



Universidade de São Paulo
Instituto de Física de São Carlos
7600017 - Introdução à Física Computacional
Docente: Francisco Castilho Alcaraz

Projeto 01

Andressa Colaço
Nº USP: 12610389

São Carlos
2022

Sumário

1	Tarefa 01	2
2	Tarefa 02	2
3	Tarefa 03	5
4	Tarefa 04	7
5	Tarefa 05	10
6	Tarefa 06	14
7	Tarefa 07	16
8	Tarefa 08	20
9	Tarefa 09	24
9.1	Pergunta A	26
9.2	Pergunta B	27

1 Tarefa 01

O primeiro programa do projeto possui como objetivo calcular e retornar a área total e o volume de um torus a partir do fornecimento da medida de seus raios interno (r_1) e externo (r_2).

Seu volume é dado pela fórmula:

$$V = 2\pi^2 r_2 r_1^2 \quad (1)$$

Por sua vez, a área superficial é dada pela equação 02:

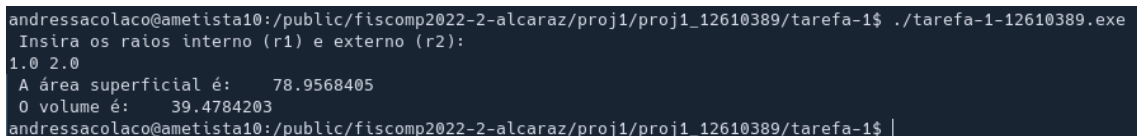
$$S = 4 * \pi^2 r_2 r_1 \quad (2)$$

O código consiste em coletar os dados do usuário, efetuar as operações descritas nas equações 01 e 02 e escrever no terminal os valores referentes aos cálculos:

```
1      PROGRAM torus
2      parameter (pi = acos(-1.e0))
3
4      write(*,*) 'Insira os raios interno (r1) e externo (r2):'
5      read(*,*) r1, r2
6
7      sarea = 4 * pi**2 * r1 * r2
8      vol = 2*(pi**2)*r2*(r1**2)
9
10     write(*,*) 'A área superficial é: ', sarea
11     write(*,*) 'O volume é: ', vol
12
13     end program torus
```

A saída do programa no terminal é:

Figura 1: Saída do código 1.



```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-1$ ./tarefa-1-12610389.exe
Insira os raios interno (r1) e externo (r2):
1.0 2.0
A área superficial é: 78.9568405
O volume é: 39.4784203
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-1$ |
```

2 Tarefa 02

O segundo programa possui como objetivo calcular a área superficial (lateral total) e o volume de um paralelepípedo definido vetorialmente a partir de três vetores fornecidos pelo usuário. As arestas deste sólido são definidas pelos vetores \vec{v}_1 , \vec{v}_2 e $\vec{v}_2 - \vec{v}_3$.

Para compreender como se dá o cálculo destas medidas, consideremos a seguinte figura:

A base do paralelepípedo será definida como estando no plano dos vetores \vec{v}_1 e \vec{v}_2 . Sabemos que a área compreendida no quadrado de arestas \vec{v}_1 e \vec{v}_2 é o módulo do produto vetorial entre eles. Desta maneira, a área da base do paralelepípedo é:

$$A_b = |\vec{v}_1 \times \vec{v}_2| \quad (3)$$

Lembrando que o módulo de um vetor é calculado como:

$$|\vec{A}| = \sqrt{A_x^2 + A_y^2 + A_z^2} \quad (4)$$

Para encontrar o volume, é necessário multiplicar a área da base pela altura do sólido. Esta medida é encontrada através do produto vetorial encontrado anteriormente (vetor ortogonal à base), que indica a direção para a qual devemos considerar a altura. Para tal, devemos fazer o produto escalar do vetor fora do plano da base ($\vec{v}_2 - \vec{v}_3$, que chamaremos de \vec{v}_4 a partir de agora) com o produto vetorial da base e considerar seu módulo:

$$h = |(\vec{v}_1 \times \vec{v}_2) \cdot \vec{v}_4| \quad (5)$$

O volume então será:

$$V = h * |\vec{A}_b| \quad (6)$$

As áreas laterais são encontradas da mesma forma que a área da base: considerando o módulo do produto vetorial entre os vetores que formam suas arestas. Assim, a área lateral total será simplesmente a soma de todos esses valores, cada um multiplicado por 2 para considerar as duas faces com mesma área:

$$A_s = 2 * (A_b + A_{l1} + A_{l2}) \quad (7)$$

A implementação do código é a a seguir:

```
1  PROGRAM prisma
2
3  dimension v1(1:3), v2(1:3), v3(1:3), v4(1:3)
4  dimension pvet12(1:3), pvet14(1:3), pvet24(1:3)
5
6  write(*,*) 'Insira o primeiro vetor:'
7  read(*,*) v1(1), v1(2), v1(3)
8
9  write(*,*) 'Insira o segundo vetor:'
10 read(*,*) v2(1), v2(2), v2(3)
11
```

```

12     write(*,*) 'Insira o terceiro vetor:'
13     read(*,*) v3(1), v3(2), v3(3)
14
15     do 10 i = 1, 3
16         v4(i) = v2(i)-v3(i)
17 10    continue
18
19     !area da base:
20     call calc_produto_vet(v1, v2, pvet12)
21
22     area_base = sqrt(pvet12(1)**2 + pvet12(2)**2 + pvet12(3)**2)
23     alt = abs(pvet12(1)*v4(1) + pvet12(2)*v4(2) + pvet12(3)*v4(3))
24     volume = area_base*alt
25
26     !calculando a área lateral:
27     call calc_produto_vet(v1, v4, pvet14)
28     call calc_produto_vet(v2, v4, pvet24)
29
30     area_lat1 = sqrt(pvet14(1)**2 + pvet14(2)**2 + pvet14(3)**2)
31     area_lat2 = sqrt(pvet24(1)**2 + pvet24(2)**2 + pvet24(3)**2)
32
33     area_superficial = 2*(area_base + area_lat1 + area_lat2)
34
35     write(*,*) 'Área lateral total:', area_superficial
36     write(*,*) 'Volume: ', volume
37
38     end program
39
40     subroutine calc_produto_vet(a, b, res)
41     dimension a(1:3), b(1:3), res(1:3)
42
43     res(1) = a(2)*b(3) - a(3)*b(2)
44     res(2) = a(3)*b(1) - a(1)*b(3)
45     res(3) = a(1)*b(2) - a(2)*b(1)
46
47     return
48     end

```

São definidos três vetores para a entrada de dados e um quarto para a operação entre \vec{v}_2 e \vec{v}_3 . Também são definidos três auxiliares para guardar o produto vetorial necessário

para o cálculo das áreas. É requisitado para o usuário inserir os vetores; após, calcula-se o vetor resultante mencionado.

