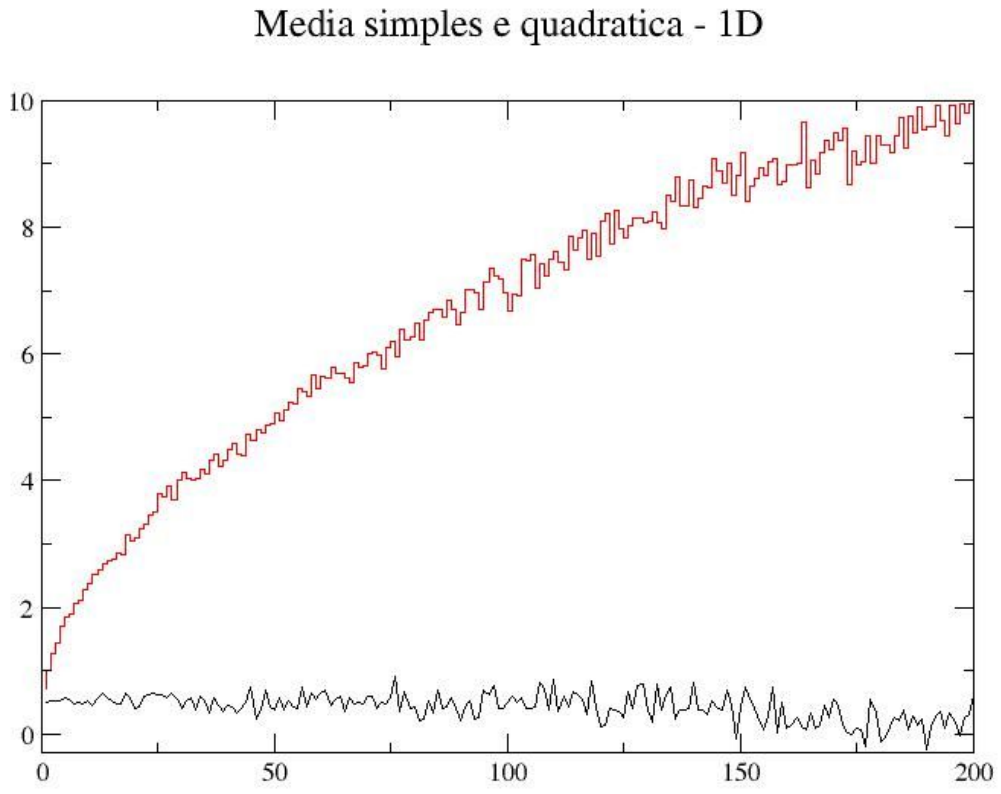
Distribuiço 1D - T=100s



Distribuiço 1D - T=200s



Para o caso unidimensional, obtém-se o caso seguinte, onde nota-se que a média quadrática vai crescendo(ele sempre anda o valor de 1 ou -1, logo sempre há um aumento no quadrado do número), já a média simples tende a se manter constante com algumas oscilações em torno de 0, visto que a chance de a partícula se deslocar na direção negativa ou positiva é a mesma(+1 ou -1).



