Instituto de Física de São Carlos
Universidade de São Paulo

[7600017] - Introdução à Física Computacional

Projeto 1

Docente:
Francisco Castilho Alcaraz

Aluno:
Breno Henrique Pelegrin da Silva (13687303)

31 de agosto de 2023

Sumário

Lista de Figuras

1 Contexto

Após a execução das tarefas do primeiro projeto de Introdução à Física Computacional, foi possível perceber que o objetivo do projeto é exercitar os conhecimentos da linguagem *Fortran 77*, como a declaração implícita de variáveis, estruturas de decisão, laços, leitura e escrita de arquivos, algoritmos básicos e métodos estatísticos básicos. Assim, realizar esse trabalho contribuiu de forma positiva para o aprendizado da linguagem, reforçando os conceitos já vistos anteriormente.

Para a leitura do relatório, vale ressaltar ao leitor que todos os códigos foram elaborados utilizando as especificações da linguagem *Fortran 77*, portanto, não foi permitida a utilização de alocação dinâmica de vetores e declaração explícita de variáveis (práticas e ferramentas geralmente utilizadas no *Fortran 90* e versões mais recentes). Além disso, os códigos foram testados após sua compilação utilizando o compilador *gfortran* do GNU (*Operating System and the Free Software Movement*), nas máquinas do servidor *basalto.ifsc.usp.br* e enviados utilizando a padronização de nomes estabelecida pelo docente.

2 Tarefa 1

Nessa tarefa, foi criado um programa que lê os coeficientes a, b e c de uma equação de segundo grau na forma exposta pela equação 1, e em seguida, calcula as raízes reais dessa equação, se houver alguma, utilizando a equação 2. As raízes de uma equação desse tipo são os valores x tais que a equação é zero, ou seja, se graficarmos a função em um plano cartesiano, as raízes são os valores de x tais que o gráfico intersecta o eixo x do plano.

$$ax + bx + c = 0 \tag{1}$$

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \tag{2}$$

Para verificar se existem uma, duas, ou zero raízes reais, o programa inicialmente calcula o discriminante Δ, dado pela equação 3. Se $\Delta = 0$, então existe apenas uma raíz real. Se $\Delta > 0$, então existem duas raízes reais diferentes. Por fim, se $\Delta < 0$, então não existem raízes reais para a equação. O código-fonte criado é exibido na figura 1 e a explicação para ele é dada logo abaixo do código, no verbete 1.

$$\Delta = b - 4ac \tag{3}$$

Figura 1: Código-fonte da tarefa 1

```
      program Bhaskara
c     Tarefa 1 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         write(*,100)
 100     format("Digite os coef. a, b, c da equação ax²+bx+c=0: ", $)
         read(*,*) a, b, c
         delta = b**2 - 4*a*c
         if (delta .lt. 0.0e0) then
            write(*,*) "A equação não possui raízes reais."
         else
            if (delta .eq. 0.0e0) then
               x1 = (-b + sqrt(delta))/2*a
               write(*,*) "1 raíz real:", x1
            else if (delta .gt. 0.0e0) then
               x1 = (-b + sqrt(delta))/2*a
               x2 = (-b - sqrt(delta))/2*a
               write(*,*) "2 raízes reais:", x1, x2
            end if
         end if
      end program
```

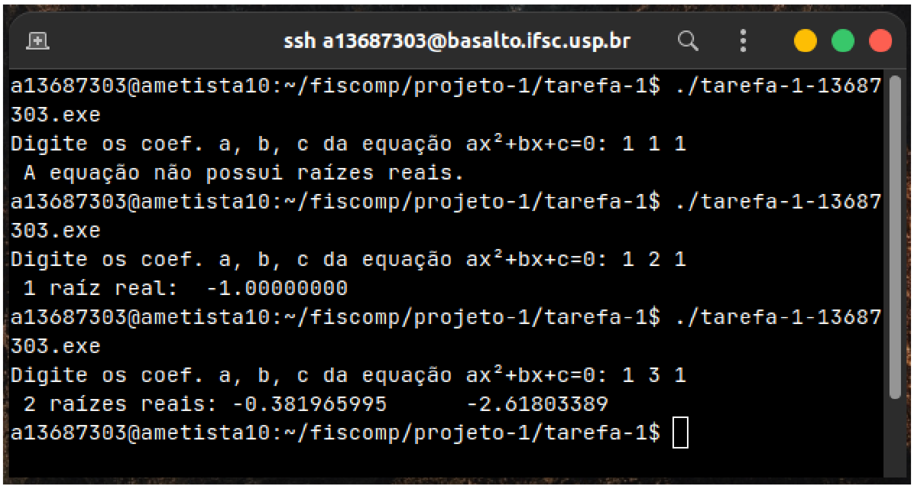
Verbete 1: Explicação do código da tarefa 1

Inicialmente, é pedido ao usuário os coeficientes a, b, c da equação e seus valores são armazenados nas respectivas variáveis. O cifrão $ foi utilizado no *FORMAT* para colocar o cursor na linha, sinalizando que deve ser feita uma entrada de dados pelo usuário.
Em seguida, o Δ é calculado e armazenado na variável. Se $\Delta > 0$, o programa calcula as duas raízes e as mostra. Se $\Delta = 0$, o programa calcula a única raíz e a mostra. Se $\Delta < 0$, o programa exibe uma mensagem de erro, pois não existem raízes reais e não mostra nenhum valor. Por fim, o programa termina.

2.1 Exemplo de funcionamento

Na figura 2, são expostos alguns testes de entrada/saída do programa da tarefa 1, para vários valores de coeficientes a, b, c evidenciando os casos $\Delta = 0$, $\Delta > 0$ e $\Delta < 0$.

Figura 2: Testes de entrada e saída da tarefa 1



3 Tarefa 2

Nessa tarefa, foi criado um programa que, dados dois vetores do espaço vetorial $\mathbb{R}^3$, calcula a área do triângulo formado por esses vetores. Sejam dois vetores $\vec{u} = (u_x, u_y, u_z)$ e $\vec{v} = (v_x, v_y, v_z)$ do $\mathbb{R}^3$, a área do triângulo formado entre eles é dada pela equação 6, onde $\vec{R} = \vec{u} \wedge \vec{v}$ é o produto vetorial entre $\vec{u}$ e $\vec{v}$, dado pela equação 4 e $\|\vec{u} \wedge \vec{v}\|$ é a norma do vetor resultante $\vec{R}$ do produto vetorial, dada pela equação 5. Os vetores só formam um triângulo se eles não forem colineares, ou seja, se a norma do produto vetorial for diferente de zero.

$$\vec{R} = \vec{u} \wedge \vec{v} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} = \hat{x}(u_y v_z - u_z v_y) + \hat{y}(u_z v_x - u_x v_z) + \hat{z}(u_x v_y - u_y v_x) \quad (4)$$

$$\left\| \vec{R} \right\| = \|\vec{u} \wedge \vec{v}\| = \sqrt{R_x^2 + R_y^2 + R_z^2} \quad (5)$$

$$A = \frac{1}{2} \|\vec{u} \wedge \vec{v}\| \quad (6)$$

Na figura 3, é exibido o código-fonte dessa tarefa e, logo abaixo, no verbete 2, o código é explicado de forma sucinta.