Como o produto vetorial é requisitado várias vezes, é implementada uma subrotina para calculá-lo. É necessário passar a referência do vetor onde será guardado o resultado, uma vez que a implementação de subrotinas depende da passagem dos parâmetros por referência (ou seja, passa a variável original para ser alterada).

Após terminados os cálculos mencionados nas equações, imprime-se na tela os dois valores requisitados.

Figura 2: Saída do código 2.

```
andressacolaco@metista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-2$ f77 tarefa-2-12610389.f -o tarefa-2-12610389.exe
andressacolaco@metista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-2$ ./tarefa-2-12610389.exe
Insira o primeiro vetor:
1.0 0.0 0.0
Insira o segundo vetor:
0.0 1.0 1.0
Insira o terceiro vetor:
0.0 0.0 1.0
Área lateral total: 6.82842731
Volume: 1.41421354
andressacolaco@metista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-2$ |
```

3 Tarefa 03

O objetivo da tarefa 03 é construir um programa que ordene os M menores números de uma lista com N números, dada em um arquivo de entrada.

A implementação começa com essa lista de números sendo armazenada em um vetor de números reais. Esse vetor possuirá um tamanho máximo de 1000 números, mas pode-se ajustar esse limite conforme a demanda do algoritmo. Assim, a lista a ser ordenada terá um limite de 1000 entradas.

É utilizado um loop para ler os dados contidos no arquivo de entrada. a variável i , iterada na leitura dos dados, pode ser usada para retornar o tamanho efetivo da lista que será ordenada. Porém, é importante notar que na última iteração, que detecta o fim do arquivo, a variável é incrementada mas não lê nenhum número novo. Portanto, o tamanho n da lista é $n = i - 1$. É então chamada uma subrotina que recebe o número de elementos a serem ordenados, o tamanho da lista e o vetor que contém a lista.

A lógica do algoritmo da subrotina é varrer a lista de trás para frente, trocando o elemento com índice maior com o elemento no índice vizinho inferior caso ele seja menor: isso resulta que, ao final de uma iteração, o menor elemento é carregado para o começo da lista.

Uma vez que só precisamos ordenar m elementos, não é necessário fazer esse processo até a lista inteira estar ordenada: fazendo-o m vezes garantimos que os m primeiros números da nova lista estão ordenados. Por conta disso, o loop mais externo da subrotina só realiza o procedimento m vezes.

Por fim, escreve-se em um arquivo de saída a quantidade m ordenada, na primeira linha, e os m primeiros elementos que foram ordenados.

O código implementado é o seguinte:

```
1      PROGRAM lista_num
2
3      dimension vetor(1:1000)
4
5      write(*,*) 'Digite m: '
6      read(*,*) m
7
8      open(10, FILE='entrada-3-12610389.dat')
9      open(20, FILE='saida-3-12610389.dat')
10
11     do i=1, 1000
12         read(10, *, end = 89) vetor(i)
13     end do
14 89    continue
15
16     n = i - 1 !i faz mais uma iteração até achar o EOF
17
18     call ordenar(m, n, vetor)
19
20     write(20,*) m
21     do k=1, m
22         write(20,*) vetor(k)
23     enddo
24
25     close(10)
26     close(20)
27
28     end program lista_num
29
30
31     subroutine ordenar(m, itam, vetor)
32     dimension vetor(1:1000)
33
34     do i = m, 1, -1
35         do j = itam, 2, -1
36             if(vetor(j-1) .gt. vetor(j)) then
37                 aux = vetor(j-1)
38                 vetor(j-1) = vetor(j)
```

```

39         vetor(j) = aux
40     endif
41 enddo
42 enddo
43
44 return
45 end

```

4 Tarefa 04

A tarefa 4 tem como objetivo calcular com precisão $eprec = 10^{-5}$ o valor de $\cos(x)$ para um $x \in \mathbb{R}$ com a série:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \quad (8)$$

que nada mais é do que a aproximação dada pela série de Taylor da função cosseno em torno de $x = 0$, cuja forma geral é:

$$\cos(x) = \sum_0^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad (9)$$

Para calculá-la com a precisão requerida, foi construída uma estrutura de repetição que será executada enquanto os termos da série forem maiores do que a precisão almejada. A estrutura de repetição é construída por variáveis auxiliares: o i representa a variável $2n$ na série, por isso é incrementado de 2 em 2; a variável $ifat$ armazena o valor do fatorial de i ; a variável soma armazena o valor total da série até então; e o $termo_{novo}$ é o valor do termo que será adicionado à série.

O loop consiste em adicionar o termo previamente calculado à soma, se este for maior que a precisão (sendo que começamos o loop a partir do segundo termo, $-x^2/2!$), incrementar o fatorial para o valor atual de $2n$ e calcular o novo termo a partir da fórmula dada na equação acima. Por fim, a variável contadora é incrementada e é feita uma nova comparação entre o termo calculado na iteração e a precisão: caso ele for menor que a precisão, não é adicionado à soma da série. Após isso, são impressos na tela os valores do cosseno calculados e o valor do erro entre a função intrínseca do fortran e a nossa aproximação.

O mesmo procedimento é repetido para a dupla precisão ($deprec = 1e - 16$), onde as variáveis são adaptadas para a nova precisão, porém a lógica é idêntica ao que foi realizado no caso anterior.

