Projeto 1: Introdução à programação

7600017 - Introdução à Física Computacional - 2023/02
27/08/2023

Prof. Dr. José Abel Hoyos
Gabriel de Freitas de Azeredo (11810964)

Resumo

Este projeto visa ser um primeiro contato dos estudantes com a computação científica e solução de problemas variados com o computador. Além disso, servirá como introdução à linguagem escolhida para o curso: Fortran 77. Os problemas vão desde método de Monte Carlo até ordenação de listas, fazendo com que o estudante utilize diversos artifícios desta linguagem que servirão como base durante todo o curso.

---

1 Tarefa 1

1.1 Introdução

Esta é uma simples tarefa com o objetivo de aprender as operações básicas na linguagem Fortran 77, para isso calculou-se as raízes reais da equação $ax^2 + bx + c = 0$, dados os coeficientes a, b e c.

1.2 Estratégia de resolução

1.2.1 Entradas e saídas de dados

O programa espera do usuário os coeficientes a, b e c reais como entrada e como saída ele imprime na tela uma, duas ou informa que não há solução nos reais para a equação do segundo grau com os coeficientes indicados.

1.2.2 Desenvolvimento do código

Com a, b e c dados de entrada. Utilizei a fórmula,

$$\Delta = b^2 - 4ac \tag{1}$$

então verifico se $\Delta < 0$ (caso sem raízes gerais), caso contrário utilizo $x = \frac{-b+\sqrt{\Delta}}{2a}$ para calcular a primeira raiz. Finalmente, verifico se $\Delta > 0$ a fim de descobrir se mais um cálculo é necessário. Se sim, aplico $x' = \frac{-b-\sqrt{\Delta}}{2a}$ para encontrar a segunda raiz.

1.3 Resultados

A figura 1 contém o código implementado e a figura 2 representa um simples caso para $a = 1$, $b = 0$ e $c = -9$.

```fortran
      program tarefa1

      real*8 :: a, b, c, delta, x1, x2

      write(*,*) "Coeficientes do polinômio: ax^2 + bx + c = 0"
      read(*,*) a, b, c

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

      if (delta .lt. 0) then

         write(*,*) "Nenhuma solução nos reais."

      else
         x1 = (-b + sqrt(delta)) / 2
         write(*,*) "x = ", x1

         if (delta .gt. 0) then

            x2 = (-b - sqrt(delta)) / 2
            write(*,*) "x' = ", x2

         end if
      end if

      end program tarefa1
```

Figura 1: Código da tarefa 1.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-1> ./tarefa-1.exe
 Coeficientes do polinômio: ax^2 + bx + c = 0
1 0 -9
 x =     3.0000000000000000
 x' =   -3.0000000000000000
```

Figura 2: Caso teste da tarefa 1.

2 Tarefa 2

2.1 Introdução

O objetivo da tarefa 2 foi utilizar o conceito dentro de vetores no Fortran 77, para isso foi calculada a área de um triângulo formado por dois vetores $\vec{v}_1$ e $\vec{v}_2$.

2.2 Estratégia de resolução

2.2.1 Entradas e saídas de dados

O programa espera pelas coordenadas de dos vetores pelo usuário $v_1 = (x_1, y_1, z_1)$ e $v_2 = (x_2, y_2, z_2)$ e imprime na tela a área do triângulo formado por eles.

2.2.2 Desenvolvimento do código

Dados os vetores de entrada, a área do triângulo foi calculada como sendo o módulo do produto vetorial entre eles dividido por dois, bem como na álgebra linear. Ou seja,

$$A = \frac{||\,\vec{q}\,||}{2} \tag{2}$$

onde $\vec{q} = \vec{v}_1 \times \vec{v}_2$, então $A = \sqrt{(y_1 z_2 - y_2 z_1)^2 + (z_1 x_2 - z_2 x_1)^2 + (x_1 y_2 - x_2 y_1)^2}/2$. Nesse programa tomei um cuidado especial, pois as variáveis de entrada foram declaradas como reais de dupla precisão e é desejado que a área também seja. Por isso, a função *dsqrt* foi utilizada.

2.3 Resultados

A figura 3 contém o código implementado e a figura 4 representa um simples caso para $\vec{v}_1 = (1, 2, 3)$ e $\vec{v}_2 = (1, 0, 0)$.

