Universidade de São Paulo

Instituto de Física de São Carlos

Introdução à Física Computacional

—

Projeto 3

Levy Bruno do Nascimento Batista – 11212550

Prof. Francisco Castilho Alcaraz

São Carlos

10/2021 – 11/2021

1 Tarefa A

Nessa primeira tarefa, estudou-se diferentes métodos para calcular a derivada primeira da função $f(x) = cosh(4x)sen(x/4)$, além de um método para encontrar as segunda e terceira derivada; com isso, foi possível analisar os desvios desses valores em relação ao valor esperado e observar como eles variam conforme o parâmetro h é alterado.

No programa abaixo, o arquivo de entrada consiste em todos os valores de h que serão utilizados, enquanto no arquivo de saída se encontram o módulo dos desvios apresentados por cada método e para cada h, além dos valores exatos das derivadas primeira, segunda e terceira com 10^{-11} de precisão. Cada método e os valores exatos utilizados são calculados a partir de diferentes *functions*, que em sua maioria recebem o parâmetro h e o ponto em que estamos calculando as derivadas, no caso $x = 0,25$, como argumentos.

tarefa-A-11212550.f:

```
      program tarefaA

      real*8 f1, f2, f3, d1, d2, d3, ds3, df2, dp2, ds5, dss5, dtas5, x
     +       , h
      open(1, file='entrada-A-11212550')
      open(2, file='saida-A-11212550')

      x = 0.25d0
      f1 = d1(x)
      f2 = d2(x)
      f3 = d3(x)

      write(2, 700)"h|", "ds3|", "df2|", "dp2|", "ds5|", "dss5|",
     +             "dtas5|"
      do i = 1, 14
c       lê os valores de h que estão em um arquivo de entrada
        read(1,*)h
c       o módulo das diferenças são escritos na tabela no arquivo de saída
        write(2, 900)h, "|", abs(ds3(h, x)-f1), "|", abs(df2(h, x)-f1),
     +               "|", abs(dp2(h, x)-f1), "|", abs(ds5(h, x)-f1),
     +               "|", abs(dss5(h, x)-f2), "|", abs(dtas5(h, x)-f3),
     +               "|"
      end do

c     os valores exatos das derivadas primeira, segunda e terceira também são escritos
```

```
      write(2,*)"exatos:"
      write(2,800)"f'(0,25)␣=␣", f1
      write(2,800)"f''(0,25)␣=␣", f2
      write(2,800)"f'''(0,25)␣=␣", f3

      close(1)
      close(2)

c     as diferentes formatações utilizadas na saída
 700  format(A11, A8, A8, A8, A8, A8, A13)
 800  format(A, F14.11)
 900  format(F10.8, A, F7.4, A, F7.4, A, F7.4, A, F7.4, A,
     F7.4, A,
     +       F12.4, A)

      end

c     função que calcula a derivada simétrica de 3 pontos
      real*8 function ds3 (h, x)
        real*8 h, x
        ds3 = (dcosh(4*(x+h))*dsin((x+h)/4) - dcosh(4*(x-h))
     +         *dsin((x-h)/4))/(2*h)
        return
      end

c     função que calcula a derivada para frente de 2 pontos
      real*8 function df2 (h, x)
        real*8 h, x
        df2 = (dcosh(4*(x+h))*dsin((x+h)/4) -
            dcosh(4*x)*dsin(x/4))/h
        return
      end

c     função que calcula a derivada para trás de 2 pontos
      real*8 function dp2 (h, x)
        real*8 h, x
        dp2 = (dcosh(4*x)*dsin(x/4) -
            dcosh(4*(x-h))*dsin((x-h)/4))/h
        return
      end

c     função que calcula a derivada simétrica de 5 pontos
      real*8 function ds5 (h, x)
        real*8 h, x
        ds5 = (dcosh(4*(x-2*h))*dsin((x-2*h)/4) -
            8*dcosh(4*(x-h))
     +        *dsin((x-h)/4) + 8*dcosh(4*(x+h))*dsin((x+h)/4)
     +        - dcosh(4*(x+2*h))*dsin((x+2*h)/4))/(12*h)
        return
      end
```

```fortran

c     função que calcula a derivada segunda simétrica de 3 pontos
      real*8 function dss5 (h, x)
        real*8 h, x
        dss5 = (-dcosh(4*(x-2*h))*dsin((x-2*h)/4) + 16*dcosh(4*(x-h))
     +         *dsin((x-h)/4) + 16*dcosh(4*(x+h))*dsin((x+h)/4)
     +         - dcosh(4*(x+2*h))*dsin((x+2*h)/4) - 30*dcosh(4*x)
     +         *dsin(x/4))/(12*h**2)
        return
      end

c     função que calcula a derivada terceira anti-simétrica de 5 pontos
      real*8 function dtas5 (h, x)
        real*8 h, x
        dtas5 = (-dcosh(4*(x-2*h))*dsin((x-2*h)/4) + 2*dcosh(4*(x-h))
     +          *dsin((x-h)/4) - 2*dcosh(4*(x+h))*dsin((x+h)/4)
     +          + dcosh(4*(x+2*h))*dsin((x+2*h)/4))/(2*h**3)
        return
      end

c     função que calcula o valor exato da derivada primeira
      real*8 function d1 (x)
        real*8 x
        d1 = 4*dsinh(4*x)*dsin(x/4) + (1/4)*dcosh(4*x)*dcos(x/4)
        return
      end

c     função que calcula o valor exato da derivada segunda
      real*8 function d2 (x)
        real*8 x
        d2 = (255/16)*dcosh(4*x)*dsin(x/4) + 2*dsinh(4*x)*dcos(x/4)
        return
      end

c     função que calcula o valor exato da derivada terceira
      real*8 function d3 (x)
        real*8 x
        d3 = (253/4)*dsinh(4*x)*dsin(x/4) + (767/64)*dcosh(4*x)
     +       *dcos(x/4)
        return
      end
```

