Universidade de São Paulo

Introdução à Física Computacional - Projeto 1

João Victor Dell Agli Floriano - 10799783

1. Programa 1 - Raízes da Eq. de Segundo Grau

Dada a equação de segundo grau $ax^2 + bx + c = 0$, calcular suas raízes não é um trabalho muito complicado, podendo recorrer às fórmulas já conhecidas:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

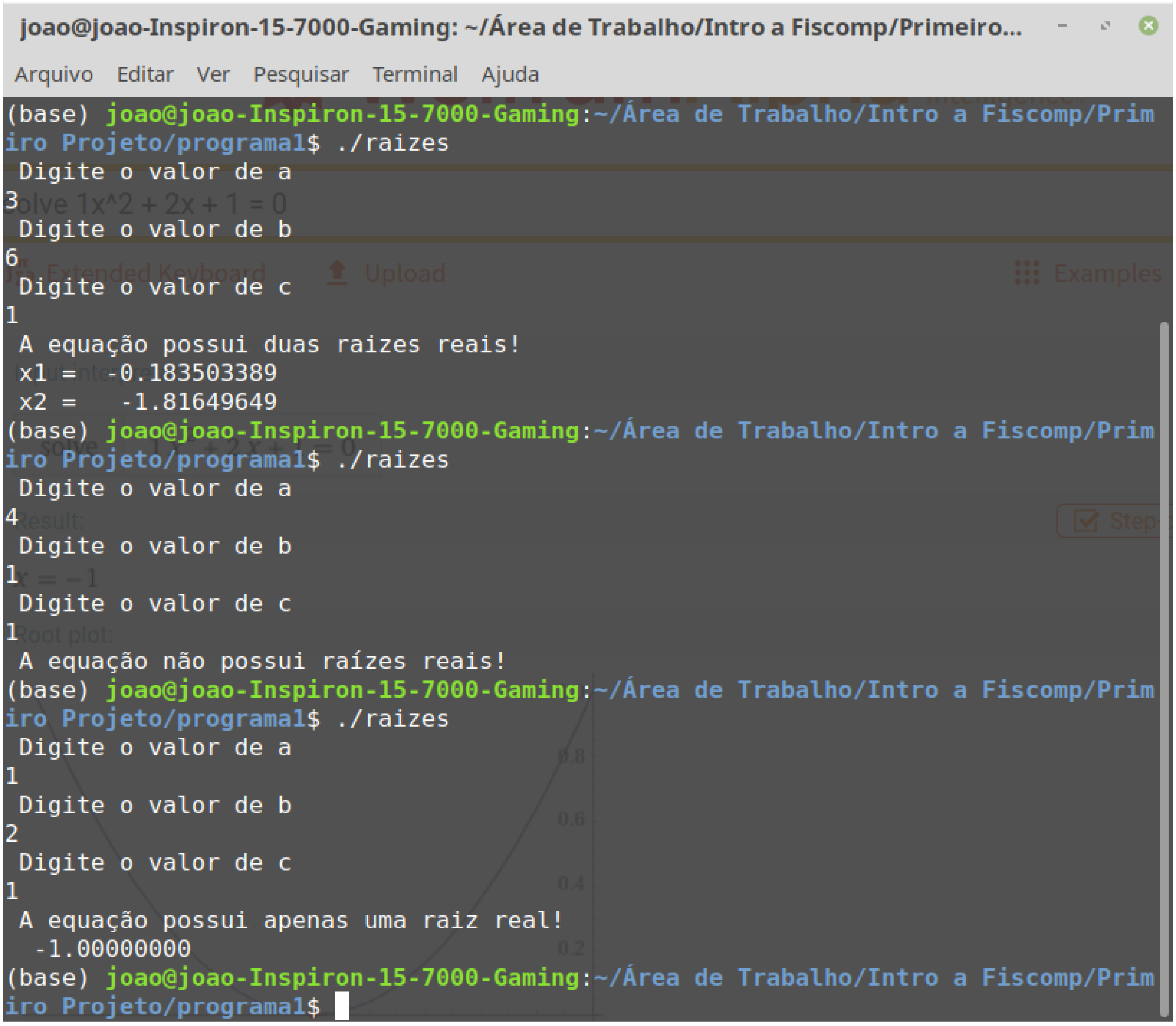
Com a fórmula de raízes de equações de segundo grau conhecida, é possível assim adaptá-la a um programa em FORTRAN 77:

```
program main

       real*4 a
       real*4 b
       real*4 c

       real*4 delta
       real*4 pos_result
       real*4 neg_result

       write(*,*) "Digite o valor de a"
       read(*,*) a
       write(*,*) "Digite o valor de b"
       read(*,*) b
       write(*,*) "Digite o valor de c"
       read(*,*) c


       delta = b**2 -4*a*c

       if (delta.LT.0) then
              write(*,*) "A equação não possui raízes reais!"
       else

              if (delta.EQ.0) then
                     write(*,*) "A equação possui apenas uma raiz real!"
                     write(*,*) "x = ", -b/2*a
              else

                     pos_result = (-b + sqrt(delta))/(2*a)
                     neg_result = (-b - sqrt(delta))/(2*a)

                     write(*,*) "A equação possui duas raízes reais!"
                     write(*,*) "x1 = ", pos_result
                     write(*,*) "x2 = ", neg_result

              end if
       end if

       end program main
```

Este programa tem a função de, ao receber os três coeficientes, **a**, **b** e **c** da equação, calcular suas raízes, e informar ao usuário quantas e quais são elas. Para isso, ele separa o cálculo das raízes em duas partes: a primeira, que consiste no cálculo da parte de dentro da raiz da equação (1), também chamada do **cálculo do delta (Δ)**, é feita antes do cálculo geral, pois o mesmo pode evitar contas desnecessárias dependendo do seu valor. Tendo o delta calculado, chega-se à segunda parte, que consiste na avaliação do valor do mesmo. Caso $\Delta < 0$, sabe-se assim que a equação de segundo grau não possui nenhuma raíz real, o que é informado ao usuário, e assim é terminado o programa, sem a necessidade de mais nenhum cálculo. Caso $\Delta = 0$, a equação terá apenas uma raíz real, sendo calculada como:

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

Caso $\Delta > 0$, a equação terá assim duas raízes reais, sendo calculadas assim por (1) e informadas ao usuário. Segue exemplo abaixo da execução do programa:



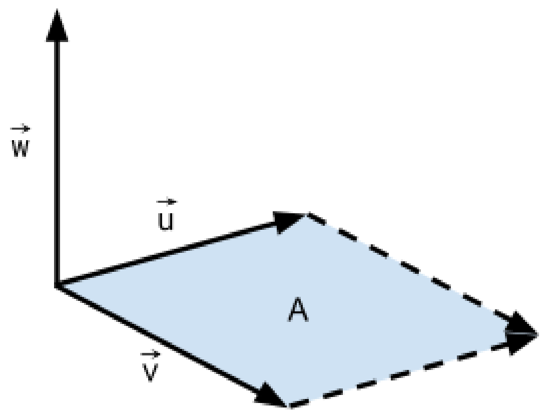
```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Primeiro...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Primeiro Projeto/programa1$ ./raizes
 Digite o valor de a
3
 Digite o valor de b
6
 Digite o valor de c
1
 A equação possui duas raizes reais!
 x1 =  -0.183503389
 x2 =   -1.81649649
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Primeiro Projeto/programa1$ ./raizes
 Digite o valor de a
4
 Digite o valor de b
1
 Digite o valor de c
1
 A equação não possui raízes reais!
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Primeiro Projeto/programa1$ ./raizes
 Digite o valor de a
1
 Digite o valor de b
2
 Digite o valor de c
1
 A equação possui apenas uma raiz real!
  -1.00000000
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Primeiro Projeto/programa1$
```

2. Programa 2 - Área do Triângulo

Neste programa, usamos conceitos de álgebra linear para calcular a área de um triângulo. Ao realizar um produto vetorial entre dois vetores em 3 dimensões, um vetor ortogonal aos dois vetores originais é dado como resultado:

$$\begin{gathered}\vec{v} \times \vec{u} = \vec{w} \\ \vec{w} \perp \vec{v}, \vec{u}\end{gathered} \tag{3}$$

Vetor este útil não apenas para se obter uma direção ortogonal ao plano formado pelos dois referidos, mas também para se obter a área do paralelogramo formado pelos mesmos:



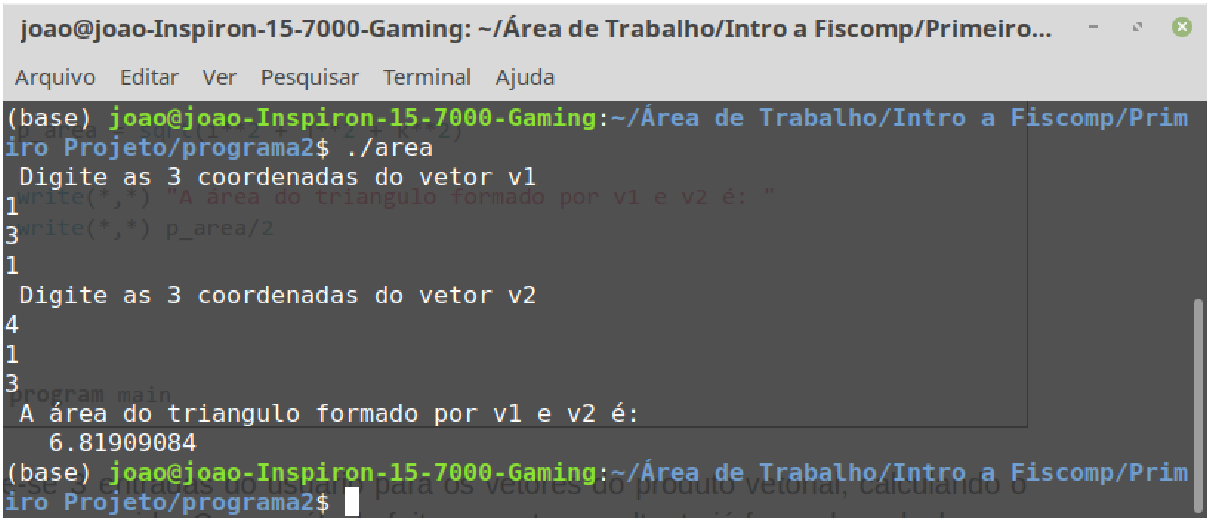
$$||\vec{w}|| = A \tag{4}$$

Sabendo então a área do paralelogramo, é possível calcular a área do triângulo como metade da mesma. O código a seguir implementa os passos citados:

```
program main

    real v1(3)
    real v2(3)

    real i
    real j
    real k

    real p_area = 0


    write(*,*) "Digite as 3 coordenadas do vetor v1"
    read(*,*) v1(1)
    read(*,*) v1(2)
    read(*,*) v1(3)


    write(*,*) "Digite as 3 coordenadas do vetor v2"
    read(*,*) v2(1)
    read(*,*) v2(2)
    read(*,*) v2(3)
```

```
    i = v1(2)*v2(3) - v2(2)*v1(3)
    j = v1(3)*v2(1) - v2(3)*v1(1)
    k = v1(1)*v2(2) - v2(1)*v1(2)


    p_area = sqrt(i**2 + j**2 + k**2)

    write(*,*) "A área do triangulo formado por v1 e v2 é: "
    write(*,*) p_area/2



end program main
```

Recebe-se 3 entradas do usuário para os vetores do produto vetorial, calculando o mesmo em seguida. Com o cálculo feito e o vetor resultante já formado, calcula-se o módulo do vetor, guardado na variável “p_area”, a qual é dividida por dois, fornecendo o resultado da área do triângulo. Segue a seguir uma imagem do programa sendo executado:



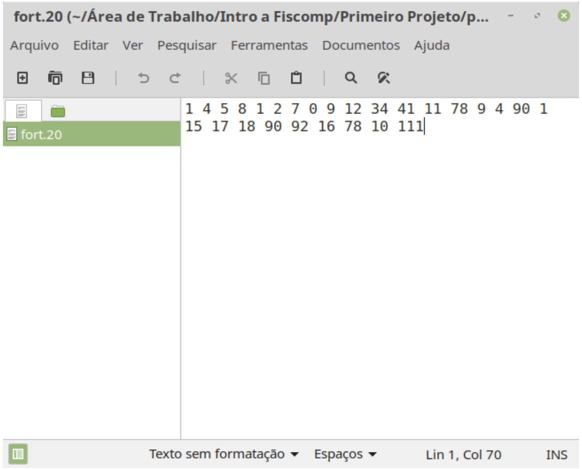
3. Programa 3 - Ordenação

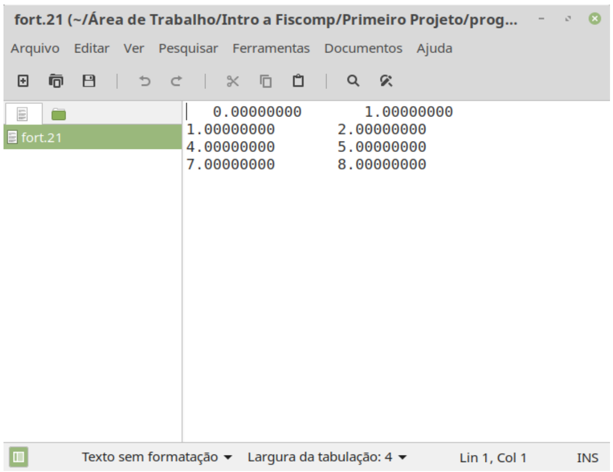
Este programa recebe como entrada um arquivo, *“fort.20”*, o qual desse lê-se N números, armazenando-os em um vetor. Dos N, o usuário escolhe quantos números desses N (ou seja, o número deve ser igual ou menor a N) serão ordenados. Obtendo-se o número M de números a serem ordenados, usa-se um algoritmo de ordenação conhecido como “bubble sort” para ordená-los. Este algoritmo, implementado com dois loops, compara o valor de dois elementos adjacentes do vetor, caso o elemento comparado seja maior ou igual ao seguinte, inverte-se suas posições, repetindo essa operação de comparação M vezes, o que garante que a quantidade de elementos do vetor a serem ordenados terão sido verificados. Após essa operação, escreve-se os M números ordenados em um arquivo de saída, *“fort.21”*. O código a seguir implementa esse algoritmo descrito:

```fortran
program main


    !Aqui estaremos lendo apenas 20 números de um arquivo
    parameter(N = 20)
    integer M
    real list(N)
    real aux

    open(20, file = "entrada-1-10799783")
    read(20, *) (list(i), i = 1, N)
    close(20)


    write(*,*) "Quantos numeros irá querer ordenar?"
    read(*,*) M


    do i = 1, M
          do j = 1, M-1
                if(list(j) .GE. list(j+1)) then

                aux = list(j+1)
                list(j+1) = list(j)
                list(j) = aux

                end if
          end do
    end do
    open(21, file = "saida-1-10799783")
    write(21,*) (list(k), k = 1, M)
    close(21)




end program main
```

Segue a seguir uma imagem do programa sendo executado, e dos arquivos citados:

```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Primeiro...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa3$ ./ordenacao
 Quantos numeros irá querer ordenar?
8
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa3$
```