É importante ressaltar um aspecto sobre este programa: para atingir precisões pequenas, calculamos muitos termos. A cada termo novo, o fatorial é multiplicado pelos dois números seguintes, crescendo de forma muito rápida e, dessa maneira, estourando o limite

de precisão das variáveis. Para aumentar o intervalo de valores que podem ser utilizados em x , declarou-se os dois fatoriais utilizados nas funções de simples e dupla precisão como sendo inteiros de dupla precisão. Mesmo assim, ainda existe um limite para a variável inteira de dupla precisão e, desta forma, um limite de valores que podem assumir o lugar de x . De forma manual, verificou-se que esses valores são aproximadamente 5 e 1.4 para o cálculo em simples e dupla precisão, respectivamente. Esta última foi colocada em uma estrutura de controle para, caso seja inserido um número entre 1.4 e 5.0, seja possível determinar o valor com precisão simples e o programa não quebre ao chegar na precisão dupla.

O código implementado segue abaixo:

```
1      PROGRAM aproximacao_cos
2
3      real*8 :: res_dupla, deprec, dx, dsoma, dtermo_novo, derro
4      integer*8 :: ifat, idfat
5
6      eprec = 1e-5
7      deprec = 1e-16
8
9      write(*,*) 'Insira o valor de x:'
10     read(*,*) x
11
12     c      Cálculo direto
13     res_simples = cos(x)
14     write(*,*) 'Valor da função intrínseca:', res_simples
15
16     c      Cálculo por aproximação
17     i = 2
18     ifat = 1
19     soma = 0e0
20     termo_novo = 1e0
21
22     10    if(abs(termo_novo)>eprec) then
23         soma = soma + termo_novo
24         ifat = ifat*(i-1)*i
25         termo_novo = (-1.0)**(i/2) * (x**i/ifat)
26         i = i+2
27         goto 10
28     endif
29
```

```

30     erro = abs(res_simples-soma)
31     write(*,*) 'Valor da aproximação:', soma
32     write(*,*) 'Valor do erro:', erro
33
34 c     Dupla precisão -----
35
36     if (abs(x) >= 1.4) then
37         write(*,*) 'Os valores precisam ser mais próximos de 0 para
38 $ o cálculo com dupla precisão.'
39
40     else
41         dx = x
42
43 c     Cálculo direto
44         res_dupla = dcos(dx)
45         write(*,*) 'Valor intrínseco (dupla precisão):', res_dupla
46
47 c     Cálculo por aproximação
48         j = 2
49         idfat = 1
50         dsoma = 0d0
51         dtermo_novo = 1d0
52
53 20     if(abs(dtermo_novo)>deprec) then
54         dsoma = dsoma + dtermo_novo
55         idfat = idfat*(j-1)*j
56         dtermo_novo = (-1.0)**(j/2) * (dx**j/idfat)
57         j = j+2
58         goto 20
59     endif
60
61     derro = abs(res_dupla-dsoma)
62     write(*,*) 'Valor da aproximação (dupla precisão):', dsoma
63     write(*,*) 'Valor do erro (dupla precisão):', derro
64
65     endif
66     end program aproximacao_cos

```

A saída do programa no terminal é:

Figura 3: Saída do código 4.

```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/taref
a-4$ ./tarefa-4-12610389.exe
  Insira o valor de x:
0.63
Valor da função intrínseca: 0.808027506
Valor da aproximação: 0.808026910
Valor do erro: 5.96046448E-07
Valor intrínseco (dupla precisão): 0.8080275112141302
Valor da aproximação (dupla precisão): 0.8080275112141302
Valor do erro (dupla precisão): 0.0000000000000000
```

5 Tarefa 05

O objetivo da tarefa 05 é gerar um arquivo que contenha as permutações de $N+1$ elementos e suas respectivas paridades através do fornecimento de um arquivo com as permutações de N elementos.

Para compreender o problema, temos que uma permutação de um conjunto é uma lista de n elementos onde cada um aparece exatamente uma vez. Uma lista de n elementos possui $n!$ permutações possíveis.

Pode-se pensar em cada permutação como uma troca de posição entre os elementos do conjunto (uma transposição). Essas trocas de posição (que consideramos apenas o "estado" inicial, ordenado, e um final, que é a permutação em si) podem ser realizadas de muitas maneiras, porém o número de transposições sempre será composto por um número de paridade igual. Dentre a lista de possíveis permutações, é possível notar que sempre haverá quantidades iguais com paridade par (que denotaremos com o valor 1) e ímpar (que denotaremos com o valor -1). Isto dá a intuição inicial para a resolução do problema.

Existem várias formas de gerar permutações de n elementos e verificar sua paridade computacionalmente. Aqui, se inicia de uma tabela base, dada em um arquivo de entrada. Sabemos que as permutações de $n+1$ elementos serão, em número, $n+1$ vezes a quantidade de permutações para n elementos. Portanto, pode-se pensar que a cada permutação de n elementos é possível adicionar um elemento extra para produzir as de $n+1$. No entanto, esse procedimento pode ser feito de $n+1$ maneiras: o novo elemento pode estar no final, no início ou em qualquer outra posição da permutação base e todas essas possibilidades são novas permutações. É notável que a paridade da permutação base pode ser alterada dependendo da posição do novo elemento.

O algoritmo implementado utiliza este conceito, trabalhando com grupos formados pelas permutações base (de n). É adicionado o elemento $n+1$ primeiramente no final (onde não troca de posição com ninguém, logo a paridade não é alterada), em sequência na posição final menos uma, e prossegue-se até o novo elemento ser adicionado no início das permutações.

