UNIVERSIDADE DE SÃO PAULO - USP

INTRODUÇÃO À FÍSICA COMPUTACIONAL – 7600017 – 2025/2
PROF. FRANCISCO C. ALCARAZ

RELATÓRIO DO 2° PROJETO

JOÃO VITOR LIMA DE OLIVEIRA - 12694394

São Carlos

2025

Parte I

Introdução Geral

MOTIVAÇÃO

O problema do caminhante aleatório é um modelo simples, mas de enorme importância na física. Ele descreve o movimento de uma partícula que, a cada passo, escolhe sua direção de forma aleatória. Apesar da simplicidade, esse modelo captura a essência de processos estocásticos presentes em muitos sistemas naturais e serve como ponto de partida para o estudo de fenômenos mais complexos. Além disso, no contexto da Física Computacional, a geração de números pseudo-aleatórios desempenha papel fundamental, pois permite a implementação eficiente de simulações que reproduzem esse tipo de processo probabilístico em computadores.

Na física estatística, o caminhante aleatório está diretamente relacionado à difusão, um processo fundamental que explica como partículas se espalham em fluidos e sólidos. A equação da difusão e a equação de Fokker-Planck, por exemplo, podem ser derivadas a partir desse modelo discreto. Isso mostra como um conceito probabilístico simples pode se conectar a leis físicas que regem o transporte de calor, carga elétrica e até a propagação de sinais em materiais. Nesse sentido, os objetivos principais dos estudos computacionais incluem a implementação de geradores pseudo-aleatórios confiáveis, a simulação de caminhantes aleatórios em uma e duas dimensões, e a análise da entropia como medida da desordem do sistema.

Além disso, o estudo do caminhante aleatório tem aplicações que vão além da difusão clássica, alcançando áreas como mecânica quântica, teoria de polímeros e até sistemas biológicos. Ele oferece uma linguagem matemática unificada para descrever flutuações, processos de relaxação e até o comportamento coletivo de sistemas complexos. Assim, ao unir teoria, simulação computacional e análise estatística, o problema do caminhante aleatório revela-se uma ferramenta essencial para compreender e modelar a física em múltiplas escalas.

Parte II

Desenvolvimento

QUESTÃO 1

ENUNCIADO

1. A fim de testarmos o gerador de números aleatórios calculemos alguns momentos da distribuição “aleatória” gerada, isto é:

$$\langle x^n \rangle, \quad \text{para } n = 1, 2, 3, 4. \tag{1}$$

Faça a média acima gerando um número grande N de números aleatórios (escolha apropriadamente N). Que resultado você esperaria? Compare com os resultados esperados e explique os obtidos.

MÉTODO UTILIZADO

Na primeira simulação, foi pedido para encontrar a média da distribuição de números aleatórios, que variam de 0 a 1 os quais estão sendo elevados por uma potência n, Equação 2. Desse modo, foi utilizado a função **rand()**, para gerar números aleatórios entre 0 e 1, ademais, foi dado uma *seed* para ela, o que permite a função gerar números pseudo randômicos baseados no valor dado.

$$\langle x^n \rangle, \quad \text{para } n = 1, 2, 3, 4. \tag{2}$$

Além disso, foram dadas 10^6 interações, ou seja, foram gerados 10^6 números pseudo-aleatórios. Dessa forma, primeiro foi feito um *loop* na qual o exponencial dos números aleatórios, *n*, varia de 1 a 4, e para cada expoente é chamado uma função que realiza o cálcula da média dos valores aleatórios elevados n. Por fim, o resultado final é divido pelo número de interações, o que da a média, e exibido na tela, Fig. 4.

CÓDIGO

O programa `main` tem como objetivo calcular os momentos de ordem $n = 1, 2, 3, 4$ de uma distribuição uniforme de números aleatórios no intervalo $(0, 1)$, isto é, calcular $\langle x^n \rangle$ para N amostras. No início do código é definido o parâmetro `iseed=1154`, que funciona como semente para o gerador de números pseudo-aleatórios, garantindo reprodutibilidade caso o mesmo valor seja utilizado em execuções futuras. A chamada `rr = rand(iseed)` tem exatamente esse papel: inicializar a sequência de números pseudo-aleatórios.

Em seguida, o programa define `m = 1e6`, ou seja, será gerado um milhão de números aleatórios para garantir que a lei dos grandes números leve os valores médios obtidos aos esperados teoricamente. Essa quantidade é impressa na tela usando a instrução `write(*,2) m`. A parte central do programa é o laço `do i = 1,4`, no qual são calculados os momentos $\langle x^n \rangle$ para $n = 1, 2, 3, 4$ por meio da chamada à função `calc(m,i)`. Cada resultado é então escrito na tela no formato `n=... -> ...`.

A função `calc(m,n)` é responsável por realizar o cálculo da média de x^n. Ela inicializa o acumulador `calc = 0` e, em um laço de $i = 1$ até m, soma o valor `rand()**n`, ou seja, o número aleatório elevado à potência n. Ao final, o acumulador é dividido por m (convertido para real em `rm`) e retornado como resultado. Em termos matemáticos, essa função implementa

$$\langle x^n \rangle \approx \frac{1}{m} \sum_{i=1}^{m} (x_i)^n, \tag{3}$$

onde x_i são números pseudo-aleatórios uniformes em $(0, 1)$. Já a função `calc2(m,n)` tem um papel diferente: em vez de calcular a média, ela grava em um arquivo os valores $(x_i)^n$ gerados, um por linha, até um total de m números. O comando `open(unit=1,file=...)` abre o arquivo para escrita, e o laço interno escreve cada valor formatado com quatro casas decimais (`F6.4`). Ao final, o arquivo é fechado com `close(1)`. Portanto, o programa principal realiza duas tarefas:

(i) Calcula e mostra na tela os momentos de ordem 1 a 4 da distribuição uniforme, aproximando os valores teóricos esperados $\langle x^n \rangle = \frac{1}{n+1}$;

(ii) Gera um arquivo de saída contendo m valores de x^n (no caso $n = 1$) para posterior análise.

Figura 1 – Função principal do código.

```
1          program main
2              parameter(iseed=1154)
3
4              rr = rand(iseed)
5
6              m = 1e6
7              write(*,2) m
8  2           format('Para␣m␣=', I8)
9              do i = 1,4
10             write(*,7) i,calc(m,i)
11             end do
12 7           format('n=', I1, '␣->␣', F6.4)
13
14             x=calc2(m,1)
15         end program main
```

Fonte: Compilado pelo Autor.

Figura 2 – Função que realiza cálcula $\langle x^n \rangle$.

```
1  function calc(m,n)
2      calc = 0
3      do i = 1,m
4          calc = calc + (rand()**n)
5      end do
6      rm = m
7      calc =calc/rm
8      return
9  end function calc
```

Fonte: Compilado pelo Autor.