Figura 3: Código-fonte da tarefa 2

```fortran
      program Triangulo
c     Tarefa 2 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         dimension v1(3)
         dimension v2(3)
         dimension vetorial(3)
         write(*,*) "Digite as componentes x,y,z do vetor v1:"
         read(*,*) v1
         write(*,*) "Digite as componentes x,y,z do vetor v2:"
         read(*,*) v2
c     Calcula o produto vetorial v1 x v2
         vetorial(1) = v1(2)*v2(3) - v2(2)*v1(3) ! i*(y1*z2 - y2*z1)
         vetorial(2) = v1(3)*v2(1) - v2(3)*v1(1) ! j*(z1*x2 - z2*x1)
         vetorial(3) = v1(1)*v2(2) - v2(1)*v1(2) ! k*(x1*y2 - x2*y1)
c     Area = 0.5*modulo(vetorial)
         area = 0.5e0 * sqrt(vetorial(1)**2 + vetorial(2)**2 +
     &   vetorial(3)**2)

         write(*,*) "Área:", area
      end program
```

Verbete 2: Explicação do código da tarefa 2

Inicialmente, são alocados os vetores com três dimensões reais, $v1$, $v2$ e $vetorial$. Depois, é perguntado ao usuário as componentes x, y e z dos vetores $v1$ e $v2$ que compõem o triângulo.
Com os valores obtidos, o produto vetorial é calculado utilizando a equação 4 e armazenado no vetor $vetorial$.
Por fim, calcula-se a norma do produto vetorial com a equação 5, utilizando a função nativa *SQRT* do *Fortran 77*, e a área utilizando a equação 6. Após efetuado o cálculo, a área é mostrada ao usuário.

3.1 Exemplo de funcionamento

Na figura 4, são realizados alguns testes de entrada/saída no programa da tarefa 2, utilizando diversos valores para as coordenadas dos vetores que formam o triângulo, para os casos onde eles são colineares ou não colineares.

Figura 4: Testes de entrada e saída da tarefa 2

4 Tarefa 3

Nessa tarefa, foi criado um programa que lê um arquivo de entrada com N números reais, e em seguida, ordena apenas os M primeiros valores lidos, escrevendo-os em um arquivo de saída, junto com o número M. Para ordenar os números, foi utilizado um algoritmo baseado no famoso *selection sort*. A explicação do algoritmo é exibida no verbete 3. O código-fonte do programa escrito está exposto na figura 5 e a explicação sucinta do código é provida no verbete 4.

Verbete 3: Explicação do algoritmo de ordenação da tarefa 3

Seja A uma lista com n números reais, indexada com início em 1.
Para ordenar os m primeiros números reais de A, usamos o seguinte algoritmo:

1. Para cada elemento $A[i], i \in [1, m]$, estabeleça um chute inicial d para o menor número, por exemplo, $d = A[1]$.
2. Para cada elemento $A[j], j \in [i+1, n]$, verifique se ele é menor que d. Se for maior, então trocamos o valor de d e $A[j]$.
3. Quando $j = n$, salvamos o valor final de d no arquivo, pois esse é o mínimo global da seção $[i+1, n]$ da lista A.

Após terminados os laços de repetição, teremos no arquivo de saída os m menores números da lista, pois, a cada iteração de i, descobrimos o mínimo global da lista desconsiderando os valores com índice menor que $i+1$ (que já foram ordenados).

Figura 5: Código-fonte da tarefa 3

```
      program MenoresNumeros
c     Tarefa 3 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)

c     Como não é permitido usar alocação dinâmica, então
c     define-se um valor máximo de tamanho para o vetor.
         parameter(maxlinhas = 1000)
         dimension alista(maxlinhas)
         write(*,100)
 100     format("Quantos M menores números você quer na saída: ", $)
         read(*,*) m
c     Lê arquivo de input e coloca dentro de uma lista
         ilinhas = 0
         open(unit=50, file='entrada-3-13687303', status='old')
         do
            read(50, *, end=10) alista(ilinhas+1)
            ilinhas = ilinhas + 1
         end do
 10      close(unit=50)
c     Ordena os M primeiros e salva em um arquivo de output
         open(unit=51, file='saida-3-13687303', status='new')
         write(51, *) "M", m
         do i=1, m, 1
            amenor = alista(1)
            do j=i+1, ilinhas, 1
               if (alista(j) < amenor) then
                  aux = amenor
                  amenor = alista(j)
                  alista(j) = aux
               end if
            end do
            write(51, *) amenor
         end do
         close(unit=51)
      end program
```

Verbete 4: Explicação do código da tarefa 3

Inicialmente, define-se um parâmetro *maxlinhas*, que indica o número máximo de reais no arquivo. Esse parâmetro é utilizado para declarar estaticamente um vetor *alinhas*, que irá armazenar os valores do arquivo.
Esse método de declaração estática foi utilizado pois não era permitido utilizar alocação dinâmica nesse projeto. Todavia, vale notar que, a fim de evitar a reescrita do código-fonte para permitir outros tamanhos de arquivo, é recomendado utilizar a alocação dinâmica do vetor, com os comandos *ALLOCATE* e *DEALLOCATE*.
Em seguida, o arquivo *"entrada-3-13687303"* é lido e conta-se o número de linhas dele utilizando o contador *ilinhas* e os valores do arquivos são armazenados no vetor. A contagem de linhas é necessária pois, como o vetor é alocado estaticamente, se acessarmos um índice maior que o número de linhas do arquivo, obteremos valores de "lixo" da memória, que são flutuações nos espaços não utilizados do vetor. Após ler o arquivo, ele é fechado.
Nesse código, a marcação de *STATUS='old'* foi utilizada na hora da abertura do arquivo (*OPEN*), para sinalizar que ele é um arquivo de entrada que já deveria estar presente no diretório antes da execução e *STATUS='new'* para sinalizar que ele é um arquivo de saída que não deve existir no diretório antes da execução. Assim, se os arquivos não satisfazerem essas condições, o compilador do *Fortran 77* acusará um erro, avisando o usuário.
Por fim, abre-se o arquivo de saída *"saida-3-13687303"* e então, o algoritmo de ordenação do verbete 3 é executado, e durante sua execução, os menores valores são escritos no arquivo. Após terminado o algoritmo, fecha-se o arquivo e encerra-se o programa.

4.1 Exemplo de funcionamento

Nas figuras 6, 7 e 8 são expostas a entrada/saída do terminal, o arquivo de entrada do programa, e o arquivo de saída do programa, respectivamente, a fim de testar o resultado do programa para uma entrada específica. Ao observar as figuras, pode-se perceber que o programa funcionou corretamente, ordenandos os 6 primeiros menores números do arquivo de entrada, lidando bem com os casos de números repetidos e números negativos.

Figura 6: Entrada e saída do terminal para a tarefa 3

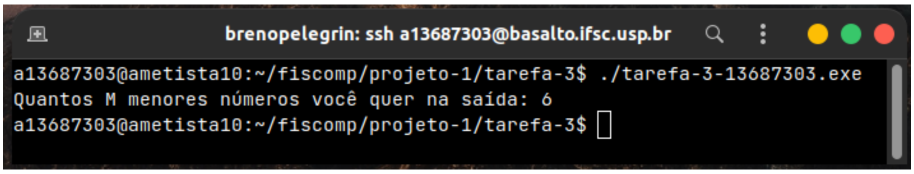


Figura 7: Arquivo de entrada da tarefa 3

```
1.125
2.45
3.367775
0.0
0.0
5.2
98.0
7.0
-4.0
-5.0
-10.0
-8.0
-8.0
4.0
9.0
12.0
```

Figura 8: Arquivo de saída da tarefa 3

```
 M           6
  -10.0000000
  -8.00000000
  -8.00000000
  -5.00000000
  -4.00000000
   0.00000000
```

5 Tarefa 4

Nessa tarefa, foi criado um programa que, dado um número inteiro N informado pelo usuário, calcula todos os números primos menores ou iguais a N e salva esses números em um arquivo de saída. Um número primo é um inteiro $x > 2$ que só é divisível por 1 e por x, ou seja, por 1 e por ele mesmo. Assim, para verificar se um número x qualquer é um número primo, basta verificar se ele é divisível por $(x-1, x-2, x-3, \dots, x-n)$ até $x-n=2$. Se for divisível, então o resto da divisão inteira (% ou MOD) de x por $x-n$ será zero e, portanto não é um número primo. Não é necessário verificar se ele é divisível por 1 ou por x pois todo inteiro é divisível por 1 e por ele mesmo.

O código feito para essa tarefa é exposto na figura 9 e uma explicação sucinta para ele é dada no verbete 5.

