UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Projeto 01

Introdução ao Fortran

Jefter Santiago Mares
n° USP:12559016

17 de setembro de 2022

Conteúdo

São Carlos
2021

Tarefa 1

- A área do tórus é dada por: $A = (2\pi R)(2\pi r)$
- O volume do tórus é dado por: $V = (\pi r^2)(2\pi R)$

Onde R é o raio externo e r é o raio interno.

```
!     Tarefa 01
!     Calcula área e volume de um tórus a partir de raios dados.
      write(*,*) "Digite os valores dos raios (interno, externo):"
      read(*,*) ri, re
      pi = acos(-1e0)

      aArea = 4.e0 * pi ** 2 * re * ri
      aVolume = (pi * ri ** 2) * (2 * pi * re)

      write(*,*) "Area = ", aArea
      write(*,*) "Volume = ", aVolume
      end
```

Resultados

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa1$ ./tarefa1.exe
Digite os valores dos raios (interno, externo):
3.1 3.4
Area =    416.102570
Volume =    644.958923

jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa1$ ./tarefa1.exe
Digite os valores dos raios (interno, externo):
2.3 33
Area =    2996.41187
Volume =    3445.87402
```

Tarefa 2

Explicação

Sejam $u = (x_1, y_1, z_1)$, $v = (x_2, y_2, z_2)$ e $w = (x_3, y_3, z_3)$. Quero calcular a área lateral e volume do paralelepipedo formado pelos vetores u, v e $w = (x_3 - x_2, y_3 - y_2, z_3 - z_2)$. Para o cálculo da área temos

$$A_L = 2 \cdot [\langle u, v\rangle + \langle u, w\rangle + \langle v, w\rangle]$$

$$A_L = 2 \cdot [(x_1x_2 + y_1y_2 + z_1z_2) + (x_3 - x_2) \cdot (x_1 + x_2) + (y_3 - y_2) \cdot (y_1 + y_2) + (z_3 - z_2) \cdot (z_1 + z_2)]$$

Para o cálculo do volumes usamos o produto misto entre u, v e w, então $V = [u, v, w] = \langle u \wedge v, w\rangle$. O produto vetorial $u \wedge v$ é $(y_1z_2 - y_2z_1, x_2z_1 - x_1z_2, x_1y_2 - x_2y_1)$. Temos então o volume dado por

$$V = (x_3 - x_2)(y_1z_2 - y_2z_1) + (y_3 - y_2)(x_2z_1 - x_1z_2) + (z_3 - z_2)(x_1y_2 - x_2y_1)$$

Código

```
!     Tarefa 02
!     Dados 3 vetores (u, v, w) calcula o volume do paralelepipedo das arestas
!     definidas por u, v e w - v.
      dimension u(1:3), v(1:3), w(1:3)
      dimension aVec(1:3)
      write(*,*)"Digite as coordenadas de cada vetor"
      read(*,*) u(1), u(2), u(3)
      read(*,*) v(1), v(2), v(3)
      read(*,*) w(1), w(2), w(3)
      w(1) = w(1) - v(1)
      w(2) = w(2) - v(2)
      w(3) = w(3) - v(3)

!     A = 2 ( A1 + A2 + A3)
      aVec(1) = (u(2)*v(3)-v(2)*u(3))
      aVec(2) = (v(1)*u(3)-u(1)*v(3))
      aVec(3) = (u(1)*v(2)-v(1)*u(2))

      a1 = aVec(1)**2 + aVec(2)**2 + aVec(3)**2
      a1 = sqrt(a1)

      a2 = (u(2)*w(3) - u(1)*w(2))**2 + (u(3)*w(1) - u(1)*w(3))**2
     +     + (u(1)*w(2) - u(2)*w(1))**2
      a2 = sqrt(a2)

      a3 = (v(2)*w(3)-v(3)*w(2))**2 + (v(3)*w(1)-v(1)*w(3))**2
     +     + (v(1)*w(2) - v(2)*w(1))**2
      a3 = sqrt(a3)

      area = 2 * (a1 + a2 + a3)
      write(*,*) "Area do paralelepipedo: ", area

      volume = aVec(1) * w(1) + aVec(2) * w(2) + aVec(3) * w(3)

      write(*,*) "Volume do paralelepipedo: ", abs(volume)
```

Resultados

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa2$ ./tarefa2.exe
Digite as coordenadas de cada vetor
1 0 0
0 1 0
0 1 1
Area do paralelepipedo:    6.00000000
Volume do paralelepipedo:    1.00000000
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa2$ ./tarefa2.exe
Digite as coordenadas de cada vetor
```

```
2 3 1
2 2 1
0 1 3
Area do paralelepipedo:    42.1373100
Volume do paralelepipedo:    6.00000000
```

Tarefa 3

Foi definido como valor máximo de entrada $N = 100$, ou seja, a maior quantidade possível de valores em um arquivo deve ser 100. O input M deverá ser de tal modo que $M \leq N$.

Os dados utilizados para testar o programa estão no arquivo `input-tarefa3.dat` e foram gerados utilizando a rotina abaixo, que gera o arquivo que contém os N números aleatórios, dos quais M deles serão ordenados. Essa rotina está implementada no arquivo `./tarefa3/gerar_numeros.f`.

