

UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE FÍSICA DE SÃO CARLOS

INTRODUÇÃO À FÍSICA COMPUTACIONAL

# **Projeto 1**

Levy Bruno do Nascimento Batista — 11212550

Prof. Francisco Castilho Alcaraz

São Carlos  
08/2021 — 09/2021

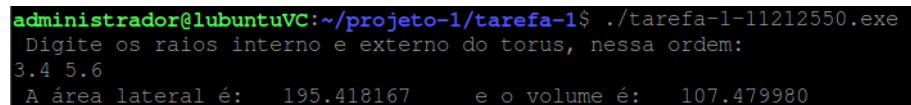
## Tarefa 1. Torus

Nessa primeira tarefa, o objetivo é, dados os raios interno e externo de um torus, que aqui foram chamados de  $r_1$  e  $r_2$ , respectivamente, calcular a sua área total e o seu volume. Todas as variáveis utilizadas ( $r_1$ ,  $r_2$ ,  $area$ ,  $volume$ ) são do tipo real de simples precisão e foram declaradas implicitamente, enquanto o valor de  $\pi$  foi definido com a função `acos()`. Observe que os raios são lidos diretamente do terminal, e a saída consiste em uma frase que contém os valores de área total e volume, que também é escrita no terminal.

```
tarefa-1-11212550.f:
1      program tarefa1
2
3      parameter (pi = acos(-1.0))
4      print*, 'Digite os raios interno e externo do torus,
          nessa ordem:'
5      read(*,*) r1, r2
6      area = pi**2*(r2**2 - r1**2)
7      volume = pi**2*(r2 - r1)**2*(r1 + r2)/4
8      write(*,*) 'A área lateral é:', area, 'e o volume é:',
          volume
9
10     end
```

Segue em anexo um exemplo de entrada testado e a respectiva saída gerada pelo programa.

Figura 1: Entrada e saída da tarefa-1.



```
administrador@ubuntuVC:~/projeto-1/tarefa-1$ ./tarefa-1-11212550.exe
Digite os raios interno e externo do torus, nessa ordem:
3.4 5.6
A área lateral é: 195.418167 e o volume é: 107.479980
```

Fonte: gerado pelo autor

## Tarefa 2. Prisma

Nessa tarefa, a área lateral e o volume de um prisma formado pelos vetores  $\mathbf{v}_1$ ,  $\mathbf{v}_2$  e  $\mathbf{v}_3$  -  $\mathbf{v}_2$  são calculados a partir das coordenadas desses vetores fornecidas na entrada pelo terminal. Enquanto os arrays  $v_1$ ,  $v_2$  e  $v_3$  correspondem aos respectivos vetores, o array  $v_4$  recebe as coordenadas do vetor  $\mathbf{v}_3 - \mathbf{v}_2$ . Observe que a área de cada face lateral distinta é obtida pela função `areaface`, que calcula a norma do produto vetorial entre dois vetores, cujas coordenadas são passadas como parâmetros da função; da mesma forma, o volume é obtido por meio

da função prodmisto que, recebendo 9 parâmetros reais (as coordenadas de 3 vetores), retorna o resultado do chamado produto misto  $\mathbf{v}_1 \cdot (\mathbf{v}_2 \times \mathbf{v}_3)$ .

tarefa-2-11212550.f:

```

1      program tarefa2
2
3      real v1(3), v2(3), v3(3), v4(3)
4      print*, 'Digite as coordenadas do vetor v1:'
5      read(*,*) v1(1), v1(2), v1(3)
6      print*, 'Digite as coordenadas do vetor v2:'
7      read(*,*) v2(1), v2(2), v2(3)
8      print*, 'Digite as coordenadas do vetor v3:'
9      read(*,*) v3(1), v3(2), v3(3)
10 c   v4 é implementado como o vetor v3 - v2
11      do i=1, 3
12          v4(i)=v3(i)-v2(i)
13      end do
14
15      area=2*(areaface(v1(1), v1(2), v1(3), v2(1), v2(2),
16 +          v2(3))+
17 +          areaface(v1(1), v1(2), v1(3), v4(1), v4(2),
18 +          v4(3))+
19 +          areaface(v2(1), v2(2), v2(3), v4(1), v4(2),
20 +          v4(3)))
21      volume=prodmisto(v4(1), v4(2), v4(3), v1(1), v1(2),
22 +          v1(3),
23 +          v2(1), v2(2), v2(3))
24
25      write(*,*) 'A área lateral do prisma formado é:', area,
26 +          'e o volume é:', volume
27
28      end
29 c   função que retorna a área de uma face do prisma
30      real function areaface(x1, y1, z1, x2, y2, z2)
31
32      areaface=sqrt((y1*z2-z1*y2)**2+(z1*x2-x1*z2)**2+
33 +          (x1*y2-y1*x2)**2)
34      return
35
36      end
37 c   função que retorna o volume do prisma a partir do
38 produto misto de 3 vetores
39      real function prodmisto(x4, y4, z4, x1, y1, z1, x2,
40 +          y2, z2)
41
42      prodmisto=x4*(y1*z2-z1*y2)+y4*(z1*x2-x1*z2)+z4*(x1*y2-y1*x2)
43      return
44
45      end

