UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Introdução à Física Computacional - Projeto 1

Silvio Lacerda de Almeida

Agosto, 2023
São Carlos, SP

Sumário

Lista de Figuras

1 Introdução

O primeiro projeto de Introdução à Física Computacional contempla 8 tarefas iniciais, e delas explicarei detalhadamente o pensamento que originou o código, assim como discutirei o resultado de alguns deles. Em particular a tarefa 8 possui uma análise mais detalhada de alguns casos especiais.

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 Raizes
2
3               write(*,*) "Escreva os coeficientes da equacão de segundo grau"
4               write(*,*) "Dada a forma f(x) = axˆ2 + bx + c"
5               read(*,*) a, b, c
6
7               r = -b/(2*a)
8               delta = (b**2) - (4*a*c)
9
10              if (delta .EQ. 0) then
11                      write(*,*) r
12              else if (delta .LT. 0) then
13                      write(*,*) "Não existem raízes reais"
14              else
15                      rr = SQRT(delta)/(2*a)
16                      r1 = r + rr
17                      r2 = r - rr
18
19                      write(*,*) r1
20                      write(*,*) r2
21              end if
22
23              end program
```

1 Explicando a tarefa 1

Pede-se para o usuário dar os coeficientes da equação de segundo grau, e colocamos seus valores nas variáveis a, b e c. Com isso começo a escrever a equação que soluciona a equação do segundo grau, associando a variável r ao valor que representa a parte que não possui o Δ. Sabendo que $\Delta = b^2 - 4ac$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Com isto, basta calcular delta para saber a quantidade de raízes. Se $\Delta > 0$, então existem duas raízes, se $\Delta = 0$ então existe apenas uma raíz, que é a variável r já definida, e se $\Delta < 0$ então não existem raízes reais.

2.1.1 Exemplo de uso

Figura 2: Saída no terminal - Tarefa 1

```
silvio$ ./tarefa-1-13783203.exe
 Escreva os coeficientes da equacão de segundo grau
 Dada a forma f(x) = axˆ2 + bx + c
1 4 1
 -0.267949224
  -3.73205090
```

Essa primeira tarefa é bem simples, não existe muita possibilidade de desenvolver sobre ela, além de algumas convenções óbvias. As variáveis a, b e c são reais, por isso as raízes possuem precisão simples mesmo o usuário inserindo "aparentemente"uma entrada inteira. E claro, não considerei o caso em que $a = 0$, pois se é uma equação de segundo grau, evidentemente $a \neq 0$.

2.2 Tarefa 2

Figura 3: Tarefa 2

```
1           program Triangulo
2
3           dimension v1(3), v2(3)
4           dimension vet(3)
5
6           write(*,*) "Escreva as coordenadas do primeiro vetor"
7           read(*,*) v1
8           write(*,*) "Escreva as coordenadas do segundo vetor"
9           read(*,*) v2
10
11          vet(1) = v1(2)*v2(3) - v2(2)*v1(3)
12          vet(2) = v1(3)*v2(1) - v2(3)*v1(1)
13          vet(3) = v1(1)*v2(2) - v2(1)*v1(2)
14
15          area = 0.5e0*SQRT(vet(1)**2 + vet(2)**2 + vet(3)**2)
16
17          write(*,*) "A área do triângulo é: ", area
18
19          endprogram
```

2 Explicando a tarefa 2

Começo o programa declarando dois vetores de dimensão 3, uma vez que é dito que usaremos apenas 3 coordenadas. Peço para o usuário declarar os valores de cada valor e após isso calcular a área se torna trivial. Com isto, basta lembrar que o produto vetorial entre dois vetores, pode nos dar área do paralelogramo que os dois formam. Basta calcular o produto vetorial, em seguida o seu módulo, e por fim dividir por 2, conforme foi feito. E assim, temos a área do triângulo.

2.2.1 Exemplo de uso

Figura 4: Saída no terminal - Tarefa 2

```
1  silvio$ ./tarefa-2-13783203.exe
2   Escreva as coordenadas do primeiro vetor
3  1 1 0
4   Escreva as coordenadas do segundo vetor
5  2 3 0
6   A área do triângulo é:   0.500000000
7
```

Essa tarefa também não tem muito o que desenvolver, apenas alguns comentários. Como dizer que haviam funções nativas que facilitariam a trativa de vetores, mas por via das dúvidas executei tudo manualmente, como o produto escalar e o cálculo de seu módulo.

2.3 Tarefa 3

Figura 5: Tarefa 3