O programa principal faz a leitura do arquivo `./tarefa3/input-tarefa3.dat` e ordena os M primeiros menores números, após isso escreve os M números ordenados no arquivo `./tarefa3/output-tarefa3.dat`.

```
!     Tarefa 03
      parameter(N = 100)
      dimension rList(N)

      in = 10 ! unidade para arquivo de entrada.
      iout = 20 ! unidade para arquvio de saída
      open(unit=in, file="input-tarefa3.dat")
      open(unit=iout, file="output-tarefa3.dat")

      write(*,*) "Quantidade de números a serem ordenados (M  100):"
      read(*,*) M

!     Lê o arquivo com os N números aleatórios
      do i = 1, N
         read(in, *, end=1) rList(i)
      end do
1     continue
```

Na sequência foi implementado a ordenação dos M menores números do vetor em ordem decrescente utilizando o algoritmo **bubble sort**.

```
!     Implementacao do algoritmo bubble sort
      do i = 1, M
         do j = N, 2, -1
            if(rList(j) < rList(j-1)) then
               tmp = rList(j)
               rList(j) = rList(j-1)
               rList(j-1) = tmp
            end if
         end do
      end do
```

Após a ordenação os M primeiros menores números foi então escrito no arquivo `./tarefa3/output-tarefa3.dat`

```
!     Salva no arquivo "output-tarefa3.dat"
      do i = 1, M
         write(iout, *) rList(i)
      end do
      write(iout, 999) M
 999  format(i2 , ' numeros.')

      close(in)
      close(iout)
```

Resultados

Pode ser testado o programa mudando os valores no arquivo `input-tarefa3.dat`, rodando o programa no terminal e depois analisando o arquivo gerado `output-tarefa3.dat`, que conterá os M primeiros menores valores do arquivo original, em ordem.

Exemplo

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ ./tarefa3.exe

Quantidade de números a serem ordenados (M   100):
15
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ cat output-tarefa3.dat
0.108032703
0.108789563
0.112783670
0.117679000
0.124440551
0.194475591
0.202756226
0.211118400
0.211621881
0.211978197
0.219443142
0.246241093
0.246837378
0.255061328
0.257259607
15 numeros.
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ ./tarefa3.exe
Quantidade de números a serem ordenados (M   100):
10
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ cat output-tarefa3.dat
0.108032703
0.108789563
0.112783670
0.117679000
```

```
0.124440551
0.194475591
0.202756226
0.211118400
0.211621881
0.211978197
10 numeros.
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ ./tarefa3.exe
Quantidade de números a serem ordenados (M   100):
3
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa3$ cat output-tarefa3.dat
0.108032703
0.108789563
0.112783670
3 numeros.
```

Tarefa 4

Queremos cálcular a série

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

que pode ser escrita como a série de potências

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{2n}}{(2n)!}$$

Primeiramente, fazemos alguns ajustes que facilitam a programação do cálculo da série, manipulando a equação temos

$$\cos x = 1 + \sum_{i=1}^{\infty} (-1)^i \cdot \frac{x^{2i}}{(2i)!}$$

onde i será a iteração do loop. Escrevemos também o termo $(2i)!$ como $i(i+1)$, que será multiplicado sempre pelo termo da iteração anterior, dessa forma todo o programa fica mais otimizado, com todo o cálculo da série sendo feito em um mesmo loop.

Teremos então, para o cálculo da série, o código abaixo:

```
1    !     Tarefa 4
2   !     Implementação da série de potências
3   !     cos(x) = (-1)^n * (x^(2n))/(2n)!
4
5         double precision eprecd
6         double precision cossd
7         double precision accd
8         double precision tmpd
9         double precision xd
10        double precision axd
11
12        eprecd = 1e-16
13        accd = eprecd * 2
```

```
14
15        eprec = 1e-5
16        acc = eprec * 2
17
18        ifat = 1
19
20        coss = 1e0
21        tmp = coss
22
23        write(*,*) "Informe x em radianos:"
24        read(*,*) xd
25
26        x = xd
27        ax = x**2
28
29        i = 1
30        do while (eprec  <=  acc)
31           ifat = (i * (i + 1))
32
33           tmp = tmp * (-1) * ax / ifat
34           coss = coss + tmp
35
36           acc = abs(tmp) - eprec
37
38           i = i+2
39        end do
40
41        cossd = 1d0
42        tmpd = cossd
43
44        axd = xd**2
45        ifat = 1
46        i = 1
47
48        do while (eprecd  <=  accd)
49           ifat = (i * (i + 1))
50
51           tmpd = tmpd * (-1) * axd / ifat
52           cossd = cossd + tmpd
53
54           accd = abs(tmpd) - eprecd
55
56           i = i+2
57        end do
58
59        write(*,*) "=================="
60        write(*,*) "=   Resultados   ="
61        write(*,*) "=================="
62
```

```
63        write(*,*) "Precisão simples"
64        write(*,*) "cos(x) = ", coss
65
66        errs = abs(cos(x)) - abs(coss)
67        write(*,*) "Erro: " , errs
68
69        write(*,*) "--------------"
70
71        write(*,*) "Precisão dupla"
72        write(*,*) "cos(x) = ", cossd
73
74        errd = abs(dcos(xd)) - abs(cossd)
75        write(*,*) "Erro: " , errs
76
77        end
```

Resultados

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa4$ ./tarefa4.exe
 Informe x em radianos:
- 1
=================
=   Resultados   =
=================
Precisão simples
cos(x) =   0.540302277
Erro:    0.00000000
--------------
Precisão dupla
cos(x) =   0.54030230586813965
Erro:    0.00000000
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa4$ ./tarefa4.exe
Informe x em radianos:
3.14159265
=================
=   Resultados   =
=================
Precisão simples
cos(x) =  -0.999999881
Erro:    1.19209290E-07
--------------
Precisão dupla
cos(x) =   -1.0000000000000002
Erro:    1.19209290E-07
```

Tarefa 5

Paridade de permutação

Seja σ uma permutação de n elementos, o par $[i, j]$, onde $1 < i < j < n$ é chamado de inversão se $i < j$ e $\sigma(i) > \sigma(j)$. Uma permutação é um mapa do tipo $\sigma : \mathbb{N} \mapsto \mathbb{N}$. A paridade de uma permutação σ é dada por $(-1)^{\lambda}$, onde λ é o número de inversões em σ. Permutações pares ocorrem se λ é par. O sinal da paridade é representado por $P(\sigma)$.