Abaixo se encontra o código implementado. Cabe notar que, por conta das especifici-

dades do programa, o usuário deve definir as matrizes que recebem os dados de entrada. A primeira tem dimensões $(n!, n + 1)$ e a segunda é a sua transposta (é necessária pois o fortran armazena dados por coluna).

```

1      PROGRAM paridade_permutacao
2
3      C      Esse programa recebe as permutações permitidas para n elementos e
4      C      suas respectivas paridades e retorna as permutações e as
5      C      paridades de um conjunto com n+1 elementos. A dimensão
6      C      das matrizes M e M_aux deve ser ajustada manualmente alterando os
7      C      parâmetros.
8
9      parameter (N=3)
10     parameter(Nfat = 6)
11     dimension M(Nfat,N+1), M_aux(N+1,Nfat), MN(1000,1000)
12
13     m_lin = Nfat
14     m_col = N
15
16     mn_lin = Nfat*(N+1)
17     mn_col = N+1 !Matriz de permutações SEM indicador de paridade
18
19     open(10, FILE = 'entrada-5-N3-12610389.dat')
20     open(20, FILE = 'saida-5-N3-12610389.dat')
21
22     read(10, *, end = 99) M_aux
23     M = transpose(M_aux)
24 99    continue
25
26     do i = 0, N
27         isinal = i
28         ipos = mn_col - i
29         i_mn = m_lin*i
30
31         do j = 1, ipos-1
32             do i_aux = 1, m_lin
33                 MN(i_mn+i_aux, j) = M(i_aux, j)
34             enddo
35         enddo
36

```

```

37         do i_aux = 1, m_lin
38             MN(i_mn+i_aux, ipos) = N+1
39         enddo
40
41         do j = ipos+1, mn_col
42             do i_aux = 1, m_lin
43                 MN(i_mn+i_aux, j) = M(i_aux, j-1)
44             enddo
45         enddo
46
47         do i_aux = 1, m_lin
48             MN(i_mn+i_aux, mn_col+1) = M(i_aux, mn_col)*(-1)**isinal
49         enddo
50
51     enddo
52
53     do i=1, mn_lin
54         write(20,*) (MN(i,j), j=1, mn_col+1)
55     enddo
56
57     close(10)
58     close(20)
59
60     end program paridade_permutacao

```

Os loops implementados servem para seguir a seguinte lógica: tendo a matriz de permutações base, consideramos como um grupo permutações base que recebem o elemento novo na mesma posição. O contador mais externo serve para indicar em qual grupo, ou melhor, em qual posição estamos. A variável *isinal* guarda a informação sobre se o sinal da paridade muda ou não: começando da posição final, com $i = 0$, a cada iteração troca-se em uma posição o elemento e, assim, a paridade deve inverter. Isso será efetivado logo antes do final da iteração, com $(-1)^{isinal}$ dizendo se a paridade muda ou não (não muda caso *isinal* for par). A variável i_{mn} é um ajuste de posição dos grupos no conjunto de dados que será gerado.

Após, são copiadas as colunas até logo antes da posição do elemento para a matriz que armazena as novas permutações (MN). Para-se o loop. É inserido o novo elemento através de um novo loop. Por fim, copiam-se as colunas restantes no novo local. Por fim, a informação da paridade é adicionada. O processo continua até que todas as permutações sejam geradas.

Quando os procedimentos terminam, os dados são transferidos para um arquivo novo.

Durante as iterações, é bastante utilizada a informação sobre as dimensões dos dados, que dependem de n . Para isso, é definida uma função fatorial iterativa, que calcula e retorna os valores correspondentes.

No diretório entregue há 4 arquivos de entrada com informações das permutações de 2, 3, 4 e 5 elementos que geram 4 arquivos de saída com as permutações de 3, 4, 5 e 6 elementos, respectivamente. Eles serão utilizados nas próximas tarefas, portanto também se encontram nos diretórios das tarefas 6 e 7.

Exemplo de entrada:

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

Exemplo de saída produzida pelo código:

1	1	2	3	4	1
2	2	1	3	4	-1
3	1	3	2	4	-1
4	2	3	1	4	1
5	3	1	2	4	1
6	3	2	1	4	-1
7	1	2	4	3	-1
8	2	1	4	3	1
9	1	3	4	2	1
10	2	3	4	1	-1
11	3	1	4	2	-1
12	3	2	4	1	1
13	1	4	2	3	1
14	2	4	1	3	-1
15	1	4	3	2	-1
16	2	4	3	1	1
17	3	4	1	2	1
18	3	4	2	1	-1
19	4	1	2	3	-1
20	4	2	1	3	1
21	4	1	3	2	1
22	4	2	3	1	-1
23	4	3	1	2	-1
24	4	3	2	1	1

6 Tarefa 06

As paridades das permutações mencionadas na tarefa anterior refletem propriedades muito interessantes observadas em matrizes e na resolução de sistemas lineares. A respeito da tarefa 06, temos que as permutações e suas paridades podem ser utilizadas para calcular determinantes.

O determinante de uma matriz A de dimensão $n \times n$ pode ser definido como a soma de $n!$ termos, sendo um para cada permutação (que vamos denotar por σ) de um conjunto $\{1, 2, \dots, n\}$.

Os termos são produzidos da seguinte maneira:

$$sgn\sigma * A_{1\sigma_1} * \dots * A_{n\sigma_n} \quad (10)$$

onde i é pego de cada linha e σ_i é pego de acordo com a coluna. A paridade é descrita no elemento $sgn\sigma$. Como o determinante é a soma desses termos, sua fórmula será:

$$|A| = \sum_{\sigma} sgn\sigma * A_{1\sigma_1} * \dots * A_{n\sigma_n} \quad (11)$$

Para melhor visualização, é útil montar uma tabela com as permutações e sinais das paridades; de fato, é dessa maneira com a qual é implementada o algoritmo. Para $n=4$, temos:

Figura 4: Tabela de permutações para $N=4$.

1234	+	2134	-	3124	+	4123	-
1243	-	2143	+	3142	-	4132	+
1324	-	2314	+	3214	-	4213	+
1342	+	2341	-	3241	+	4231	-
1423	+	2413	-	3412	+	4312	-
1432	-	2431	+	3421	-	4321	+

Nesse caso, o determinante é calculado como:

$$\begin{aligned}
 &+a_{11}a_{22}a_{33}a_{44} - a_{11}a_{22}a_{34}a_{43} \\
 &+a_{11}a_{23}a_{32}a_{44} - a_{11}a_{23}a_{34}a_{42} \\
 &+a_{11}a_{23}a_{32}a_{43} - a_{11}a_{24}a_{33}a_{42} + \dots
 \end{aligned}$$

onde os 18 termos restantes são omitidos aqui considerando seu padrão de escrita.