```

Segue em anexo um exemplo de entrada testado e a respectiva saída gerada pelo programa.

Figura 2: Entrada e saída da tarefa-2.

```
administrador@lubuntuVC:~/projeto-1/tarefa-2$ ./tarefa-2-11212550.exe
Digite as coordenadas do vetor v1:
3.4 4.5 5.6
Digite as coordenadas do vetor v2:
8.378 5.832 3.812
Digite as coordenadas do vetor v3:
12.81 15.93 5.82
A área lateral do prisma formado é: 317.520721 e o volume é: 238.2812
96
```

Fonte: gerado pelo autor

### Tarefa 3. Ordenação

Nessa tarefa, o objetivo é gerar um arquivo de saída com os  $M$  menores números ordenados do arquivo de entrada, que contém  $N$  números reais ( $M \leq N$ ). O programa foi escrito para  $N = 20$ , mas basta alterar o valor de "idmax" definido pelo comando *parameter* para que o mesmo funcione para arquivos de entrada com uma quantidade diferente de números. Inicialmente, o valor de  $M$  é fornecido pelo terminal e todos os  $N$  números da entrada são armazenados no array "lista", e após isso o programa vai buscando os  $M$  menores números presentes em "lista" da seguinte forma: a variável "tmp" recebe o primeiro elemento de lista e seu valor vai sendo comparado com os demais presentes em "lista", caso encontre algum menor, então "tmp" assume esse valor. Além disso, a posição desse valor é armazenada na variável "ipos", e após todos os valores serem checados, o valor de "tmp" é escrito no arquivo de saída e o valor armazenado na posição "ipos" é mudado para  $rmax + 1$ , "rmax" sendo o maior valor presente no arquivo de entrada.

tarefa-3-11212550.f:

```
1      program tarefa3
2
3      parameter (idmax = 20)
4      real lista(idmax)
5      rmax = 0.0
6      open(unit=1, file='entrada-3-11212550')
7      open(unit=2, file='saida-3-11212550')
8      write(*,*) 'Quantos números você quer ordenar? menos
          que ', idmax
9      read(*,*) n
```

```

10 c      vetor lista recebe todos os números que estão no
      arquivo de entrada
11      do i = 1, idmax
12          read(1,*)lista(i)
13          if (lista(i).gt.rmax) rmax = lista(i) !rmax recebe
      o maior número do vetor
14      end do
15
16
17      do k = 1, n
18          tmp = lista(1)
19          do j = 2, idmax
20 c      tmp recebe o atual menor número e ipos armazena a
      posição dele
21          if (tmp.gt.lista(j)) then
22              tmp = lista(j)
23              ipos = j
24          endif
25      end do
26 c      caso o menor número seja o primeiro da lista, ipos
      recebe 1
27          if (tmp.eq.lista(1)) ipos = 1
28          write(2,*) tmp
29 c      as posições ordenadas tem o valor alterado para não
      serem contadas de novo
30          lista(ipos) = rmax+1
31      end do
32
33      write(2,*)'Você quis ordenar', n, 'números'
34      close(1)
35      close(2)
36
37      END

```

Seguem em anexo um exemplo de arquivo de entrada e a respectiva saída gerada, além de um imagem mostrando a tela do terminal quando o programa é executado.

entrada-3-11212550:

```

1  4.8548
2  31.473
3  18.93874
4  1.3727
5  83.74
6  47.2737
7  56.37472
8  60.9845
9  70.783
10 21.874

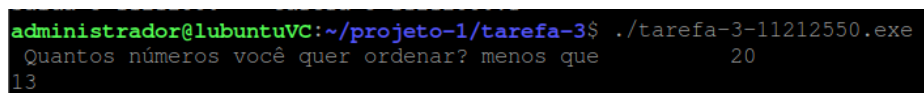
```

```

11 17.73
12 79.2421
13 14.8523
14 26.64
15 65.098
16 36.505
17 86.42
18 58.4847
19 93.9
20 99.395

