UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Introdução à Física Computacional - Projeto 2

Silvio Lacerda de Almeida

Setembro, 2023
São Carlos, SP

Sumário

Lista de Figuras

1 Introdução

O segundo projeto de Introdução à Física Computacional contempla 4 tarefas, e delas explicarei detalhadamente o pensamento que originou o código, assim como discutirei o resultado de alguns deles.

Todos os códigos foram escritos em Fortran 77, aderindo à sintaxe implícita, ou seja, quaisquer variáveis declaradas tendo inicialmente letras no intervalo [i-n] são inteiras, e o resto são reais. A declaração de variáveis entretanto se fez obrigatória quando os algoritmos estudavam reais com precisão dupla, ou quando foi necessário criar uma função.

2 Tarefas

2.1 Tarefa 1

Figura 1: Tarefa 1

```
1           program ValorEsperado
2           n = 1000000
3           aMedia = 0.0e0
4           do i=1, n
5             aMedia = aMedia + rand()
6           end do
7           aMedia = aMedia/real(n)
8
9           write(*,*) "Média de ", n, " números aleatórios: "
10          write(*,*) aMedia
11          end program
```

2.1.1 Explicando a tarefa 1

Inicialmente defino uma variável n que contenha a quantidade de números aleatórios que será gerada, e logo em seguida faço um loop acrescentando na variável aMedia todos os valores aleatórios a partir da função rand, que gera números aleatórios entre 0 e 1, para por fim conseguir a média de fato, fazendo aMedia / n e obtendo assim uma média de todos os valores aleatórios que foram colocados.

2.1.2 Exemplo de uso

Figura 2: Saída no terminal - Tarefa 1

```
1  silvio$ ./tarefa-1-13783203.exe
2   Média de      1000000  números aleatórios:
3    0.500028431
```

No exemplo podemos notar que com 1000000 números aleatórios, o valor esperado é 0.5, que é a média entre 0 e 1. No caso da função *rand*, suas cotas inferior e superior são respectivamente

0 e 1. Quanto maior a quantidade de números aleatórios mais próximo o valor esperado estará de 0.5, e no caso do nosso exemplo ele não é exatamente 0.5, pois a partir quinta casa decimal os valores já são maiores que 0.

2.2 Tarefa 2

Figura 3: Tarefa 2

```
1           program Andarilho
2           integer i_path
3           parameter (n_passos=1000)
4           dimension vec_hist(-n_passos:n_passos)
5
6           vec_hist = 0.0e0
7           aMedia = 0.0e0
8           aMediaQ = 0.0e0
9
10          write(*,*) "Digite a quantidade de andarilhos"
11          read(*,*) m
12
13          do j=1, m
14            i_x = i_path(n_passos)
15            aMedia = aMedia + real(i_x)
16            aMediaQ = aMediaQ + (real(i_x)**2)
17
18            vec_hist(i_x) = vec_hist(i_x) + 1
19          end do
20
21
22          open(unit=1, file="saida-2-13783203.dat", status="new")
23          do i=-n_passos, n_passos
24            write(1,*) i, vec_hist(i)
25          end do
26          close(unit=1)
27
28          aMedia = aMedia/real(m)
29          aMediaQ = aMediaQ/real(m)
30
31          write(*,*) "Valor esperado de <x>: ", aMedia
32          write(*,*) "Valor espeerado de <x^2>: ", aMediaQ
33
34          end program
35
36          function i_path(n)
37          logical goFoward
38
39          p_foward = 1.e0/2.e0 ! essa parte é alterada conforme a necessidade
40          goFoward = .FALSE.
41          i_path = 0
42
43          do i=1, n
44            goFoward = rand() .LE. p_foward
45
46            if (goFoward .EQV. .FALSE.) then
47              i_path = i_path - 1
48            else
49              i_path = i_path + 1
50            end if
51          end do
52          return
53          end function
```

2.2.1 Explicando a tarefa 2

Inicialmente defino a função *i_path* que está no final do código, mas como ela será utilizada na metade do código, ela será explicada antes. A função *i_path* começa com a declaração da variável *goFoward* que será responsável por determinar se o andarilho irá para frente ou não, se for **.TRUE.** então o andarilho vai para frente. A função também recebe um parâmetro n que é a quantidade de passos que o andarilho vai dar. Também inicializo o valor de *i_path* para poder fazer o somatório do mesmo ao longo da função, e defino p_foward que é a probabilidade do andarilho ir para frente.