A matriz que será fornecida deve ter as dimensões corretas para concordar com a quantidade de elementos no conjunto de permutações. Portanto, considerando o programa da tarefa anterior, que gerou os dados utilizados nesta, devemos usar a saída adequada. No diretório entregue, há mais arquivos exemplo mas pode-se utilizar o programa 5 para

outros valores e então calcular o determinante. O programa está configurado para a entrada padrão 'entrada-6-12610389.dat', com $N=4$ e pode-se alterar somente esse arquivo para utilizar outros valores de N .

A lógica utilizada no código é a seguinte: primeiro, armazena-se a matriz de permutações na matriz MQ ; em sequência, coleta-se a matriz do usuário, que deve ser quadrada de ordem N . O determinante é inicializado em 0 e então começa-se a somar os termos descritos acima. Entra-se em um loop que indica em qual linha das matrizes estamos e se inicializa a variável *soma* com o valor 1, pois é composta de multiplicações. Essa variável é cada termo da soma ($a_{11}a_{23}a_{32}a_{44}$, por exemplo): a cada iteração, considera-se uma nova linha do arquivo de permutações, ou seja, de acordo com a tabela da Figura 4, é como se estivessemos em um de seus quadrados. Em sequência, entra-se em um loop que percorre as colunas da matriz, ou seja, pega elemento por elemento. Porém, o termo é construído da seguinte forma: os primeiros índices de cada a estão em sequência e vão de 1 até N (por isso recebem o iterador j). Já, o segundo índice é um elemento de uma linha da matriz de permutação. Percorrendo todas as colunas, temos o termo construído. O último passo é adicionar o sinal da paridade, que está na coluna $N+1$ da matriz de permutações e adicionar o termo à variável geral do determinante.

Por fim, é impresso na tela o valor do determinante.

O código se encontra a seguir:

```
1      PROGRAM determinante_matriz
2      c      Este programa utiliza uma matriz de permutações de N para
3      c      calcular o determinante. No diretório estão contidos os arquivos
4      c      gerados para N=3, N=4, N=5 e N=6, pegos das saidas da tarefa 5.
5      c      Para a utilização de N=4 ou outro N
6      c      qualquer é necessário ajustar os parâmetros e fornecer o arquivo
7      c      que pode ser gerado com o outro código.
8
9      parameter(N=4)
10     parameter(Nfat=24)
11     dimension RM(N,N), MQ(Nfat, N+1), MQ_aux(N+1, Nfat)
12
13     mq_lin = Nfat
14     mq_col = N
15     indice_par = N+1
16
17     open(10, FILE='entrada-6-12610389.dat')
18
19     read(10, *, end = 99) MQ_aux
20     MQ = transpose(MQ_aux)
```



```

21  99  continue
22
23  write(*,*) 'Configurado pra N= ', N
24  write(*,*) 'Forneça a matriz: '
25  do i = 1, N
26      read(*,*) (RM(i, j), j=1, N)
27  end do
28
29  det = 0
30
31  do i = 1, mq_lin
32      soma = 1
33      do j = 1, mq_col
34          soma = soma * RM(j, MQ(i,j))
35      end do
36      det = det + MQ(i, indice_par) * soma
37  end do
38
39  write(*,*) 'O determinante vale: ', det
40
41  close(10)
42
43  end program determinante_matriz

```

A saída do programa é:

Figura 5: Saída do código 6 para N=4.



```

andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-6$
./tarefa-6-12610389.exe
Configurado para N=          4
Forneça a matriz:
1 1 0 1
2 0 -1 0
0 2 2 -3
0 1 0 -1
O determinante vale:      9.00000000

```

7 Tarefa 07

O objetivo da tarefa 07 é utilizar os dois últimos programas para calcular a solução de um sistema linear de ordem N , que possui a forma

$$AX = Y \tag{12}$$

sendo A a matriz dos coeficientes das incógnitas das equações, X um vetor com as incógnitas e Y o vetor de igualdades das equações.

Aqui, assumimos uma matriz de números reais de ordem $N \times N$, com determinante diferente de 0. Y é um vetor real de tamanho N e o conjunto de equações possui apenas uma solução.

A partir desses pressupostos, pode-se utilizar a Regra de Cramer para resolver o sistema - teremos que as N soluções do sistema, contidas no vetor x , terão a forma:

$$x_i = \frac{\det(A_i)}{\det(A)}, i = 1, 2, \dots, N \quad (13)$$

onde x_i é um elemento na i -ésima posição do conjunto solução e $\det(A_i)$ é o determinante da matriz A com sua i -ésima coluna trocada pelo vetor Y .

A implementação do código é feita utilizando o programa 6 dentro da subrotina *calcular_det*, já explicado no item anterior. A função *ifat*, que calcula o fatorial, também é trazida para auxiliar na definição das variáveis. Além disso, a subrotina que calcula o determinante precisa que lhe seja passado o arquivo correspondente gerado na tarefa 05, que aqui é feito de forma manual.

A parte de resolver o sistema linear em si começa com a inserção da matriz de coeficientes por parte do usuário (o valor de N é dado no começo do programa e pode ser alterado ali, uma vez que desta forma pode-se definir os tamanhos das matrizes mais facilmente). O determinante da matriz de coeficientes original, RM_{coef} é inicializado em 0 e chama-se a função para calcular e armazenar seu determinante na variável \det_{rm} .

Em sequência, inicia-se um laço para determinar o valor de cada incógnita do vetor X : em primeiro momento, a coluna de índice i , que será trocada, é armazenada temporariamente em um vetor auxiliar *aux* e, após, essa coluna recebe os valores de X . Logo em sequência, é chamada a subrotina que calcula o determinante, cujo valor é armazenado no i -ésimo índice de um vetor *det_coord*, que armazenará, em sequência, todos os valores que serão utilizado no cálculo das soluções.

Após o cálculo, entra-se em um laço para desfazer a troca inicialmente feita e passar a matriz original de coeficientes para a próxima iteração. Por fim, a solução é calculada dividindo o valor do determinante da matriz com sua i -ésima coluna trocada pelo determinante da matriz original.