```
1           program OrdenaNumeros
2
3           dimension bNum(100000)
4           iQtdLinha = 0
5
6           write(*,*) "Digite quantos números devem ser imprimidos"
7           read(*,*) m
8
9           open(unit=1, file="entrada-3-13783203.txt")
10          do
11            read(1,*, end=5) bNum(iQtdLinha+1)
12            iQtdLinha = iQtdLinha + 1
13          end do
14  5       close(unit=1)
15
16          open(unit=2, file="saida-3-13783203.txt", status="new")
17          do i=1, m
18            auxMenorNum = bNum(1)
19            do j=1, iQtdLinha
20              if (bNum(j) < auxMenorNum) then
21                aTemp = auxMenorNum
22                auxMenorNum = bNum(j)
23                bNum(j) = aTemp
24              else
25                continue
26              end if
27            end do
28            write(2,*) auxMenorNum
29          end do
30          close(unit=2)
31
32          end program
33
```

3 Explicando a tarefa 3

Começo o código declarando um vetor capaz de alocar até 100000 números e uma variável inteira chamada iQtdLinha, que servirá de contador mais tarde. A razão pela qual se cria a variável bNum com 100000 espaços livres é porque não foi permitido usar ALLOCATE e DEALLOCATE, ou seja, alocação dinâmica de espaços, portanto, se fez necessário apenas estipular um número muito grande.
Depois disso lê-se quantos números devem ser ordenados, abrimos o arquivo "entrada-3-13783203.txt"e lemos todos os números que tem lá, enquanto incrementando a variável iQtdLinha, para mais tarde utilizarmos no loop. Dizemos que o ao final é necessário fazer um salto para a linha identificador 5, que seria a linha 14.
Agora podemos de fato escrever a saída, abrindo o arquivo "saida-3-13783203.txt"na unidade 2, e indicando que é um arquivo novo. Começo o laço indicando que o final é a quantidade de números a serem ordenados, e indico que a variável real auxMenorNum vai assumir temporariamente o "posto"de menor número, pegando sempre o primeiro valor da lista. Agora, entramos no segundo laço, que tem como limite a quantidade de linhas dentro do arquivo de entrada. Verifica-se então se cada item desta lista é menor que auxMenorNum, se for menor então fazemos a ordenação, trocando bNum(j) e auxMenorNum de lugar. A variável aTemp dentro deste bloco de IF tem como intenção apenas facilitar a troca de variáveis. Repetimos isso para cada valor dentro do arquivo de entrada, que agora está dentro do vetor, e escrevemos dentro da unidade 2, que é o arquivo de saída. Por fim, fechamos a unidade 2.

2.3.1 Exemplo de uso

Figura 6: entrada-3-13783203.txt

```
32
22
11
4
14
5
9922
23
5
0

```

Figura 7: saida-3-13783203.txt

```
     0.00000000
     4.00000000
     5.00000000
     5.00000000
     11.0000000
     14.0000000
     22.0000000
     23.0000000
     32.0000000
     32.0000000

```

Como é possível ver, existem casas decimais excessivas sendo colocadas no arquivo de saída, mas isso não é atoa. A descrição da tarefa exigia que fosse feito um programa que lesse números reais, logo, a ideia é que o programa ordene mesmo caso o arquivo de entrada possua um número como 232.4839.

2.4 Tarefa 4

Figura 8: Tarefa 4