```

Figura 3: Tela do terminal.



```

administrador@lubuntuVC:~/projeto-1/tarefa-3$ ./tarefa-3-11212550.exe
Quantos números você quer ordenar? menos que      20
13

```

Fonte: gerado pelo autor

```

saida-3-11212550:
1      1.37269998
2      4.85480022
3      14.8522997
4      17.7299995
5      18.9387398
6      21.8740005
7      26.6399994
8      31.4729996
9      36.5050011
10     47.2737007
11     56.3747215
12     58.4846992
13     60.9845009
14  Você quis ordenar      13 números

```

#### Tarefa 4. Convergência de séries

Nessa tarefa, objetiva-se calcular o valor de  $\cosh x$  para um valor qualquer de  $x$  real através do método de séries. Isso pode ser feito de duas maneiras, a depender da precisão que se deseja obter na resposta, de forma que foi realizado o mesmo procedimento duas vezes, uma para variáveis reais de simples precisão e outra para as de dupla precisão. O valor retornado pela função `cosh()` do FORTRAN é armazenado na variável "cosh", enquanto "y" vai recebendo os

termos da série que, no infinito, converge pra função cosh; observe que "y" recebe termos até que a diferença dela para o valor de "cosh" seja menor que "eprec", que é a precisão a qual se deseja obter no resultado. Para o caso de simples precisão, essa variável recebe o valor de  $10^{-5}$ , como dito no enunciado, enquanto no caso de dupla precisão foi variando-se o valor de "eprec" até que a precisão de "y" fosse a mesma do resultado retornado pela função dcosh(), chegando-se então no valor de  $10^{-13}$ .

tarefa-4-11212550.f:

```

1      program tarefa4
2
3      double precision xd, y2, cosh2, tmp2
4  c    xr, y1, cosh1, tmp1 são as variáveis análogas de
      simples precisão
5      parameter (eprec1 = 1.e-5, eprec2 = 1.d-13)
6  c    precisão para o cosh em simples e dupla precisão,
      respectivamente
7
8      y1 = 1.e0
9      y2 = 1.d0
10     tmp1 = 1.e0
11     i = 1
12     j = 1
13     tmp2 = 1.d0
14     write(*,*)'Digite o valor x do qual se deseja'
           calcular cosh:'
15     read(*,*)xr
16     xd = dble(xr)
17     cosh1 = cosh(xr)
18     cosh2 = dcosh(xd)
19
20  c    atualiza o valor de y1 até que fique dentro da
      precisão eprec1
21  10   if (abs(y1 - cosh1).gt.eprec1) then
22         tmp1 = tmp1*xr**2/(2*i*(2*i-1))
23         y1 = y1 + tmp1
24         i = i + 1
25         goto 10
26     endif
27
28     write(*,*)'O valor de cosh(x) é (precisão simples):',
           y1
29
30  c    atualiza o valor de y2 até que fique dentro da
      precisão eprec2
31  20   if (abs(y2 - cosh2).gt.eprec2) then
32         tmp2 = tmp2*xd**2/(2*j*(2*j-1))
33         y2 = y2 + tmp2
34         j = j + 1

```

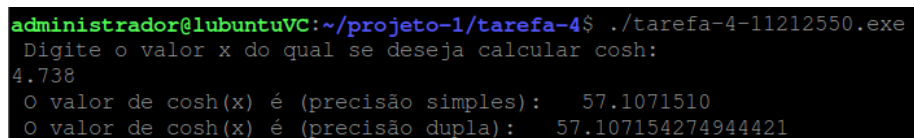
```

35         goto 20
36     endif
37
38     write(*,*)'O valor de cosh(x) é (precisão dupla):', y2
39
40     end

```

Segue em anexo um exemplo de entrada testado e a respectiva saída gerada pelo programa.