Ao terminar de percorrer o tamanho N , o conjunto solução é impresso na tela e encerra-se o programa. Ele se encontra a seguir:

```

1      PROGRAM sistema_linear
2
3  c    É necessário alterar o arquivo de entrada para o arquivo com as
4  c    permutações de N ao utilizar.
5
6      parameter(N=5)
```

```

7      dimension RM_coef(N, N), vet(N), det_coord(N), aux(N), sol(N)
8
9      write(*,*) 'Configurado pra N= ', N
10     write(*,*) 'Insira a matriz de coeficientes:'
11
12     do i=1, N
13         read(*,*) (RM_coef(i,j), j=1, N)
14     enddo
15
16     det_rm = 0
17     Nfat = ifat(N)
18     call calcular_det(N, Nfat, RM_coef, det_rm)
19
20     write(*,*) 'Insira o vetor de resultados'
21     read(*,*) (vet(i), i=1, N)
22
23     do k=1, N
24         do i=1, N
25             aux(i) = RM_coef(i, k)
26             RM_coef(i,k) = vet(i)
27         enddo
28
29         call calcular_det(N, Nfat, RM_coef, det_coord(k))
30
31         do i=1, N
32             RM_coef(i,k) = aux(i)
33         enddo
34
35         sol(k) = det_coord(k)/det_rm
36     enddo
37
38
39     write(*,*) 'Conjunto solução:', (sol(k), k=1, N)
40
41     end program sistema_linear
42
43     subroutine calcular_det(N, Nfat, RM, det)
44     dimension RM(N, N), MQ(Nfat, N+1), MQ_aux(N+1,Nfat)
45     mq_lin = ifat(N)
46     mq_col = N

```

```

47     indice_par = N+1
48
49     open(10, FILE='entrada-7-12610389.dat')
50
51     read(10, *, end = 99) MQ_aux
52     MQ = transpose(MQ_aux)
53 99    continue
54
55     det = 0e0
56
57     do i = 1, mq_lin
58         soma = 1e0
59         do j = 1, mq_col
60             soma = soma * RM(j, MQ(i,j))
61         end do
62         det = det + MQ(i, indice_par) * soma
63     end do
64
65
66     close(10)
67     return
68 end
69
70 integer function ifat(n)
71 ifat = 1
72
73 do i = 1, n
74     ifat = ifat * i
75 end do
76
77 return
78 end
79

```

Para $N=4$, temos:

Figura 6: Saída do código 7 para $N=4$.

```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-7$  
./tarefa-7-12610389.exe  
N configurado:      4  
Insira a matriz de coeficientes:  
1 1 0 1  
2 0 -1 0  
0 2 2 -3  
0 1 0 -1  
Insira o vetor de resultados  
6 -2 3 -1  
Conjunto solução:  1.00000000    2.00000000    4.00000000    3.00000000
```

Para $N=5$, temos:

Figura 7: Saída do código 7 para $N=5$.

```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/taref  
a-7$ ./tarefa-7-12610389.exe  
N configurado:      5  
Insira a matriz de coeficientes:  
1 1 1 1 0  
0 1 0 -2 0  
-9 0 1 0 1  
0 -1 0 0 3  
0 1 -2 3 -1  
Insira o vetor de resultados  
15 0 20 -8 -19  
Conjunto solução:  -1.00000000    2.00000000    13.00000000    1.00000000  
-2.00000000
```

Para $N=6$, temos:

Figura 8: Saída do código 7 para $N=6$.

```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-7$  
./tarefa-7-12610389.exe  
N configurado:      6  
Insira a matriz de coeficientes:  
1 -2 1 0 -2 1  
-2 0 3 -4 0 0  
0 1 0 2 -1 2  
1 0 0 0 3 -1  
0 -1 2 0 0 -3  
0 0 0 1 -1 0  
Insira o vetor de resultados  
-4 -9 17 10 -14 -1  
Conjunto solução:  1.00000000    2.00000000    3.00000000    4.00000000  
5.00000000    6.00000000  
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-7$
```

8 Tarefa 08

Na tarefa 08, é buscado aproximar os volumes de esferas em n dimensões. Para tal, considera-se uma região limitada de um plano com n dimensões que contém a esfera. São gerados n números aleatórios entre 0 e 1, que indicam coordenadas espaciais de pontos nesse plano. Se a distância entre eles e a origem é menor do que a área limitada pela

esfera, conta-se esses pontos como "acertos" ou, simplesmente, pontos contidos na esfera. Considerando os pontos uniformemente gerados, espera-se que a razão entre os pontos que verificamos ser contidos na esfera e os pontos totais que foram gerados nos dê uma boa aproximação do volume desta conforme aumenta-se o número de pontos gerados.

Esta relação se resumiria na razão se estivéssemos considerando o volume total da esfera na simulação; no entanto, consideramos-a na origem, possuindo raio unitário, e consideramos um "cubo" de raio também unitário a envolvendo. Assim, em duas dimensões, nossa simulação dá um quarto do volume total; em três, um oitavo; e assim por diante. Portanto, a relação a ser considerada é:

$$V_d = \frac{n_{acertos}}{n_{total}} * 2^d \quad (14)$$

onde d denota o número de dimensões.

O algoritmo implementado para esse método, contido na função *volume* inicia com um vetor que será utilizado para guardar as coordenadas de cada ponto. O número de acertos é inicializado em 0 e então entra-se em um loop cujo objetivo é gerar M pontos (a amostragem), sendo que para cada um deles são geradas d coordenadas. A distância do ponto à origem é calculada na variável raio, que tem sua norma generalizada em um loop, contemplando as n coordenadas. Por fim, a variável contadora de acertos aumenta um ponto caso o raio esteja na região da esfera.