Logo em seguida faço um laço que verifica se um número aleatório a partir da função *rand*, e se este número form menor ou igual à probabilidade de ir para frente, então *goFoward* se torna **.TRUE.**. Depois disso apenas verifico se devo ir para frente ou não, assumindo que ir para frente significa somar 1, e ir para trás significa diminuir 1. Por fim, retorno o valor de *i_path*.

O programa principal começa definindo o parâmetro *n_passos* que será a quantidade de passos dada por cada andarilho. E logo em seguida defino a variável *vec_hist* que será um array que vai conter os dados de quanto cada andarilho andou para no fim podermos colocar em um gráfico um histograma, mostrando assim uma curva que mostra exatamente um comportamento que se aproxima do valor esperado de um anadarilho aleatório parar após andar os *n_passos*.

Defino os valores iniciais da média normal e da média ao quadrado, assim como o valor de todos os dados do histograma, e peço para o usuário retornar o valor m que representa a quantidade de andarilhos que serão utilizados.

Faço um laço aonde a posição final de cada andarilho *i_x* será o valor de *i_path*, e começo a calcular o valor da média e da média quadrada para mostrar os valores esperados ao final do código.

Logo em seguida abro o arquivo *saida-2-13783203.dat* e escrevo a posição final de cada andarilho para posteriormente fazer o gráfico. E por fim mostro $< x >$ e $< x^2 >$.

2.2.2 Exemplo de uso

Figura 4: Saída no terminal - Tarefa 2

```
1  silvio$ ./tarefa-2-13783203.exe
2   Digite a quantidade de andarilhos
3  10000
4   Valor esperado de <x>:    9.16000009E-02
5   Valor espeerado de <x^2>:    999.865601
```

Neste exemplo de uso, podemos ver que o valor esperado de x se aproxima de 0, sendo ele 9.16000009E-02, e o valor esperado de x^2 999.865601, e isso faz muito sentido, pois analiticamente:

$$< x^2 >= 4Npq + N^2 - 4N^2pq \tag{1}$$

$$< x >^2= N^2 - 4N^2pq \tag{2}$$

Sendo N a quantidade de passos, p a probabilidade de ir para frente, e q a probabilidade de ir para trás. Sendo assim temos que para $p = 1/2$, e como $q = 1 - p = 1/2$, podemos estimar seus respectivos valores como sendo:

$$< x^2 >= 4.1000.\tfrac{1}{2}.\tfrac{1}{2} + 1000^2 - 4.1000^2.\tfrac{1}{2}.\tfrac{1}{2} = 1000$$

$$< x >= \sqrt{1000^2 - 4.(1000^2)\tfrac{1}{2}.\tfrac{1}{2}} = 0$$

o $< x >$ de cada andarilho é $< x >= (+1).(1/2) + (-1).(1/2) = 0$, enquanto o $< x^2 >= (+1)^2.(1/2) + (-1)^2.(1/2) = 1$, então o $< x^2 >$ para todos os andarilhos, sabendo que foram 1000 passos, deve ser $< x^2 >= 1000$, que também é muito próximo do valor obtido a partir de valores aleatórios.

É possível também fazer um histograma podendo facilitar assim a visualização de quantos andarilhos tem ao longo da reta x.