Figura 9: Código-fonte da tarefa 4

```
      program NumerosPrimos
c     Tarefa 4 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         write(*,100)
 100     format("Digite um inteiro N: ", $)
         read(*,*) n
         open(unit=50, file='saida-4-13687303', status='new')
c     Se n <= 1, não existem primos.
         if (n .gt. 1) then
            iatual = 2
c     Laço while (numero_atual <= numero_maximo)
 101        if(iatual .le. n) then
c     Todo número é divisível por 1 e por ele mesmo, então
c     podemos excluir esses casos para otimizar o código.
               do i=2, iatual-1, 1
                  if(MOD(iatual, i) .eq. 0) then
                     iatual = iatual+1
                     goto 101
                  endif
               enddo
c     Como o laço concluiu sem redirecionar, então é um número
c     primo.
               write(50,*) iatual
               iatual = iatual+1
               goto 101
            endif
         else
            write(*,*) "Não existe primo <= 1"
         endif
         close(unit=50)
      end program
```

Verbete 5: Explicação do código da tarefa 4

Inicialmente, pede-se ao usuário o inteiro N e abre-se o arquivo de saída “saida-4-13687303”, utilizando a flag *STATUS='new'*. Em seguida, é verificado se $N > 1$, pois não existem primos menores ou iguais a 1. Caso $N < 1$, o programa exibe uma mensagem de erro e encerra. Caso contrário, a execução continua.
Após, é iniciado um laço de repetição *while* (identificado pelo *label* 101), com um contador $iatual$ que irá contar de 2 até N, e a cada repetição, o laço verifica se $iatual \leq N$. Esse contador é o candidato à número primo.
Dentro do laço *while*, é utilizado um laço *do*, que itera o contador i de 2 a $iatual - 1$, verificando se a divisão inteira de $iatual$ por i é zero. Note que os casos $i = 1$ e $i = iatual$ são descartados, pois todo inteiro é divisível por 1 e por ele mesmo (um primo é divisível **apenas** por 1 e por ele mesmo)
Se for divisível, então $iatual$ não é um número primo, portanto incrementamos $iatual$ e retornamos ao label 100. Se não for divisível por nenhum i, então o laço *do* é encerrado sem redirecionar e $iatual$ é um número primo, que é então escrito no arquivo de saída “saida-4-13687303”.
Ao final do laço *while*, o programa terá terminado e todos os números primos menores ou iguais a N estarão escritos no arquivo, e então o arquivo é fechado.

5.1 Exemplo de funcionamento

Nas figuras 6, 7 e 8 são expostas a entrada/saída do terminal, e o arquivo de saída da tarefa 4, respectivamente, sendo possível observar os números primos gerados para o N especificado na entrada do terminal.

Figura 10: Entrada e saída do terminal para a tarefa 4

Figura 11: Arquivo de saída da tarefa 4

```
          2
          3
          5
          7
         11
         13
         17
         19
         23
         29
         31
         37
         41
         43
         47
         53
         59
```

5.2 Testes para $N = 100, 1000, 10000$

Em todos os casos, os arquivos de saída estavam coerentes. Eles não serão anexados no relatório pois são muitos números primos, o que iria ocupar um número grande de páginas. A fim de observar a velocidade do programa, as perguntas ao usuário foram removidas e N foi definido diretamente no código.

A figura 12 mostra o tempo de execução dos códigos *n1e2*, *n1e3* e *n1e4*, para $N = 10^2, 10^3, 10^4$, respectivamente, utilizando o comando *time* do *Linux*. Ao observar a figura, nota-se que o algoritmo executa em um tempo muito pequeno, na ordem de 0,01 s, mas conforme N aumenta, maior é o tempo de execução.

Figura 12: Testes para $N = 10^2, 10^3, 10^4$ na tarefa 4

6 Tarefa 5

Nessa tarefa, foi criado um programa para calcular o logaritmo natural de um número qualquer, utilizando uma expansão de Taylor centrada em 0, expressa na equação 7, que converge para números $x \in [0, 2]$. Inicialmente, o programa foi implementado utilizando inteiros de simples precisão. O cálculo da série foi feito utilizando um laço de repetição, até que a diferença entre o valor atual e o valor anterior seja menor ou igual a uma precisão pré-definida como 10^{-5}, chamada de *eprec*.

$$ln(x) = -\sum_{n=1}^{\infty}(1-x)^n \tag{7}$$

Como a série não converge para números maiores que 2, para $x > 2$, pode-se usar a propriedade de logaritmos expressa na equação 8, pois $\frac{1}{x}$ é sempre menor que 1 quando $x > 1$.

$$ln(x) = -ln\left(\frac{1}{x}\right) \tag{8}$$

O código-fonte da implementação do programa para precisão simples e usando $eprec = 10^{-5}$ é exposto na figura 13, e a explicação sucinta para o código é dada no verbete 6.

Figura 13: Código-fonte da tarefa 5 (precisão simples)

```
      program LnSimples
c     Tarefa 5 (simples precisão) - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         write(*,100)
 100     format("Digite x para calcular ln(x): ", $)
         read(*,*) x
         aLnFortran = log(x)

         if (x .le. 0.0e0) then
            write(*,*) "Erro: ln(x) não está definido para x <= 0."
            call exit()
         else if (x .le. 1.0d0) then
            aLn = fLnSerie(x)
         else if (x .gt. 1.0d0) then
c     Como a série só converge para x entre 0 e 2, podemos fazer
c     -ln(1/x) = ln(x), já que 1/x < 1 quando x > 1
            aLn = -1.0e0*fLnSerie(1.0e0/x)
         end if

         diff = abs(aLn - aLnFortran)
         write(*,*) "ln(x) simples precisão = ", aLn
         write(*,*) "ln(x) fortran simples precisão = ",
     &aLnFortran
         write(*,*) "Diferença: ", diff

      end program

      function fLnSerie(x)
c        Função válida apenas para 0 <= x <= 2
         parameter( eprec = 1.0E-5)
         soma = 0.0e0
         i = 1
         do
            anterior = soma
            soma = soma - ((1.0e0-x)**i)/i
            if(abs(soma - anterior) .lt. eprec) then
               exit
            else
               i = i + 1
            end if
         end do
         fLnSerie = soma
         return
      end function
```

Verbete 6: Explicação do código da tarefa 5 (precisão simples)

Inicialmente, é perguntado ao usuário o valor de x para calcular $ln(x)$. Em seguida, se $x \leq 0$, o programa exibe uma mensagem de erro e termina, chamando a subrotina *exit* nativa do *Fortran 77* para efetuar um *graceful shutdown*.
A fim de comparar depois o resultado do $ln(x)$ do programa com a função nativa *LOG(x)* do fortran, é calculado o *LOG(x)* e armazenado em uma variável.
Caso $x \in [0, 1]$, o programa irá chamar a função $fLnSerie(x)$, responsável por calcular o $ln(x)$ utilizando a série da equação 7. Caso $x > 1$, o programa irá calcular o $ln(x)$ fazendo $ln(x) = -ln\left(\frac{1}{x}\right)$, conforme exposto na equação 8.
Por fim, é exibido ao usuário o $ln(x)$ calculado pelo programa, o $ln(x)$ calculado utilizando a função nativa *LOG(x)* e a diferença absoluta entre os dois resultados.
A função $fLnSerie(x)$ foi implementada da seguinte forma:
Primeiro, define-se o parâmetro $eprec = 10^{-5}$, e uma variável para armazenar o resultado do somatório, chamada $soma = 0$.
Depois, é realizado um laço de repetição enquanto a diferença entre o termo atual e o termo anterior for maior que *eprec*. Dentro do laço, é somado o novo termo da série na variável *soma*. Quando a condição de parada é satisfeita, o programa sai do laço de repetição e retorna o valor obtido pela série.

6.1 Dupla precisão

O código-fonte da implementação do programa para precisão dupla e usando *eprec* como uma variável informada pelo usuário é exposto na figura 14, e a explicação sucinta para o código é dada no verbete 7.