```
          program NumerosPrimos
          write(*,*) "Digite um número inteiro"
          read(*,*) n

          open(unit=1, file="saida-4-13783203.txt", status="new")
          if (n .GT. 1) then
            j = 2
  1         if (j .LE. n) then
              do i=2, j-1, 1
                if (MOD(j, i) .EQ. 0) then
                  j = j+1
                  goto 1
                end if
              end do
              write(1,*) j
              j = j+1
              goto 1
            end if
          else
            write(*,*) "Não há primos"
          end if
          close(unit=1)
          end program

```

4 Explicando a tarefa 4

Começo o código pedindo para o usuário digitar um número inteiro n, em seguida verifico se ele é maior que 1, se for eu crio uma variável j com seu valor inicial sendo 2, e se 2 for menor ou igual a n, então podemos começar o laço, aonde verifico se na divisão de um pelo outro eles tem resto igual a 0. Se tiver, aumentamos e vamos de novo para a linha 1, e aí recomeçamos a verificação para poder avançar. O raciocínio é simples, basta lembrar que um número primo só divisível por 1 e por ele mesmo, por isso verifica-se o resto da divisão. Todos os valores que são primos são escritos dentro do arquivo *saida-4-13783203.txt*.

2.4.1 Exemplo de uso

Figura 9: Entrada no terminal - Tarefa 4

```
1  silvio$ ./tarefa-4-13783203.exe
2   Digite um número inteiro
3  30
```

Figura 10: saida-4-13783203.txt

```
1              2
2              3
3              5
4              7
5             11
6             13
7             17
8             19
9             23
10            29
```

2.5 Tarefa 5

Figura 11: Tarefa 5

```
1           program Log
2           real*8 x, aLnFortran, aLnSerie
3           real*8 logNatty
4
5           write(*,*) "Digite um número: "
6           read(*,*) x
7           aLnFortran = dLog(x)
8
9           if (x .GT. 1) then
10            aLnSerie = - logNatty(1.0d0/x)
11          else
12            aLnSerie = logNatty(x)
13          end if
14
15          write(*,*) "Log do fortran: ", aLnFortran
16          write(*,*) "Log da série  : ", aLnSerie
17
18          end program
19
20          real*8 function logNatty(a)
21          real*8 a, aux, eperc
22          real*8 aLnS
23          i = 1
24          eperc = 1e-15
25          aLnS = 0.0d0
26
27          do
28            aux = ((1.0d0-a)**i)/i
29            aLnS = aLnS + aux
30            i = i + 1
31            if (abs(aux) .LT. eperc) then
32              exit
33            end if
34          end do
35          logNatty = -aLnS
36          return
37          end function
```

5 Explicando a tarefa 5

No início do código declaro todas as variáveis que terão precisão dupla no código, inclusive a função *logNatty*, que se localiza ao final do código, fazendo exatamente a expansão em série proposta no enunciado da tarefa. Só que há um problema, a função converge apenas para valores menores que 2, por isso existe uma coisa simples que pode ser feita. Com x¿1, todo número da forma 1/x será sempre menor que 1, portanto, por meio da propriedade de log, podemos fazer:

$$log(x) = -log(1/x)$$

E com isso podemos calcular o log de qualquer número. A precisão está setada para 1e-15, mas pode ser tão pequena ou tão grande quanto quiser.

2.5.1 Exemplo de uso

Figura 12: Saída no terminal - Tarefa 5

```
silvio$ time ./tarefa-5-13783203.exe
 Digite um número:
1000000
 Log do fortran:    13.815510557964274
 Log da série  :    13.815510557150269

real    0m3.552s
user    0m1.531s
sys     0m0.002s
silvio$ ./tarefa-5-13783203.exe
 Digite um número:
2
 Log do fortran:   0.69314718055994529
 Log da série  :   0.69314718055994451
```

Em um dos exemplos eu executo o programa junto com um *time* para poder checar o quão rápido meu algoritmo estava indo. A priori, não há nada de surpreendente que este código seja rápido, mas isso porque não há o contexto das tentativas passadas. Antes de notar que isso poderia ser feito, eu testei vários outros algoritmos que demoravam muito, inclusive podendo passar de 7 minutos a execução caso a precisão fosse menor que 1e-9.

2.6 Tarefa 6

Esse exercício em específico é importante falar um algebrismo por trás da equação do enunciado. Sabendo que $N \in \mathbb{R}$, podemos desenvolver da seguinte forma:

$$(z-2)^N = 3$$
$$z-2 = 3^{\frac{1}{N}}$$
$$z = 3^{\frac{1}{N}} + 2$$

Sabemos que $3^{\frac{1}{N}} + 2 = (3^{\frac{1}{N}} + i0) + (2 + i0)$ e a partir daí podemos abrir:

$$3^{\frac{1}{N}} = \rho e^{2k\pi i} = 3^{\frac{1}{N}}[\cos(2k\pi) + i\sin(2k\pi)]$$
$$z = [3^{\frac{1}{N}}\cos(2k\pi) + 2] + i[3^{\frac{1}{N}}\sin(2k\pi)]$$

Figura 13: Tarefa 6