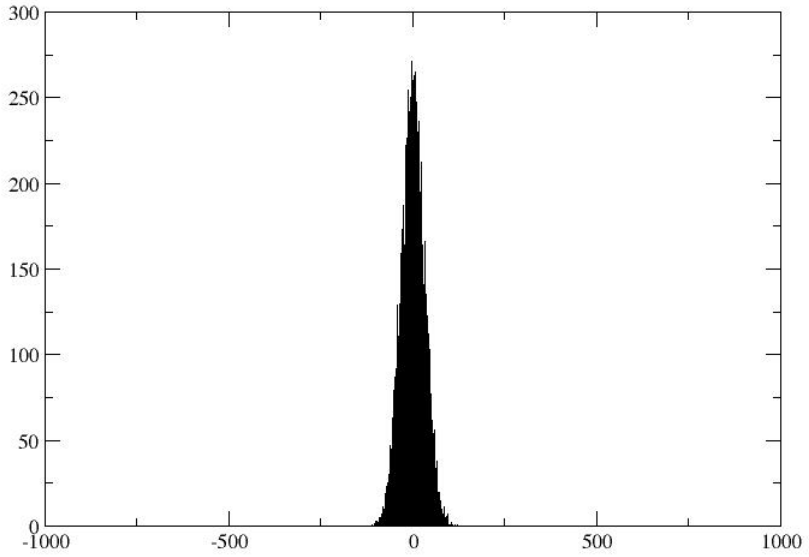

Figura 5: Número de andarilhos x Posição. Probabilidade de ir para frente 1/2

Podemos repetir o código e analisar o comportamento, comparando-o com os mesmos métodos analíticos aplicados no exemplo anterior, mas agora alterando algumas coisas.

Podemos escolher outras probabilidades para o andarilho ir para frente, ao invés de fixar em 1/2, podemos escolher 1/3, 1/4 e 1/5, e o interessante é que para todos eles, as equações 1 e 2 valem e se mostram verdadeiras.

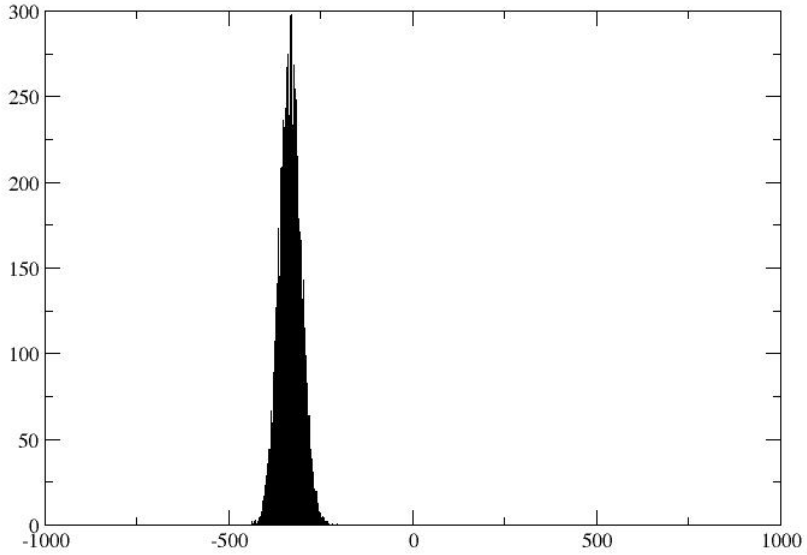

Figura 6: Número de andarilhos x Posição. Probabilidade de ir para frente 1/3. $< x >$ = -333.355194 e $< x^2 >$ = 112014.898.

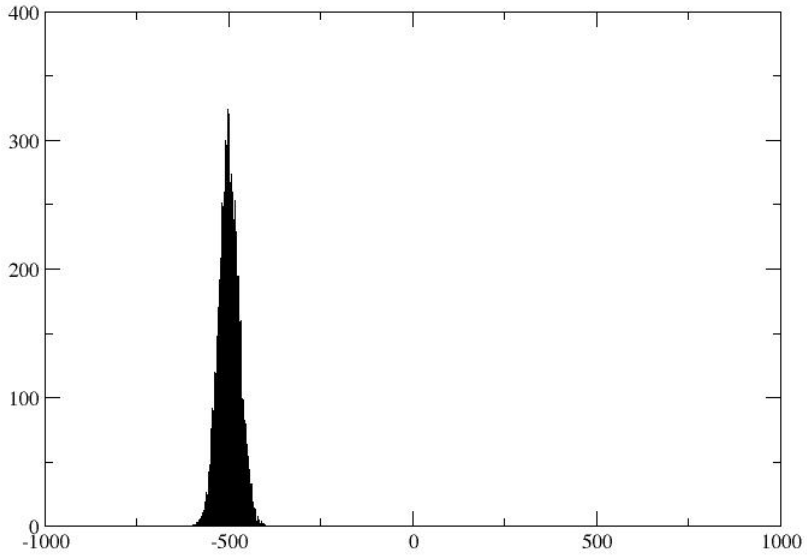

Figura 7: Número de andarilhos x Posição. Probabilidade de ir para frente 1/4. $< x >$ = -499.930389 e $< x^2 >$ = 250687.188.