Figura 4: Entrada e saída da tarefa-4.



```

administrador@lubuntuVC:~/projeto-1/tarefa-4$ ./tarefa-4-11212550.exe
Digite o valor x do qual se deseja calcular cosh:
4.738
O valor de cosh(x) é (precisão simples): 57.1071510
O valor de cosh(x) é (precisão dupla): 57.107154274944421

```

Fonte: gerado pelo autor

## Tarefa 5. Permutações

Nessa tarefa, a ideia é escrever um programa que lê um arquivo de entrada com todas as permutações de 1, 2, ..., N e a respectiva paridade e gera um arquivo de saída análogo, porém com os números 1, 2, ..., N+1. No código exposto, a entrada esperada é para N = 3, e consequentemente a saída terá permutações de 1, 2, 3 e 4, mas isso pode ser facilmente modificado variando-se o parâmetro "idmax" (e também o limite do "do", como está explicado na linha 8 do código). Em cada iteração, uma linha do arquivo de entrada é armazenada no array "ient" e, dentro de um outro loop aninhado, são geradas as linhas da saída variando-se a posição do novo número N+1 em cada permutação da entrada, lembrando também de trocar a paridade toda vez que a posição do novo número adicionado é modificada. Cada nova permutação gerada é armazenada no array "isai" e então é escrita no arquivo de saída.

tarefa-5-11212550.f:

```

1      program tarefa5
2
3      parameter (idmax = 4)
4      dimension ient(idmax), isai(idmax+1)
5      open(1, file='entrada-5-11212550')
6      open(2, file='1saida-5-11212550')
7
8      c      o arquivo de entrada tem (idmax-1)! linhas
9      do i = 1, 6

```



```

10         read(1,*) (ient(j), j = 1, idmax)
11 c         ipar armazena a paridade da nova permutação
12         ipar = ient(idmax)
13 c         em cada loop, uma permutação nova é formada a
        partir de uma linha da entrada
14         do m = idmax, 1, -1
15 c         é montada a parte anterior à inserção do novo
        termo
16         do k = 1, m-1
17             isai(k) = ient(k)
18         end do
19 c         novo termo inserido
20         isai(m) = idmax
21 c         é montada a parte posterior à inserção do novo
        termo
22         do l = m+1, idmax
23             isai(l) = ient(l-1)
24         end do
25         isai(idmax+1) = ipar
26         ipar = -ipar ! o sinal da próxima permutação é
        invertido
27         write(2,*)(isai(n), n = 1, idmax+1)
28     end do
29 end do
30
31 close(1)
32 close(2)
33
34 end

```

Seguem em anexo um exemplo de arquivo de entrada e a respectiva saída gerada.

entrada-5-11212550:

```

1  1  2  3  1
2  2  1  3 -1
3  1  3  2 -1
4  2  3  1  1
5  3  2  1 -1
6  3  1  2  1

```

1saida-5-11212550:

1	1	2	3	4	1
2	1	2	4	3	-1
3	1	4	2	3	1
4	4	1	2	3	-1
5	2	1	3	4	-1
6	2	1	4	3	1

7	2	4	1	3	-1
8	4	2	1	3	1
9	1	3	2	4	-1
10	1	3	4	2	1
11	1	4	3	2	-1
12	4	1	3	2	1
13	2	3	1	4	1
14	2	3	4	1	-1
15	2	4	3	1	1
16	4	2	3	1	-1
17	3	2	1	4	-1
18	3	2	4	1	1
19	3	4	2	1	-1
20	4	3	2	1	1
21	3	1	2	4	1
22	3	1	4	2	-1
23	3	4	1	2	1
24	4	3	1	2	-1

O arquivo de saída desse teste pode ser usado como entrada de um novo teste, lembrando de, antes de rodar o programa, modificar o valor de "idmax" para 5, e assim a nova saída conterá todas as permutações de 1, 2, 3, 4 e 5, e as respectivas paridades. O mesmo procedimento pode ser feito para a nova saída e assim por diante.

## Tarefa 6. Determinante

Nessa tarefa, o objetivo é escrever um programa que calcula o determinante de uma matriz  $N \times N$  com o auxílio do código da Tarefa 5.. No caso, o código abaixo foi feito para o caso em que  $N = 4$ , mas isso pode ser modificado trocando-se o valor definido no parâmetro "idmax", além do limite do "do" da linha 16, que sempre deve assumir o valor de *idmax*!. O programa recebe duas entradas: o arquivo "1entrada-6-11212550" que contém as permutações de 1, 2, ..., N e as respectivas paridades (esses arquivos podem ser gerados na Tarefa 5.), e o "2entrada-6-11212550" que contém a matriz a qual se deseja calcular o determinante, sendo que a saída é dada na tela do terminal. Observe que o programa calcula esse valor com a seguinte expressão:

$$det = \sum_{perm \pi} sinal(\pi) \prod_{i=1}^N a_{i\pi(i)} \quad (1)$$

De forma que  $\pi(i)$ ,  $i \in 1, 2, \dots, N$  são os termos de uma dada permutação  $\pi$  e  $sinal(\pi)$  é a sua respectiva paridade. Cada termo do produto é armazenado na variável "prod", que por sua vez vai sendo adicionada na variável "det", que fornece o valor final do determinante.