```
1           program RaizComplexa
2             write(*,*) "Digite um número N: "
3             read(*,*) n
4
5             PI = 4.0e0 * atan(1.0e0)
6             do i=1, n
7               am = 3.0e0**(1.0e0/n)
8               ri = am*sin((2.0e0*i*PI)/n)
9               rr = 2 + am*cos((2.0e0*i*PI)/n)
10              write(*,*) rr, " ", ri,"i"
11            end do
12          end program
```

6 Explicando a tarefa 6

Começo o código pedindo para o usuário me dar o número de raízes que ele quer, em seguida declaro o valor de PI, e depois escrevo o algoritmo que vai me dar N raízes, com base na notação trigonométrica de um número complexo, e utilizando o raciocínio apresentado antes da exibição do código, encontramos o valor de z para cada k, sendo $k \in \mathbb{Z}^*$.

2.6.1 Exemplo de uso

Figura 14: Saída no terminal - Tarefa 6

```
1  silvio$ ./tarefa-6-13783203.exe
2   Digite um número N:
3  6
4     2.60046840          1.04004192     i
5     1.39953148          1.04004180     i
6    0.799063087         -1.04989240E-07 i
7     1.39953160         -1.04004192     i
8     2.60046887         -1.04004169     i
9     3.20093679          2.09978481E-07 i
```

2.7 Tarefa 7

Figura 15: Tarefa 7

```
1           program MonteCarlo
2           real*8 n_dentro, n_total, vol
3           real dist
4           dist = 0.d0
5           n_dentro = 0.0d0
6           n_total = 0.0d0
7
8           write(*,*) "Digite o número de dimensões da esfera"
9           read(*,*) iDimen
10          write(*,*) "Digite o número de partículas"
11          read(*,*) iPart
12
13          do i=1, iPart
14            dist = 0.0d0
15            do j=1, iDimen
16              a = rand()
17              dist = dist + (a-0.5d0)**(2.0d0)
18            end do
19            if (SQRT(dist) .LE. 0.5) then
20              n_dentro = n_dentro + 1.0d0
21            end if
22            n_total = n_total + 1.0d0
23          end do
24          vol = n_dentro/n_total
25          write(*,*) "O volume em ", iDimen, " dimensões é ", vol
26          end program
27
```

7 Explicando a tarefa 7

Nesta tarefa começo o código declarando as variáveis que usarão precisão dupla e logo depois inidoc seus valores iniciais. Logo em seguida, solicito ao usuário que escreva quantas dimensões tem a esfera, e quantas partículas devem ser utilizadas. O número de partículas poderia ser uma constante dentro do código, a ideia era simplesmente facilitar os testes.
A ideia deste código é muito simples, vou partir do princípio que a esfera tem raio $1/2$, e com seu eixo deslocado também $1/2$ em todas as dimensões, fazendo com que haja uma forma simples de utilizar o método de Monte Carlo para resolver o problema. Crio d coordenadas, sendo d a quantidade de dimensões da esfera, e digo que cada uma dessas coordenadas será um número aleatório entre 0 e 1 através da função *rand()* (daí a ideia de usar $r = 1/2$). Depois eu verifico a distância deste ponto de d dimensões até o centro da esfera, através da métrica de distância usual.

$$d(a,b) = \sqrt{\sum_{n=1}^{d}(a_i - b_i)^2}$$

Disto isto, basta verificar se a distância é menor ou igual ao raio da esfera, que é $1/2$, e com isso sabemos que a partícula caiu dentro da esfera. Se não, a partícula caiu fora. O volume da

esfera é aproximadamente V_{dentro}/V_{fora}, e com isso podemos estimar um valor.

2.7.1 Exemplo de uso

A priori, não há necessidade de usar *real*8*, mas eu achei que seria mais interessante manter os valores assim, para ter uma noção do quão próximo dos valores tabelas podemos chegar. Para notar esta diferença, basta utilizarmos cada vez mais partículas.

Podemos verificar os valores de acordo com o site ⟨https://en.wikipedia.org/wiki/Volume_of_an_n-ball⟩ que já nos mostra uma tabela com a qual podemos facilitar as contas. E com ele podemos fazer a seguinte tabela, e comparar com os casos de uso.

Dimensão	Equação de Volume	Volume (R = 1/2)
2	πR^2	$0,7853$
3	$\frac{4\pi}{3}R^3$	$0,5235$
4	$\frac{\pi^2}{2}R^4$	$0,3084$
5	$\frac{8\pi^2}{15}R^5$	$0,1644$

Tabela 1: Valores de volume para diferentes dimensões.

E utilizando o método de Monte Carlo, tentamos ver a aproximação do volume para estas dimensões.

Figura 16: Saída no terminal - Tarefa 7