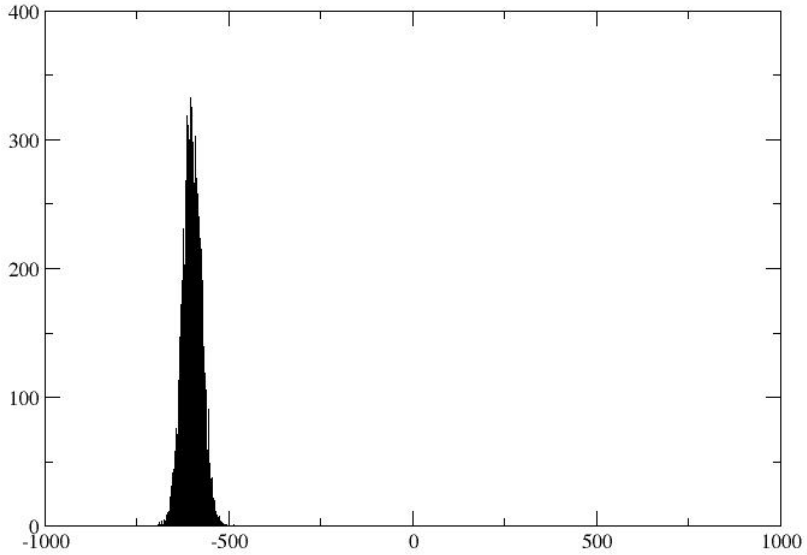

Figura 8: Número de andarilhos x Posição. Probabilidade de ir para frente 1/5. $< x >$ = -599.992798 e $< x^2 >$ = 360641.688.

2.3 Tarefa 3

Figura 9: Tarefa 3

```
1           program Andarilh2D
2           parameter (n_andarilhos=1000)
3           parameter (n_passos=1000000)
4           dimension vec_x(n_andarilhos)
5           dimension vec_y(n_andarilhos)
6           dimension i_r(2)
7
8           vec_x = 0
9           vec_y = 0
10
11          p_walk = 1.e0/4.e0
12
13          aMedia = 0.0e0
14          aMediaQ = 0.0e0
15
16          do j=1, n_andarilhos
17            i_r = 0
18
19            do i=1, n_passos
20              prob = rand()
21              if (prob .LE. p_walk) then
22                i_r(1) = i_r(1) - 1
23              else if (prob .GT. p_walk .AND. prob .LE. p_walk*2) then
24                i_r(1) = i_r(1) + 1
25              else if (prob .GT. p_walk*2 .AND. prob .LE. p_walk*3) then
26                i_r(2) = i_r(2) - 1
27              else
28                i_r(2) = i_r(2) + 1
29              end if
30            end do
31
32            r = SQRT(real(i_r(1))**2 + real(i_r(2))**2)
33            aMedia = aMedia + r
34            aMediaQ = aMediaQ + r**2
35
36            vec_x(j) = i_r(1)
37            vec_y(j) = i_r(2)
38          end do
39
40          open(unit=1, file="saida-3-13783203-1000000.dat")
41          do i=1,n_andarilhos
42            write(1,*) vec_x(i), vec_y(i)
43          end do
44          close(unit=1)
45
46          aMedia = aMedia/real(n_andarilhos)
47          aMediaQ = aMediaQ/real(n_andarilhos)
48
49          write(*,*) "Valor esperado de <r>: ", aMedia
50          write(*,*) "Valor espeerado de <r^2>: ", aMediaQ
51
52          end program
```

2.3.1 Explicando a tarefa 3

Neste programa começamos definindo tudo que será utilizado ao longo do código, pois haverão muitos vetores que serão úteis para facilitar o processo. Sendo eles: *vec_x* responsável por guardar a posição de todos os andarilhos no eixo x; *vec_y* responsável por guardar a posição de todos os andarilhos no eixo y; *i_r* responsável por guardar a posição temporária de cada andarilho. Guardando também a quantidade de andarilhos em *n_andarilhos* e a quantidade de passos em *n_passos*.