Exemplo: cálculo de paridades

Para $n = 3 \Rightarrow n! = 6$, ou seja, o conjunto de permutações $S = \{123, 132, 213, 231, 312, 321\}$.

- $\sigma\left(\begin{smallmatrix}1&2&3\\1&2&3\end{smallmatrix}\right) \Rightarrow N = 0 \Rightarrow P(\sigma) = (-1)^N = (-1)^0 = 1$
- $\sigma\left(\begin{smallmatrix}1&2&3\\1&3&2\end{smallmatrix}\right) \Rightarrow (2, 3) \Rightarrow N = 1 \Rightarrow P(\sigma) = -1$
- $\sigma\left(\begin{smallmatrix}1&2&3\\2&1&3\end{smallmatrix}\right) \Rightarrow (1, 2) \Rightarrow N = 1 \Rightarrow P(\sigma) = -1$
- $\sigma\left(\begin{smallmatrix}1&2&3\\2&3&1\end{smallmatrix}\right) \Rightarrow (1, 3), (2, 3) \Rightarrow N = 2 \Rightarrow P(\sigma) = 1$
- $\sigma\left(\begin{smallmatrix}1&2&3\\3&1&2\end{smallmatrix}\right) \Rightarrow (1, 2), (1, 3) \Rightarrow N = 2 \Rightarrow P(\sigma) = 1$
- $\sigma\left(\begin{smallmatrix}1&2&3\\3&2&1\end{smallmatrix}\right) \Rightarrow (1, 2), (1, 3), (2, 3) \Rightarrow N = 3 \Rightarrow P(\sigma) = -1$

  Podemos então colocar as permutações de n em com suas respectivas paridades

Tabela 1: Permutações para $n = 3$ e suas paridades.

σ	$P(\sigma)$
1 2 3	1
1 3 2	-1
2 1 3	-1
2 3 1	1
3 1 2	1
3 2 1	-1

Solução

Para o caso de $n + 1$, com $n = 3$ temos 24 permutações, podemos escrever como $4 \cdot 3!$, ou seja, 4 conjuntos de permutações a partir das permutações de 3. Então podemos escrever esses conjuntos levando em conta que o 4 vai ocupar uma posição diferente em cada grupo:

$$\left|\begin{array}{cccc} 1\,2\,3\,4 & 1\,2\,4\,3 & \cdots & 4\,1\,2\,3 \\ 1\,3\,2\,4 & 1\,3\,4\,2 & \ldots & 4\,1\,3\,2 \\ \vdots & \vdots & \vdots & \vdots \end{array}\right|$$

Note que, o número de linhas é 3! e de colunas são $n + 1$. A ánalise das paridades pode ser feita apenas observando as colunas. O valor $n + 1$ começa na coluna mais à direita, fazendo então as comparações que definem paridade vemos que para a primeira coluna todas paridades se mantém, então as paridades são as mesmas que para 3!. Para a segunda coluna vemos que em todas linhas temos que $\sigma(3) = 4 > \sigma(4)$, onde 3 e 4 são os indices (ou posição na coluna), então o número de

inversões será ímpar, pois a paridade será $(-1)^1$. Para o caso seguinte, de 2 temos que $\sigma(2) > \sigma(3)$ e $\sigma(2) > \sigma(4)$, pois 4 é maior que qualquer elemento (1, 2 ou 3) nas colunas seguintes. Então a paridade será $(-1)^2 = 1$.

Generalizando, seja $j > 1$ o indice da coluna, a nova paridade será dada por (-1) . Segue a descrição do algoritmo em pseudo-código:

Algorithm 1 Gera $(n+1)$ permutações e suas respectivas paridades.

```
n_Perm ← lista com P = n! permutações e suas paridades
for j ← n_Cols + 1 to 1, step = −1 do
    if j < nCol + 1 then
        for i ← 1 to n_Linhas, step = −1 do
            aux = n_Perm(i, j)
            n_Perm(i, j) = n_Perm(i, j − 1)
            n_Perm(i, j − 1) = aux
            n_Perm(i, n_Col + 1) = (−1) · n_Perm(i, n_Col + 1)
        end for
    end if
    Armazena(aux_Perm)
end for
```

Prova de corretude do algoritmo

No algoritmo, n_{Perm} é uma matriz de dimensões $n! \times (n+1)$, todas as linhas da coluna n possui inicialmente valores iguais à n e as linhas da coluna $(n+1)$ as paridades de cada permutação.