entrada-A-11212550:

```
0.5
0.2
0.1
0.05
0.01
0.005
0.001
0.0005
0.0001
0.00005
0.00001
0.000001
0.0000001
0.00000001
```

saida-A-11212550:

```
         h|    ds3|    df2|    dp2|    ds5|   dss5|
             dtas5|
0.50000000| 1.6794| 3.2669| 0.0919| 1.7782| 2.2400|
    61.4173|
0.20000000| 0.5466| 0.9688| 0.1245| 0.3519| 0.0493|
    7.6453|
0.10000000| 0.4240| 0.6222| 0.2257| 0.3831| 0.0879|
    2.9675|
0.05000000| 0.3947| 0.4922| 0.2971| 0.3849| 0.0902|
    1.8870|
0.01000000| 0.3854| 0.4048| 0.3660| 0.3850| 0.0904|
    1.5484|
0.00500000| 0.3851| 0.3948| 0.3754| 0.3850| 0.0904|
    1.5379|
0.00100000| 0.3850| 0.3870| 0.3831| 0.3850| 0.0904|
    1.5345|
0.00050000| 0.3850| 0.3860| 0.3840| 0.3850| 0.0904|
    1.5344|
0.00010000| 0.3850| 0.3852| 0.3848| 0.3850| 0.0904|
    1.5343|
0.00005000| 0.3850| 0.3851| 0.3849| 0.3850| 0.0904|
    1.5344|
0.00001000| 0.3850| 0.3850| 0.3850| 0.3850| 0.0904|
    1.5345|
0.00000100| 0.3850| 0.3850| 0.3850| 0.3850| 0.0903|
    13.1294|
0.00000010| 0.3850| 0.3850| 0.3850| 0.3850| 0.0906|
    21.5651|
0.00000001| 0.3850| 0.3850| 0.3850| 0.3850|
    0.2793|6938872.3388|
 exatos:
```

```
17 f'(0,25)␣=␣␣0.29360905953
18 f''(0,25)␣=␣␣3.79150970917
19 f'''(0,25) = 21.56508833786
```

Para a análise dos resultados, é importante destacar algumas considerações provenientes da literatura: os erros de truncamento são funções de h, indicados pelo termo adicional nas fórmulas presentes no roteiro na notação O; a diminuição no valor de h acarreta em desvios menores até certo limite, e depois disso os devios voltam a crescer. Nesse sentido, um valor "ideal"para h depende do método escolhido, da função $f(x)$ e do ponto no qual se está calculando as derivadas, além da ordem da derivada em questão; mesmo assim, é possível obter algumas conclusões a partir dos resultados obtidos.