tarefa-6-11212550.f:

```

1      program tarefa6
2
3      parameter (idmax = 4)
4      dimension ivet(idmax+1)
5      real mat(idmax, idmax)
6      det = 0
7      prod = 1
8      open(1, file='1entrada-6-11212550')
9      open(2, file='2entrada-6-11212550')
10
11 c      a matriz que queremos calcular o det é lida de um
      arquivo de entrada
12      do i = 1, idmax
13          read(2,*)(mat(i,1), 1 = 1, idmax)
14      end do
15
16      do j = 1, 24
17          read(1,*) (ivet(k), k = 1, idmax+1)
18 c      prod calcula os termos do somatório que dá o det em
      termo das permutações
19          do m = 1, idmax
20              prod = prod*mat(m, ivet(m))
21          end do
22 c      ivet(idmax+1) armazena a paridade da permutação
      utilizada
23          prod = prod*ivet(idmax+1)
24          det = det + prod
25 c      antes de utilizar uma nova permutação, prod retorna
      ao seu valor inicial
26          prod = 1
27      end do
28
29      write(*,*)'Determinante da matriz real', idmax, 'x',
          idmax, ':',
30      +      det
31      close(1)
32      close(2)
33
34      end

```

Segue em anexo um exemplo de entradas testadas e a respectiva saída gerada pelo programa.

1entrada-6-11212550:

```

1  1 2 3 4 1
2  1 2 4 3 -1
3  1 3 2 4 -1
4  1 3 4 2 1
5  1 4 2 3 1
6  1 4 3 2 -1

```

```

7  2  1  3  4 -1
8  2  1  4  3  1
9  2  3  1  4  1
10 2  3  4  1 -1
11 2  4  1  3 -1
12 2  4  3  1  1
13 3  1  2  4  1
14 3  1  4  2 -1
15 3  2  1  4 -1
16 3  2  4  1  1
17 3  4  1  2  1
18 3  4  2  1 -1
19 4  1  2  3 -1
20 4  1  3  2  1
21 4  2  1  3  1
22 4  2  3  1 -1
23 4  3  1  2 -1
24 4  3  2  1  1

```

2entrada-6-11212550:

```

1  8.434  9.2  1.32  0.232
2  7.34  2.9  4.689  5.86
3  5.743  2.64  9.43  4.3
4  4.55  7.384  5.1  8.485

```

Figura 5: Saída da tarefa-6.

```

administrador@ubuntuVC:~/projeto-1/tarefa-6$ gfortran -o tarefa-6-11212550.exe
e tarefa-6-11212550.f
administrador@ubuntuVC:~/projeto-1/tarefa-6$ ./tarefa-6-11212550.exe
Determinante da matriz real      4 x      4 : -2829.25586

```

Fonte: gerado pelo autor

## Tarefa 7. Sistemas lineares

Nessa tarefa, o objetivo é solucionar um sistema de equações lineares na forma abaixo, por meio da chamada regra de Cramer:

$$AX = Y \quad (2)$$

De sorte que as matrizes reais  $A$   $N \times N$  e  $Y$   $N \times 1$  são dadas em arquivos de entrada distintos; além disso, o valor de  $N$  é lido do terminal, podendo ser 4, 5 ou 6, sendo que, a depender da escolha do usuário, diferentes pares de arquivos serão abertos. Inicialmente, a matriz  $A$  é atribuída à variável "mat\_den" e  $Y$  é

atribuída à "y", e então a subrotina "calc\_det" é chamada, que basicamente é o programa da Tarefa 6. modificado para receber matrizes quadradas de ordem 4, 5 ou 6, para calcular o determinante da matriz A. Já dentro do loop, a variável "mat\_num", que inicialmente recebeu "mat\_den", é modificada para corresponder à matriz cujo determinate entra no cálculo da linha que estamos trabalhando (regra de Cramer). No fim de cada iteração, o programa imprime na tela do terminal o valor de uma linha da matriz coluna X que satisfaz o problema.