Figura 3 – Função auxiliar que salva em um arquivo de saida o valor de um número aleatório entre 0 e 1 elevado a n.

```
1  function calc2(m,n)
2      open(unit=1,file='saida-1-12694394.txt')
3      do i = 1,m
4          write(1,7) (rand()**n)
5      end do
6      close(1)
7  7           format(F6.4)
8  end function calc2
```

Fonte: Compilado pelo Autor.

RESULTADOS E DISCUÇÃO

Portanto, através dos resultados obtidos, Tabela 1, percebe-se que a média dos valores aleatórios é inversamente proporcional ao valor de, *n*, o que não era esperado pelo estudante. Uma explicação para esse comportamento é o fato de que uma fração

elevado a um número real positivo, maior que 1, sempre vai ser menor ou igual ao seu valor original,

$$(1/x)^m \leq 1/x$$

onde *m* é um número real maior que 1. Desse modo, é intuitivo perceber que o valor dos números aleatórios deve decrescer com o aumento do número no seu expoente e por conseguinte a média desses números também diminui.

Tabela 1 – Valor das médias de números aleatórios entre 0 e 1, elevados a um expoente *n*, variando de 1 a 4. Dados obtidos utilizando o código da Figura 2.

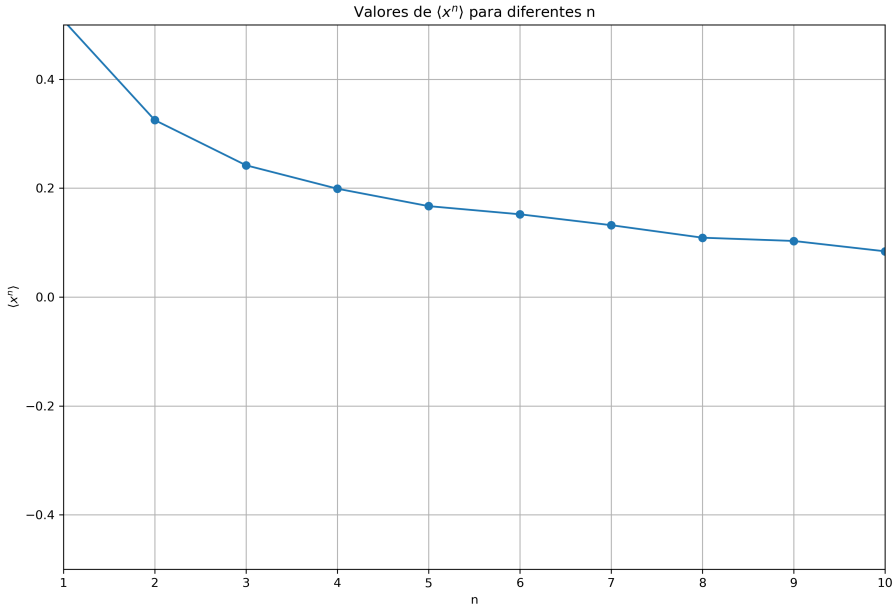
n	$\langle x^n \rangle$
1	0.50
2	0.33
3	0.25
4	0.20

Fonte: Compilado pelo Autor

Figura 4 – Resultado exibido na tela após a execução do código.

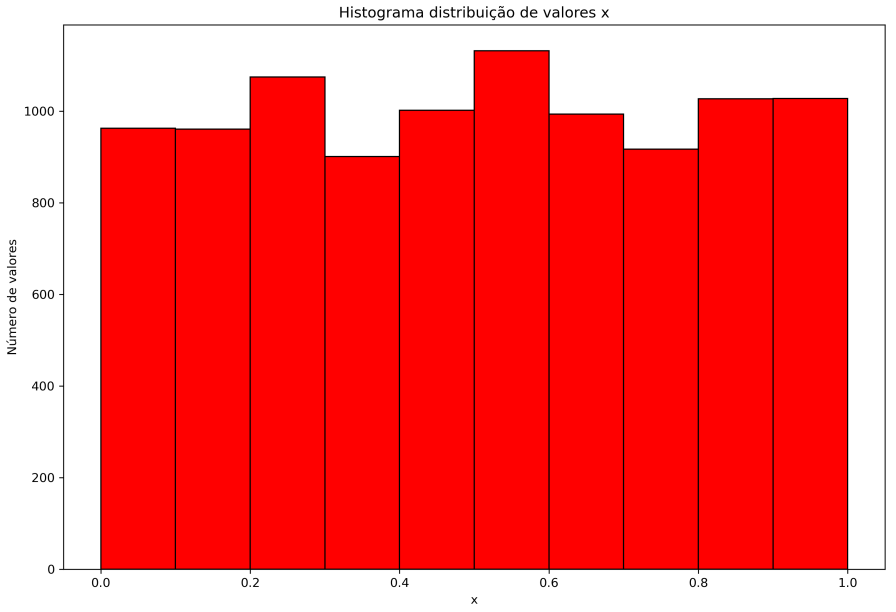
Fonte: Compilado pelo Autor.

Figura 5 – Valores de $\langle x^n \rangle$ para n entre 1 e 10



Fonte: Compilado pelo Autor.

Figura 6 – Histograma que mostra a distribuição de valores para a função **rand()**, variando entre 0 e 1.



Fonte: Compilado pelo Autor.

QUESTÃO 2

ENUNCIADO

2. Vamos considerar agora o problema de andarilhos aleatórios em uma dimensão. Aqui, em cada unidade de tempo, cada caminhante, independentemente onde esteja, dá um passo à direita (esquerda) com probabilidade p ($q = 1 - p$). O caso $p = q = 1/2$ corresponde a um caminhante tão desnorteado que ele não se lembra de onde veio e nem qual o rumo certo a tomar. O caso em que $p \neq q$ corresponde ao viajante aleatório em uma ladeira. A questão nesta tarefa é calcular $\langle x \rangle$ e $\langle x^2 \rangle$ após um certo número N de passos.

(a) Considere $p = q = \frac{1}{2}$ e um número grande M de andarilhos todos partindo da origem ($x = 0$) no tempo inicial ($t = 0$). Após $N = 1000$ passos faça um histograma do número de andarilhos $n(x)$ em função de x. Que tipo de curva você obteve? Calcule $\langle x \rangle$ e $\langle x^2 \rangle$.

(b) Refaça o item anterior considerando $p = \frac{1}{3}, \frac{1}{4}, \frac{1}{5}$. Qual seria a forma analítica em termos de N, p e q para $\langle x \rangle$ e $\langle x^2 \rangle$?