Primeiramente, em relação aos métodos para derivada primeira, observa-se que os métodos de derivada para frente de 2 pontos e derivada para trás de 2 pontos têm um termo adicional $O(h)$, enquanto o da derivada simétrica de 3 pontos é $O(h^2)$ e o da derivada simétrica de 5 pontos é $O(h^4)$, ou seja, espera-se que, para valores menores de h, $O(h^4)$ vá a 0 mais rapidamente, seguido por $O(h^2)$ e, por fim, $O(h)$. De fato, é observado que, para a precisão de 4 casas decimais e até o valor máximo de h utilizado, os desvios desses 4 métodos aparentam convergir para o mesmo valor, sendo que isso ocorre primeiro para a derivada simétrica de 5 pontos (ds5, linha 6), seguida pela derivada simétrica de 3 pontos (ds3, linha 8) e as derivadas para frente e para trás de 2 pontos (dp2 e df2, ambas na linha 12). Para ds3 e df2, o h ideal é o menor possível, no caso $h = 10^{-8}$, já que os desvio só diminuem no intervalo testado, enquanto para dp2 esse valor ideal é $h = 0,5$, uma vez que os desvios tendem a crescer com a diminuição de h; por outro lado, ds5 não apresenta um comportamento monotônico no intervalo estudado, e o desvio mínimo se dá quando $h = 0,2$.

Agora, analisando os casos de derivadas superiores, para a derivada segunda simétrica de 5 pontos (dss5), vemos que ela tem um comportamento similar à ds5, uma vez que o termo adicional de ambas é $O(h^4)$, e também possui o valor ideal de h como $h = 0,2$; observe ainda que, para dss5, há um salto no valor do desvio para $h = 10^{-8}$, o que não ocorre em ds5. Por fim, para a derivada terceira anti-simétrica de 5 pontos (dtas5), de início observa-se que os seus desvios são, em média, maiores do que os obtidos nos casos anteriores, o que mostra que esse método provavelmente é o menos preciso; nesse contexto, destaca-se que o valor de h que minimiza o desvio para dtas5 é $h = 10^{-4}$, enquanto $h = 10^{-8}$ gera um desvio que torna o método inutilizável para esse valor de h.

2 Tarefa B

Já nessa segunda tarefa, o objetivo é estudar diferentes métodos de integração numérica, e comparar os desvios resultantes para cada um deles e para cada valor de N, que indica em quantas partes o intervalo de integração foi dividido. No caso, destaca-se que a integral estudada foi:

$$\int_0^1 e^{x/2} sen(\pi x) dx$$

No programa exposto, foi utilizado um arquivo de entrada que contém todos os valores de N analisados, além da constante pi ter sido definida com o auxílio do comando *common*, uma vez que ela também será usada nas *functions* implementadas; sobre elas, cada uma retorna o valor da integral calculada com um método, dado o valor de N como parâmetro. Além disso, há também uma *function* que calcula a integral de forma mais precisa por meio de uma expressão analítica. Todos os desvios e o resultado "exato"são escritos em um arquivo de saída.

tarefa-B-11212550.f:

```fortran
      program tarefaB

      real*8 valor, y, exato, trapezio, simpson, boole, pi
c     definindo pi como uma variável comum a ser usada nas demais funções
      common /cte/pi
      open(1, file='entrada-B-11212550')
      open(2, file='saida-B-11212550')

      pi = dacos(-1.d0)
c     guarda o valor exato da integral na variável valor
      valor = exato()
      write(2, 700)"N|", "h=(b-a)/N|", "Trapézio|", "Simpson|",
     +             "Boole|"
      do l = 1, 10
c       lê os valores de N diretamente do arquivo de entrada
        read(1,*)y
c       o módulo das diferenças para cada método vai sendo escrito na saída
        write(2, 900) int(y), "|", 1/y, "|", abs(trapezio(y) - valor),
     +                "|", abs(simpson(y) - valor), "|",
     +                abs(boole(y) - valor), "|"
      end do

c     escreve o valor exato da integral no arquivo de saída
      write(2, 800)"exato␣=␣", valor

 700  format(A5, A12, A13, A12, A12)
 800  format(A, F13.11)
 900  format(I4, A, D11.4, A, D11.4, A, D11.4, A, D11.4, A)

      close(1)
      close(2)

      end

c     função que calcula a integral pelo método do trapézio
      real*8 function trapezio (x)
```

```
        real*8 h, x, pi
        common /cte/pi
        h = 1/x
        trapezio = 0.d0
        do i = 1, int(x)-1
          trapezio = trapezio + h*dexp(i/(2*x))*dsin(pi*i/x)
        end do
        return
      end

c     função que calcula a integral pelo método de Simpson
      real*8 function simpson (x)
        real*8 h, x, pi
        common /cte/pi
        h = 1/x
        simpson = 0.d0
        do j = 1, int(x)/2
          simpson = simpson + (2*h/3)*dexp(j/x)*dsin(pi*2*j/x) +
     +  (4*h/3)*dexp((2*j-1)/(2*x))*dsin(pi*(2*j-1)/x)
        end do
        return
      end

c     função que calcula a integral pelo método de Boole
      real*8 function boole (x)
        real*8 h, x, pi
        common /cte/pi
        h = 1/x
        boole = 0.d0
        do k = 1, int(x)/4
          boole = boole + (64*h/45)*dexp((4*k-3)/(2*x))*
     +            dsin(pi*(4*k-3)/x) + (8*h/15)*dexp((4*k-2)/(2*x))*
     +            dsin(pi*(4*k-2)/x) + (64*h/45)*dexp((4*k-1)/(2*x))*
     +            dsin(pi*(4*k-1)/x) + (28*h/45)*dexp(2*k/x)*
     +            dsin(pi*4*k/x)
        end do
        return
      end

c     função que calcula a integral de forma mais exata
      real*8 function exato ()
        real*8 pi
        common /cte/pi
        exato = 4*pi*(1 + dexp(0.5d0))/(4*pi**2 + 1)
        return
      end
```