Figura 14: Código-fonte da tarefa 5 (precisão dupla)

```fortran
      program LnDupla
         implicit real*8 (a-h, o-z)
c     Tarefa 5 (simples precisão) - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         write(*,100)
 100     format("Digite o valor de eprec: ", $)
         read(*,*) eprec

         write(*,101)
 101     format("Digite x para calcular ln(x): ", $)
         read(*,*) x

         aLnFortran = dlog(x)

         if (x .le. 0.0d0) then
            write(*,*) "Erro: ln(x) não está definido para x <= 0."
            call exit()
         else if (x .le. 1.0d0) then
            aLn = fLnSerie(x, eprec)
         else if (x .gt. 1.0d0) then
c     Como a série só converge para x entre 0 e 2, podemos fazer
c     -ln(1/x) = ln(x), já que 1/x < 1 quando x > 1
            aLn = -1.0d0*fLnSerie(1.0d0/x, eprec)
         end if

         diff = abs(aLn - aLnFortran)
         write(*,*) "ln(x) dupla precisão = ", aLn
         write(*,*) "ln(x) fortran dupla precisão = ",
     &aLnFortran
         write(*,*) "Diferença: ", diff

      end program

      function fLnSerie(x, eprec)
c        Função válida apenas para 0 <= x <= 2
         implicit real*8 (a-h, o-z)
         soma = 0.0d0
         i = 1
         do
            anterior = soma
            soma = soma - ((1.0d0-x)**i)/i
            if(abs(soma - anterior) .lt. eprec) then
               exit
            else
               i = i + 1
            end if
         end do
         fLnSerie = soma
         return
      end function
```

Verbete 7: Explicação do código da tarefa 5 (precisão dupla)

O código implementado para dupla precisão é extremamente parecido com o exposto no verbete 7, e portanto, não será explicado novamente. Apenas algumas modificações foram feitas:
Ao invés de definir o parâmetro *eprec*, a função $fLnSerie$ recebe esse parâmetro como argumento do programa principal. No programa principal, a variável *eprec* é perguntada ao usuário, e então passada para a função $fLnSerie(x, eprec)$ no momento em que o $ln(x)$ é calculado.
Além disso, todas as variáveis reais do programa foram definidas com dupla precisão.

6.2 Exemplo de funcionamento

Na figura 15, são exibidos alguns casos de teste para $x = 0, 5, 1, 5, 10, 1000$ do programa de precisão simples com $eprec = 10^{-5}$ em todos os casos, e na figura 16 são exibidos os mesmos testes, mas utilizando o programa de precisão dupla, com $eprec = 10^{-16}$ em todos os casos.

Figura 15: Entrada e saída do terminal para a tarefa 5 (precisão simples)

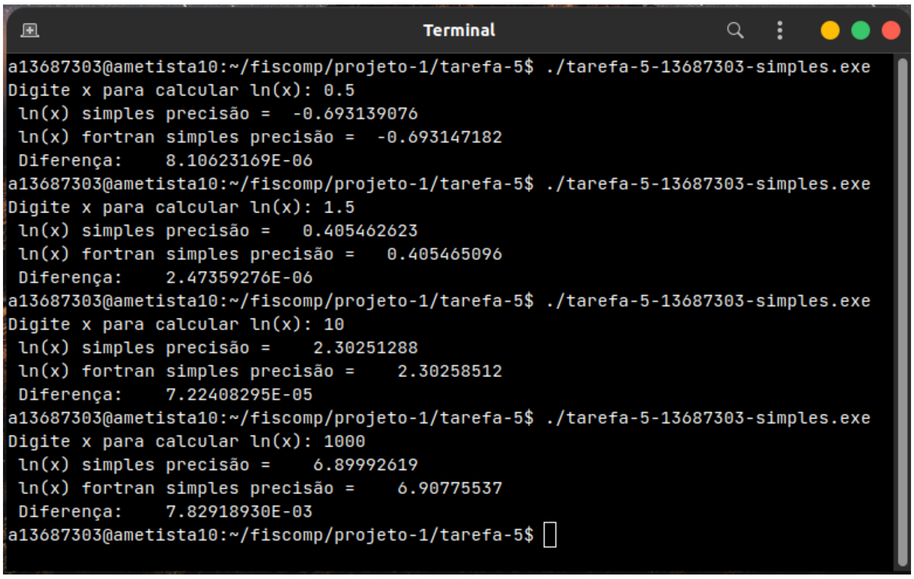
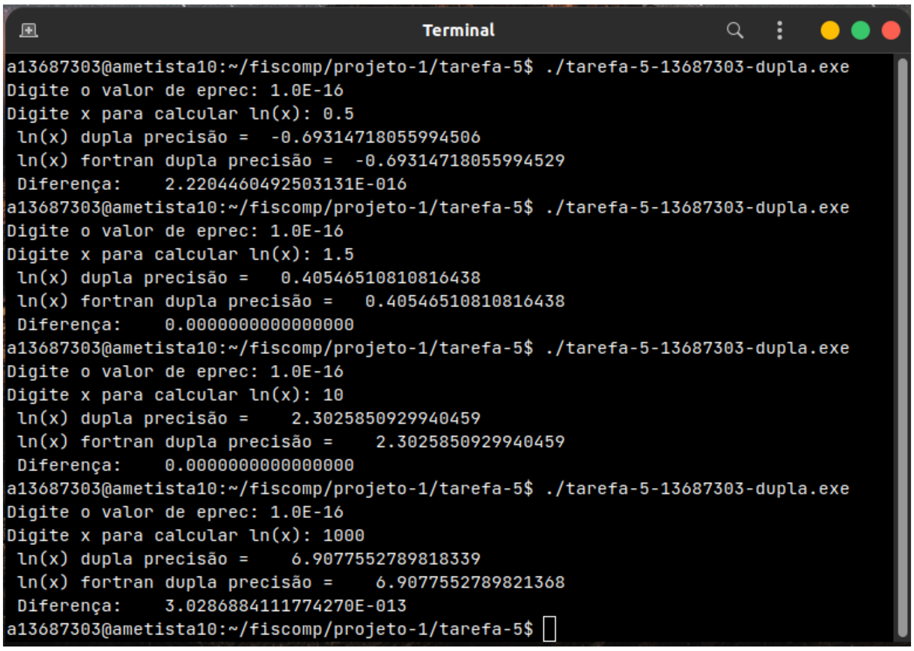


Figura 16: Entrada e saída do terminal para a tarefa 5 (precisão dupla)



Analisando as figuras, é possível observar que, no caso da precisão simples, existe uma diferença significativa entre o $ln(x)$ do programa e o $ln(x)$ calculado pela função *LOG(x)* do *Fortran 77*. Essa diferença se acentua quanto maior é o argumento x da função.

Já no caso da precisão dupla, observa-se que, para pequenos valores de x (até $x = 10$), a diferença entre o $ln(x)$ do programa de dupla precisão e a função *DLOG(x)* do *Fortran 77* é nula. Todavia, para valores maiores, na ordem de $x = 10^3$, existe uma diferença na ordem de 10^{-13}, que é desprezível. Essa diferença acontece porque $\frac{1}{x}$ tende a 0 quando x tende ao infinito, e assim, se aproxima muito do limite de convergência da série.

7 Tarefa 6

Nessa tarefa, foi codificado um programa para calcular os números complexos z que satisfazem a equação 9, com N inteiro maior que zero.

$$(z-2)^N = 3 \tag{9}$$

Exponenciando os dois lados da equação e, depois, somando 2 nos dois lados, obtém-se a expressão abaixo.

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

Todavia, utilizando a forma de Euler para um número complexo, exposta na equação

10, é possível expressar o número real 3 como um complexo de parte imaginária igual a zero, e de módulo $\rho = 3$.