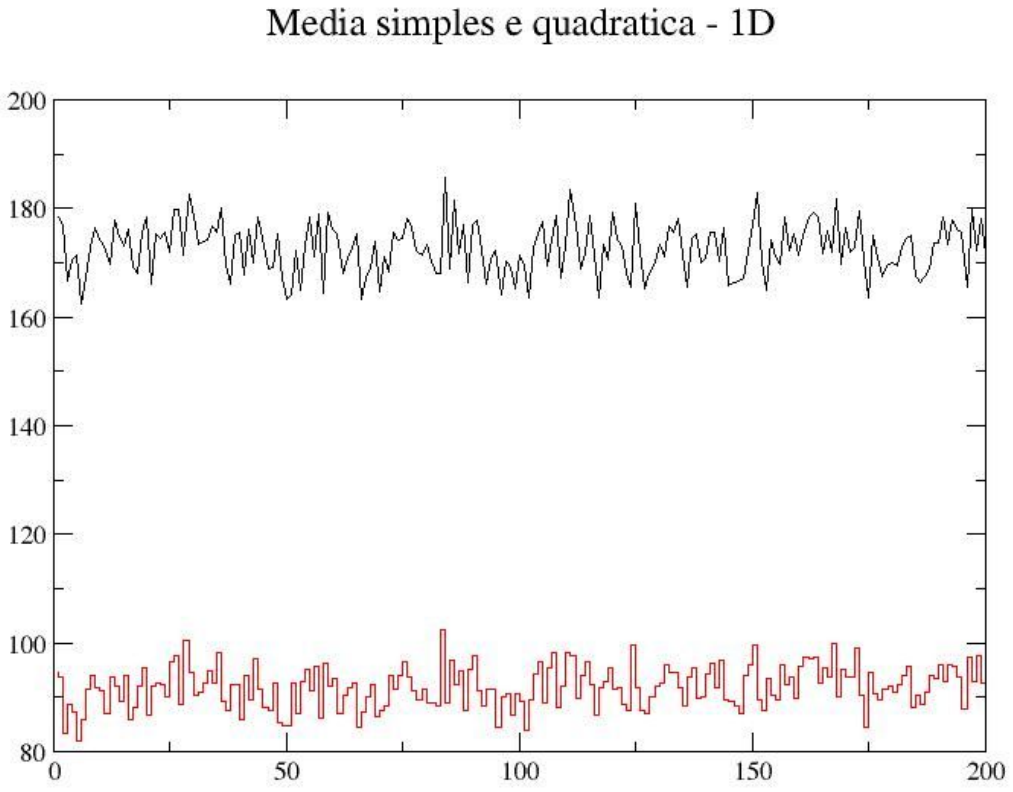

```
program passeio

integer :: i,j,l
real*8 :: x,med,quad,erromed,erroquad
real*8 :: random1,random2
integer, parameter :: k=10**3
real*8,dimension(k,200) :: p
```

```
!realiza-se a simulação k vezes para t=0 até t=200 e salva-se em
um vetor p
do i=1,k
      x=0
      do j=1,200
            call Random_Number(random)
            if(random>0.5d0) then
                  x = x+1.d0
                  p(i,j) = x
            else
                  x = x-1.d0
            endif
      enddo
enddo

!plota-se num gráfico a distribuição para os instantes de tempo
t=50,t=100 e t=200

open(10,file="passeio1d50.dat")
open(20,file="passeio1d100.dat")
open(30,file="passeio1d200.dat")

do j=1,k
      write(10,*) p(j,50)
enddo

do j=1,k
      write(20,*) p(j,100)
enddo

do j=1,k
      write(30,*) p(j,200)
enddo

!calcula-se o deslocamento médio e o deslocamento quadrático médio
e plota-se num gráfico

open(40,file="medio1d.dat")
open(50,file="quadratico1d.dat")

do j=1,200
      med = 0
      quad = 0
      do i=1,k
            med = med + p(i,j)
            quad =  quad + p(i,j)**2.d0
```

```
      enddo
      med = med/real(k,8)
      quad = sqrt(quad/real(k,8))
      write(40,*)j,med
      write(50,*)j,quad
enddo


end program
```

3.2- Passeio bidimensional

Para o caso bidimensional, é interessante notar que o gráfico contendo a média simples e quadrática seguem a mesma direção. Note que elas oscilam de forma semelhante, porém a média simples é menor que a média quadrática.



```
program passeio

integer :: i,j,l
real*8 :: x,y,med,quad,erromed,erroquad
real*8 :: random1,random2
integer, parameter :: k=10**3
real*8,dimension(k,200) :: px,py

!realiza-se a simulação k vezes para t=0 até t=200 e salva-se em
um vetor p
do i=1,k
     x=0
     do j=1,200
          call Random_Number(random)
          if(random<0.25d0) then
               x = x+1.d0
               px(i,j) = x
          elseif(0.25d0<random .and. random<0.5d0) then
               x = x-1.d0
               px(i,j) = x
          elseif(0.5d0<random .and. random<0.75d0) then
               y = y+1.d0
               py(i,j) = y
          else
               y = y-1.d0
               py(i,j) = y
          endif
     enddo
enddo

!calcula-se o deslocamento médio e o deslocamento quadrático médio
e plota-se num gráfico

open(40,file="medio2d.dat")
open(50,file="quadratico2d.dat")

do j=1,200
     med = 0
     quad = 0
     do i=1,k
          med = med + sqrt(px(i,j)**2 + py(i,j)**2)
          quad =  quad + px(i,j)**2.d0 + py(i,j)**2
     enddo
     med = med/real(k,8)
     quad = sqrt(quad/real(k,8))
```

```
      write(40,*)j,med
      write(50,*)j,quad
enddo


end program
```