MÉTODOLOGIA

Referencial teórico questão 2

Nos próximos projetos é pedido para calcular a média de passos de um *andarilho aleatório*, ou seja, uma pessoa que não segue um padrão nas suas passadas, nessa simulação o indivíduo tem apenas um grau de liberdade, ou seja, ele só pode andar para direita ou esquerda. Vale notar, que é fixado uma probabilidade do andarilho dar um passo para direita como sendo, *p*, portanto, a probabilidade desse indivíduo dar um passo a esquerda é $q = 1 - p$.

$$p(n_d) = \frac{N!}{n_d! n_e!} p^{n_d} q^{n_e} \tag{4}$$

Desse modo, se a probabilidade de dar um passo para a direita é dada pela Equação 4, onde N é o número total de passos, n_d é o número de passos a direita e n_e é o número de passos a esquerda, a conta para $p(n_e)$ segue a mesma fórmula.

Assim sendo, é possível encontrar a média do números de passos a direita como sendo o número de passos totais multiplicado pela probabilidade do andarilho dar um passos a direita.

$$< n_d >= \sum_{n_d=0}^{N} n_d \cdot p(n_d) \tag{5}$$

Utilizando as Equações 5 e 4, temos que,

$$< n_d >= \sum_{n_d=0}^{N} n_d \frac{N!}{n_d! n_e!} p^{n_d} q^{n_e}$$

utilizando o teorema dos binômios podemos reescrever a equação da seguinte maneira,

$$< n_d >= \sum_{n_d=0}^{N} \binom{N}{n_d} p^{n_d} q^{n_e}$$

o que é equivalente a,

$$F(p,q) = (p+q)^N$$

portanto,

$$< n_d >= p\frac{\partial F(p,q)}{\partial p} \ = pN(p+q)^N$$

se $p+q=1$, então:

$$< n_d >= pN$$

Por fim, para n_e, $< n_e >= qN$. Além disso, para realizar o cálculo da média de passos totais, é necessário subtrair $< n_e >$ e $< n_d >$, $< x >=< n_d > - < n_e >$, tendo em vista que no referencial escolhido a esquerda é considerado como sentido negativo. Ademais, para encontrar a média quadrática de passos $< x^2 >$ é necessário utilizar as mesmas equações, porém, utilizando a segunda derivada parcial da função $F(p,q)$, realizando essas derivações temos que, $< x^2 >= 4Npq$. Assim sendo, encontramos as equações 7 e 6 que nos dão a média de passos do andarilho e a média quadrática respectivamente.

$$< x^2 >= 4Npq \tag{6}$$

$$< x >= N(p-q) \tag{7}$$

CÓDIGO

Função principal do código, nela eu dou o número de bêbados, o número de passos e a probabilidade de dar um passo a direita.

Figura 7 – Função principal do código.

```
1  program main
2
3    ! Eu vou definir as constantes
4    m = 1e4 ! Número de bebados
5    n = 1e2 ! Número de passos
6    p = 0.33 ! Probabilidade de andar para direita
7    x = calc(n,m,p)
8
9  end program main
```

Fonte: Compilado pelo Autor.

A função `calc`, implementada em Fortran77, realiza uma simulação de caminhantes aleatórios em uma dimensão, também conhecida como *random walk*. A ideia do programa é acompanhar o movimento de um grande número de caminhantes independentes, cada um partindo da origem e dando uma sequência de passos para a direita ou para a esquerda, de acordo com uma probabilidade pré-definida.

O funcionamento pode ser descrito da seguinte maneira: inicialmente, o vetor de posições possíveis é zerado, de modo que nenhuma posição final esteja ocupada. Define-se ainda um vetor de passos que associa o valor +1 ao movimento para a direita e -1 ao movimento para a esquerda. Em seguida, a simulação percorre todos os caminhantes. Cada um deles começa na origem e realiza um total de n passos. A cada passo, um número aleatório é gerado e, se for menor que a probabilidade p, o caminhante desloca-se uma unidade para a direita; caso contrário, desloca-se uma unidade para a esquerda.

Ao final do percurso de cada caminhante, a posição resultante é registrada, e duas grandezas estatísticas são atualizadas: a soma das posições finais e a soma dos quadrados dessas posições. Repetindo esse processo para os m caminhantes, obtém-se ao final a média da posição $\langle x \rangle$ e a média do quadrado da posição $\langle x^2 \rangle$, dividindo-se os acumuladores pelo número total de caminhantes. Essas duas medidas são impressas na tela, representando, respectivamente, a posição média e a largura da distribuição dos resultados.

Além disso, a função gera um arquivo de saída que contém o histograma das posições finais, isto é, para cada posição entre $-n$ e n é registrado o número de

caminhantes que terminou naquele ponto. Esse histograma é a base para visualizar a distribuição espacial resultante após a simulação, que, para o caso simétrico $p = 0.5$, tende a assumir a forma de uma curva gaussiana centrada na origem. Para valores diferentes de p, aparece um viés que desloca a média $\langle x \rangle$ para a direita ou para a esquerda.

Porm fim, é pedido para utilizar o código gerado, para calcular a distribuição de andarilhos, com a probabilidade de dar um passo à direita, p, variando de $1/2$, $1/3$, $1/4$ a $1/5$.

Figura 8 – Função calc que realiza a simulação de caminhantes aleatórios em uma dimensão.

```
1
2       function calc(n,m,p)
3           parameter(iseed=1154)
4           dimension ipos(-n:n),istp(0:1)
5
6           ! Da o seed para a func rand()
7           rr = rand(iseed)
8           ! Eu vou iniciar o vetor posição
9           do i = -n,n
10              ipos(i) = 0
11          end do
12          ! Vou definir algumas variaveis
13          istp(0) = 1
14          istp(1) = -1
15
16          rmed = 0
17          rmed2 = 0
18          ! Vou criar o arquivo de saida
19          open(unit=1,file='saida-1-12694394.txt')
20          ! Vou calcular as pos
21          do i = 1,m
22              ix = 0
23              do j = 1,n
24                  if (rand() .lt. p) then
25                      irr = 0
26                  else
27                  irr = 1
28                  end if
29
30                  ix = ix + istp(irr)
31              end do
32              rmed = rmed + ix
33              rmed2 = rmed2 + (ix*ix)
34              ipos(ix) = ipos(ix) + 1
35          end do
36              rm = m
37              rmed = rmed/rm
38              rmed2 = rmed2/rm
39
40          !
41          write(*,*) "<x>,<x^2>"
42          write(*,9) rmed,rmed2
43          !
44          do i = -n,n
45              write(1,7) i, ipos(i)
46          end do
47  7           format(I12,',',I12)
48          close(1)
49          return
50      end function calc
```

Fonte: Compilado pelo Autor.