Para a parte imaginária ser zero, o argumento do seno deve ser zero, então $\theta = 2k\pi$, com $k \in \mathbb{N}^*$. É necessário utilizar um inteiro k no argumento pois as funções seno e cosseno são cíclicas, logo $cos(2k\pi) = 1$ e $sin(2k\pi) = 0$, $\forall k \in \mathbb{N}^*$. Assim, o número complexo $w = (3 + 0i)$ pode ser expresso utilizando a equação 11.

$$\rho e^{i\theta} = \rho \left[cos(\theta) + isin(\theta)\right] \tag{10}$$

$$3 = 3e^{2k\pi i} = 3 \left[cos(2k\pi i) + isin(2k\pi i)\right] \tag{11}$$

Exponenciando os dois lados da equação 11 por $\frac{1}{N}$, obtém-se a equação 12. Todavia, é necessário impor um limite ao inteiro k, para evitar repetir as raízes, já que z terá 6 raízes. Assim, $k \in [1, N]$, $k \in \mathbb{N}^*$. Por fim, para obter a forma trigonométrica do número complexo $z = 3^{\frac{1}{N}} + 2$, basta somar (2 + 0i) na expressão obtida para $(3^{\frac{1}{N}} + 0i)$, obtendo, por fim, a equação 13.

$$(3^{\frac{1}{N}} + 0i) = 3^{\frac{1}{N}} e^{\frac{2k\pi}{N}} = \left[3^{\frac{1}{N}} cos\left(\frac{2k\pi}{N}\right)\right] + i\left[3^{\frac{1}{N}} sin\left(\frac{2k\pi}{N}\right)\right],\ k \in [1, N], k \in \mathbb{N}^* \tag{12}$$

$$z = \left[3^{\frac{1}{N}} cos\left(\frac{2k\pi}{N}\right) + 2\right] + i\left[3^{\frac{1}{N}} sin\left(\frac{2k\pi}{N}\right)\right],\ k \in [1, N], k \in \mathbb{N}^* \tag{13}$$

Dessa forma, o problema reduz-se a resolver a equação 13, dado o número N informado pelo usuário. O código-fonte do programa elaborado é exposto na figura 17, e uma explicação sucinta de seu funcionamento é dada no verbete 8.

Figura 17: Código-fonte da tarefa 6

```fortran
      program Complexos
c     Tarefa 6 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         complex :: z
         write(*,100)
 100     format('Digite o valor de N: ', $)
         read(*,*) n
         pi = 4.0e0*atan(1.0e0)
         do i=1, n, 1
c     Transforma Re(z) e Im(z) em um número complexo do fortran
            zmod = 3.0e0**(1.0e0/N)
            preal = 2 + zmod*cos(2.0e0*pi*i/N)
            pimag = zmod*sin(2.0e0*pi*i/N)
            z = cmplx(preal, pimag)
            write(*,101) i, z
 101        format('z'(1i0) ' = ' (1f12.6) ' + ' (1f12.6)' i')
         enddo
      end program
```

Verbete 8: Explicação do código da tarefa 6

Inicialmente, pergunta-se ao usuário o inteiro N. Em seguida, define-se π para ser usado posteriormente nos cálculos, utilizando precisão simples.
Após, é implementada a equação 13, utilizando um laço *do* para variar k (no programa, é chamado de i) de 1 a N. A variável *zmod* armazena $3^{\frac{1}{N}}$, que é o módulo de $(3^{\frac{1}{N}} + 0i)$ e as variáveis *preal* e *pimag* armazenam as partes real e imaginária de z, respectivamente.
Para cada iteração, o programa gera um número complexo z a partir das partes real e imaginária, utilizando a função *CMPLX* do *Fortran 77*, e mostra o número na tela, na forma cartesiana $z_i = Re(z) + iIm(z)$.

7.1 Exemplo de funcionamento

Na figura 18, é exposto um exemplo de funcionamento do programa para $N = 6$.

Figura 18: Entrada e saída do terminal para a tarefa 6 para $N = 6$

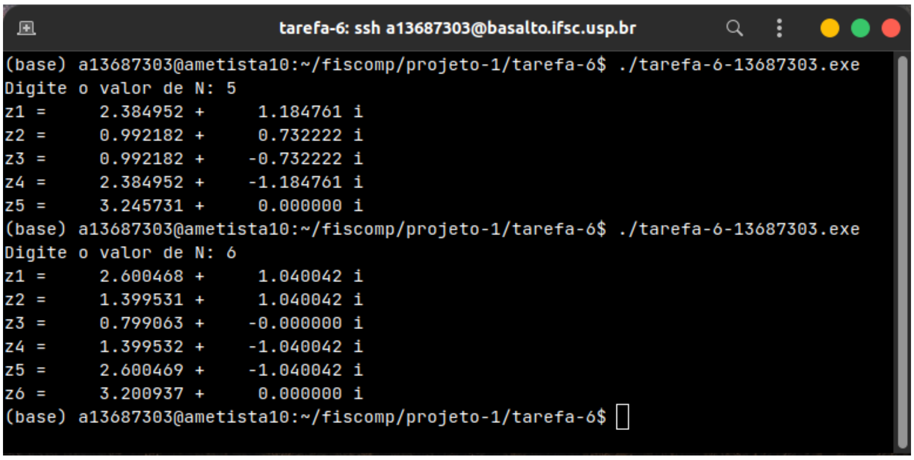
```
tarefa-6: ssh a13687303@basalto.ifsc.usp.br
(base) a13687303@ametista10:~/fiscomp/projeto-1/tarefa-6$ ./tarefa-6-13687303.exe
Digite o valor de N: 6
z1 =     2.600468 +     1.040042 i
z2 =     1.399531 +     1.040042 i
z3 =     0.799063 +    -0.000000 i
z4 =     1.399532 +    -1.040042 i
z5 =     2.600469 +    -1.040042 i
z6 =     3.200937 +     0.000000 i
(base) a13687303@ametista10:~/fiscomp/projeto-1/tarefa-6$
```

7.2 Testes para $N = 1, 2, 3, 4, 5, 6$

Nas figuras 19 e 20, são expostos os testes do programa para N variando de 1 a 6.

Figura 19: Entrada e saída do terminal para a tarefa 6 para N de 1 a 4

Figura 20: Entrada e saída do terminal para a tarefa 6 para N de 5 a 6



8 Tarefa 7

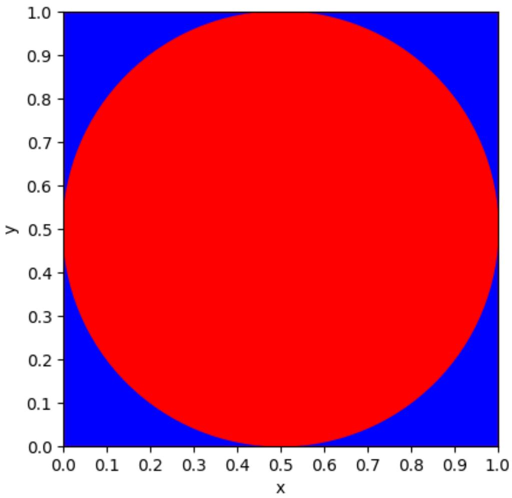
Nessa tarefa, foi criado um programa para calcular o volume de uma d-esfera em um espaço euclidiano de d dimensões. Para isso, foi utilizado o algoritmo de Monte Carlo generalizado para d dimensões. Inicialmente, suponha $d = 2$ para simplificar o raciocínio e considere um espaço euclidiano E^2 dotado de uma métrica euclidiana e um sistema de coordenadas (x, y) com origem $(0, 0)$. Assim, a distância entre um ponto $p_2 = (x_2, y_2)$ e outro ponto $p_1 = (x_1, y_1)$ nesse espaço é dada pela equação 14.

$$f(p_2, p_1) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{14}$$

Seja Q um quadrado (em azul) de lado $L = 1$ centrado em $(0,5,0,5)$ e uma circunferência C (em vermelho) de raio $R = 0,5$ inscrita nesse quadrado, ou seja, centrada em $(0,5,0,5)$, conforme mostra a figura 21. Suponha um ponto aleatório $p_n = (x_n, y_n)$ com coordenadas variando de 0 a 1, ou seja, dentro do quadrado. Assim, o ponto p_n está dentro da circunferência se a sua distância em relação ao centro da circunferência é menor que o raio da circunferência, ou seja, se a equação 15 é satisfeita.