- **I)** O loop externo itera até que $j \leftarrow 1$, ou seja, essa é a sua condição de parada.
- **II)** A condição $j < n_{\text{Col}} + 1$ garante que as colunas só vão começar a ser trocadas a partir da segunda iteração.
- **III)** O loop interno vai iterar até que $i \leftarrow n_{\text{Linhas}}$, então, por todas as linhas da matriz.
  - No loop interno é feita a troca de posições das colunas, ou seja, para $j = n_{\text{Col}}$ teremos o seguinte loop

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \\ \vdots & \vdots & \vdots & \vdots \end{array} \overset{j \leftarrow j-1}{\longrightarrow} \begin{array}{cccc} 1 & 2 & 4 & 3 \\ 1 & 3 & 4 & 2 \\ \vdots & \vdots & \vdots & \vdots \end{array} \overset{j \leftarrow j-2}{\longrightarrow} \begin{array}{cccc} 1 & 4 & 2 & 3 \\ 1 & 4 & 3 & 2 \\ \vdots & \vdots & \vdots & \vdots \end{array} \overset{j \leftarrow j-3}{\longrightarrow} \begin{array}{cccc} 4 & 1 & 2 & 3 \\ 4 & 1 & 3 & 2 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

    Como esse procedimento é feito para todas colunas (exceto a das paridades), no fim do loop teremos permutações de $(n+1)$ com $n!$ linhas. A paridade de cada permutação, ou seja, cada linha da matriz, é calculada por $(-1) \cdot n_{\text{Perm}}(j, n_{\text{Col}} + 1)$, a nova paridade é então um produto da antiga por (-1), como descrito acima na descrição matemática.
- **IV)** Quando o loop interno finaliza tem-se uma matriz com as permutações de $n!$ com $(n+1)$ na posição j. É então armazenado esse valor na num arquivo de saída.
- **V)** Com o fim do loop externo temos um arquivo de saída com $(n+1)!$ permutações e suas devidas paridades.

  Portanto, esse algoritmo computa as $(n+1)!$ permutações e suas respectivas paridades. □

Código em Fortran

Preparação dos dados

Primeiramente é feita a declaração das variaveis e realizada a leitura do arquivo `input-tarefa3.dat`, que contém as permutações para $n = 3$ e suas respectivas paridades.

```
!     Lendo o arquivo 'input-tarefa3.dat'
      parameter(nRow = 6, nCol = 4)
      dimension iPerm(nRow,nCol), nAuxPerm(nCol, nRow)
      dimension nPerm(nRow, nCol + 1)
      dimension iauxPerm(nRow, nCol + 1)

      open(unit=10, file="input-tarefa3.dat")
      open(unit=20, file="output-tarefa3.dat")

      read(in, *) nAuxPerm
      iPerm = transpose(nAuxPerm)
```

Esse bloco de código gera a matriz com as $n + 1$ colunas além da coluna com as paridades.

```
!   Gera a matrix n! x (n + 1)
      i = 1
      do while(i < nCol)
        do j = 1, nRow
          nPerm(j, nCol + 1) = iPerm(j, nCol)
          nPerm(j, nCol) = nCol
          nPerm(j, i) = iPerm(j, i)
       end do
      i =  i + 1
      end do
```

Implementação do algoritmo (1) em Fortran

As linhas 31 à 33 não são equivalentes ao termo "Armazena(n_{Perm})", presente na descrição do algoritmo. Elas escrevem no arquivo `output-tarefa5.dat`.

```
!    Por fim escreve no arquivo 'output-tarefa5.dat'
     do j = nCol, 1 , -1
        iauxPerm = nPerm
        do i = 1, nRow
           iaux = iauxPerm(i, j)
           iauxPerm(i, j) = iauxPerm(i, nCol)
           iauxPerm(i, nCol) = iaux

           iauxPerm(i, nCol + 1) = (-1)**(j + 2)*iauxPerm(i, nCol + 1)
        end do
        do k = 1, nRow
           write(20,*) (iauxPerm(k,ih), ih=1, nCol + 1)
        end do
     end do
     close(10)
```

```
36        close(20)
37        end
```

Resultados

A tabela abaixo é o resultado para o input de permutações da tabela 1.

i	σ	$P(\sigma)$
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

Foi gerado o código executável para os casos de $n = 3, 4, 5$ e 6 e disponibilizados na pasta `./tarefa5/`, os arquivos das permutações $n = 4, 5$ e 6 foram usados para o cálculo do determinante da tarefa 6.

Replicabilidade e generalização do código em Fortran

Para inputs diferentes de $n = 3$, é necessário alterar o arquivo `tarefa5.f`. Esse código foi escrito pensando na implementação especifica de $n = 3$, escrever um para um n qualquer envolveria um código com algumas especifidades maiores, por exemplo, como o `fortran77` não possui alocação dinâmica de memória, seria necessário programar condições especificas para que o algoritmo não tentasse acessar indíces além do número de elementos que a matriz possui.

Então, para casos gerais, é necessário alterar a linha um do programa, alterar o número de linhas, $n!$, e o número de colunas $n+1$, além, é claro, de substituir o arquivo `input-tarefa3.dat` pelo arquivo com as permutações de $n!$.

Tarefa 6

Definição do determinante através de permutações

Seja a matriz quadrada $A = (a_{ij})$, de dimensões $n \times n$, podemos cálcular o seu determinante pela formúla

$$\det A = \sum_{\sigma \in S} P(\sigma) \cdot A_{1,\sigma_1} \cdot A_{2,\sigma_2} \cdot \cdots \cdot A_{n,\sigma_n}$$

Onde σ é uma permutação em um conjunto S de permutações e $P(\sigma)$ é a paridade de cada permutação.