RESULTADOS E DISCUSSÃO

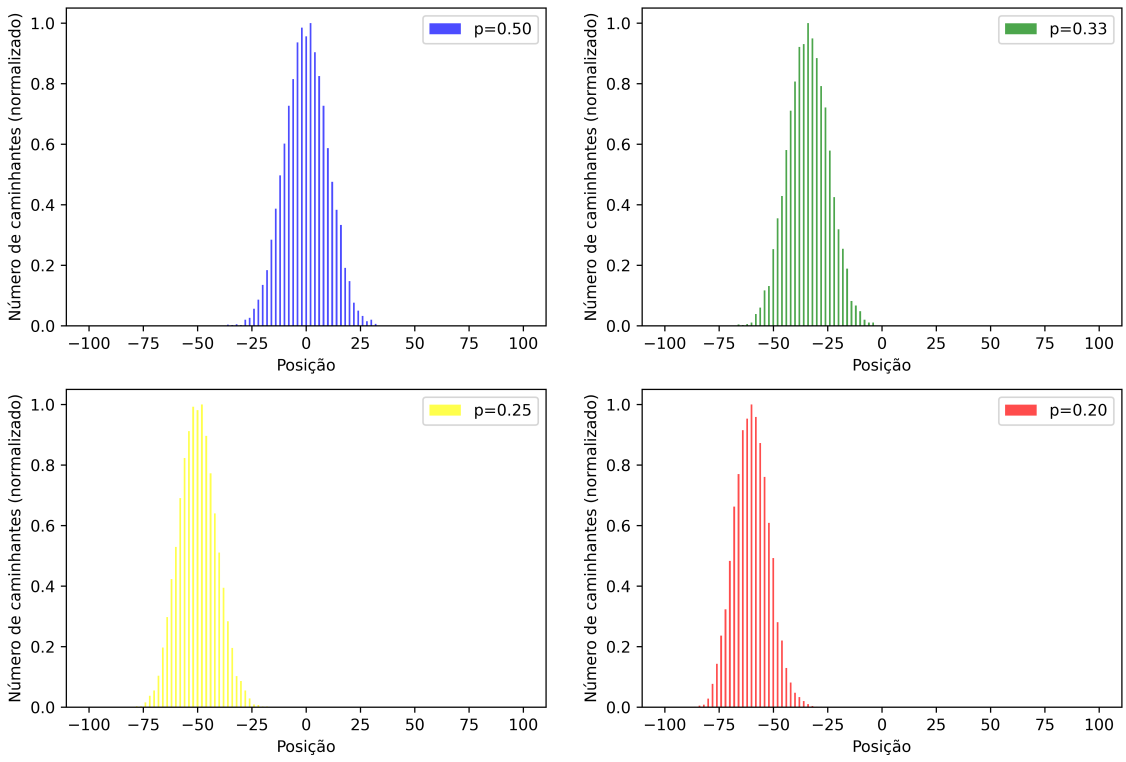
São representados nas figuras 10, 9 e 10 respectivamente a distribuição de andarilhos em diferentes pontos do eixo-x para diferentes valores de *p*, vale notar que, ambas distribuições têm pico no valor $< x >$ dado pela tabela 2.

Tabela 2 – Valores da média $\langle x \rangle$ e $\langle x^2 \rangle$ para diferentes valores *p*, com o número de bêbados igual a $m = 10^4$ e o número de passos $n = 100$.

p	$\langle x \rangle$	$\langle x^2 \rangle$
1/2	0.049	99.303
1/3	-33.953	1241.171
1/4	-49.983	2573.408
1/5	-59.970	3660.482

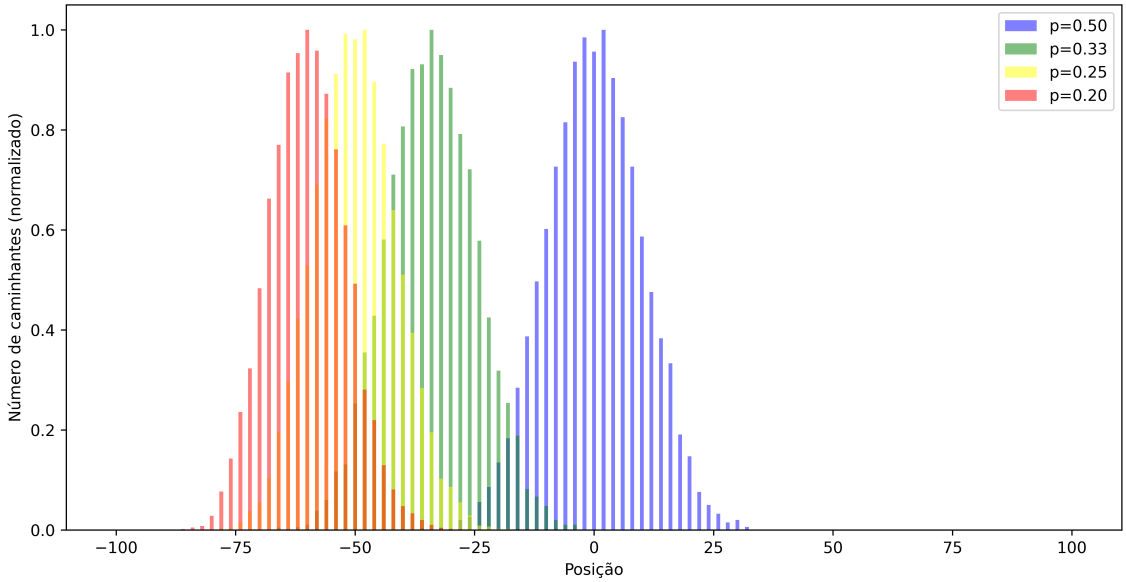
Fonte: Compilado pelo Autor

Figura 9 – Histograma da distribuição de andarilhos em diferentes posições no eixo-x, para $M = 10^4$ e $n = 100$, para diferentes valores de p.



Fonte: Compilado pelo Autor.

Figura 10 – Histograma da distribuição de andarilhos em diferentes posições no eixo-x, para $M = 10^4$ e $n = 100$.



Fonte: Compilado pelo Autor.

Figura 11 – Resultado exibido pela simulação no terminal, onde $m = 10^4$, $n = 100$ e $p = 0.50$.

```
~/r/G/S/IntroFiscomp/Projeto-2/tarefa-2  ./tarefa-2-12694394.exe
<x>,<x^2>
    0.0494     99.3036
~/r/G/S/IntroFiscomp/Projeto-2/tarefa-2
```

Fonte: Compilado pelo Autor.

QUESTÃO 3

ENUNCIADO

3. Considere agora o caso do andarilho bidimensional não enviesado, i.e., com iguais chances $\left(\frac{1}{4}\right)$ ele dá um passo em qualquer direção dos pontos cardeais: norte, sul, leste e oeste. Calcule $\langle\vec{r}\rangle$ e $\Delta^2 = \langle\vec{r}\cdot\vec{r}\rangle - \langle\vec{r}\rangle\cdot\langle\vec{r}\rangle$. Repare que estes andarilhos perfazem o mesmo tipo de movimento que moléculas no processo de difusão, como por exemplo a difusão de um pingo de leite numa xícara de café. Faça um diagrama das posições das moléculas após um número N de passos $(N = 10, 10^2, 10^3, 10^4, 10^5, 10^6)$.