tarefa-7-11212550.f:

```

1      program tarefa7
2
3      dimension y(6)
4      real mat_den(6, 6), mat_num(6,6)
5      write(*,*) 'Qual a dimensão da matriz quadrada A?(4, 5 ou 6) '
6      read(*,*) i
7
8      c   verifica quais arquivos de matrizes A e Y serão usados
9      if (i.eq.4) then
10         open(1, file='1entrada-7-11212550')
11         open(2, file='2entrada-7-11212550')
12     elseif (i.eq.5) then
13         open(1, file='3entrada-7-11212550')
14         open(2, file='4entrada-7-11212550')
15     else
16         open(1, file='5entrada-7-11212550')
17         open(2, file='6entrada-7-11212550')
18     endif
19
20     do l = 1, i
21         read(1,*) (mat_den(l, m), m = 1, i)
22         read(2,*) y(l)
23     end do
24
25     mat_num = mat_den
26     write(*,*) 'O vetor coluna X que resolve o sistema é:'
27
28     c   o determinante da matriz do denominador é calculado
29     call calc_det(mat_den, i, det_den)
30
31     c   cada matriz do numerador é montada e tem seu
        determinante calculado
32     do icon = 1, i
33     c   a cada iteração a matriz do numerador é alterada
34         if (icon.eq.1) then
35             do jcont = 1, i
36                 mat_num(jcont, icon) = y(jcont)
37             end do

```

```

38         else
39             do kcont = 1, i
40                 mat_num(kcont, icon - 1) = mat_den(kcont,
41                     icon - 1)
42                 mat_num(kcont, icon) = y(kcont)
43             end do
44         end if
45 c         o determinante da matriz do numerador é calculado
46         call calc_det(mat_num, i, det_num)
47         write(*,*) det_num/det_den
48     end do
49
50     close(1)
51     close(2)
52
53     END
54
55     subroutine calc_det(real_m, n, det)
56         dimension ivet(n+1)
57         dimension real_m(6, 6)
58         det = 0
59         prod = 1
60 c         verifica qual arquivo de permutações será utilizado
61         if (n.eq.4) then
62             open(12, file='1saida-5-11212550')
63             do kcont = 1, 24
64                 read(12,*) (ivet(lcont), lcont = 1, n+1)
65 c                 prod calcula os termos do somatório que dá o
66                 det em termo das permutações
67                 do mcont = 1, n
68                     prod = prod*real_m(mcont, ivet(mcont))
69                 end do
70 c                 ivet(n+1) armazena a paridade da permutação
71                 utilizada
72                 prod = prod*ivet(n+1)
73                 det = det + prod
74 c                 antes de utilizar uma nova permutação, prod
75                 retorna ao seu valor inicial
76                 prod = 1
77             end do
78             close(12)
79         elseif (n.eq.5) then
80 c             mesmo procedimento de n = 4
81             open(13, file='2saida-5-11212550')
82             do kcont = 1, 120
83                 read(13,*) (ivet(lcont), lcont = 1, n+1)
84                 do mcont = 1, n
85                     prod = prod*real_m(mcont, ivet(mcont))
86                 end do

```

```

84         prod = prod*ivet(n+1)
85         det = det + prod
86         prod = 1
87     end do
88     close(13)
89 else
90 c     mesmo procedimento de n = 4
91     open(14, file='3saida-5-11212550')
92     do kcont = 1, 720
93         read(14,*) (ivet(lcont), lcont = 1, n+1)
94         do mcont = 1, n
95             prod = prod*real_m(mcont, ivet(mcont))
96         end do
97         prod = prod*ivet(n+1)
98         det = det + prod
99         prod = 1
100     end do
101     close(14)
102 endif
103
104     return
105 end

```

Segue em anexo um exemplo de entradas que correspondem a  $N = 4$  e a respectiva saída gerada pelo programa. Observe que, na subrotina, o arquivo que contem as permutações de 1, 2, 3 e 4 e a respectiva paridade é aberto, mas foi omitido aqui porque é o mesmo gerado pelo programa da Tarefa 5., tendo sido já exibido naquela seção.