```
1  silvio$ ./tarefa-7-13783203.exe
2   Digite o número de dimensões da esfera
3  2
4   Digite o número de partículas
5  10000
6   O volume em            2  dimensões é   0.78029999999999999
7  silvio$ ./tarefa-7-13783203.exe
8   Digite o número de dimensões da esfera
9  3
10  Digite o número de partículas
11 10000
12  O volume em            3  dimensões é   0.52749999999999997
13 silvio$ ./tarefa-7-13783203.exe
14  Digite o número de dimensões da esfera
15 4
16  Digite o número de partículas
17 10000
18  O volume em            4  dimensões é   0.31230000000000002
19 silvio$ ./tarefa-7-13783203.exe
20  Digite o número de dimensões da esfera
21 5
22  Digite o número de partículas
23 10000
24  O volume em            5  dimensões é   0.16489999999999999
```

Como pode-se ver, a aproximação é muito boa. E a ideia é que quanto mais partículas sejam escolhidas, maior a precisão do volume.

2.8 Tarefa 8

Figura 17: Tarefa 8

```
1           program VolumeEsfera
2           real*8 g, vol
3           real*8 r, PI
4           real*8 aux1, aux2
5
6           PI = 4.0d0*ATAN(1.0)
7
8           write(*,*) "Digite quantas dimensões tem a esfera"
9           read(*,*) iDimen
10          write(*,*) "Digite o raio da esfera"
11          read(*,*) r
12
13          open(unit=1,file="dimensoes-esferas.dat", status="new")
14          do i=2, iDimen
15            d_i = real(i,8)
16            aux1 = g(d_i/2.0d0 + 1.0d0)
17            aux2 = ((PI**(d_i/2.0d0))*(r**d_i))
18
19            vol = aux2/aux1
20            write(1,*) i, vol
21          end do
22          close(unit=1)
23          end program
24
25          recursive real*8 function g(x) result(seq)
26          real*8 x, PI
27          PI = 4.0d0*ATAN(1.0)
28
29          if (x .EQ. 0.5d0) then
30            seq = SQRT(PI)
31          else if (x .EQ. 1.0d0) then
32            seq = 1.0d0
33          else
34            seq = (x-1.0d0)*g(x-1.0d0)
35          end if
36          return
37          end function g
38
```

8 Explicando a tarefa 8

Para uma precisão maior, declaramos as variáveis como reais de precisão dupla. As funções gamma e de volume também precisaram ser declaradas para podermos utilizalas. A função gamma é uma função recursiva que segue precisamente o que o enunciado exige.

$$\Gamma(1/2) = \sqrt{\pi}, \Gamma(1) = 1, \Gamma(x+1) = x\Gamma(x)$$

Logo após as declarações em precisão dupla eu instancio PI para utilizar no cálculo do volume. Logo em seguida eu leio as variáveis *iDimen* e *r* para saber as dimensões que o usuário quiser, e o raio da esfera. Logo em seguida, crio o arquivo *dimensoes-esfera.dat* que irá conter o volume de cada esfera, juntamente com o valor de sua dimensão. A equação para o cálculo do volume é a mesma dada pelo enunciado.

$$V_p = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} R^d$$

Com estes valores, o problema fica simples, rodo um laço que cria as linhas no arquivo de saída indicando *dimensao - volume*.

2.8.1 Exemplo de uso

Rodei o programa passando $iDimen = 14$ e $\mathbf{r} = 1$, e com isso o arquivo saída ficou:

Figura 18: dimensoes-esfera.dat