METODOLOGIA

Nesse problema, é pedido para generalizar o caso do andarilho aleatório das simulações **2A** e **2B** para o plano 2D, portanto, o código criado é similar ao da questão **2**. Vale notar que, ao contrário do problema anterior, nesse problema, o andarilho tem igual chance de caminhar na direção norte, sul, leste e oeste, ou seja, a probabilidade dele dar um passo em cada uma dessas direções é 1/4.

CÓDIGO

O programa `main` é responsável por chamar a função `calc(m,n)` com os parâmetros $m = 10^2$ (número de caminhantes) e $n = 10^3$ (número de passos de cada caminhante). A função `calc` implementa o movimento de andarilhos bidimensionais (caminhantes aleatórios), que dão passos em direções norte, sul, leste ou oeste com probabilidades iguais ($\frac{1}{4}$ cada). O objetivo é calcular as quantidades $\langle\vec{r}\rangle$, $\langle\vec{r}\cdot\vec{r}\rangle$ e $\langle(\Delta r)^2\rangle = \langle\vec{r}\cdot\vec{r}\rangle - \langle\vec{r}\rangle\cdot\langle\vec{r}\rangle$, bem como salvar a distribuição final das posições em um arquivo externo.

Logo no início da função, define-se `iseed=12`, a semente do gerador pseudo-aleatório. A chamada `rr = rand(iseed)` inicializa a sequência de números aleatórios. Em seguida, é criado o vetor `istep(1:2)` com os possíveis incrementos de posição: $+1$ e -1. O array bidimensional `ipos(-n:n,-n:n)` é inicializado com zeros, sendo ele responsável por registrar a frequência de caminhantes que terminaram em cada posição (i, j) do reticulado bidimensional.

A etapa principal do cálculo ocorre em dois laços aninhados: (i) para cada um dos m caminhantes, inicializa-se a posição em $(0, 0)$; (ii) em cada um dos n passos, sorteia-se uma direção usando o par de variáveis `idir` e `irand`. O valor `idir = 2e0*rand()` (truncado para 0 ou 1) decide se o passo ocorrerá no eixo x ou no eixo y, enquanto

`irand = 2e0*rand() + 1` escolhe o sentido (índice para `istep`), resultando em $+1$ ou -1. Ao final dos n passos, o contador `ipos(ix,iy)` é incrementado, armazenando quantos caminhantes terminaram na posição (ix, iy).

Com a distribuição final das posições construída, o código calcula primeiro o vetor médio $\langle\vec{r}\rangle$. Isso é feito somando-se todas as posições finais (i, j) ocupadas (pesadas pela frequência em `ipos`) e dividindo pelo número de caminhantes m. Em seguida, calcula-se $\langle\vec{r}\rangle \cdot \langle\vec{r}\rangle = r_1$.

Depois, a função calcula $\langle\vec{r} \cdot \vec{r}\rangle = r_2$, obtido pela soma dos quadrados das coordenadas $(i^2 + j^2)$ de todas as posições finais ocupadas, também dividido por m. A diferença entre r_2 e r_1 fornece o desvio quadrático médio:

$$\langle(\Delta r)^2\rangle = \langle\vec{r} \cdot \vec{r}\rangle - \langle\vec{r}\rangle \cdot \langle\vec{r}\rangle.$$

Por fim, os resultados são gravados em um arquivo `saida-1-12694394.txt`, onde cada linha contém as coordenadas finais (i, j) e a frequência `ipos(i,j)` de caminhantes que terminaram naquela posição. O formato usado é `I4,',',I4,',',I4`, permitindo futura análise gráfica ou estatística da distribuição espacial.

Em resumo, esse programa realiza a simulação de um conjunto de caminhantes aleatórios em duas dimensões, calcula grandezas médias associadas ao processo de difusão ($\langle\vec{r}\rangle$, $\langle r^2\rangle$, $\langle(\Delta r)^2\rangle$) e armazena a distribuição final das posições em arquivo para inspeção posterior.

Figura 12 – Função principal do código.

```
1 program main
2     n = 3000
3     m = 1000
4     x = calc(m,n)
5 end program main
```

Fonte: Compilado pelo Autor.

Figura 13 – Função que realiza os cálculos.

```
1
2  function calc(m,n)
3  parameter(iseed=12)
4  dimension istep(1:2), ipos(-n:n,-n:n)
5
6  ! Da o seed para o rand()
7  rr = rand(iseed)
8
9  ! Inicia o vetor istep
10 istep(1) = 1
11 istep(2) = -1
12
13 ! Inicia o vetor posição
14 do i =-n,n
15     do j = -n,n
16         ipos(i,j) = 0
17     end do
18 end do
19
20 ! Cálculos
21 ixm = 0
22 iym = 0
23 do i =1,m
24     ix = 0
25     iy = 0
26     do j = 1,n
27         ! Escolho se vou na direção x ou y
28         idir = 2e0*rand()
29         irand = 2e0*rand() + 1
30         if (idir .EQ. 0) then
31             ix = ix + istep(irand)
32         else
33             iy = iy + istep(irand)
34         end if
35     end do
36     ipos(ix,iy) = ipos(ix,iy) + 1
37 end do
```

Fonte: Compilado pelo Autor.

Figura 14 – continuação da função que realiza os cálculos.

```
1
2           ! Cálcula o valor médio da posição <r>
3           r1x = 0
4           r1y = 0
5           do i =-n,n
6               do j =-n,n
7                   if (ipos(i,j) .NE. 0) then
8                       r1x = r1x + i
9                       r1y = r1y + j
10                  end if
11              end do
12          end do
13          r1x = r1x/m
14          r1y = r1y/m
15          rr1 = r1x + r1y
16          r1 = (r1x*r1x) + (r1y*r1y)
17
18          ! Cálcula o valor médio de <r.r>
19          r2=0
20          do i =-n,n
21              do j =-n,n
22                  if (ipos(i,j) .NE. 0) then
23                      r2 = r2 + (i*i) + (j*j)
24                  end if
25              end do
26          end do
27          r2 = r2/m
28
29          ! Calcula o valor de <(delta_r)^2> = <r.r> - <r>.<r>
30          dr2 = r2 - r1
31
32          ! Escreve na tela <(delta_r)^2> e <r>
33          write(*,*)"␣<r>,␣<(delta_r)^2>"
34          write(*,11) rr1,dr2
35          ! Salva os resultados
36          open(unit=1,file='saida-1-12694394.txt')
37          do i =-n,n
38              do j = -n,n
39              if (ipos(i,j) .GT. 0) then
40                  write(1,7) i,j,ipos(i,j)
41              end if
42              end do
43          end do
44  7       format(I4,',',I4,',',I4)
45  11      format(F12.4, F12.4)
46          close(1)
47          end function calc
```

Fonte: Compilado pelo Autor.