1entrada-7-11212550:

```

1  51.384  73.293  9.27  38.3872
2  84.579  64.743  15.639  76.73
3  45.28  34.3278  84.497  6.49
4  53.389  8.93  98.742  48.348

```

2entrada-7-11212550:

```

1  10.0
2  20.0
3  30.0
4  40.0

```

Figura 6: Saída da tarefa-7.

```
administrador@lubuntuVC:~/projeto-1/tarefa-7$ ./tarefa-7-11212550.exe
Qual a dimensão da matriz quadrada A?(4, 5 ou 6)
4
O vetor coluna X que resolve o sistema é:
0.131428793
-3.96739952E-02
0.293866545
8.93609896E-02
```

Fonte: gerado pelo autor

## Tarefa 8. Volume d-dimensional (aproximação)

Nessa tarefa, o objetivo é calcular o volume de uma esfera d-dimensional de raio unitário por meio de um método de simulação, isto é, com o auxílio da função `rand()`, podemos "sortear" pontos no interior do cubo circunscrito à esfera, sendo a razão do número desses pontos que estão dentro da esfera também e do total é próxima da razão entre o volume da esfera e do cubo para uma quantidade razoável de pontos utilizados. A entrada do programa consiste no valor de `d`, dado no terminal, e a saída é o volume da esfera de raio 1 na respectiva dimensão.

tarefa-8-11212550.f:

```
1      program tarefa8
2
3  c      numero de pontos que serão utilizados na simulação
4      parameter (m = 1000)
5      n_pts = 0
6      write(*,*) 'Em qual dimensão deseja calcular o volume?'
7      read(*,*) id
8
9      do i = 1, m
10 c      r será o quadrado da distância de um ponto à origem
11      r = 0
12      do j = 1, id
13 c      coord sempre será um valor entre -1 e 1
14      coord = 2*rand() - 1
15      r = r + coord**2
16      end do
17 c      verifica se o ponto está no interior da esfera de
18 raio 1
19      if (sqrt(r).le.1.0) n_pts = n_pts + 1
20      end do
```



```

21         write(*,*)'0_volume_da_esfera_unitária_é:',
22             n_pts*2.0**id/m
23     end

```

É possível comparar os valores de volume obtidos pelo programa conforme varia-se a quantidade de pontos utilizados, ou ainda comparar com o valor esperado, calculado a partir da expressão:

$$V_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2} + 1)} R^d \quad (3)$$

Sendo  $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ ,  $\Gamma(1) = 1$ ,  $\Gamma(x+1) = x\Gamma(x)$ .

Tabela 1: Volume calculado pelo programa e o número de pontos utilizados.

Pontos	2D	3D	4D
$10^3$	3,164000	4,303999	5,216000
$10^4$	3,121200	4,219999	4,996799
$10^5$	3,142560	4,204559	4,919680
$10^6$	3,143352	4,192647	4,942207

Fonte: gerado pelo autor

Tabela 2: Volume calculado pelo programa para  $10^6$  pontos e o valor esperado pela expressão (3).

Dimensão	Programa	Esperado
2D	3,143352	3,141592
3D	4,192647	4,188790
4D	4,942207	4,934802

Fonte: gerado pelo autor

Observa-se que todos os resultados tiveram pelo menos uma casa decimal de precisão para  $10^6$  pontos, e que o aumento na quantidade de pontos utilizados vai tornando o resultado mais próximo do esperado.

Seguem em anexo exemplos de entradas e as respectiva saídas gerada pelo programa para  $10^6$  pontos.

Figura 7: Entrada e saída da tarefa-8.

```
administrador@lubuntuVC:~/projeto-1/tarefa-8$ ./tarefa-8-11212550.exe
Em qual dimensão deseja calcular o volume?
2
O volume da esfera unitária é: 3.14335203
administrador@lubuntuVC:~/projeto-1/tarefa-8$ ./tarefa-8-11212550.exe
Em qual dimensão deseja calcular o volume?
3
O volume da esfera unitária é: 4.19264793
administrador@lubuntuVC:~/projeto-1/tarefa-8$ ./tarefa-8-11212550.exe
Em qual dimensão deseja calcular o volume?
4
O volume da esfera unitária é: 4.94220781
```