$$f((x_n, y_n), (0,5,0,5)) = \sqrt{(x_n - 0,5)^2 + (y_n - 0,5)^2} < 0,5 \tag{15}$$

Figura 21: Geometria do método de Monte Carlo para $d = 2$



Fonte: Autoria própria

A probabilidade P_{circ} de um ponto aleatório p_n cair dentro da circunferência é dada pela equação 16, e no caso onde são gerados N pontos aleatórios p_n com $n \in [1, N]$, a razão entre o número de pontos n_{dentro} que caíram dentro da circunferência e o número de pontos aleatórios gerados n_{total} será proporcional à razão entre a área da circunferência A_{circ} e a área do quadrado A_{quad}. No limite em que $n \to \infty$, a razão não é apenas proporcional, mas exatamente igual, conforme mostra a equação 17.

$$P_{circ} = \frac{A_{circ}}{A_{quad}} \propto \frac{n_{dentro}}{n_{total}} \tag{16}$$

$$\lim_{n\to\infty} \left(\frac{n_{dentro}}{n_{total}} \right) = P_{circ} = \frac{A_{circ}}{A_{quad}} = \frac{\pi R^2}{1} \tag{17}$$

Generalizando para d dimensões, devem ser gerados pontos aleatórios p_n de coordenadas $(a_1, a_2, \ldots, a_d)$ em um espaço euclidiano E^d, com um sistema de coordenadas $(e_1, e_2, \ldots, e_d)$ de origem no ponto nulo (todas as coordenadas iguais a zero). Nesse espaço, a distância entre dois pontos $p_2 = (a_1, a_2, \ldots, a_d)$ e $p_1 = (b_1, b_2, \ldots, b_d)$ é dada pela métrica da equação 18.

$$f(p_2, p_1) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \ldots + (b_d - a_d)^2} \tag{18}$$

Assim, uma d-esfera nesse espaço é um objeto geométrico caracterizado por uma única propriedade: o conjunto de todos os pontos de E^d que distam R do centro da d-esfera. Dessa forma, para um ponto p_n qualquer estar dentro da d-esfera, é necessário que ele satisfaça a equação 19. No caso $d = 2$, uma esfera 2-dimensional é uma circunferência, que é o caso particular utilizado para formular o raciocínio para o método de Monte Carlo. Para d arbitrário, a equação 17 é reformulada considerando uma d-esfera, obtendo a equação 20.

$$f = \sqrt{(a_1 - 0,5)^2 + (a_2 - 0,5)^2 + \ldots + (a_d - 0,5)^2} < 0,5 \tag{19}$$

$$\lim_{n \to \infty} \left(\frac{n_{dentro}}{n_{total}} \right) = P_{d-esfera} = \frac{V_{d-esfera}}{V_{d-cubo}} = V_{d-esfera} \tag{20}$$

Utilizando essa generalização, foi possível criar o programa desejado, cujo código fonte é exposto na figura 22. A explicação sucinta para o código é dada logo em seguida, no verbete 9.

Figura 22: Código-fonte da tarefa 7

```
      program EsferasMonteCarlo
c     Tarefa 7 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
         write(*,100)
 100     format("Digite o numero M de pontos aleatórios: ", $)
         read(*,*) m
         write(*,101)
 101     format("Digite o numero d de dimensões: ", $)
         read(*,*) idim
         ndentro = 0
         ntotal = 0
         do i=1, m, 1
            dist = 0
c     Gera d coordenadas aleatórias e calcula a distância euclidiana
            do j=1, idim, 1
               coord = rand()
               dist = dist + (coord - 0.5e0)**2
            enddo
c     Se satisfaz a equação da d-esfera, então caiu dentro da d-esfera
            if(sqrt(dist) .lt. 0.5e0) then
               ndentro = ndentro + 1
               ntotal =  ntotal + 1
            else
               ntotal = ntotal + 1
            endif
         enddo
         volume = real(ndentro)/ntotal
         write(*,*) "Volume da d-esfera:", volume
      end program
```

Verbete 9: Explicação do código da tarefa 7

Inicialmente, o número de pontos aleatórios M a serem gerados é definido pelo usuário. Em seguida, inicializa-se o número de pontos dentro da d-esfera e pontos totais como zero.
Em seguida, é realizado um laço *do* com o iterador i de 1 a M, para gerar M pontos aleatórios. Dentro desse laço, é realizado outro laço *do*, com o iterador j de 1 a d, para gerar d coordenadas aleatórias para cada ponto i. Cada coordenada é gerada utilizando a função nativa *RAND* do *Fortran 77*, que gera um número aleatório real $x \in [0, 1)$.
Após gerar os números aleatórios, verifica-se se a distância $\sqrt{(x_1 - 0,5)^2 + (x_2 - 0,5)^2 + \ldots + (x_d - 0,5)^2} < 0,5$. Se for verdadeiro, então o ponto está dentro da d-esfera, e *ndentro* e *ntotal* são incrementados. Caso falso, o ponto está fora da d-esfera, e apenas *ntotal* é incrementado.
Por fim, o volume é calculado fazendo a razão entre *ndentro* e *ntotal* e é mostrado ao usuário. A função *REAL* foi utilizada para converter *ndentro* em um real e evitar a divisão inteira, já que o resultado da divisão deve ser um real.

8.1 Exemplo de funcionamento

Na figura 23 é exibido um exemplo de funcionamento do programa para 3 dimensões (uma esfera comum), utilizando 10^6 números aleatórios. A fim de observar o tempo de execução do programa para $d = 3$ e $M = 10^6$, um novo programa foi compilado, com as perguntas removidas e a dimensão e quantidade de números aleatórios inseridos diretamente no código. A entrada/saída para esse programa, com seu tempo de execução mensurado pelo comando *time* do *Linux*, é exibido na figura 24.

Figura 23: Entrada e saída do terminal para a tarefa 7 para $N = 10^6$ e $d = 3$

Figura 24: Tempo de execução da tarefa 7 para $M = 10^6$ e $d = 3$

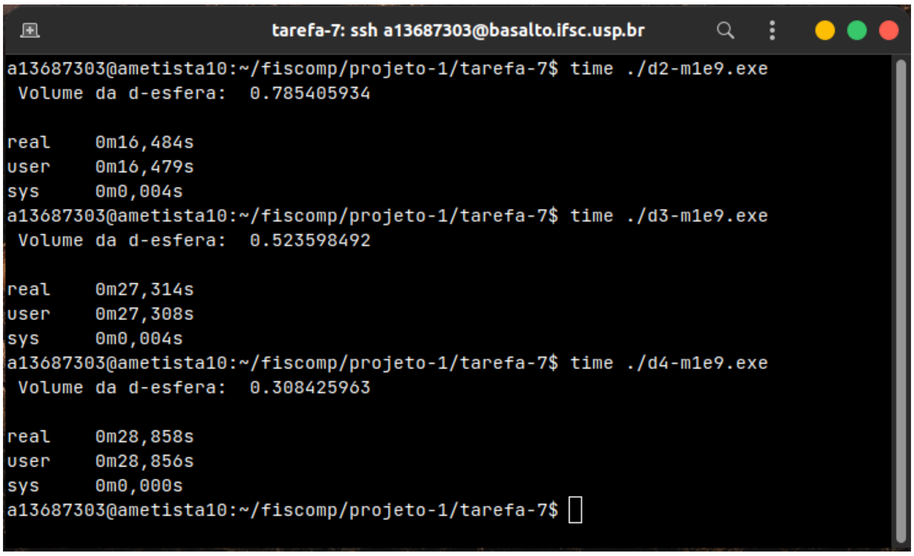
8.2 Testes para $d = 2, 3, 4$

O programa foi testado para d de 2 a 4, com o tempo de execução medido utilizando o comando *time* do *Linux* e os resultados para os volumes foram comparados com a expressão 21, a fim de verificar se estão corretos. Para os testes, as perguntas ao usuário foram removidas do código e a dimensão e quantidade de números aleatórios foram definidas diretamente no código.