```fortran
      program tarefa2

      real*8 :: v1(3), v2(3)

      write(*,*) "Digite o primeiro vetor (v1): "
      read(*,*) v1

      write(*,*) "Digite o segundo vetor (v2): "
      read(*,*) v2

      abs_crossp = dsqrt((v1(2)*v2(3) - v2(2)*v1(3))**2 + (v1(3)*v2(1)
     *   - v2(3)*v1(1))**2 + (v1(1)*v2(2) - v2(1)*v1(2))**2)

      write(*,*) "A = ", abs_crossp / 2.d0

      end program tarefa2

```

Figura 3: Código da tarefa 2.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-2> ./tarefa-2.exe
 Digite o primeiro vetor (v1):
1 2 3
 Digite o segundo vetor (v2):
1 0 0
 A =    1.8027756214141846
```

Figura 4: Caso teste da tarefa 2.

3 Tarefa 3

3.1 Introdução

O objetivo dessa tarefa é aprender a manipular arquivos no Fortran 77. Especificamente, utilizar os comandos *WRITE* e *READ*. Para isso, li todos os números de um arquivo, contando quantos números em uma variável e ordenei os m primeiros, com o m sendo dado no terminal.

3.2 Estratégia de resolução

3.2.1 Entradas e saídas de dados

O programa espera o número m do usuário e um arquivo contendo n números. A saída é um outro arquivo com os m menores números ordenados.

3.2.2 Desenvolvimento do código

Primeiro, foi definido um vetor capaz de armazenar grandes quantidades de números do arquivo de entrada. Então, utilizando o parâmetro *end* do coando *READ* do Fortran 77, todos os números foram lidos e sendo contados. Após isso o programa recebe o número m e ordena os números seguindo o algoritmo *bubble sort*. Definitivamente esse algoritmo não é o mais eficiente para grandes quantidades de números, porém sua lógica é facilmente entendida e funciona bem para a tarefa em questão.

O algoritmo funciona selecionando o último elemento do vetor e comparando com o elemento anterior, caso for menor, troca-se a posição dos dois. Fazendo isso percorrendo a lista inteira, temos o vetor ordenado.

Mesmo não sendo muito eficiente, o *bubble sort* é introduzido nos cursos de introdução a computação como na disciplina oferecida para o IFSC, pois estimula o pensamento algorítmico e solução de problemas pelo computador.

3.3 Resultados

A figura 5 contém o código implementado do *bubble sort* e a figura 6 representa o inicio do arquivo de saída para $m = 100$.

```
      do i = 1, m
         do j = (n - 1), 1, -1

            if (values(j + 1).lt.values(j)) then

               vtemp = values(j)
               values(j) = values(j + 1)
               values(j + 1) = vtemp

            end if
         end do
      end do
```

Figura 5: Código da tarefa 3.

```
     500000
   1.2451782822608948E-006
   5.5236741900444031E-006
   6.6380016505718231E-006
   6.6407956182956696E-006
   7.6387077569961548E-006
   7.8030861914157867E-006
   1.0204501450061798E-005
   1.0691117495298386E-005
   1.1510215699672699E-005
   1.2444797903299332E-005
   1.5242025256156921E-005
   1.9087921828031540E-005
   2.2660940885543823E-005
   2.3015774786472321E-005
   2.3539178073406219E-005
   3.1753443181514740E-005
   3.3237971365451813E-005
   3.9324164390563965E-005
   4.1628722101449966E-005
   4.1754916310310364E-005
   4.3858308345079422E-005
   4.4111628085374832E-005
   4.7957990318536758E-005
   4.8963818699121475E-005
   5.1964540034532547E-005
   5.2090734243392944E-005
   5.3675845265388489E-005
   6.1507336795330048E-005
   6.1914790421724319E-005
   6.3926447182893753E-005
   6.6818203777074814E-005
```

Figura 6: Caso teste da tarefa 3.

4 Tarefa 4

4.1 Introdução

O objetivo dessa atividade foi calcular todos os números primos menores ou iguais a um número dado. Além do número totla de números primos calculados.