Fonte: gerado pelo autor

## Tarefa 9. Volume d-dimensional (fórmula)

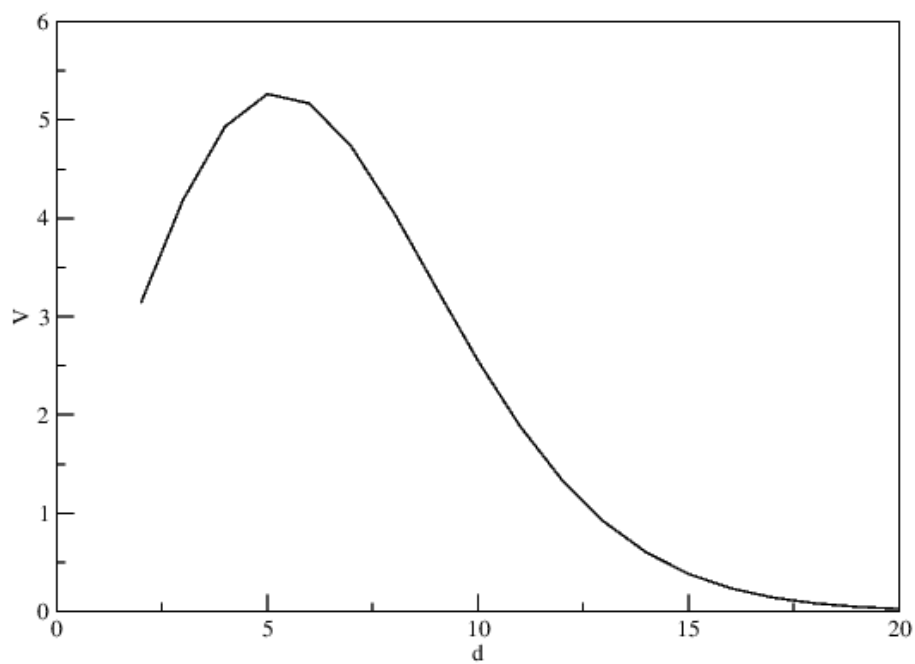
Nessa última tarefa, foi escrito um código que calcula o volume das esferas de raio 1 nas dimensões 2, 3, ..., 20 e escreve em um arquivo de saída cada par dimensão/volume por linha. Esse mesmo arquivo pode ser utilizado para montar um gráfico por meio do software *xmgrace*, o que possibilita uma melhor visualização do comportamento do volume conforme a dimensão aumenta.

tarefa-9-11212550.f:

```
1      program tarefa9
2
3      parameter (pi = acos(-1.0))
4      Vd = pi
5      tmp = 1.0
6      open(1,file='dimensões-esferas')
7
8      do id = 2, 20
9          Vd = Vd*tmp
10     c      escreve a dimensão e o respectivo volume da esfera
11     de raio unitário
12         write(1,*)id, Vd
13         tmp =
14             (sqrt(pi)*id/(id+1))*gamma(id/2.0)/gamma((id+1)/2.0)
15     end do
16
17     close(1)
18
19     end
```

Seguem em anexo os arquivos de saída e o respectivo gráfico.

Figura 8: Volume da esfera de raio 1 por dimensão.



Fonte: gerado pelo autor

dimensões-esferas:

1	2	3.14159274
2	3	4.18879032
3	4	4.93480301
4	5	5.26378965
5	6	5.16771364
6	7	4.72476721
7	8	4.05871344
8	9	3.29851007
9	10	2.55016518
10	11	1.88410485
11	12	1.33526325
12	13	0.910629034
13	14	0.599264860
14	15	0.381443501
15	16	0.235330760
16	17	0.140981197
17	18	8.21459293E-02

18	19	4.66216281E-02
19	20	2.58069057E-02

## Perguntas

a) Nesse caso, a razão é dada por:

$$\frac{V_{cubo}}{V_{esf}} = \frac{1}{\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}} = \frac{\Gamma(\frac{d}{2}+1)}{\pi^{d/2}} \quad (4)$$

E como foi visto na Tarefa 9., o volume da esfera tende a 0 conforme  $d \rightarrow \infty$ , então a razão  $\frac{V_{cubo}}{V_{esf}} \rightarrow \infty$ .

b) Considerando que o número de mols é proporcional ao número de átomos que estão contidos em uma célula, daí a constante de Avogadro seria dada por:

$$N_A = k \frac{1\mu^d}{1A^d} = 10^{4d} k \quad (5)$$

No caso, como é conhecido o valor de  $N_A$  para  $d = 3$ , então:

$$k = 6,02 \cdot 10^{11} mol^{-1}$$

Logo, a constante de Avogadro em uma dimensão d qualquer seria:

$$N_A = 6,02 \cdot 10^{11+4d} mol^{-1} \quad (6)$$