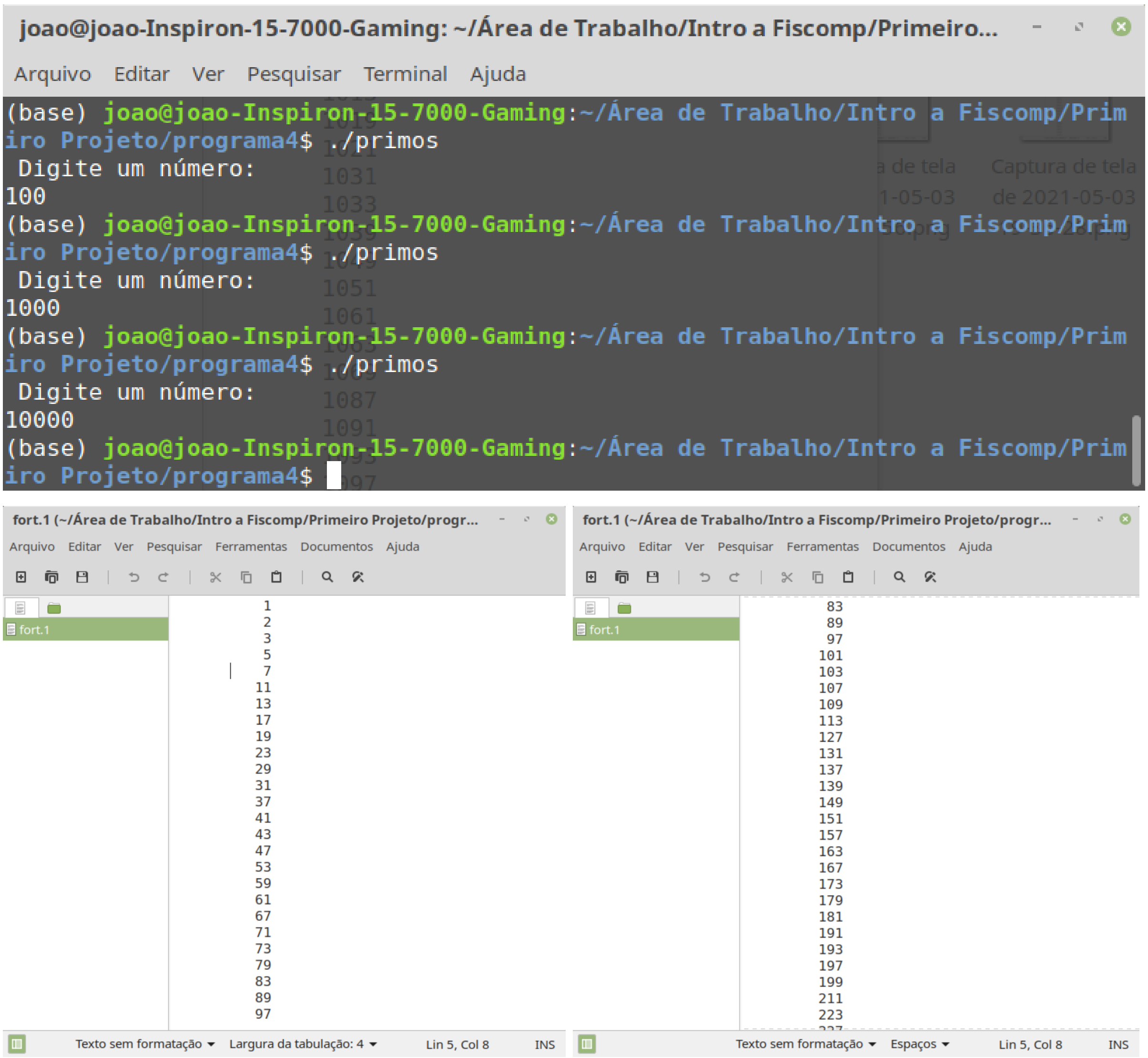
4. Programa 4 - Números Primos

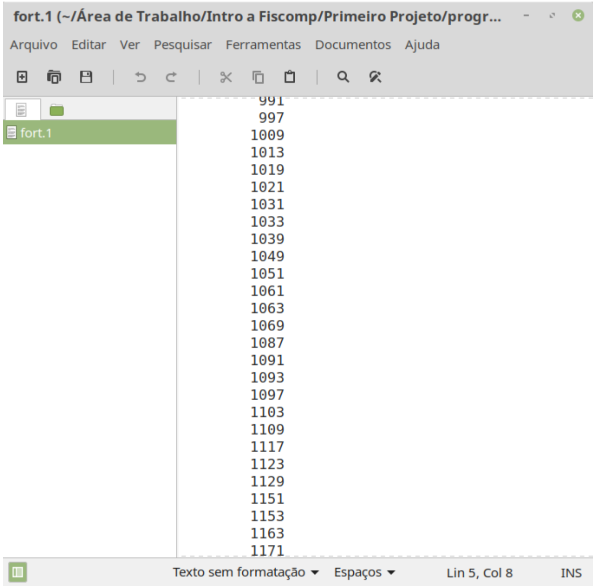
Este programa tem o trabalho de, ao receber um número inteiro N do usuário, calcular quais os primos menores ou iguais a ele. Para esta tarefa, o mesmo se utiliza de um mecanismo bem simples: cria-se uma variável inteira chamada de “counter”, responsável por guardar quantas vezes um número é divisível por outros números. Criada a variável, roda-se um loop iterando i para cada número anterior até N, inicializando o contador igual a 0, e para cada uma, roda-se outro loop iterando j, que calcula o resto da divisão de i por todos os números menores ou iguais a ele. Se o resto de i por j for igual a 0, significa que i é divisível por j, atualizando o contador = contador + 1. Ao final do segundo loop, verifica-se o valor do contador, se este número for menor ou igual a 2, o número então apenas foi dividido “com sucesso” duas vezes: por 1 e por ele mesmo, a própria definição de número primo. Sendo essa condição atingida, o número é então guardado em um arquivo. O código a seguir implementa este algoritmo:

```
program main

    integer N
    integer counter

    write(*,*) "Digite um número:"
    read(*,*) N

    open(2, file = "saida-1-10799783")
    do i = 1, N

          counter = 0

          do j = 1, i
                if ( mod(i,j) .EQ. 0) then
                counter = counter + 1
                end if
          end do

          !se o contador tiver valor de apenas 2, quer dizer que o numero foi
          !dividido por apenas 2 numeros: 1 e ele mesmo
```

```
            if ( counter .LE. 2) then
                   write(2,*) i
            end if

      end do
      close(2)


  end program main
```

Seguem imagens do programa sendo executado, e do arquivo de saída:





5. Programa 5 - Logarítmo Natural

Este programa tem como o objetivo calcular o valor aproximado do logaritmo natural Ln(x) por meio de sua série:

$$\ln(x) = -[(1-x) + (1-x)^2/2 + (1-x)^3/3 + \cdots] = -\sum_{n=1}^{\infty}(1-x)^n/n. \tag{5}$$

Para esta tarefa, receberemos um número real $x \in \mathbb{R}^{*+}$do usuário, e a partir desse número, calcularemos seu valor Ln(x). Antes de aplicá-lo na série, é importante ressaltar que o intervalo de convergência da mesma não compreende todos os reais, mas sim apenas os números tais que $|1 - x| < 1$, o que restringiria o valor que poderíamos receber do usuário. Para contornar este problema, usaremos um truque já conhecido do cálculo aproximado de Ln(x), utilizando-se das propriedades de um logaritmo:

$$ln(x) = ln(y \cdot 10^n) = ln(y) + n \cdot ln(10) \tag{6}$$

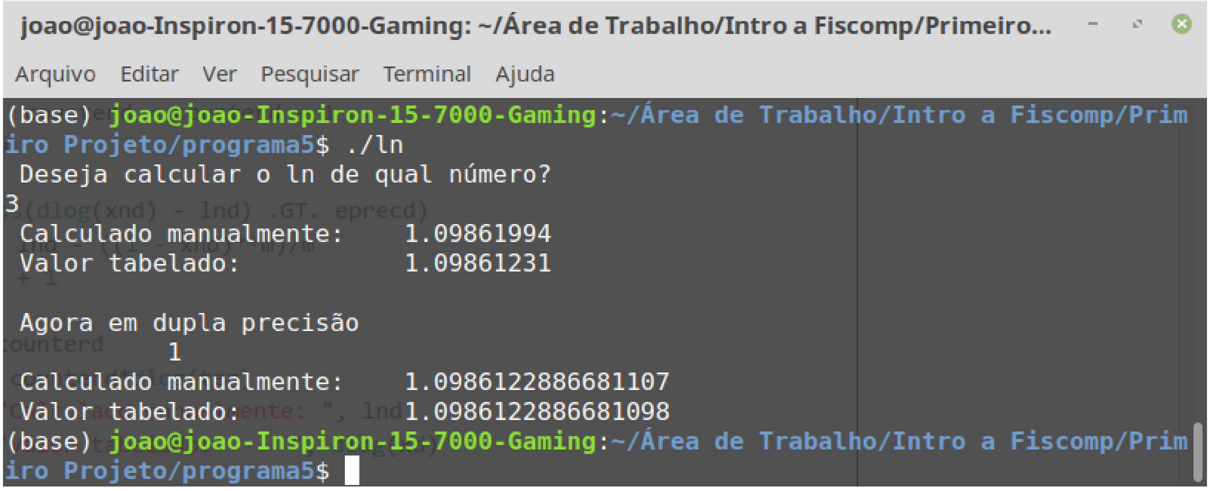
Com o valor de Ln(10) já conhecido, Ln(10) = 2.3025…, é possível calcular o logaritmo natural de qualquer x já citado anteriormente. Conhecendo este truque, é possível implementá-lo com o auxílio de um subalgoritmo simples: verifica-se se x é maior ou igual a 1, caso sim, o mesmo é atualizado com o seu valor dividido por 10, e uma variável “counter” inicializada anteriormente em 0, é atualizada como counter = counter + 1, armazenando assim que aquela operação fora feita. Repete-se esse algoritmo em um loop até que x seja menor ou igual a 1, e após essa condição ser atingida, pode-se assim calcular o logaritmo natural deste novo número pela série (5), cálculo este feito até que a diferença entre este valor e o valor tabelado pela função *log(x)* intrínseca do fortran seja menor que um valor determinado anteriormente $eprec\ =\ 10^{-5}$. Ao atingir esse valor, o valor final do logaritmo do número inserido pelo usuário é calculado por (6), com o auxílio da variável “counter”, que fará o papel do “n”. Este valor é assim comparado com o valor tabelado, impresso na tela ao usuário.

Este algoritmo é também repetido agora em dupla precisão, com novas variáveis declaradas neste formato, exclusivamente para esta operação. Este formato nos permite rodar a série (5) em ordens maiores, permitindo maior precisão nos valores finais. Realizando esta operação mais vezes, o máximo de precisão possível a ser atingida é de 1D-16, diminuindo ao passo que $|1 - x| \to 0$. Isso ocorre pois, para precisões menores que 1D-16, o número de casas extrapola o número permitido pela dupla precisão, fazendo com que o computador entenda que números menores que este alcance são iguais a 0 por conta dos arredondamentos, fazendo com que o loop da série do Ln(x) nunca atinja a condição de parada estipulada, neste caso, que a diferença entre o valor tabelado *dlog(x)* e o valor calculado por nós seja maior que a precisão estabelecida. Segue o código que implementa este conjunto de algoritmos:

```
  program main

      real x
      real*8 xd
      real xn
      real*8 xnd
      real ln
      real*8 lnd
      integer n
      integer m
      real eprec
      real*8 eprecd
      integer counter
      integer counterd
      real*8 ten

      write(*,*) "Deseja calcular o ln de qual número?"
      read(*,*) x

      xn = x
      ln = 0
      n = 1
      eprec = 0.00001
      counter = 0

      xd = x
      xnd = xd
      lnd = 0.0
      m = 1
      eprecd = 1D-15
      counterd = 0
      ten = 10
C     O máximo que consegue-se extrair de precisão é de até 1D-16, para
C     |1 -xn| tendendo a 1, o máximo que a máquina consegue computar nessa
C     precisão, com a mesma diminuindo com |~| tendendo a 0. Abaixo de 1D-16,
```

```fortran
C      o while que calcula o Ln entra em um loop eterno pois a condição de
C      parada nunca será atingida, afinal com os arredondamentos da precisão
C      dupla, a condição sempre acusará que os valores de dlog e Ln são
C      equivalentes. Para valores mais precisos, seriam necessários valores
C      de precisão quádrupla.



       if (x .GE. 1) then
              do while (xn .GE. 1)
                     xn = xn/10
                     counter = counter + 1
              end do
       end if
       do while (abs(log(xn) - ln) .GT. eprec)
              ln = ln - ((1 - xn)**n)/n
              n = n + 1
       end do
       ln = ln + counter*log(10.0)
       write(*,*) "Calculado manualmente: ", ln
       write(*,*) "Valor tabelado:      ", log(x)


       write(*,*) " "
       write(*,*) "Agora em dupla precisão"


       if (xd .GE. 1) then
              do while (xnd .GE. 1)
                     xnd = xnd/10
                     counterd = counterd + 1
              end do
       end if
       do while (abs(dlog(xnd) - lnd) .GT. eprecd)
              lnd = lnd - ((1 - xnd)**m)/m
              m = m + 1
       end do
       write(*,*) counterd
       lnd = lnd + counterd*dlog(ten)
       write(*,*) "Calculado manualmente: ", lnd
       write(*,*) "Valor tabelado:      ", dlog(xd)



  end program main
```

Segue também uma imagem da execução do programa:



6. Programa 6 - Raízes Complexas

Este programa tem como função calcular, mediante um número N, as N raízes da equação complexa:

$$(z-2)^N = 3. \tag{7}$$

Resolvendo a equação acima de maneira analítica, tem-se que suas k raízes terão o formato:

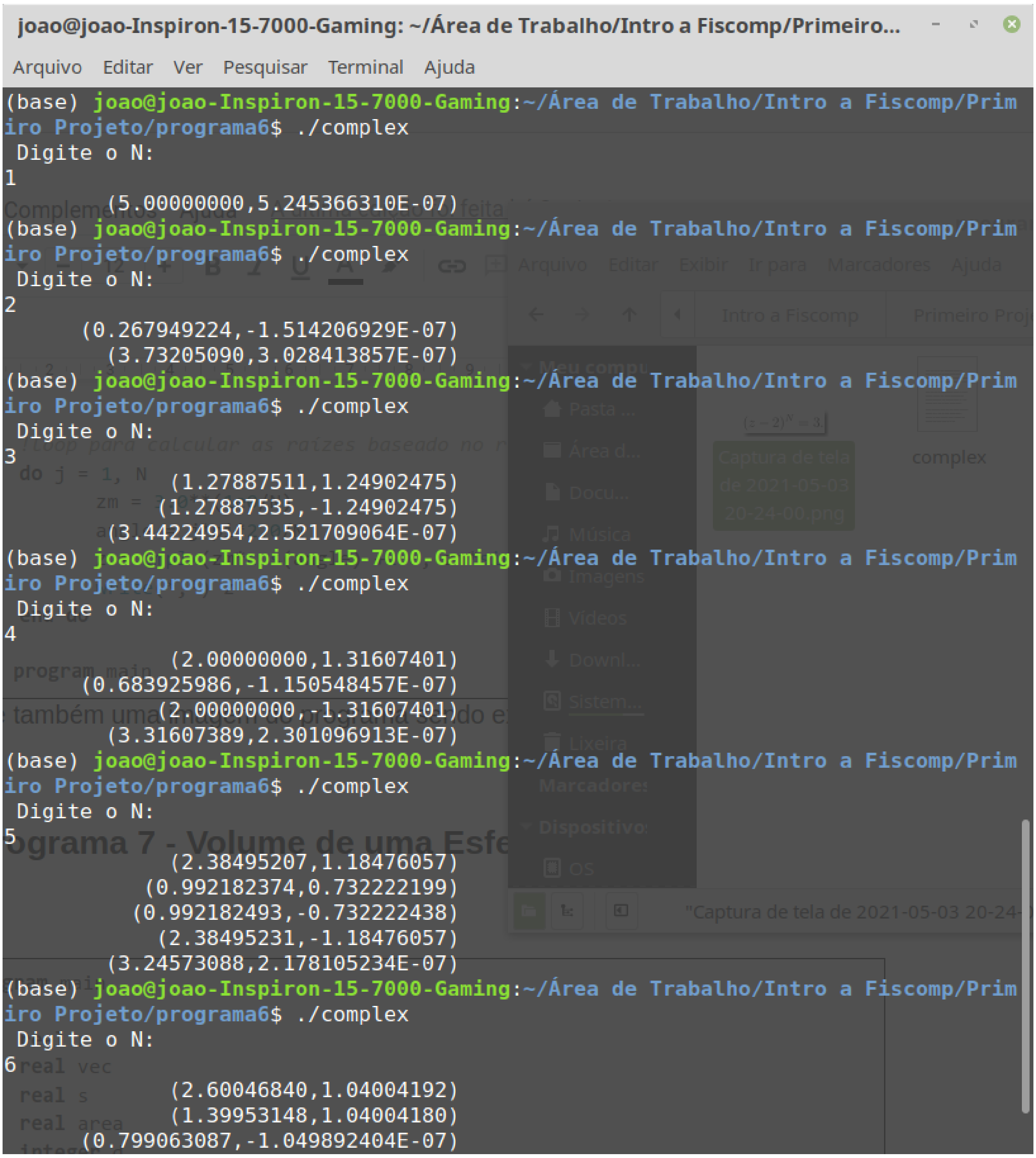
$$z_k = |z|((cos(\frac{2\pi k}{N}) + 2) + isin(\frac{2\pi k}{N})) \tag{8}$$

Com $k = 1, 2, 3, ..., N$. Para implementar esse algoritmo, recebia-se um N do usuário, e dentro de um loop se calculava as k raízes da equação, imprimindo-as na tela utilizando-se do formato de números complexos do fortran. Segue o código que implementa esse algoritmo:

```
program main


    complex z
    real zm
    real angle
    integer N
    real pi

    pi = 4*atan(1.0)
    zm = 1.0
    angle = 0.0


    write(*,*) "Digite o N:"
    read(*,*) N
```

```fortran
    !loop para calcular as raízes baseado no resultado já conhecido
    do j = 1, N
            zm = 3.0**(1e0/N)
            angle = j*pi*2.0/N
            z = cmplx(zm*cos(angle) +2.0, zm*sin(angle))
            write(*,*) z
    end do

end program main
```

Segue também uma imagem do programa sendo executado:



É importante ressaltar que em algumas raízes, verifica-se a presença de números do formato xE-7. Tais números devem ser interpretados como 0, impressos assim apenas por conta das aproximações realizadas pelo computador.

7. Programa 7 - Volume de uma Esfera

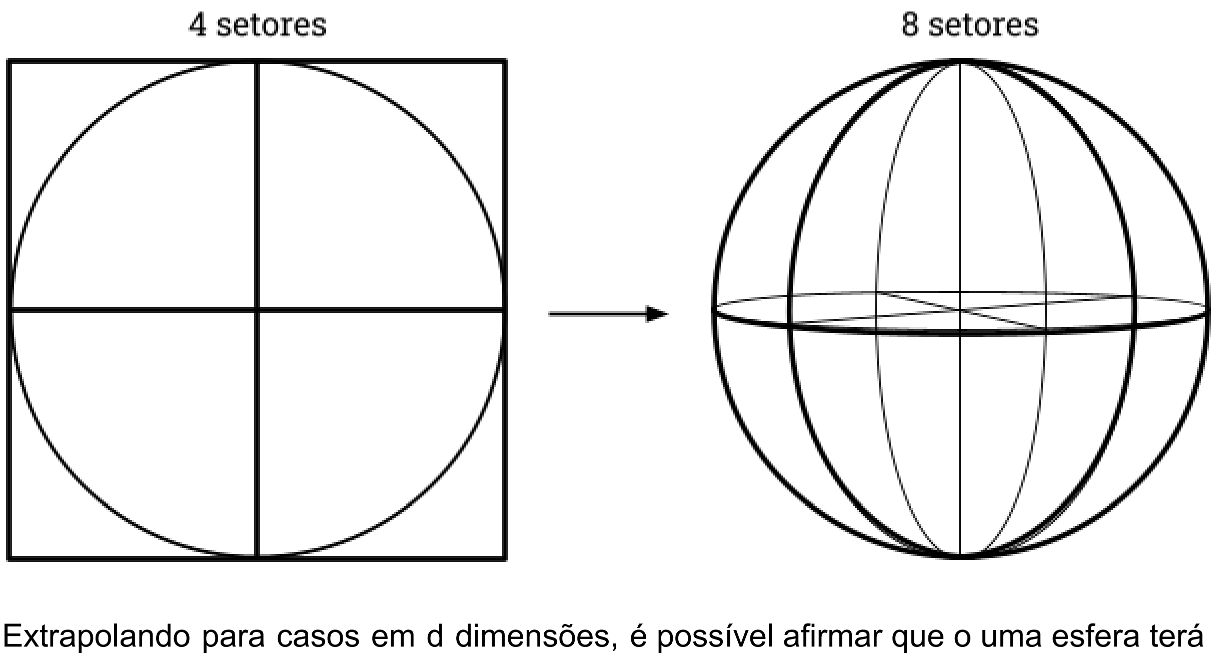
Este programa tem como objetivo realizar o cálculo do volume V_d de uma esfera de d dimensões por meio de um método conhecido como *Método de Monte Carlo.* Este método prevê que, por exemplo, em um domínio com limites bem definidos $[0, a_1] \times [0, a_2] \times ... \times [0, a_n]$; $a_1, a_2, ..., a_n \in \mathbb{R}$, ao escolher uma série de pontos aleatórios com densidade de probabilidade uniforme em todo o domínio, é possível aproximar o volume de um objeto, neste caso de uma esfera, verificando quantos pontos escolhidos estão dentro do raio da esfera, N_r em relação ao número total de pontos, N, sendo esta a taxa de ocupação do domínio S:

$$S = \frac{N_r}{N} \tag{9}$$

Conhecendo esta taxa, é possível assim calcular o "volume" desse objeto:

$$V_d = S \cdot a_1 a_2 \ldots a_n \tag{10}$$

Para implementar este algoritmo em uma esfera de d dimensões de raio R, separaremos a mesma em setores, calculando o volume de um setor, e estendendo esse volume para as demais. Nos casos 2D e 3D, teremos 4 e 8 setores:



Extrapolando para casos em d dimensões, é possível afirmar que o uma esfera terá 2^d setores. Sendo assim, seu volume final será:

$$V_d = 2^d \cdot S \cdot a_1 a_2 \ldots a_n \tag{11}$$

Com isso, e aliando-se à função intrínseca do fortran *rand(),* é possível calcular o volume de uma esfera em d dimensões por meio da geração de números aleatórios. Primeiro, recebe-se do usuário o número de dimensões que ele gostaria de calcular a esfera de raio 1. Sabendo então d, roda-se um loop iterando i até a quantidade de pontos aleatórios que se quer gerar, e dentro do mesmo roda-se outro loop para cada dimensão a qual este número será gerado. Após gerado, verifica-se se o módulo do ponto em d dimensões gerado se encontra dentro do raio da esfera, se sim, com o auxílio de uma variável counter, conta-se quantos pontos estão dentro da mesma. Ao final destes loops, realiza-se a conta da taxa s, e calcula-se o volume final por (11). Segue o código que implementa este algoritmo:

```
program main

    real r
    real vec
    real s
    real area
    integer d
    integer M
    integer counter

    r = 1.0
    vec = 0.0
    M = 10000000
    counter = 0

    write(*,*) "Quantas dimensoes gostaria de calcular o volume?"
    read(*,*) d



    !inicializando a seed do rand() baseado no horário do computador
    call srand(time())
    do i = 1, M
          do j = 1, d
                vec = vec + ((r*rand())**2)
          end do
          if( sqrt(vec) .LE. r) then
                counter = counter + 1
          end if
          vec = 0.0
    end do

    s = real(counter)/real(M)
    area = (2**d)*s*(r**d)

    write(*,*) area



end program main
```

Segue também uma imagem do programa em execução:

```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Primeiro...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa7$ ./esfera
 Quantas dimensoes gostaria de calcular o volume?
2
   3.14230323
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa7$ ./esfera
 Quantas dimensoes gostaria de calcular o volume?
3
   4.18952799
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa7$ ./esfera
 Quantas dimensoes gostaria de calcular o volume?
4
   4.93391180
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Prim
iro Projeto/programa7$
```

Para verificar se os cálculos estão corretos, comparamos estes resultados com a expressão:

$$V_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} R^d \quad (12)$$

Com $\Gamma(1/2) = \sqrt{\pi}$, $\Gamma(1) = 1$, $\Gamma(x+1) = x\Gamma(x)$. Com a ajuda do website wolframalpha.com, tem-se os cálculos:

$d = 2$

Input:

$$\frac{\pi^{2/2}}{\Gamma(\frac{2}{2}+1)} \times 1^2$$

Exact result:

π

Decimal approximation:

3.141592653589793238462643383279502884197169

...

$d = 3$

Input:

$$\frac{\pi^{3/2}}{\Gamma\left(\frac{3}{2}+1\right)} \times 1^3$$

Exact result:

$$\frac{4\pi}{3}$$

Decimal approximation:

4.1887902047863909846168578443726705122628925325 ...

$d = 4$

Input:

$$\frac{\pi^{4/2}}{\Gamma\left(\frac{4}{2}+1\right)} \times 1^4$$

Exact result:

$$\frac{\pi^2}{2}$$

Decimal approximation:

4.9348022005446793094172454999380755676568497 ...

Que correspondem, aproximadamente, aos valores calculados pelo nosso programa.