As entradas/saídas do terminal para $d = 2, 3, 4$ com $M = 10^9$ (1 bilhão de números aleatórios) são exibidas na figura 25. Note que, na figura, os programas *d2-m1e9.exe*, *d3-m1e9.exe* e *d4-m1e9.exe* representam os casos $d = 2, 3, 4$, respectivamente.

$$V_d = \frac{\pi^{\frac{d}{2}} R^d}{\Gamma(\frac{d}{2} + 1)} \tag{21}$$

Figura 25: Entrada e saída do terminal para os testes da tarefa 7 com $M = 10^9$



Observando a figura 25, é possível concluir que o programa executa em um tempo razoável para 1 bilhão de pontos. No caso $d = 2$, executou em 16,5 s, para $d = 3$ em 27,3 s e para $d = 4$ em 28,8 s. A comparação dos resultados obtidos com a equação 21 é feita na tabela 1, onde são calculadas as variações percentuais entre os valores dos dois métodos, utilizando a equação 22. Ao observar a tabela 21, nota-se que a variação percentual entre os dois métodos é da ordem de $0,001$ a $0,0001$ %, então eles fornecem resultados praticamente iguais quando $M = 10^9$.

$$var\%(a, b) = 100 \cdot \frac{max(a, b) - min(a, b)}{max(a, b)} \tag{22}$$

Vale notar que, ao reduzir M para 10^6, conforme visto no exemplo da figura 24 para $d = 3$, o tempo de execução é extremamente baixo, na ordem de $0,01$ s, ou seja, se o usuário aceitar perder precisão, o programa pode executar de maneira extremamente rápida.

Tabela 1: Comparação entre Monte Carlo e expressão analítica do volume

d	Monte Carlo	Analítica	Variação percentual
2	$0,785405934$	$0,785398185$	$0,001\%$
3	$0,523598492$	$0,523598790$	$0,0001\%$
4	$0,308425963$	$0,308425158$	$0,0003\%$

Além de os dois métodos serem muito próximos, observa-se que seus resultados também são razoáveis, pois, para $d = 2$ e $d = 3$, dimensões nas quais conhecemos o volume da d-esfera, os resultados são muito próximo das expressões conhecidas para essas dimensões, conforme é exposto abaixo.

$$V_2 = \pi(0,5)^2 = 0,785398163$$

$$V_3 = \frac{4}{3}\pi(0,5)^3 = 0,523598775$$

9 Tarefa 8

Nessa tarefa, foi criado um programa que, dado o raio R e a dimensão d, calcula o volume de uma d-esfera em $2, 3, \ldots, d$ dimensões utilizando a expressão analítica da equação 23 e salva os volumes calculados em um arquivo de saída *"dimensoes-esferas"*. Algumas propriedades da função $\Gamma(x)$ foram providenciadas no enunciado da tarefa: $\Gamma(\frac{1}{2}) = \sqrt{pi}$, $\Gamma(1) = 1$ e $\Gamma(x + 1) = x\Gamma(x)$.

$$V_d = \frac{\pi^{\frac{d}{2}} R^d}{\Gamma(\frac{d}{2} + 1)} \tag{23}$$

Na expressão da equação 23, note que no denominador há $\Gamma\left(\frac{d}{2} + 1\right)$. Como $\Gamma(x + 1) = x\Gamma(x)$, então temos que $\Gamma\left(\frac{d}{2} + 1\right) = \frac{d}{2}\Gamma\left(\frac{d}{2}\right)$. Vamos chamar a expressão obtida de uma função da dimensão, $f(d) = \frac{d}{2}\Gamma\left(\frac{d}{2}\right)$. Para $d = 0$, $f(d)$ não está definida, pois $\Gamma(0)$ diverge para infinito.

Todavia, como $\frac{d}{2}\Gamma\left(\frac{d}{2}\right) = \Gamma\left(\frac{d}{2} + 1\right)$, então pode-se considerar que $f(0) = \Gamma\left(\frac{0}{2} + 1\right) = \Gamma(1) = 1$. Assim, temos a condição inicial da função, já que não existe $d < 0$, e eliminamos a inconsistência da função para $d = 0$. Analisando os valores de $f(d)$ para $d = 1, 2, \ldots, n$, é possível obter uma relação de recorrência para $f(d)$, expressa na equação 24.

$$f(0) = \Gamma\left(\frac{0}{2}+1\right) = 1$$

$$f(1) = \frac{1}{2}\Gamma\left(\frac{1}{2}\right) = \frac{1}{2}\sqrt{\pi}$$

$$f(2) = \frac{2}{2}\Gamma\left(\frac{2}{2}\right) = \frac{2}{2}f(0)$$

$$f(3) = \frac{3}{2}\Gamma\left(\frac{3}{2}\right) = \frac{3}{2}\Gamma\left(\frac{1}{2}+1\right) = \frac{3}{2}\left[\frac{1}{2}\Gamma\left(\frac{1}{2}\right)\right] = \frac{3}{2}f(1)$$

$$f(d) = \frac{d}{2}f(d-2), \text{ para } d \geq 2, \text{ com } f(0) = 1 \text{ e } f(1) = \frac{\sqrt{\pi}}{2} \tag{24}$$

Dessa forma, como a relação de recorrência possui condições iniciais e condição de parada bem definidas, o problema do cálculo do volume d-dimensional utilizando a expressão analítica pode ser reduzido a uma simples recursão, que pode ser transformada em um laço de repetição. A partir disso, escreveu-se o código-fonte do programa, exibido na figura 26. Uma explicação sucinta do código é exposta no verbete 10.

Figura 26: Código-fonte da tarefa 8

```
      program EsferaGamma
c     Tarefa 7 - Intro. Fiscomp - Prof. Alcaraz
c     Aluno: Breno Henrique Pelegrin da Silva (13687303)
c     Como não é permitido usar alocação dinâmica, então
c     define-se um valor máximo de tamanho para o vetor.
         parameter(maxdim=1000)
         dimension vetor(0:maxdim)
         pi = 4.0e0*atan(1.0e0)
c     Define os valores iniciais da função gamma(d/2 + 1)
c     do denominador da equação de volume
         vetor(0) = 1
         vetor(1) = 0.5e0*sqrt(pi)
         open(unit=50, file='dimensoes-esferas', status='new')
         write(*,100)
 100     format("Digite o numero de dimensões d e o raio R: ", $)
         read(*,*) idim, r
c     O exercício pede para escrever apenas os volumes 2,3,...,d no
c     arquivo, então d = 0 e d = 1 não serão escritos no arquivo.
         if(idim .ge. 2 .and. idim .le. maxdim) then
            do i=2, idim, 1
               vetor(i) = (i/2.0e0)*vetor(i-2)
               write(50, *) i, (pi**(i/2.0e0) * r**i)/vetor(i)
            enddo
         else if (idim .eq. 0) then
            write(*,*) "Volume:", vetor(0)
         else if (idim .eq. 1) then
            write(*,*) "Volume:", vetor(1)
         else if (idim .lt. 0) then
            write(*,*) "Erro: Não existe volume negativo."
            call exit()
         else if (idim .gt. maxdim) then
            write(*,*) "Erro: A máxima dimensão suportada é ", maxdim
            call exit()
         endif
         close(unit=50)
      end program
```