Exemplo de como construir o somatório

Seja $n = 3$, sabemos que uma matriz A , $n \times n$, tem determinante igual à $\det A = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$. Mostramos a então que o determinante calculado pela definição é equivalente:

- $\sigma(1) = \{1, 2, 3\}, P(1) = 1$
- $\sigma(2) = \{1, 3, 2\}, P(2) = -1$
- $\sigma(3) = \{2, 1, 3\}, P(3) = -1$
- $\sigma(4) = \{2, 3, 1\}, P(4) = 1$
- $\sigma(5) = \{3, 1, 2\}, P(5) = 1$
- $\sigma(6) = \{3, 2, 1\}, P(6) = -1$

Os elementos de cada permutação vão servir para selecionar quais os indices dos elementos em cada iteração do somatório. Cada permutação σ define as posições dos elementos em A que devem ser multiplicados, fazendo o somatório temos

$$\begin{aligned}\det A = P(1) \cdot A_{1,1}A_{2,2}A_{3,3} + P(2) \cdot A_{1,1}A_{2,3}A_{3,2} + P(3) \cdot A_{1,2}A_{2,1}A_{3,3} + \\ P(4) \cdot A_{1,2}A_{2,3}A_{3,1} + P(5) \cdot A_{1,3}A_{2,1}A_{3,2} + P(6)A_{1,3}A_{2,2}A_{3,1}\end{aligned}$$

$$\det A = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}$$

Código implementado em Fortran

```
!     Calcula determinante pela definição
      parameter(nRow = 24, nCol = 5)
      dimension nPerm(nRow, nCol), nAuxPerm(nCol, nRow)
      parameter(nMatrix = 4)
      dimension rMatrix(nMatrix, nMatrix)

      open(10, file="permutacoes-n4.dat")
      open(20, file="matriz-4x4.dat")

      read(10, *) nAuxPerm
      nPerm = transpose(nAuxPerm)
```

```
12
13        read(20, *) rMatrix
14        rMatrix = transpose(rMatrix)
15
16        det = 0e0
17        do i = 1, nRow
18
19           signal = nPerm(i, nCol)
20           aux = 1e0
21           do j = 1, nCol - 1
22              aux = aux * rMatrix(j, nPerm(i, j))
23           end do
24           det = det + signal * aux
25
26        end do
27
28        write(*, *) "Determinante = ", det
29        end
```

Resultados

Na pasta que contém o código tem também os arquivos de entrada para as matrizes às quais o cálculo do determinante é realizado e também as respectivas matrizes de permutações. Foram gerados 3 arquivos execútaveis, `determinante-4x4.exe, =determinante-5x5.exe` e `determinante-6x6.exe`, cada um realiza o calculo do determinante para matrizes de dimensões $4\times4, 5\times5$ e 6×6, respectivamente. Para se alterar a matriz basta modificar os arquivos `matriz-4x4.dat`, `matriz-5x5.dat` e `matriz-6x6.dat`, também presentes na pasta `./tarefa6/`.

Abaixo alguns exemplos de uso dos programas:

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ cat matriz-4x4.dat
3 2 1 2
4 4 1 2
3 0 2 7
2 1 4 4
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ ./determinante-4x4.exe
 Determinante =   -47.0000000
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ cat matriz-5x5.dat
2 1 1 4 2
9 3 9 0 3
1 1 9 6 3
1 3 5 5 4
1 9 2 2 2
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ ./determinante-5x5.exe
 Determinante =   -4068.00000
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ cat matriz-6x6.dat
1 2 3 4 50 6
11 2 3 14 5 2
3 29 3 4 15 6
4 1 13 4 5 6
```

```
9 4 32 4 59 6
2 0 -3 4 -1 46
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa6$ ./determinante-6x6.exe
 Determinante =   -61599620.0
```

Tarefa 7

O sistema linear do tipo $AX = Y$ pode ser resolvido usando determinates usando a regra de Cramer, ou seja $X_i = \frac{\det A_i}{\det A}$, onde a matriz A_i tem a coluna i trocada pelo vetor Y. Usando o a definição de determinante explicada na tarefa anterior podemos escrever um programa para resolver um sistema desse tipo.

Segue abaixo a implementação do código para o cálculo da solução de um sistema linear para matriz de ordem 4×4, com as matrizes A e Y dadas:

```
1  !     Cálcula as soluções de um sistema linear
2  !     AX = Y com A e Y dados.
3        parameter(nRow = 24, nCol = 5)
4        parameter(nMatrix = 4)
5
6        dimension nPerm(nRow, nCol), nAuxPerm(nCol, nRow)
7        dimension rMatrix(nMatrix, nMatrix), auxMatrix(nMatrix, nMatrix)
8
9        dimension Y(1:nMatrix)
10
11       open(30, file='y.dat')
12       read(30, *) Y
13 !      print *, Y
14       close(30)
15
16       open(10, file="permutacoes-n4.dat")
17       open(20, file="matriz-4x4.dat")
18
19       read(10, *) nAuxPerm
20       nPerm = transpose(nAuxPerm)
21
22       read(20, *) rMatrix
23       rMatrix = transpose(rMatrix)
24
25       detA = 0e0
26       call det(nRow, nCol,nPerm, nMatrix, rMatrix, detA)
27
28       write(*, *) "Matriz solução X: "
29       do j =  1, nMatrix
30          auxMatrix = rMatrix
31          do i = 1, nMatrix
32             auxMatrix(i, j) = Y(i)
33          end do
34
35         detXj = 0e0
```

```
36          call det(nRow,nCol,nPerm,nMatrix,auxMatrix,detXj)
37          write(*, *) detXj / detA
38
39        end do
40
41        close(10)
42        close(20)
43        end
```

A subrotina `det` é a implementação do cálculo do determinante pela definição feito na tarefa 6 adaptada para este programa.