8. Programa 8 - Volume em d Dimensões

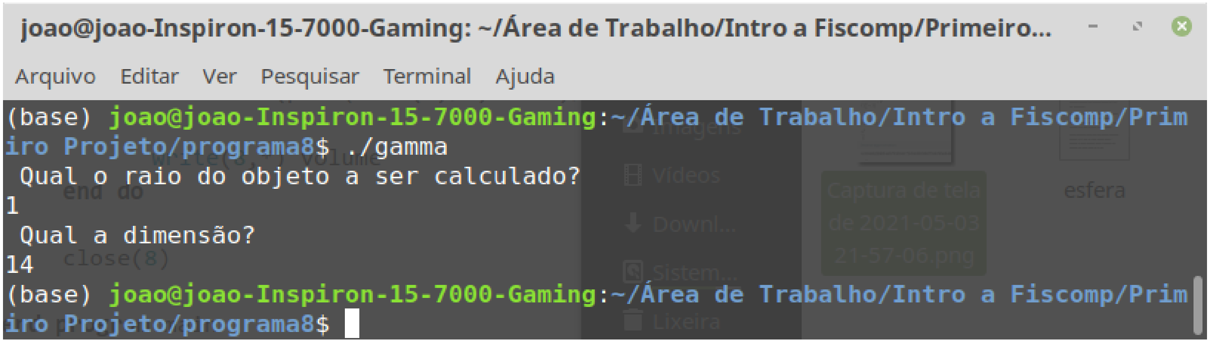
Este programa tem o objetivo de, utilizando a expressão (12), apresentada acima, estender o cálculo dos volumes para dimensões maiores de maneira fácil e rápida, nos permitindo visualizar a evolução dos volumes. Para isso, basta implementar a função $\Gamma(x)$ em um algoritmo funcional. Tal algoritmo será implementado da seguinte maneira: recebe-se o valor de x, e enquanto x > 1, este valor será atualizado como x = x - 1, e uma variável auxiliar “gamma”, inicializada em 1, será atualizada como gamma = x*gamma, tudo isso em loop. Após o fim desse loop, verifica-se o valor de x, se x < 1, pela imagem da nossa função, x = 1/2, e pelas propriedades da função gamma, $\Gamma(1/2) = \sqrt{\pi}$, portanto gamma será atualizada para *gamma* = $\sqrt{\pi}$**gamma*. Caso x = 1, novamente pelas propriedades da função gamma, $\Gamma(1) = 1$, portanto gamma será atualizada para *gamma = 1*gamma.*

Com a função gamma implementada, é possível calcular o volume final por (12). Fazendo estes cálculos para $d = 1,..., 14$, teremos o volume de esferas em até 14 dimensões, e os escreveremos em um arquivo *.dat*, que será usado pelo graficador *xmgrace* para graficar a evolução dos volumes conforme as dimensões aumentam. Segue o código que implementa este algoritmo:

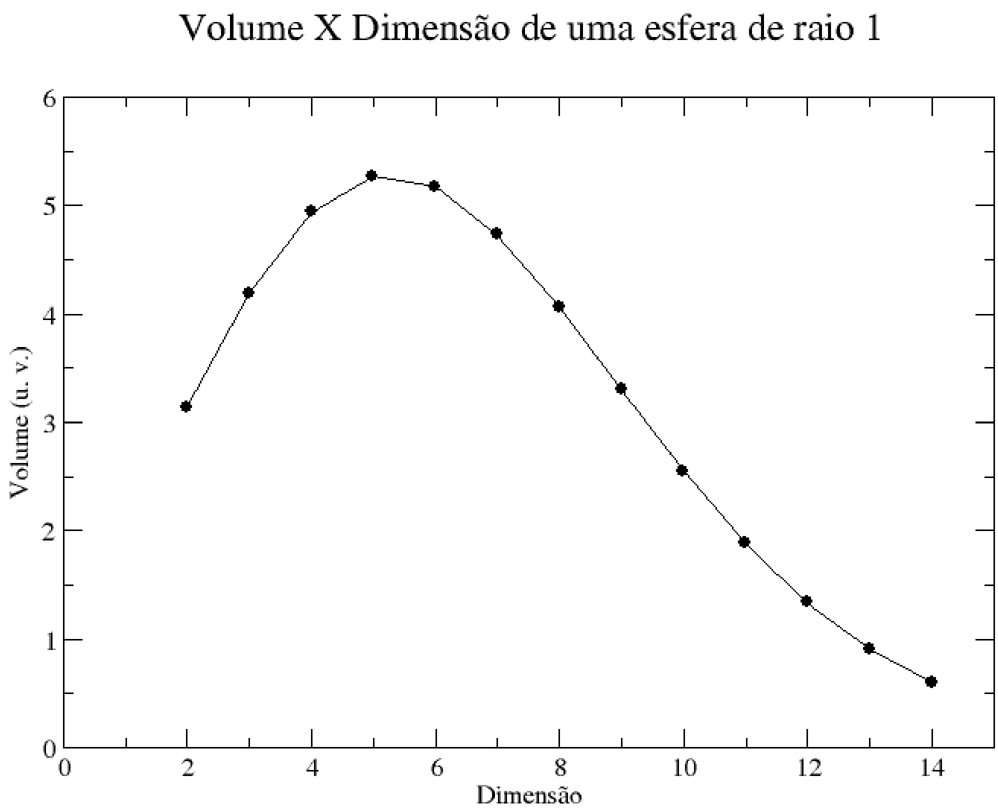
```
program main

	real r
	real pi
	real gamma
	real y
	real volume
	integer d

	!definindo pi e inicializando o volume
	pi = 4*atan(1.0)
	volume = 0.0

	write(*,*) "Qual o raio do objeto a ser calculado?"
	read(*,*) r

	write(*,*) "Qual a dimensão?"
	read(*,*) d


	open(8, file = "dimensoes-esferas.dat")


	!Loop para calcular o volume das esferas e as salvar em um arquivo
	do i = 2, d
		gamma = 1.0

		y = real(i)/2 + 1.0

		do while(y .GT. 1.0)
		y = y - 1
		gamma = y*gamma
		end do

		if (y .LT. 1) then
```

```
            gamma = sqrt(pi)*gamma
            else
            gamma = 1*gamma
            end if

            volume = (pi**(real(i)/2)*r**i)/gamma

            write(8,*) volume
      end do

      close(8)

  end program main
```

Segue também o código sendo executado:



A seguir, o gráfico gerado pelo *xmgrace:*



Perguntas:

A) De acordo com o programa 8, o volume de um cubo de raio 1, igual a $1m^d$ será T vezes maior:

$$T = \frac{1}{\frac{\pi^{d/2} R^d}{\Gamma(\frac{d}{2}+1)}} \tag{13}$$

Utilizando nosso programa, para $d \to \infty$, verifica-se que o volume da esfera $\to 0$, portando, $T \to \infty$.