Logo em seguida coloco os valores iniciais das variáveis que serão utilizadas para 0, e claro, a probabilidade do andarilho ir para qualquer lado como sendo *p_walk* que vale 1/4, ou seja, cada andarilho tem a mesma chance de ir para cada direção (esquerda, direito, baixo, cima). Também defino as variáveis que vão guardar o $< r >$ e $< r^2 >$, armazenando eles em *aMedia* e *aMediaQ*.

Começo fazendo um laço que irá percorrer um j para cada andarilho, em seguida faço um laço que irá percorrer um i para cada passo. Em cada passo eu faço um teste, onde arbitrariamente defini que entre os intervalos 0 e 1/4 o andarilho vai para esquerda, entre os intervalos 1/4 e 2/4 ele vai para a direita, entre os intervalos 2/4 e 3/4 ele vai para baixo e entre os intervalos 3/4 e 4/4 ele vai para cima. O vetor *i_r* serve justamente para indicar que seu primeiro valor representa o eixo X, e o segundo o eixo Y.

E após rodar esse laço, guardando a posição final do andarilho, eu incremento as variáveis de média que serão utilizados mais tarde com o valor do módulo de *i_r*, e por fim eu seto as posições em *vec_x* e *vec_y*.

Após repetir este processo para cada andarilho, eu abro o arquivo *saida-3-13783203-1000000.dat*, aonde eu altero seu último número de acordo com o número de passos afim de facilitar os gráficos que serão gerados. Por fim eu termino de calcular as médias e mostro os valores esperados no terminal.

2.3.2 Exemplo de uso

Nos exemplos de uso desta tarefa, repeti o código para diferentes quantidades de passos, para assim poder fazer um diagrama com todos eles. As diferentes quantidades de passos serão respectivamente 10, 10^2, 10^3, 10^4, 10^5, 10^6.

Além dos vetores para podermos ver o diagrama, também pode-se ter o $< r >$ e $< r^2 >$ de cada caso, e a partir deles podemos calcular $\Delta^2 =< r^2 > - < r >^2$. Que analiticamente pode ser calculado utilizando as equações 1 e 2, que nos revela uma nova equação:

$$\Delta^2 = 4Npq \tag{3}$$

Como todas as probabilidades são de 1/4 então fica simples calcular o Δ^2 analítico.

Tabela 1: Tabela com o valor de $< r^2 >$ e $< r >^2$ para cada número de passos.

$< r^2 >$	$< r >^2$	Δ^2	Δ^2 (analítico)	N
9.72999954	7.62214766813	2.10785187187	2.5	10
94.5839996	74.0567842223	20.5272153777	25	10^2
1039.33801	816.013305154	223.324704846	250	10^3
10165.3438	7963.08686168	2202.25693832	2500	10^4
98396.2969	77180.1074191	21216.1894809	25000	10^5
1015445.19	806193.544085	209251.645915	250000	10^6

Figura 10: Saída no terminal - Tarefa 3

```
1  silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
2  silvio$ ./tarefa-3-13783203.exe
3   Valor esperado de <r>:    2.76082373
4   Valor espeerado de <r^2>:    9.72999954
5  silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
6  silvio$ ./tarefa-3-13783203.exe
7   Valor esperado de <r>:    8.60562515
8   Valor espeerado de <r^2>:    94.5839996
9  silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
10 silvio$ ./tarefa-3-13783203.exe
11  Valor esperado de <r>:    28.5659466
12  Valor espeerado de <r^2>:    1039.33801
13 silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
14 silvio$ ./tarefa-3-13783203.exe
15  Valor esperado de <r>:    89.2361298
16  Valor espeerado de <r^2>:    10165.3438
17 silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
18 silvio$ ./tarefa-3-13783203.exe
19  Valor esperado de <r>:    277.813080
20  Valor espeerado de <r^2>:    98396.2969
21 silvio$ gfortran tarefa-3-13783203.f -o tarefa-3-13783203.exe
22 silvio$ ./tarefa-3-13783203.exe
23  Valor esperado de <r>:    897.882812
24  Valor espeerado de <r^2>:    1015445.19
```