4.2 Estratégia de resolução

O método mais eficiente para realizar essa atividade é pelo Crivo de Eratóstenes. Um algoritmo amplamente difundido que retorna os números primos de uma lista de $n$números. Como um número primo p é definido tal como sendo aquele que é divisível apenas por 1 e p, a estratégia consiste de tirar todos os múltiplos dos n números da lista. Ou seja, seja $n = 2$, tiramos da lista o $4, 6, 8, ...$ até o último múltiplo. Ao realizar isso para toda a lista, ficam somente os primos.

4.2.1 Entradas e saídas de dados

O número n que representa o tamanho da lista deve ser obtido por entrada. De saída, todos os números primos menores que n e o total de números primos.

4.2.2 Desenvolvimento do código

Para o código, primeiro iniciei um vetor com variáveis do tipo lógico como verdadeiro. Então, apliquei o algoritmo do Crivo de Eratóstenes com o *loop DO*, deixando como falso todos os elementos do vetor com índices de múltiplos dos n números. Ao final, obtive um vetor com *TRUE* em números primos e *FALSE* em números não primos. Usando isso de condição, imprimi somente os primos na tela, contando quantos são impressos para o total de números primos.

4.3 Resultados

A figura 7 contém o código implementado e a figura 8 representa o final da saída do caso teste para $n = 10000$, mostrando os números primos e no final o total de números primos impressos (1230).

```
program tarefa4

integer, parameter :: max_n = 10000
integer :: n, icount = 0
logical :: b(max_n) = .TRUE.

read(*,*) n

do i = 2, n
   j = 2
   do while((i * j).le.max_n)
      b(i * j) = .FALSE.
      j = j + 1
   end do
end do

do i = 1, n
   if (b(i)) then
      write(*,*) i
      icount = icount + 1
   end if
end do

write(*,*) icount
end program tarefa4
```

Figura 7: Código da tarefa 4.

```
        9887
        9901
        9907
        9923
        9929
        9931
        9941
        9949
        9967
        9973
        1230
gabriel@ubuntu ~/U/c/i/p/tarefa-4> |
```

Figura 8: Caso teste da tarefa 4.

5 Tarefa 5

5.1 Introdução

O objetivo desta tarefa é implementar um código que consiga calcular uma boa aproximação para a função *log* de um número pelo método da série de Taylor. A série já foi dada pelo enunciado, com uma pequena observação: com os critérios de convergência aprendidos em cálculo IV é possível calcular o raio de convergência desta série $R = 1$. Como foi expandida em torno do ponto $x = 1$, então a série representa bem o comportamento da função *log* para valores $x \in]0, 2[$.

5.2 Estratégia de resolução

5.2.1 Entradas e saídas de dados

O programa espera de entrada um valor para x, respeitando o intervalo determinado na introdução do problema. De saída, retorna o valor de *log* calculado pela série e o calculado pela função nativa do Fortran, para fins de comparação, também foi calculado o erro absoluto entre esses dois valores.

5.2.2 Desenvolvimento do código

O código é basicamente uma implementação da série em forma de um *loop DO*, inicializando a variável de tolerância *EPS* como $1.E-5$ para a primeira parte da tarefa e $1.D-16$ para a solução em dupla precisão, pois foi a tolerância que obtive melhor precisão em comparação a função *DLOG* do Fortran. O *loop* calcula até o erro, definido como a diferença de termos consecutivos da série, ser menor que a tolerância já explicitada.

5.3 Resultados

A figura 9 contém o código implementado para a tarefa 5a e a figura 10 representa um caso de teste para $x = 1.7$.

```fortran
program tarefa5a

real*4 :: ln_x = 0.e0, x = 0.e0, aux = 0.e0

eps = 1.e-5
error = 1.e0
n = 1

read(*,*) x

do while (abs(error).ge.eps)

   ln_x = ln_x + (1 - x)**real(n, 4) / real(n, 4)

   error = ln_x - aux
   aux = ln_x
   n = n + 1
end do

ln_x = -1 * ln_x

write(*,*) "Taylor: ", ln_x
write(*,*) "Fortran log(x): ", log(x)
write(*,*) "Erro absoluto", ln_x - log(x)

end program tarefa5a
```

Figura 9: Código da tarefa 5a.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-5> ./tarefa-5a.exe
1.7
 Taylor:    0.530624986
 Fortran log(x):    0.530628264
 Erro absoluto   3.27825546E-06
```

Figura 10: Caso teste da tarefa 5a.

A figura 11 contém o código implementado para a tarefa 5b e a figura 12 representa um caso de teste para $x = 1.7$.