RESULTADOS E DISCUSSÃO

Nesta seção, serão exibidos os dados resultados obtidos através da simulação da tarefa 3, nas Figuras - 15 e 16 são contemplados as posições de 1000 caminhantes aleatórios dando 3000 passos. Desses gráficos, fica evidente que a tendencia é que após 3000 passos que podem ser para qualquer direção do eixo cardial, e sem preferencia para uma direção, o andarilho tende a ficar na sua posição inicial, no nosso caso (0,0). Ademais, na Tabela - 3 são apresentados os valores da média $\langle r \rangle$ e Δ^2 para diferentes números de passos, *n*, nela é visível que quanto maior o número de passos, maior é o valor do desvio padrão e o valor de $\langle r \rangle$ tende a 0.

Tabela 3 – Valores da média $\langle r \rangle$ e Δ^2 para diferentes números de passos, *n*, com o número de andarilhos fixo igual a $m = 1000$

n	$\langle r \rangle$	Δ^2
1000	-1.5580	930.8069
2000	-2.6220	1931.1337
3000	-1.5500	2891.0312
4000	-0.3080	3876.8213
5000	-0.0760	4862.9712

Fonte: Compilado pelo Autor

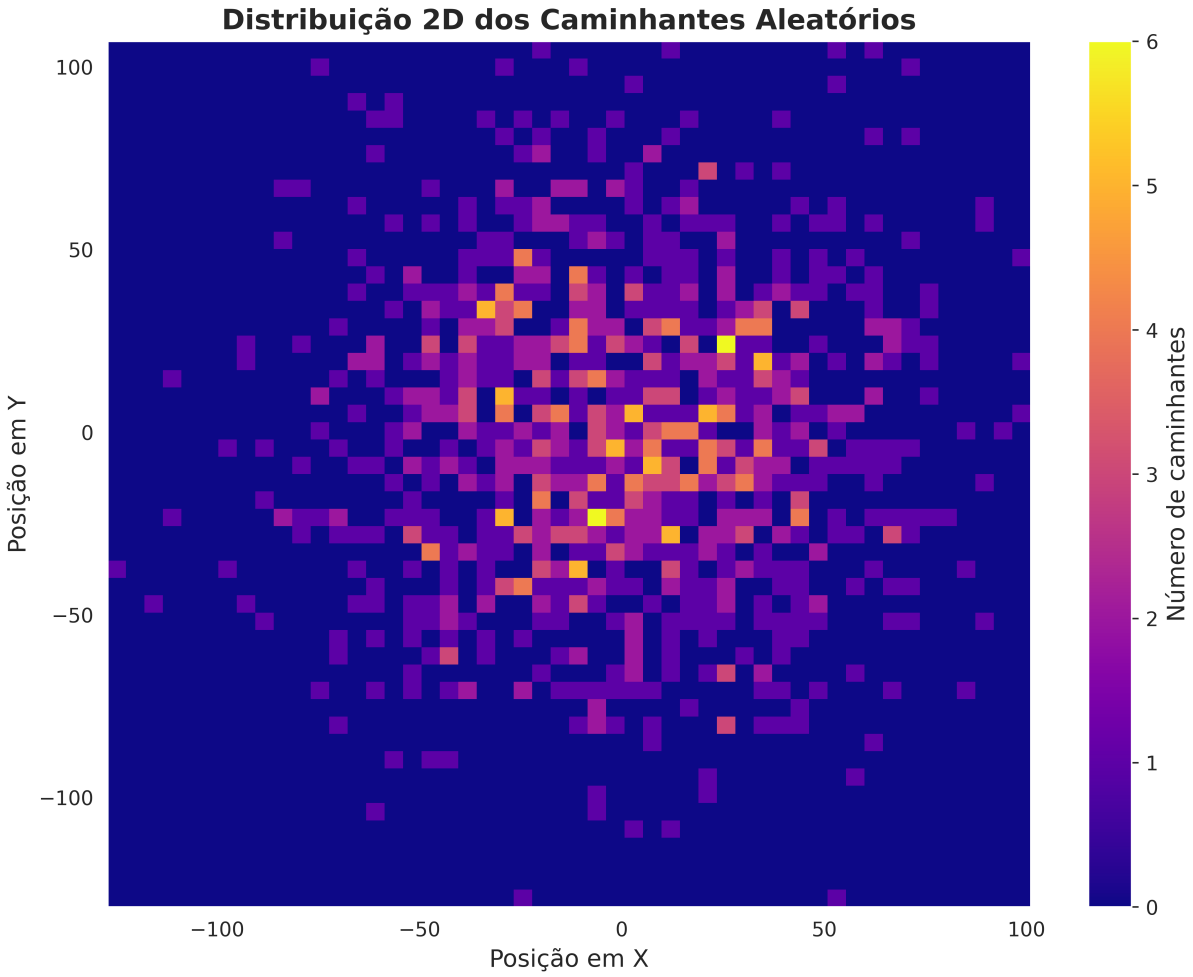


Figura 15 – Histograma 2D da posição dos caminhantes aleatórios no eixo xy.

Fonte: Compilado pelo Autor.