```
1  subroutine det(nRow,nCol,nPerm,nMatrix,rMatrix, detx)
2  dimension nPerm(nRow, nCol), nAuxPerm(nCol, nRow)
3  dimension rMatrix(nMatrix, nMatrix)
4
5  do i = 1, nRow
6     signal = nPerm(i, nCol)
7     aux = 1e0
8     do j = 1, nCol - 1
9        aux = aux * rMatrix(j, nPerm(i, j))
10    end do
11    detx = detx + signal * aux
12 end do
13 return
14 end
```

Na pasta `./tarefa3/` consta 3 arquivos `.exe`, executáveis que podem ser usados para cálcular as soluções com matrizes $4 \times 4, 5 \times 5$, e 6×6. Para testar basta rodar os arquivos e alterar os arquivos `matriz-4x4.dat`, `matriz-5x5.dat` e `matriz-6x6.dat`. Além de também alterar os inputs do arquivo `y-4x4.dat`, `y-5x5.dat` e `y-6x6.dat`.

Resultados

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat y-4x4.dat
4
3
13
1
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat matriz-4x4.dat
3 2 1 2
4 4 1 2
3 0 2 7
2 1 4 4
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ ./sistema4x4.exe
 Matriz solução X:
  4.17021275
 -3.08510637
-0.787234068
-0.276595742
```

```
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat y-5x5.dat
5
3
9
1
3
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat matriz-5x5.dat
2 1 1 4 2
9 3 9 0 3
1 1 9 6 3
1 3 5 5 4
1 9 2 2 2
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ ./sistema5x5.exe
 Matriz solução X:
  1.07964599
 0.469026536
 0.519174039
  2.59587026
 -4.26548672
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat y-6x6.dat
5
1
3
1
9
-1
jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ cat matriz-6x6.dat
1 2 3 4 50 6
11 2 3 14 5 2
3 29 3 4 15 6
4 1 13 4 5 6
9 4 32 4 59 6
2 0 -3 4 -1 46

jefter66@muaddib~/Workspace/intro_fiscomp/projeto1/tarefa7$ ./sistema6x6.exe
Matriz solução X:
0.341461062
 4.57333997E-02
 5.84237371E-03
-0.242816135
0.111934975
-1.26564093E-02
```

Tarefa 8

Breve explicação sobre Método de Monte Carlo

Podemos escrever a média de uma função f pelas expressões

$$\langle f \rangle = \frac{1}{b-a}\int_a^b f(x)dx = \frac{1}{N}\sum_i f(x_i)$$

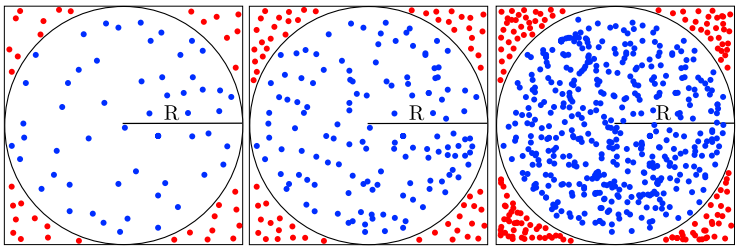
onde o termo N é o número total de pontos calculados e x_i são pontos que residem extritamente no intervalo $[a, b]$. Podemos então escrever uma aproximação para integral como

$$\int_a^b f(x)dx \approx \frac{(b-a)}{N}\sum_i f(x_i)$$

Implementação do Método de Monte Carlo para cálculo do volume da N-esfera

Vamos chegar em um formúla que foi implementada no código em Fortran.

Primeiramente para o caso básico de $d = 2$, ou seja, o volume em $2D$, que é equivalente à área do circulo, $A_c = \pi R^2$. Pelo método de Monte Carlo essa área pode ser calculada gerando N valores X_i aleatórios e contado N_i desses números que obedecem a regra $\sum_i^N X_i^2 \leq R^2$, ou seja, quantos pontos X_i estão dentro da área do circulo.



Para N muito grande a figura tende a ficar completamente preenchida e então a área do quadrado $A_q \approx N$ e a área do circulo $A_c \approx N_i$, fazendo a razão abaixo temos uma aproximação melhor para a área do circulo

$$\frac{N_i}{N} \approx \frac{A_c}{A_q} \Rightarrow A_c \approx A_q\left(\frac{N_i}{N}\right)$$

Já sabemos que a área do quadrado é dada por $A_q = (2R)^2$, queremos chegar a uma fórmula para o volume podemos então considerar $V_{cubo}(d) = (2R)^d, d \in \mathbb{N}$, volume do cubo. Generalizando a área do circulo para o volume temos $V(d) = V_{cubo}(d)\left(\frac{N_i}{N}\right)$, ou seja

$$\boxed{V(d) = (2R)^d\left(\frac{N_i}{N}\right)}$$

Essa formúla foi implementada no código abaixo e usada para calcular o volume da esfera em 2, 3 e 4 dimensões.