Podemos ver que quanto maior for a potência de 10 da quantidade de andarilhos, maior é a potência de Δ^2, o que faz bastante sentido, pois mais uma vez estamos lidado com o fato de que o Δ final depende da quantidade total de passos. Por fim, podemos construir de fato o diagrama.

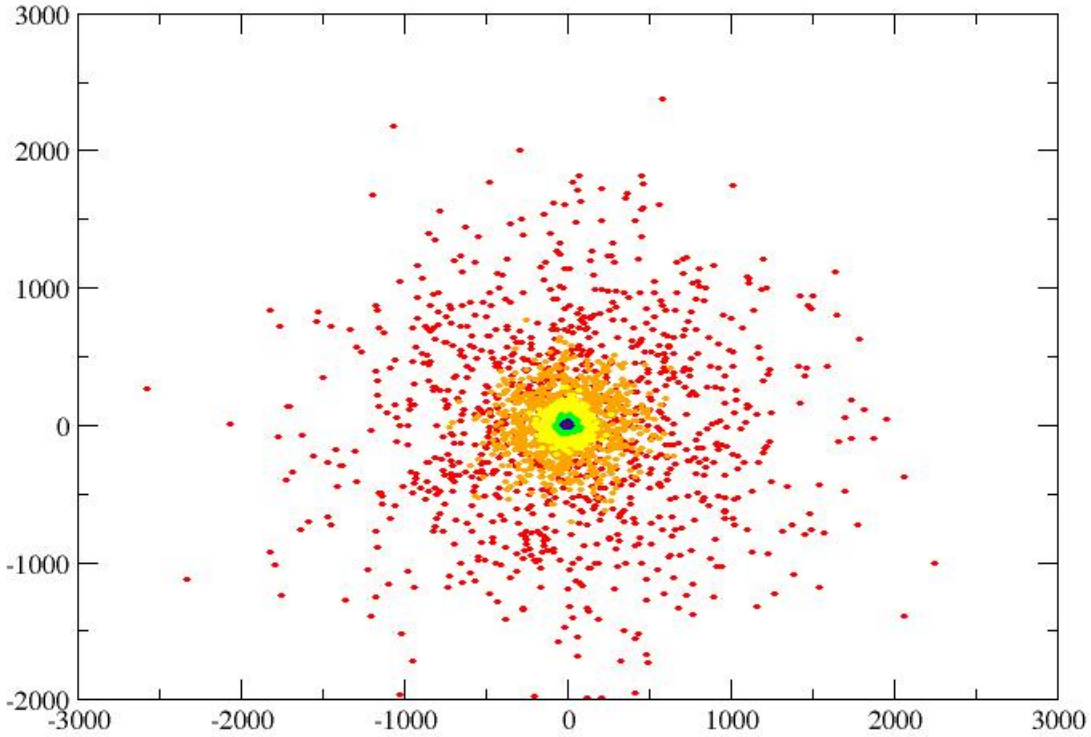

Figura 11: Posição X de cada andarilho x Posição Y de cada andarilho. Posição de cada andarilho após N passos. 10 passos = roxo, 10^2 passos = azul, 10^3 passos = verde, 10^4 passos = amarelo, 10^5 passos = laranja, 10^6 passos = vermelho.