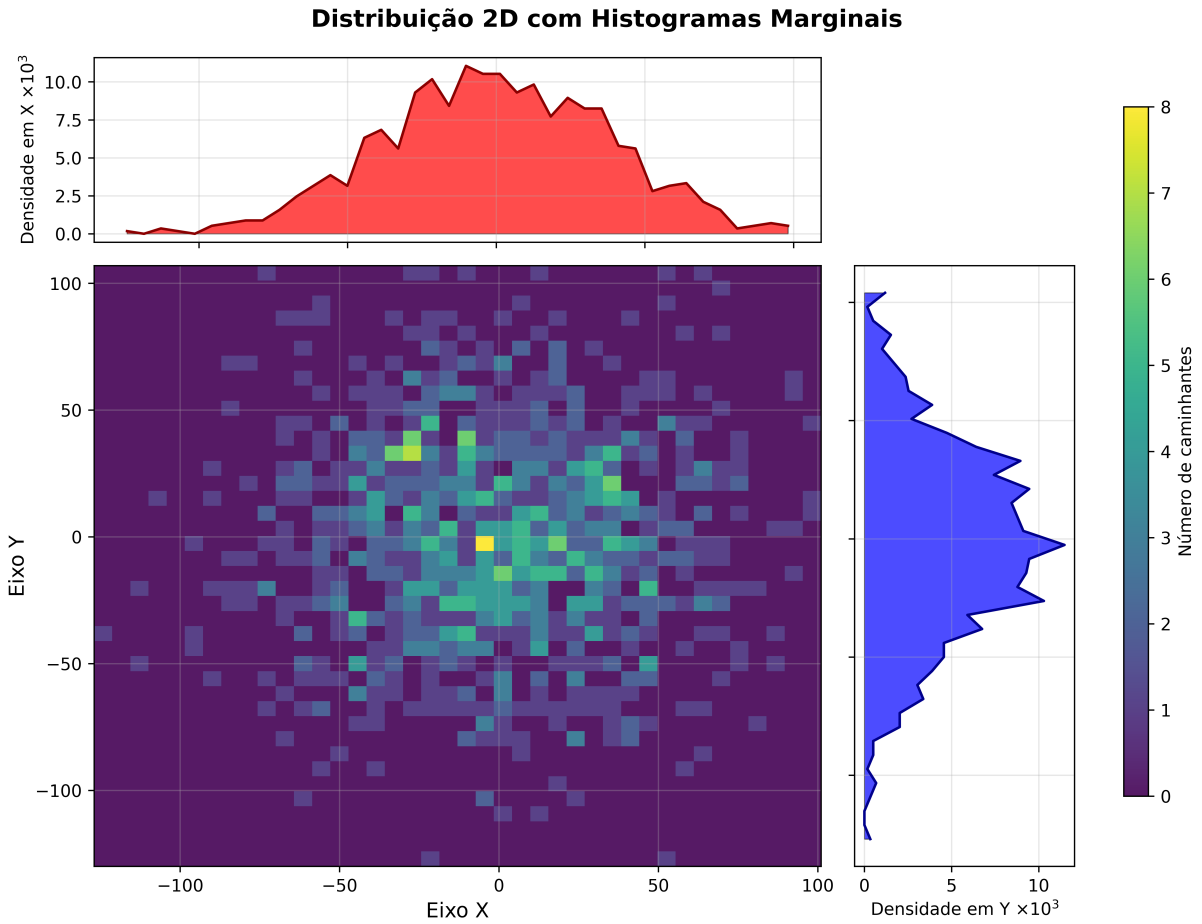


Figura 16 – Histograma 2D da posição dos caminhantes aleatórios no eixo xy, nas margens são histogramas da distribuição de caminhates aleatórios em cada eixo.

Fonte: Compilado pelo Autor.

Figura 17 – Resultado exibido pela simulação no terminal, onde os valores forem de $m = 1000$ e $n = 5000$.

```
~/r/G/S/IntroFiscomp/Projeto-2/tarefa-3 ./tarefa-3-12694394.exe
 <r>, <(delta_r)^2>
    -0.0760    4862.9712
~/r/G/S/IntroFiscomp/Projeto-2/tarefa-3
```

Fonte: Compilado pelo Autor.

QUESTÃO 4

ENUNCIADO

4. Vamos verificar o aumento da entropia e a flecha do tempo no exercício anterior. Calcule a entropia como função do número de passos N das moléculas (que é proporcional ao tempo $t = N\Delta t$, onde Δt é o intervalo de tempo médio entre passos). A entropia é dada por

$$S = -\sum_i P_i \ln P_i, \tag{8}$$

onde P_i é a probabilidade de se encontrar o sistema em um certo micro-estado i. Para se definir o micro-estado i, definimos um reticulado (muito maior que o tamanho de um passo) e verificamos quantas moléculas encontramos em cada célula do reticulado.

METODOLOGIA

Nessa simulação é pedido para calcular o valor da entropia de um sistema com *M* moléculas, realizando *N* passos.

Tendo em vista que a entropia do sistema cresce em função do tempo, este que é proporcional ao número de passos, $t = N(t - t_0)$, é possível simular o aumento de entropia utilizando um aumento no número de passos dados pelas moléculas.

$$S = -\sum_i P_i \ln P_i \tag{9}$$

Tendo em vista que a entropia de um sistema é dada pela Equação 9, onde P_i é a probabilidade de encontrar o sistema em um certo micro-estado, tal que o micro-estado é definido como sendo um reticulado de tamanho fixo no qual é medido o número de moléculas que lá estão.

$$P_i = \frac{K_i}{M} \tag{10}$$

Desse modo, através da Equação 10, onde K_i é o número de partículas no reticulado e M é o número total de moléculas, é possível encontrar a probabilidade P_i daquele micro-estado ocorrer.

CÓDIGO

A estrutura principal do código abre um arquivo de saída, percorre diferentes valores de N (número de passos) e, para cada caso, chama a função `calc`, que realiza os cálculos necessários para estimar a entropia do sistema.

A função `calc` é o núcleo do programa. Inicialmente, é definido o tamanho do reticulado em função de N, bem como os micro-estados disponíveis. Em seguida, o reticulado é zerado, e os possíveis passos são definidos como deslocamentos de ± 1 em cada direção.

A simulação é então realizada para m moléculas, que partem da origem e percorrem N passos aleatórios. A cada passo, a direção é escolhida de forma estocástica, com igual probabilidade de caminhar no eixo x ou no eixo y, e o reticulado registra a posição final das partículas.

Após a simulação, a função procede ao cálculo da entropia. O espaço é dividido em regiões do reticulado (as células), e conta-se o número de partículas em cada uma. A probabilidade P_i de encontrar uma partícula em um micro-estado i é obtida pela razão entre a ocupação da célula e o número total de partículas, normalizado pelo fator de discretização.

Com esses valores, a função aplica a fórmula

$$S = -\sum_i P_i \ln P_i,$$

acumulando a contribuição de cada célula do reticulado.

O valor de entropia calculado é então retornado para o programa principal, que o registra no arquivo de saída junto com o respectivo número de passos.

Figura 18 – Função principal do código.

```
1 program main
2         m = 1000
3         open(unit=1,file='saida-1-12694394.txt')
4         do i = 100,1000
5                 write(1,7) i,calc(m,i)
6         end do
7 7               format(I12,',',F12.4)
8         close(1)
9 end program main
```

Fonte: Compilado pelo Autor.

Figura 19 – Função que realiza os cálculos.

```
function calc(m,n)
        parameter(iseed=1154)
        dimension ipos(-n:n,-n:n),istep(0:1)

        ! Número de micro estados
        irazao = n*0.1
        ! Tamanho do reticulado
        ksize = n/irazao

        ! Da o seed para o rand()
        rr = rand(iseed)
        ! Inicia o vetor posição
        do i =-n,n
                do j = -n,n
                        ipos(i,j) = 0
                end do
        end do

        ! Valores dos passos
        istep(0) = 1
        istep(1) = -1
```

Fonte: Compilado pelo Autor.

FUNÇÃO CALC

A função calc(m,n) tem por objetivo simular m trajetórias (ou “moléculas”) realizando n passos cada, registrar a posição final de cada trajetória em um reticulado discreto e, a partir da distribuição espacial resultante, calcular uma aproximação da entropia de Shannon das probabilidades de ocupação das células macroscópicas do reticulado. O programa principal chama calc(m,i) para valores crescentes de i (de 100 a 1000) e grava o par (número de passos, entropia) em arquivo, permitindo estudar S como função de N.