```fortran
      program tarefa5b

      real*8 :: ln_x = 0.d0, x = 0.d0, aux = 0.d0, eps = 1.d-16

      error = 1.d0
      n = 1

      read(*,*) x

      do while (abs(error).ge.eps)

         ln_x = ln_x + (1 - x)**real(n, 8)/ real(n, 8)
         error = ln_x - aux

         aux = ln_x
         n = n + 1
      end do

      ln_x = -1 * ln_x

      write(*,*) "Taylor: ", ln_x
      write(*,*) "Fortran log(x): ", dlog(x)
      write(*,*) "Erro absoluto", abs(ln_x - dlog(x))

      end program tarefa5b
```

Figura 11: Código da tarefa 5b.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-5> ./tarefa-5b.exe
1.7
 Taylor:   0.53062825106216993
 Fortran log(x):    0.53062825106217038
 Erro absoluto   4.4408920985006262E-016
```

Figura 12: Caso teste da tarefa 5b.

6 Tarefa 7

6.1 Introdução

O objetivo desta atividade é utilizar uma introdução ao método de Monte Carlo para calcular o volume de uma esfera em d dimensões e depois comparar com o resultado da fórmula analítica. As duas formas foram implementadas.

6.2 Estratégia de resolução

6.2.1 Entradas e saídas de dados

Como entrada, o programa espera do usuário o número m de variáveis aleatórias que o método de Monte Carlo deve utilizar, o raio R da esfera e a dimensão que o volume deve ser calculado. Como saída, o programa imprime o volume pelo método de Monte Carlo e pela fórmula, assim possibilitando a comparação.

6.2.2 Desenvolvimento do código

Primeiro, a função $RAND$ do Fortran foi inicializada. Como ela gera números pseudo-aleatórios entre 0 e 1, salvei em um vetor de 4 dimensões (máximo aceito pelo programa), números de forma $v(n) = rand()R$, assim obtive números aleatórios entre 0 e R. Com isso, calculei a distância entre o ponto gerado na dimensão d e a origem, com a fórmula $dist = (\Delta x)^2 + (\Delta x_2)^2 + ...$ Caso a distância seja menor que o raio, então o ponto está dentro da esfera. Fazendo a proporção para um número grande de pontos, relacionando os pontos pertencentes a esfera com o número m de pontos gerados, conseguimos calcular a relação entre o volume da esfera e do cubo de lado R e dimensão d. De forma:

$$V_e = \frac{N_{dentro}}{m}(2R)^d \tag{3}$$

Para o cálculo pela fórmula, primeiro calculei a função Γ que é uma espécie de fatorial definido para números reais. Definindo $x = d/2+1$, realizei um *loop* que calcula o fatorial de $x-1$ até $x = 1/2$ ou $x = 1$. Ao sair do *loop*, caso o último valor de x seja 1/2, multiplico por $\Gamma(1/2) = \sqrt{\pi}$. Então, aplico a fórmula

$$V_d = \frac{\pi^{d/2}}{\Gamma(1+d/2)}R^d \tag{4}$$

6.3 Resultados

A figura 13 contém o trecho de código implementado para a tarefa 7 pelo método de Monte Carlo e a figura 14 representa o trecho de código para a implementação com a fórmula. E a figura 15 mostra os casos testes pedidos no enunciado, bem como um caso extra com um número de pontos desprezível $m = 100$. O resultado com um número de pontos tão pequeno diverge bastante do esperado e isso mostra a importância de casos teste no método de Monte Carlo.

```
      do i = 1,m

         dist = 0.d0

         random_vec(1) = rand() * R
         random_vec(2) = rand() * R
         random_vec(3) = rand() * R
         random_vec(4) = rand() * R

         do j = 1,id
            dist = dist + random_vec(j)**2
         end do

         if (dist.le.R) then
            icont = icont + 1
         end if

      end do

      write(*,*) 'To d = ', id, ' and R = ', R

      write(*,*) 'Monte Carlo:'
      write(*,*) real(icont, 8) / real(m, 8) * (2 * R)**id
```