2.4 Tarefa 4

Figura 12: Tarefa 4 - Parte 1

```
1           program Entropia
2           parameter (n_andarilhos=1000)
3           parameter (n_passos=1000)
4           dimension m_x(n_andarilhos)
5           dimension m_y(n_andarilhos)
6           integer iMax
7           integer iMin
8
9           p_walk = 1.e0/4.e0
10          m_x = 0
11          m_y = 0
12          i_s = 10
13
14          open(unit=1, file="saida-4-13783203.dat")
15          do m=1, n_passos
16            do n=1, n_andarilhos
17              prob = rand()
18              if (prob .LE. p_walk) then
19                m_x(n) = m_x(n) - 1
20              else if (prob .GT. p_walk .AND. prob .LE. p_walk*2) then
21                m_x(n) = m_x(n) + 1
22              else if (prob .GT. p_walk*2 .AND. prob .LE. p_walk*3) then
23                m_y(n) = m_y(n) - 1
24              else
25                m_y(n) = m_y(n) + 1
26              end if
27            end do
28
29            S = 0
30
31            imin_x = iMin(m_x, n_andarilhos)
32            imax_x = iMax(m_x, n_andarilhos)
33            imin_y = iMin(m_y, n_andarilhos)
34            imax_y = iMax(m_y, n_andarilhos)
```

Figura 13: Tarefa 4 - Parte 2

```
            do i=imin_x, imax_x, i_s
              do j=imin_y, imax_y, i_s
                n_inside = 0
                do k=1, n_andarilhos
                  if (m_x(k) .GT. i .AND. m_x(k) .LT. i+i_s) then
                    if (m_y(k) .GT. j .AND. m_y(k) .LT. j+i_s) then
                      n_inside = n_inside + 1
                    end if
                  end if
                end do

                p = real(n_inside)/real(n_andarilhos)
                if (p .GT. 0.0e0) then
                  S = S - (p * log(p))
                end if
              end do
            end do
            write(1,*) S
          end do
          close(unit=1)
          end program

          function iMax(mm, nn)
          dimension mm(nn)
          i_m = 0

          do ii=1, nn
            if (mm(ii) .GT. i_m) then
              i_m = mm(ii)
            end if
          end do
          iMax = i_m
          return
          end function

          function iMin(mm, nn)
          dimension mm(nn)
          i_m = 0

          do ii=1, nn
            if (mm(ii) .LT. i_m) then
              i_m = mm(ii)
            end if
          end do
          iMin = i_m
          return
          end function
```

2.4.1 Explicando a tarefa 4

Começo o código setando todas as funções e variáveis que serão utilizados ao longo do código, como o número de passos e de andarilhos, e claro, a posição de cada andarilho, agora em vetores de números inteiro m_x e m_y. Também seto as funções $iMax$ e $iMin$, onde elas

servem respectivamente para encontrar o maior valor e o menor valor de um vetor. Ambas são muito simples, eu faço um laço pelo vetor, e verifico se o próximo item é menor ou maior do que a minha variável de checagem.

Mais uma vez começo o código definindo os valores iniciais de tudo, mas agora defino uma nova variável, e esta representará o lado de um quadrado de tamanho *i_s*. O valor para este lado é arbitrário, ele serve exclusivamente para eu poder calcular a probabilidade de ter um andarilho num quadrado de lado *i_s* qualquer.

Abro o arquivo *saida-4-13783203.dat* e logo em seguida começo um laço percorrendo todos os passos, e logo depois eu faço um laço percorrendo todos os andarilhos. E agora para cada andarilho eu repito procedimento do exercício anterior: faço um teste, onde arbitrariamente defini que entre os intervalos 0 e 1/4 o andarilho vai para esquerda, entre os intervalos 1/4 e 2/4 ele vai para a direita, entre os intervalos 2/4 e 3/4 ele vai para baixo e entre os intervalos 3/4 e 4/4 ele vai para cima.

Mas agora, antes de dar o próximo passo, eu calculo a entropia do sistema. Se a entropia do sistema segue a fórmula:

$$S = -\sum P_i \ln P_i$$

Sendo assim, verifico a probabilidade em cada quadrado de lado *i_s* ao longo do espaço aonde os andarilhos podem estar em **cada** passo. A necessidade das funções de máximo e mínimo vem agora. Normalmente, quanto mais passos e mais andarilhos, mais lento o programa fica, porque eu vou percorrer uma malha de quadrados cada vez maior e verificar cada vez mais andarilhos. Entretanto, eu já sei que aonde não tiver andarilhos, a probabilidade P_i de ter um andarilho ali será 0, então não há necessidade de criar uma malha que percorre todo meu espaço disponível (o espaço máximo que os andarilhos podem chegar é de *-n_passos* até *n_passos* em cada eixo). Sendo assim, eu pego apenas o ponto máximo e mínimo de cada eixo aonde o andarilho pode estar, podendo otimizar o código. Isso fez o código reduzir de 5 minutos para 11 segundos para o caso de 10000 andarilhos e $i\_s = 10$. Quanto maior o lado do quadrado, mais rápido o código fica também, porque diminuímos a quantidade de quadrados na malha, facilitando os laços.

O próximo laço serve exclusivamente para verificar se cada andarilho está dentro do quadrado, vejo se a coordenada x de cada andarilho está entre i e $i+i\_s$, e se a coordenada y de cada andarilho está entre j e $j+i\_s$. Acrescento uma variável *n_inside* para indicar quantos andarilhos estão ali dentro, e por fim consigo calcular a probabilidade como sendo (quantos tem dentro)/(total), e com isso consigo calcular a entropia, para poder colocar num gráfico.

2.4.2 Exemplo de uso

Este programa não mostra nenhuma mensagem no terminal ou exige que o usuário insira parâmetros, sendo assim, todos os dados aqui serão oriundos do arquivo de saída *saida-4-13783203*. E no caso deste output podemos fazer um gráfico, facilitando assim o crescimento da visualização do crescimento da entropia para cada passo.

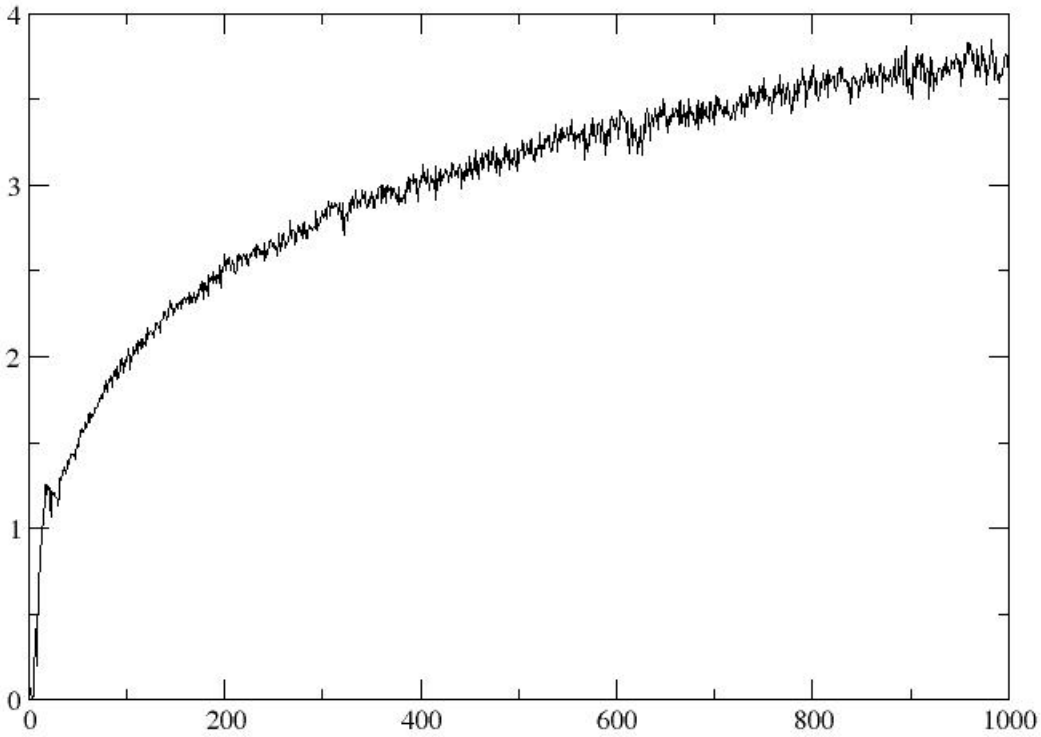

Figura 14: Entropia x N (passos).