Código

```
!     Tarefa 8 - Cálculo do volume de uma N-Esfera

      write(*, *) "Informe a quantidade N de números aleatórios."
      read(*, *) n

      write(*, *) "Resultados"
      v2 = volume(n, 2)
      v3 = volume(n, 3)
      v4 = volume(n, 4)

      print *, "Para d = 2, Volume = ", v2
      print *, "Para d = 3, Volume = ", v3
      print *, "Para d = 4, Volume = ", v4
      end

!     m - número de iteracoes
      function volume(n, id)
      ni = 1
!     Para uma esfera de raio unitário
      r = 1
      do j = 1, n
         rtmp = 0e0
         raux = 0e0
         do i = 1, id
            call random_number(rtmp)
            raux = raux + rtmp ** 2
         end do
         raux = sqrt(raux)
         if(raux <= r) then
            ni = ni + 1
         end if
      end do
      ni = ni - 1
!     V(d) = (2R)^d * (Ni)/N
      volume = (2 * r)**id * ((ni * 1e0)/ (n * 1e0))
      end
```

Resultados

Para fazer a comparação com o esperado, ou seja, o cálculo da expressão (1), foi feito o desvio padrão cada uma das 3 dimensões cálculadas. Na tabela abaixo consta os valores médios do volume nas três dimensões e seus respectivos desvios padrão. Foi utilizado o método um número $N = 1000$ de vezes com intuito de cálcular o desvio padrão do método de Monte Carlo para o volume da n-esfera.

Comparando os valores médios encontrados com os cálculados pela expressão (1), que estão na tabela (4) é a aproximação é razoável, principalmente para ordens cada vez maiores de amostras para o método de Monte Carlo.

Tabela 2: Volume da esfera cálculada em $2, 3$ e 4 dimensões para diferentes ordens de grandezas.

Ordem	2 dimensões	3 dimensões	4 dimensões
10^1	2.79999995	4.00000000	6.40000010
10^2	3.31999993	4.15999985	4.63999987
10^3	3.16799998	4.21600008	4.96799994
10^4	3.15280008	4.15999985	4.86880016
10^5	3.13836002	4.18616009	4.93279982
10^6	3.14278412	4.18742418	4.93838406
10^7	3.14097643	4.18836260	4.93273115

Tabela 3: Volume médio da esfera cálculado para um número de amostras $N = 1000$ e seu desvio padrão.

Ordem	2 dimensões	3 dimensões	4 dimensões
10^1	3.1 ± 0.5	4 ± 1	5 ± 2
10^2	3.1 ± 0.2	4.2 ± 0.4	4.9 ± 0.7
10^3	3.14 ± 0.05	4.19 ± 0.12	4.9 ± 0.2
10^4	3.14 ± 0.01	4.18 ± 0.04	4.93 ± 0.07
10^5	3.141 ± 0.005	4.18 ± 0.01	4.93 ± 0.02
10^6	3.141 ± 0.001	4.188 ± 0.004	4.934 ± 0.007
10^7	3.1415 ± 0.0005	4.188 ± 0.001	4.934 ± 0.002

Tarefa 9

O volume da n-esfera é definido como

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

Foi feito um programa para executar o cálculo do volume da n-esfera em d dimensões, d = 2, 3 , ..., 20. Os valores cálculados estão na tabela (4) abaixo.

Código

```
1       external gamma
2       open(unit=10, file="dimensoes-esferas.dat")
3
4       R = 1.0e0
5       PI = acos(-1.0e0)
6       do n = 2, 20
7  !    V(d) = [ ^(d/2) / Γ(d/2 + 1)] * R^(d)
8          Volume = (PI ** ((n * 1.0e0)/ 2) / gamma(n)) * R ** (n)
9          write(10, *) Volume
10      end do
11
12      close(10)
13      end
14
```

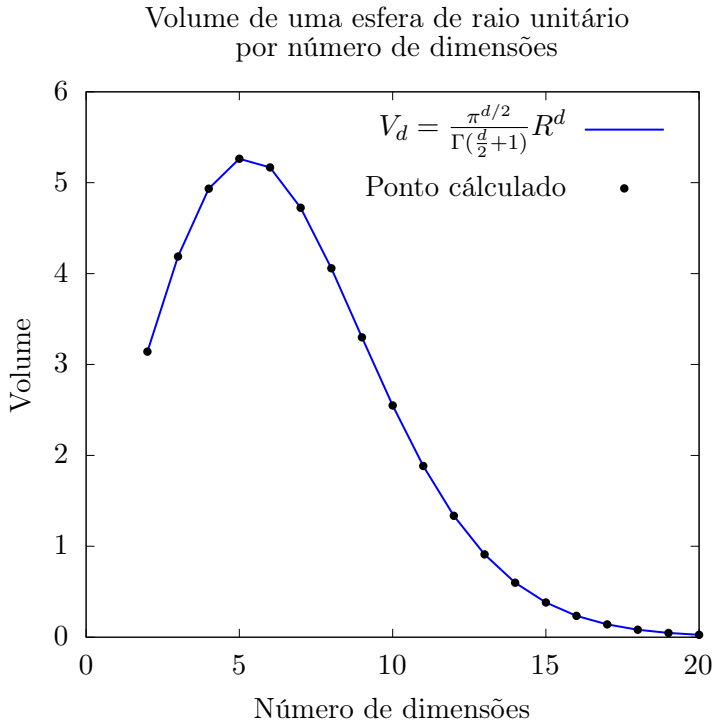
```
15        function gamma(n)
16        aux = 1e0
17        gamma = 1e0
18
19        ! (d/2 + 1)
20        x = (1.0e0 * n) / 2.0e0 + 1.0e0
21
22        do while(x >= 0e0)
23           call baseCases(x, aux)
24           if(x > 1) then
25              gamma = gamma * (x - 1)
26           end if
27           x = x - 1
28           gamma = gamma * aux
29        end do
30        end
31
32        subroutine baseCases(x, aux)
33        if(x == 0e0) then
34           aux = 1
35        else if (x == 0.5e0) then
36           aux = sqrt(acos(-1e0))
37        end if
38        return
39        end
```

Com base nos valores dessa tabela foi construido o gráfico da variação de volume em função do aumento das dimensões.