Figura 13: Código da tarefa 7, Monte Carlo.

```
      x = real(id, 8) / 2.d0 + 1.d0

      do while(x.ne.(0.5d0).and.x.ne.(1.d0))
         fun_gamma = (x - 1.d0) * fun_gamma
         x = x - 1.d0
      end do

      if (x.eq.(0.5d0)) then
         fun_gamma = fun_gamma * dsqrt(pi)
      end if

      vd_formula = pi**(real(id, 8) / 2.d0)* R**real(id, 8) / fun_gamma

      write(*,*) 'Formula:'
      write(*,*) vd_formula
```

Figura 14: Código da tarefa 7, método da fórmula.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-7> ./tarefa-7.exe
1000000 1 2
 To d =            2  and R =    1.0000000000000000
 Monte Carlo:
   3.1435599999999999
 Formula:
   3.1415926535897931
gabriel@ubuntu ~/U/c/i/p/tarefa-7> ./tarefa-7.exe
1000000 1 3
 To d =            3  and R =    1.0000000000000000
 Monte Carlo:
   4.1968719999999999
 Formula:
   4.1887902047863914
gabriel@ubuntu ~/U/c/i/p/tarefa-7> ./tarefa-7.exe
1000000 1 4
 To d =            4  and R =    1.0000000000000000
 Monte Carlo:
   4.9455999999999998
 Formula:
   4.9348022005446790
gabriel@ubuntu ~/U/c/i/p/tarefa-7> ./tarefa-7.exe
100 1 2
 To d =            2  and R =    1.0000000000000000
 Monte Carlo:
   3.0800000000000001
 Formula:
   3.1415926535897931
```

Figura 15: Caso teste da tarefa 7.

7 Tarefa 8

7.1 Introdução

Utilizando o código da tarefa anterior, o objetivo desta tarefa é calcular o volume da esfera para mais pontos e discutir mais profundamente os resultados.

7.2 Estratégia de resolução

7.2.1 Entradas e saídas de dados

O programa espera de entrada o raio R e a dimensão d da esfera. A saída é em um arquivo, contendo o raio, a dimensão e os volumes das respectivas esferas.

7.2.2 Desenvolvimento do código

O código é como do exercício anterior, porém coloquei em um loop *DO* que calcula o volume da esfera dados d e R.

7.3 Resultados

A figura 16 mostra o arquivo de saída da tarefa 8a para o caso que $d_{max} = 25$ e $R = 1$. A figura 17 mostra o resultado do gráfico da tarefa 8b.