Verbete 10: Explicação do código da tarefa 8

Inicialmente, define-se um parâmetro *maxdim*, que representará a máxima dimensão suportada pelo programa. Esse parâmetro é necessário pois não é permitido utilizar alocação dinâmica de vetores, então o vetor utilizado para a recursão precisa ser alocado de maneira estática previamente. Após, armazena-se os valores iniciais de $f(0)$ e $f(1)$ em $vetor(0)$ e $vetor(1)$, define-se o valor da constante π e é aberto o arquivo *"dimensoes-esferas"* utilizando a flag *STATUS='new'*, pois é um arquivo de saída que não deveria existir antes da execução do programa.
Após, é perguntado ao usuário a dimensão d e o raio R. Se $d > maxdim$, o programá exibirá uma mensagem de erro e irá terminar. Se d for menor que 0, o programa também irá mostrar um erro e terminar, pois não existe d-esfera com d negativo. Para terminar o programa em caso de erros, foi utilizada a subrotina *exit* nativa do *Fortran 77*, quem é responsável por fechar todas as unidades, limpar a memória e encerrar o programa.
Se $d = 0$ ou $d = 1$, o programa mostra o volume na tela e encerra, mas não escreve nada no arquivo, pois foi solicitado no enunciado da tarefa para imprimir apenas os valores para $2, 3, \ldots, d$ dimensões. Caso $d \geq 2$, o algoritmo recurso será executado utilizando um laço *do* com o contador i de 2 a d e passo 1, para calcular os volumes de 2 a d. A cada i, é calculado o volume fazendo $V_i = \frac{\pi^{\frac{d}{2}} R^d}{\frac{d}{2} vetor(i-2)}$, e então ele é escrito no arquivo *"dimensoes-esferas"*, ao lado do contador i para graficar V em função de i posteriormente.

9.1 Exemplo de funcionamento e análise gráfica

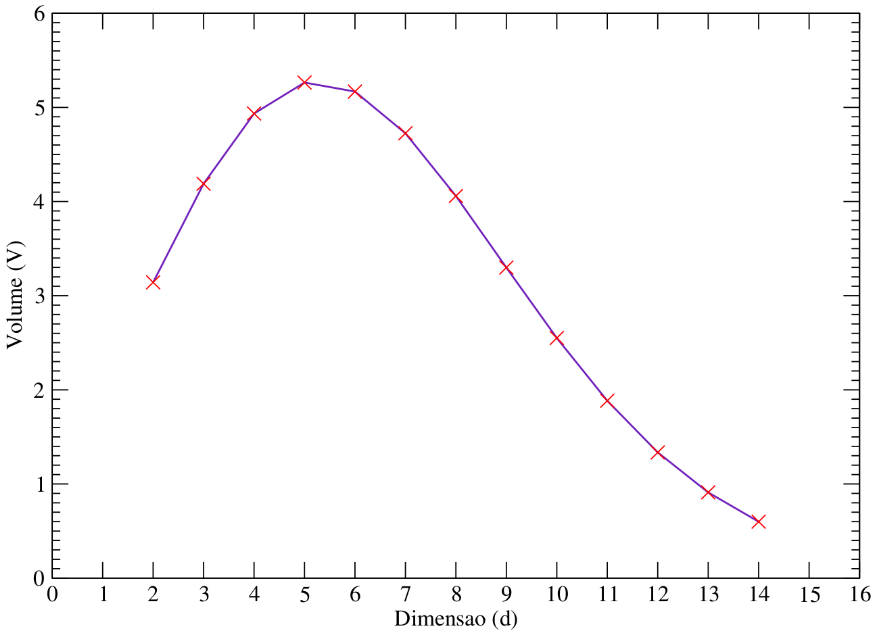
Na figura 27 é exibido um exemplo de entrada/saída do terminal para o programa da tarefa 8, e na figura 28, é exibido o conteúdo do arquivo de saída *"dimensoes-esferas"* para $d = 14$ e $R = 1$. Note que, no último comando do terminal, foi feito um teste de tempo de execução do programa utilizando o comando *time*, sendo possível observar que o programa para a função analítica executa extremamente mais rápido que o programa da Tarefa 7. Na figura 29, é exibido o gráfico dos volumes em função da dimensão, elaborado no *xmgrace* utilizando os dados do arquivo de saída.

Figura 27: Entrada e saída do terminal para a tarefa 8

Figura 28: Arquivo de saída da tarefa 8

```
           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
```

Figura 29: Gráfico do volume em função da dimensão

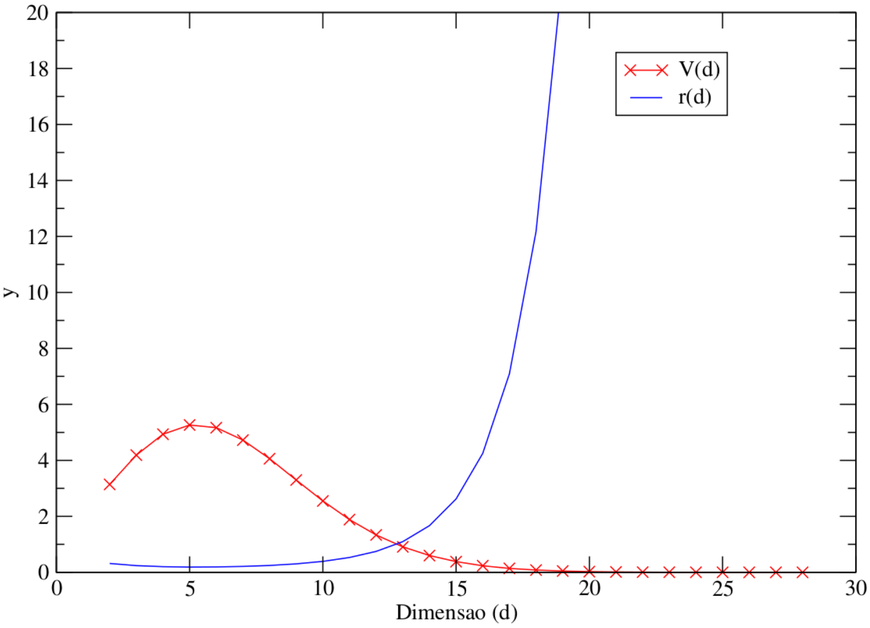


9.2 Pergunta A

Se um cubo d-dimensional tem volume $1m^d$, a razão r_d entre o volume do d-cubo e da d-esfera em uma dimensão d arbitrária, dada pela equação 25, mede quantas vezes o volume do d-cubo é maior que o volume da d-esfera. Calculando essa razão para d de 2 a 50 e graficando os resultados no *xmgrace*, obtém-se o gráfico da figura 30. Na figura, o eixo y representa o valor de V_d e r_d.

$$r_d = \frac{1}{V_d} \tag{25}$$

Figura 30: Gráfico de V_d e r_d em função da dimensão



A partir do gráfico, é possível observar que, quando $d \to \infty$, a razão $r_d \to \infty$, ou seja, o volume do d-cubo torna-se cada vez maior que o volume da d-esfera. Isso se deve ao fato de que, quanto maior a dimensão, mais difícil é um ponto do espaço satisfazer a equação da d-esfera, então o volume torna-se cada vez mais próximo de zero.

$$\lim_{d \to \infty} r_d = +\infty$$

$$\lim_{d \to \infty} V_d = 0$$

9.3 Pergunta B

O número de Avogadro N_A é um fator de conversão entre o microscópico e o macroscópico: ele pode ser definido como a quantidade de átomos de um elemento necessária para que a massa em gramas de todos os átomos seja numericamente igual à massa atômica de um átomo individual, assim, 1 mol de uma substância é a quantidade de átomos necessários para transformar x unidades de massa atômica em x gramas.

Se um volume macroscópico típico d-dimensional é 1 mm^d (ou seja, 10^{-3d}) e o volume de um átomo d-dimensional é Å^d (ou seja, 10^{-10d}), então o número de Avogadro nessa dimensão será a razão entre o volume macroscópico e o microscópico, como mostra a equação 26. Assim, para $d = 3$, temos que $N_A = 10^{21}$, que está próximo do número esperado, $6,02 \times 10^{23}$.

$$N_A = \frac{10^{-3d}}{10^{-10d}} = 10^{7d} \tag{26}$$