A partir desses cálculos foi feito o gráfico abaixo e na sequência respondido algumas perguntas acerca da n-esfera.

Tabela 4: Volume da esfera para diferentes dimensões.

Dimensão	Volume
2	3.14159274
3	4.18879032
4	4.93480253
5	5.26378918
6	5.16771317
7	4.72476625
8	4.05871248
9	3.29850912
10	2.55016422
11	1.88410401
12	1.33526301
13	0.910628915
14	0.599264622
15	0.381443352
16	0.235330686
17	0.140981153
18	0.00821459070
19	0.00466216095
20	0.00258068983



Questão A

Saber quantas vezes maior será o volume é equivalente a saber a razão entre a cubo e a esfera em n dimensões, como o volume do cubo em n dimensões é dado por $V_c = (2R)^d$ e vamos usar o volume da

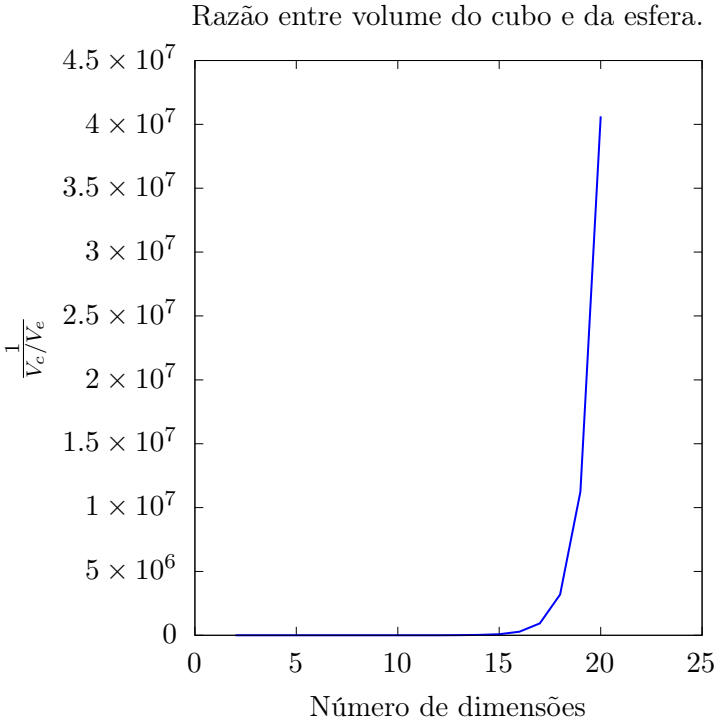
esfera (1), então

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

ou seja, a diferença dos volumes não depende do raio, apenas da dimensão. Assim, para qualquer d, a razão do cubo e da esfera será de $\frac{V_e}{V_d} = \left(\frac{\sqrt{\pi}}{2}\right)^d \frac{1}{\Gamma(\frac{d}{2}+1)}$. No caso de $d \mapsto \infty$ temos que, como $\lim_{d\to+\infty} \Gamma(\frac{d}{2}+1) = +\infty$ e $\left(\frac{\sqrt{\pi}}{2}\right) < 1$, então

$$\lim_{d\to\infty} \left(\frac{\sqrt{\pi}}{2}\right)^d \frac{1}{\Gamma(\frac{d}{2}+1)} = 0$$

Portanto, a razão $\frac{V_c}{V_e} \to \infty$, podemos dizer que para dimensões cada vez maiores, o tamanho do cubo será muito maior que o da esfera. Esse resultado pode ser melhor exemplicado através do gráfico abaixo:



Questão B

Sabemos que o número de particulas pode ser calculado pela relação $n = \frac{V_e}{V_e}$ Nessa hipótese temos que $[V_c]$ em μ^d e $[V_e]$ em ^{d}. Considerando o volume da célula como equivalente ao de um cubo em d dimensões, ou seja, $v_{\text{cel}} = (2R)^d$ e o do átomo aproximado por uma esfera, $v_{\text{atom}} = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}R^d$, sabemos que $n_p = \frac{V_{\text{cel}}}{V_{\text{atom}}}$ = número de partículas, ajustando para as unidades dadas, temos que

$$n_p = \frac{V_{\text{cel}}}{V_{\text{atom}}} \cdot \frac{1 \cdot \mu^d}{1 \cdot {}^d} = \frac{V_{\text{cel}}}{V_{\text{atom}}} \cdot \frac{\left(1 \cdot 10^{-4}\right)^d}{\left(1 \cdot 10^{-10}\right)^d}$$

$$np = \frac{V_{\text{cel}}}{V_{\text{atom}}} 10^{4d} \tag{2}$$

Sabemos que o número de Avogadro na dimensão $d = 3$ é tal que $n_p = 6 \cdot 10^{23}$, temos

$$n_p = 6 \cdot 10^{23} = \frac{V_{\text{cel}}}{V_{\text{atom}}} 10^{4 \cdot 3} = \frac{V_{\text{cel}}}{V_{\text{atom}}} 10^{12}$$

como os números não batem, precisamos fazemos um ajuste para que possamos generalizar a ordem de grandeza para o número de Avogadro em d dimensões. Para isso fazemos $\frac{10^{23}}{10^{12}} = 10^{11}$. Então, em d dimensões, o número de Avogadro é da ordem de $10^{4d} \cdot 10^{11} = 10^{4d+11}$.