```
gabriel@ubuntu ~/U/c/i/p/tarefa-8> cat tarefa-8a-saida-1.dat
   1.0000000000000000
           0   1.00000000
           1   1.99999988
           2   3.14159274
           3   4.18879032
           4   4.93480206
           5   5.26378918
           6   5.16771269
           7   4.72476625
           8   4.05871201
           9   3.29850888
          10   2.55016398
          11   1.88410389
          12   1.33526278
          13  0.910628796
          14  0.599264503
          15  0.381443292
          16  0.235330626
          17  0.140981108
          18   8.21458846E-02
          19   4.66216020E-02
          20   2.58068908E-02
          21   1.39491502E-02
          22   7.37043098E-03
          23   3.81065602E-03
          24   1.92957430E-03
          25   9.57722485E-04
```

Figura 16: Arquivo de saída da tarefa 8a.

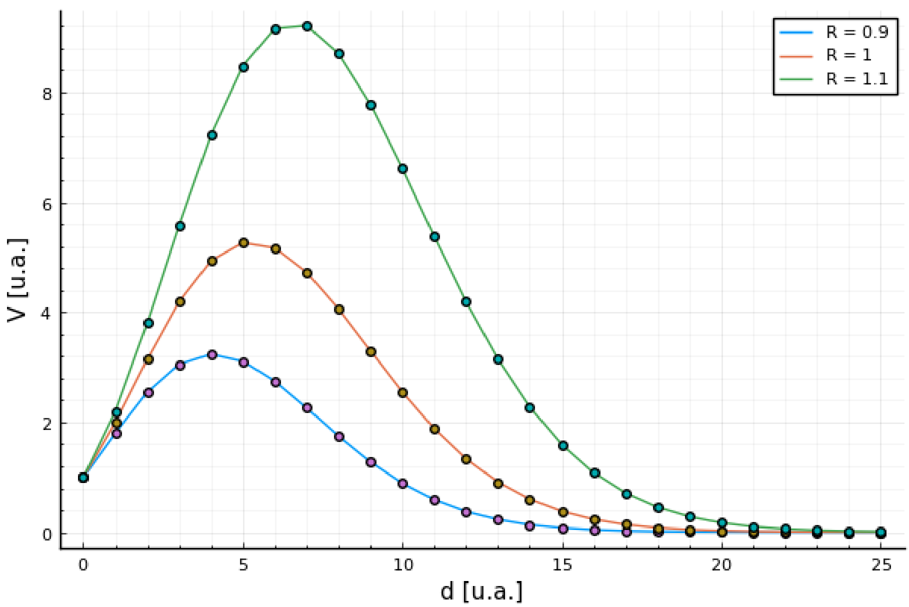

Figura 17: Tarefa 8b.

8 Tarefa 9

8.1 Introdução

Essa tarefa consiste de questões teóricas ligadas as tarefas anteriores.

8.2 Resultados

Para resolver a primeira parte, utilizei o mesmo código do exercício anterior e coloquei junto ao gráfico do volume da esfera de raio $R = 1$, o volume de um cubo em d dimensões. Olhando o gráfico da figura 18 fica claro o comportamento da diferença dos dois volumes.

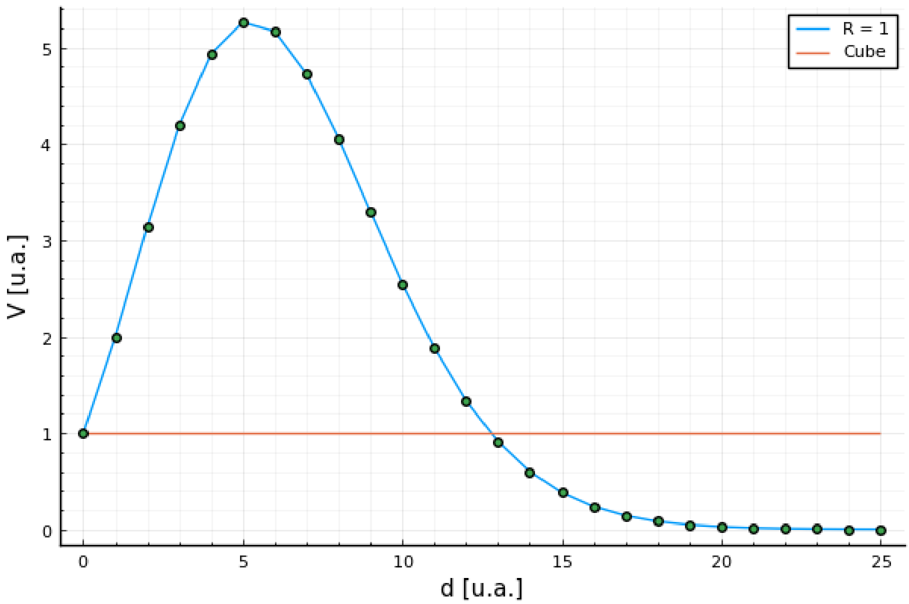

Figura 18: Volumes da esfera e do cubo em função de d.

Para o segundo problema, primeiro temos que converter as unidades, então: volume do átomo $V_a = (10^{-10})^d m^d$, volume macroscópico $V_m = (10^{-3})^d m^d$. Fazendo a razão temos o número de Avogrado: $N_a = V_m/V_a = 10^{7d}$