```
 2   3.1415927410125732
 3   4.1887903213500977
 4   4.9348024751914465
 5   5.2637893068708772
 6   5.1677132114641093
 7   4.7247663647671860
 8   4.0587125781926048
 9   3.2985092698962122
10   2.5501643947012624
11   1.8841041415397495
12   1.3352629917970349
13  0.91062890682661579
14  0.59926464605318508
```

E utilizando o *xmgrace* podemos traçar um gráfico que mostra o comportamento da aproximação para volume ao longo das 14 dimensões, conforme está ilustrado na Figura 19.

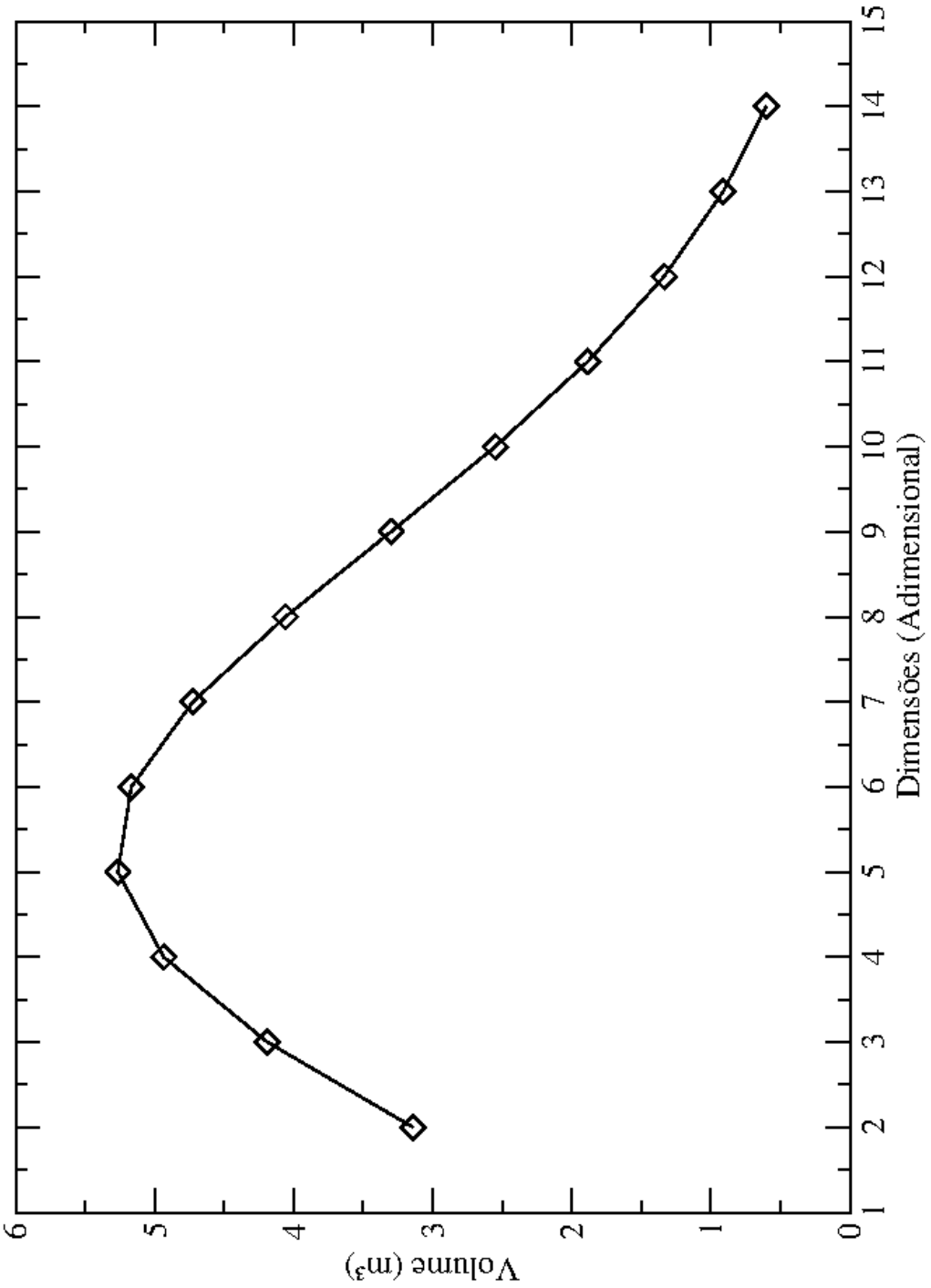

Figura 19: Gráfico dimensão x volume.

Outra parte desta tarefa envolve responder duas perguntas. A **letra A** pergunta quantas

vezes o volume de um cubo de d dimensões será maior que uma esfera de d dimensões, e para isto podemos utilizar nosso algoritmo, sendo $r = 0.5d0$, pois assim teremos sempre uma esfera contida dentro do cubo.

Figura 20: dimensoes-esfera.dat