Por ser um exercício para verificar um método de aproximação, foram logo definidas 3 dimensões e 3 valores de amostragem para verificar a acurácia alcançada. Além disso, é implementada a função que dá o valor exato do volume:

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)} R^d \quad (15)$$

Sendo que as esferas e cubos considerados nos códigos possuem raio unitário, o que permite omitir esse termo dos cálculos. Tal função é implementada da seguinte maneira: definindo-se a função *vol_exato*, que recebe o número de dimensões, primeiro é feito o cálculo da função gamma e, ao final, utiliza-se a expressão acima para encontrar o volume e retorná-lo ao código. Para calcular gamma, utiliza-se as informações conhecidas sobre ela: $\Gamma(1/2) = \sqrt{\pi}$, $\Gamma(1) = 1$ e $\Gamma(x + 1) = x\Gamma(x)$. Sabemos que, se x for diferente de 1 ou $1/2$, pode-se sempre escrever que $\Gamma(x) = (x - 1)\Gamma(x - 1)$ até que cheguemos em um dos valores com a função definida. Portanto, começa-se a função definindo π , útil para o final do cálculo, *aux*, uma variável auxiliar que armazena o último valor de x antes deste ser reduzido e *arg*, que carrega o argumento da função gamma que vamos calcular.

É definido um loop para controlar os cálculos, que devem parar caso $x = 1/2$ ou $x = 1$. Entrando no loop, uma subrotina auxiliar controla o valor da variável auxiliar para os primeiros casos da lista, $x = 1/2$ e $x = 1$, os quais não encontramos resposta fazendo $\Gamma(x) = (x - 1)\Gamma(x - 1)$. Caso o argumento ainda não seja um desses valores, temos a

execução da condição onde a função gamma é multiplicada por $(x-1)$, tal como mencionado acima nas relações obedecidas por esses valores. Saindo dessa condicional, o argumento é decrementado em 1, para o próximo termo ser obtido, e Γ é multiplicada por aux : caso seja a última iteração, aux terá os valores necessários para a base da função; caso não seja, aux ainda vale 1, portanto o valor de gamma não muda.

Terminando esses processos, a função retorna o valor exato do volume de uma esfera de n -dimensões com raio unitário.

Para as dimensões 2, 3 e 4 são feitas simulações com três valores de M para cada. O erro da cada uma é dado com a diferença do valor do volume calculado e do volume aproximado com os pontos aleatórios. Esses resultados são impressos na tela ao usuário, na forma de uma tabela.

A implementação é a seguir:

```

1      PROGRAM vol_hipersfera
2      dimension r(100), M(3), idim(3)
3
4      write(*,*) '          DIM          M          VOLUME          ERRO'
5      do i = 1, 3
6          idim(i) = 1+i
7
8          volume_exato = vol_exato(idim(i))
9
10         do j=1, 3
11             M(j) = 100**j
12             volume_aprox = volume_mc(idim(i), M(j))
13             erro = abs(volume_exato-volume_aprox)
14             write(*,*) idim(i), M(j), volume_aprox, erro
15         enddo
16     enddo
17
18     end program vol_hipersfera
19
20
21     function volume_mc(idim, M)
22         dimension r(100)
23
24         hit = 0.0
25         do i= 1, M
26
27             do j=1, idim

```

```

28         r(j) = rand()
29     enddo
30
31     raio = 0
32
33     do j =1, idim
34         raio = raio + r(j)**2
35     enddo
36
37     raio = raio**0.5
38
39     if (raio <= 1) then
40         hit = hit + 1
41     endif
42 enddo
43
44 volume_mc = 2**idim*(hit/M)
45 return
46 end
47
48 function vol_exato(idim)
49     pi = acos(-1.0e0)
50     aux = 1
51     fgamma = 1
52     arg = (idim/2.0e0)+1
53
54 99     if (arg .gt. 0) then
55         call gamma_aux(aux, arg)
56         if (arg .gt. 1) then
57             fgamma = fgamma*(arg-1)
58         end if
59
60         arg = arg - 1
61         fgamma = fgamma*aux
62     goto 99
63     endif
64
65     vol_exato = (pi**((idim/2.0e0)))/fgamma
66 return
67 end

```



```

68
69      subroutine gamma_aux(aux, arg)
70          pi = acos(-1.0e0)
71
72          if (arg == 1) then
73              aux = 1
74          else if (arg == 0.5) then
75              aux = pi**0.5
76          endif
77      return
78  end

```

Por fim, temos a saída:

Figura 9: Saída do código 8.

```
andressacolaco@ametista10:/public/fiscomp2022-2-alcaraz/proj1/proj1_12610389/tarefa-8$
./tarefa-8-12610389.exe
```

DIM	M	VOLUME	ERRO
2	100	2.83999991	0.301592827
2	10000	3.15280008	1.12073421E-02
2	1000000	3.14187598	2.83241272E-04
3	100	4.00000000	0.188790321
3	10000	4.21920013	3.04098129E-02
3	1000000	4.18799210	7.98225403E-04
4	100	6.07999992	1.14519739
4	10000	4.91200018	2.28023529E-02
4	1000000	4.94147205	6.66952133E-03

Observa-se que a simulação chega cada vez mais perto do valor calculado ao aumentar-se o volume da amostragem, conforme já esperado, portanto, quanto maior este, podemos dizer com mais segurança que chegamos próximos ao valor real.

9 Tarefa 09

Na tarefa 09, os cálculos feitos para comparação são implementados em um código que visa gerar os volumes da esferas com dimensões variando de 2 até 20 em um novo arquivo, que servirá de base para gerar um gráfico correspondente.

Para isso, é construído um loop para dar à variável N (número de dimensões) os valores no intervalo $[2, 20]$. Para cada dimensão, o volume da esfera de raio unitário é calculado a partir da mesma função utilizada na tarefa anterior e então armazenado em uma variável que será escrita, juntamente com o número de dimensões, no arquivo de destino.