Logo no início da função aparecem definições importantes: parameter (iseed=1154) e dimension ipos(-n:n,-n:n),istep(0:1). O parâmetro iseed é usado para inicializar o gerador de números pseudo-aleatórios, garantindo reprodutibilidade quando o mesmo iseed for usado sempre; a chamada rr = rand(iseed) efetua essa semente (dependendo da implementação de rand do compilador). O array ipos é um contador bidimensional de posições, com índices que vão de $-n$ até n em cada dimensão — portanto o reticulado fino considerado tem $(2n+1) \times (2n+1)$ células e pode armazenar posições negativas (o centro da origem é (0,0)). istep(0:1) é reservado para os incrementos ± 1 que serão aplicados aos deslocamentos em cada passo.

A variável irazao é calculada como n*0.1 (portanto, em termos inteiros, corresponde a 10% de n) e, segundo o comentário, representa o “número de micro-estados”.

Figura 20 – Função que realiza os cálculos.

```
1
2         ! Realiza os cálculos
3         do i = 1,m
4                 ix = 0
5                 iy = 0
6                 do j = 1,n
7                 idir = 2e0*rand()
8                 irand = 2e0*rand()
9
10                if (idir .EQ. 0) then
11                        ix = ix + istep(irand)
12                else
13                        iy = iy + istep(irand)
14                endif
15                end do
16                ipos(ix,iy) = ipos(ix,iy) + 1
17        end do
18
19        ! Calculo reticulado
20        calc = 0
21        n1 = -n
22        n2 = n1 + ksize
23        do k =1,2*irazao
24                rpro= 0e0
25
26                do i = n1,n2
27                        do j = n1,n2
28                                rpro = rpro + ipos(i,j)
29                        end do
30                end do
31
32                rpro = rpro/(irazao)
33
34                if (rpro .NE. 0e0) then
35                        calc = calc - rpro*log(rpro)
36
37                end if
38                ! Muda o valor dos indices
39                n1 = n2
40                n2 = n2 + ksize
41        end do
42        return
43 end function calc
```

Fonte: Compilado pelo Autor.

Em seguida ksize = n/irazao define o tamanho (em células finas) de cada bloco macroscópico: cada célula macroscópica conterá ksize $\times$ ksize células do reticulado fino. Em outras palavras, a malha fina (-n:n, -n:n) será agrupada em $2 \cdot irazao$ blocos ao longo de cada dimensão (total de $(2 \cdot irazao)^2$ blocos macroscópicos).

A função inicializa ipos com zeros usando dois laços aninhados do tipo do j = -n,n; do i = -n,n. Isso garante que antes da simulação não haja contagens residuais e

que ipos(i,j) passe a ser o acumulador correto das frequências absolutas de trajetórias que terminaram na posição (i, j).

Figura 21 – Função principal do código.

```
do i = 1,m
    ix = 0
    iy = 0
    do j = 1,n
        idir = 2e0*rand()
        irand = 2e0*rand()
        if (idir .EQ. 0) then
            ix = ix + istep(irand)
        else
            iy = iy + istep(irand)
        endif
    end do
    ipos(ix,iy) = ipos(ix,iy) + 1
end do
```

Fonte: Compilado pelo Autor.

Aqui cada trajetória começa em (ix,iy) = (0,0) e executa n passos. As variáveis idir e irand são usadas para decidir, em cada passo, (i) em qual eixo o passo ocorrerá (x ou y) e (ii) o sinal do passo (+1 ou -1). Ao atribuir 2.0*rand() (um real entre 0 e 2) a idir e irand, a parte fracionária é truncada e o valor fica em [0,1] com (aproximadamente) 50% de chance para cada. Assim idir=0 seleciona um deslocamento em x, caso contrário em y; irand serve de índice para istep(irand) onde istep(0)=-1 e istep(1)=1, produzindo passos de unidade para a direita/para cima ou para a esquerda/para baixo. Ao final dos n passos a posição final (ix,iy) é incrementada no contador ipos(ix,iy).

Depois de simular m trajetórias, ipos(i,j) contém a frequência absoluta (número de trajetórias) que terminaram em cada posição do reticulado fino. Para obter a entropia o código soma as contagens ipos em blocos macroscópicos e aplica a fórmula de Shannon. O trecho relevante é:

Para cada bloco (variando k), rpro acumula o número total de trajetórias que caíram nas células finais pertencentes àquela célula macroscópica. Isso feito, é aplicado a Equação - 9, com o rpro = rpro/(irazao), que nos da a probabilidade de uma partícula estar no retículado.

Figura 22 – Função principal do código.

```
calc = 0
n1 = -n
n2 = n1 + ksize
do k = 1, 2*irazao
    rpro = 0e0
    do i = n1,n2
        do j = n1,n2
            rpro = rpro + ipos(i,j)
        end do
    end do
    rpro = rpro/(irazao)
    if (rpro .NE. 0e0) then
        calc = calc - rpro*log(rpro)
    end if
    n1 = n2
    n2 = n2 + ksize
end do
```

Fonte: Compilado pelo Autor.

RESULTADOS E DISCUÇÃO

Na Figura - 23, é apresentado o gráfico gerado pelo arquivo de saida, é evidente que o acrescésimo no número de passos faz com que a entropia do sistema aumente.

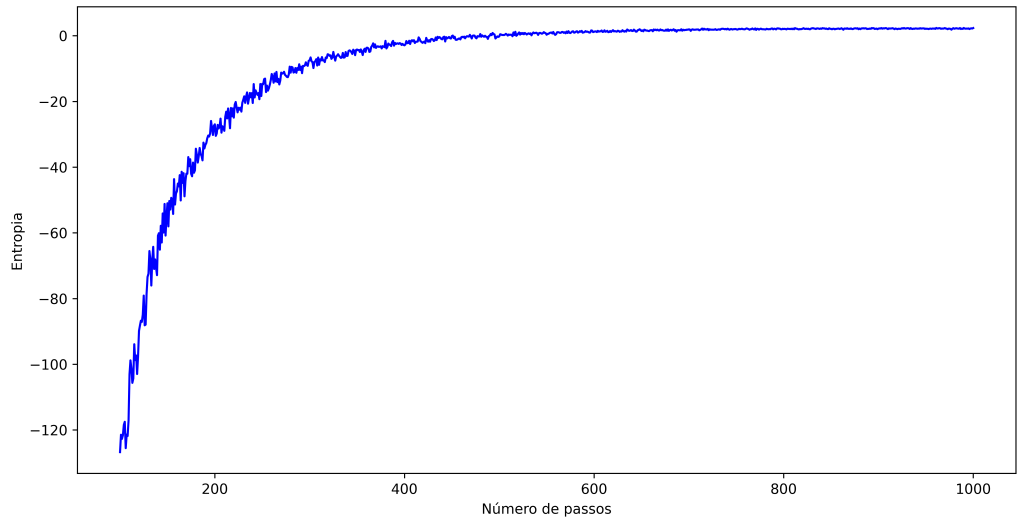


Figura 23 – Entropia do sistema vs número de passos

Fonte: Compilado pelo Autor.