```
 2  0.78539818525314331
 3  0.52359879016876221
 4  0.30842515469946541
 5  0.16449341583971491
 6   8.0745518929126708E-002
 7   3.6912237224743641E-002
 8   1.5854346008564862E-002
 9   6.4424009177660394E-003
10   2.4903949167004516E-003
11   9.1997272536120580E-004
12   3.2599194135669797E-004
13   1.1116075522785837E-004
14   3.6576211306957097E-005
15   1.1640727390310513E-005
16   3.5908612479899510E-006
17   1.0756007255736983E-006
18   3.1336176751857167E-007
19   8.8923669255004920E-008
20   2.4611376353680353E-008
21   6.6514750913268189E-009
22   1.7572482113420612E-009
23   4.5426577965083217E-010
24   1.1501162968561558E-010
25   2.8542361516829431E-011
26   6.9484557875453092E-012
27   1.6605273287523698E-012
28   3.8980746898927051E-013
29   8.9943113832093835E-014
30   2.0410271916152934E-014
31   4.5574940890157034E-015
32   1.0018869077168568E-015
33   2.1693621738271884E-016
34   4.6287068185276640E-017
35   9.7360749398896434E-018
36   2.0196544085224873E-018
37   4.1333489671772612E-019
38   8.3485942489056432E-020
39   1.6647819373533205E-020
40   3.2784853862526607E-021
41   6.3781303045829775E-022
42   1.2261507013056145E-022
43   2.3299404495476969E-023
44   4.3773478893286295E-024
45   8.1330266703229069E-025
46   1.4947656906523404E-025
47   2.7181550585051170E-026
48   4.8916094200708730E-027
49   8.7136083681084511E-028
50   1.5367444645963379E-028
```

O volume do cubo de lado 1, sempre será $1m^d$, dito isto, basta dividir 1 pelo volume da esfera, e logo percebe-se que a tendência é que o cubo continue sendo muito maior que a esfera, fazendo com que no infinito, o volume do cubo seja infinitas vezes maior que o volume da esfera.

A **letra B** por outro lado é mais simples. Conversando em sala de aula com o professor sobre esta questão, foi informado que poderia ser desconsiderado o volume da proteína e então que a constante de avogadro R_a deveria ser da forma $R_a = \frac{V_{macroscopio}}{V_{atomo}} = \frac{10^{-3d}}{10^{-10d}} = 10^{7d}$. Para termos uma ideia se isso é coerente, basta comparamos com a nossa dimensão, de acordo com o cálculo que fizemos, $R_a = 10^{7.3} = 10^{21}$. O valor atual é $6,022.10^{23}$, um valor que está quase na mesma ordem de grandeza.

3 Conclusão

O projeto inteiro foi escrito em Fortran, e serviu para ter uma noção muito clara do quão poderosa esta linguagem pode ser, principalmente no que tange poder de processamento. A velocidade com que a maioria dos algoritmos é executada, sempre passa da velocidade caso o mesmo fosse compilado em *python* por exemplo. Alguns problemas tinham detalhes não muito óbvios, mas com um pouco de esforço foi possível chegar neles, sem muita dificuldade. No entanto, no geral, as tarefas do projeto 1 continham problemas relativamente simples, mas muito interessantes para se ter um domínio dos pormenores de fortran 77, que pelo menos para mim, foi um primeiro contato. Aprendi a usar **functions**, **routines** e várias outras funcionalidades, que deixaram os códigos bem mais legíveis e bem organizados.