O código implementado se encontra a seguir:

```

1      PROGRAM dimensoes_esferas
2      parameter(pi=acos(-1.0e0))

```

```

3
4      open(10, FILE='dimensoes-esferas-12610389.dat')
5
6      do i = 2, 20
7
8          vol_esf = volume(i)
9          write(10,*) i, vol_esf
10
11      enddo
12
13      close(10)
14
15      end program dimensoes_esferas
16
17      function volume(idim)
18          pi = acos(-1.0e0)
19          aux = 1
20          fgamma = 1
21          arg = (idim/2.0e0)+1
22
23      99      if (arg .gt. 0) then
24              call gamma_aux(aux, arg)
25              if (arg .gt. 1) then
26                  fgamma = fgamma*(arg-1)
27              end if
28
29              arg = arg - 1
30              fgamma = fgamma*aux
31          goto 99
32      endif
33
34      volume = (pi**((idim/2.0e0)))/fgamma
35      return
36      end
37
38      subroutine gamma_aux(aux, arg)
39          pi = acos(-1.0e0)
40
41          if (arg == 1) then
42              aux = 1

```

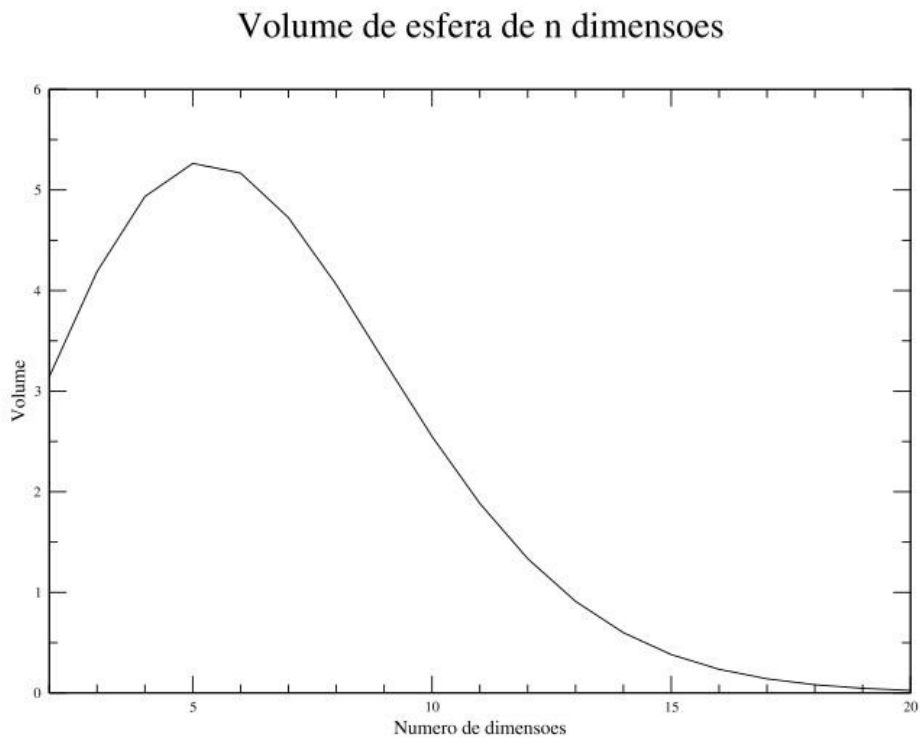
```

43         else if (arg == 0.5) then
44             aux = pi**0.5
45         endif
46     return
47 end

```

O gráfico gerado pelo *gnuplot* a partir do arquivo de saída se encontra abaixo:

Figura 10: Gráfico *Volume vs. n° dimensões*.



9.1 Pergunta A

Considerando um cubo de raio $1m^d$ (lado $2R$ ou $2m^d$) em d dimensões, queremos saber quantas vezes é maior do que uma esfera de raio unitário na mesma dimensão. Para isso, precisamos determinar a razão entre esses dois valores: o volume de um cubo será dado por $V_c = (2R)^d$ e o da esfera, pela equação:

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

Assim, a razão V_c/V_s , que indica quantas vezes o cubo é maior que a esfera, se dá por:

$$razao = \frac{2^d R^d}{\frac{\pi^{d/2}}{\Gamma(d/2+1)} R^d} = \frac{2^d}{\frac{\pi^{d/2}}{\Gamma(d/2+1)}} = \left(\frac{2}{\sqrt{\pi}}\right)^d \Gamma(d/2 + 1)$$

ou seja, não depende do raio deles, desde que sejam iguais.

Sabemos que quando d tende ao infinito, o volume da esfera tende a zero. Portanto, os dois termos da razão aumentam conforme as dimensões aumentam e o volume do cubo se distancia cada vez mais do volume da esfera. Podemos estender essa ideia para dizer que a razão tende a infinito, por continuar aumentando indefinidamente, ou melhor,

$$\lim_{d \rightarrow \infty} \left(\frac{2}{\sqrt{\pi}} \right)^d \Gamma(d/2 + 1) = \infty$$

9.2 Pergunta B

Para estimar a ordem do número de Avogadro em um mundo de d -dimensões, parte-se do pressuposto onde o número de partículas será a relação entre o volume do cubo (célula) e o volume das átomos (esferas) que estão presentes ali:

$$n_p = \frac{V_{clula}}{V_{tomos}} \quad (16)$$

Uma vez que na fórmula da razão consideramo-os com a mesma ordem de grandeza, deve-se ajustar os valores para:

$$n_p = \frac{2^d R^d}{\frac{\pi^{d/2}}{\Gamma(d/2+1)} (R)^d} = \frac{2^d (1 \cdot 10^{-6})^d}{\frac{\pi^{d/2}}{\Gamma(d/2+1)} (1 \cdot 10^{-10})^d} = \frac{2^d}{\frac{\pi^{d/2}}{\Gamma(d/2+1)}} (1 \cdot 10^4)^d$$

Ou seja,

$$n_p = \frac{V_{celula}}{V_{atomos}} (1 \cdot 10^4)^d \quad (17)$$

Para ajustar essa equação para termos que já conhecemos, podemos testar o caso onde $N = 3$, com a constante original:

$$n_p = \frac{2^3}{\pi^{3/2}} \Gamma(3/2 + 1) (1 \cdot 10^4)^3 = 1,91 \cdot 10^{12}$$

Para ajustar a equação para coincidir com o número de Avogadro, temos que adicionar um fator

$$\frac{6,02 \cdot 10^{23}}{1,91 \cdot 10^{12}} = 3,15 \cdot 10^{11}$$

Portanto, em d dimensões poderíamos dizer que o número de Avogadro seria de ordem

$$n_p = \frac{2^d}{\pi^{d/2}} \Gamma(d/2 + 1) (1 \cdot 10^4)^d (3,15 \cdot 10^{11}).$$