entrada-B-11212550:

```
12
24
48
96
192
384
768
1536
3072
6144
```

saida-B-11212550:

```
    N|  h=(b-a)/N|   Trapézio|    Simpson|      Boole|
   12| 0.8333D-01| 0.4821D-02| 0.2047D-04| 0.4411D-06|
   24| 0.4167D-01| 0.1204D-02| 0.1273D-05| 0.6771D-08|
   48| 0.2083D-01| 0.3010D-03| 0.7945D-07| 0.1053D-09|
   96| 0.1042D-01| 0.7524D-04| 0.4964D-08| 0.1644D-11|
  192| 0.5208D-02| 0.1881D-04| 0.3102D-09| 0.2598D-13|
  384| 0.2604D-02| 0.4703D-05| 0.1939D-10| 0.1110D-15|
  768| 0.1302D-02| 0.1176D-05| 0.1212D-11| 0.6661D-15|
 1536| 0.6510D-03| 0.2939D-06| 0.7494D-13| 0.2220D-15|
 3072| 0.3255D-03| 0.7348D-07| 0.6217D-14| 0.7772D-15|
 6144| 0.1628D-03| 0.1837D-07| 0.2776D-14| 0.1110D-14|
exato = 0.82228543287
```

Assim como na seção anterior, é possível estudar o comportamento dos desvios a partir dos termos adicionais que são dados na notação O. No caso, o método do trapézio tem um termo que depende de $O(h^3)$, o de Simpson depende de $O(h^5)$ e o de Boole depende de $O(h^7)$; note que aqui $h = \frac{1}{N}$, ou seja, um aumento de N diminui o valor de h.

Primeiramente, para o método do trapézio, observa-se que os desvios tendem a cair conforme se aumenta o N para todos os valores testados, de forma que o valor apropriado de N nesse caso é o maior possível, isto é, $N = 6144$; da mesma forma, os desvios no método de Simpson também caem conforme o N aumenta, e o valor ideal é novamente $N = 6144$, a diferença está no fato de que os desvios nesse último possuem ordem bem menor em comparação com os correspondentes calculados pelo método do trapézio (veja que o menor desvio no trapézio é da ordem de 10^{-7}, enquanto no Simpson ele é da ordem de 10^{-14}). Por fim, nota-se que os desvios no método de Boole se comportam de uma forma um pouco diferente: eles diminuem até $N = 384$, que é o valor apropriado de N para esse caso, oscila uma vez e depois volta a subir. Isso se dá porque, embora os erros tendem a diminuir com o aumento de h, eles são somados em cada iteração realizada no cálculo do método, de forma que, para muitas operações, eles podem voltar a crescer. Observa-se também que a ordem dos desvios de Boole é, em média, menor do que os encontrados no método de Simpson, e essas relações entre a ordem dos desvios estão de acordo com os termos adicionais na notação O, já citados anteriormente.

3 Tarefa C

Por fim, nessa última tarefa, o objetivo é estudar diferentes métodos para calcular as raízes de equações. No caso, destaca-se que a equação estudada é um polinômio de grau 3, que pode ter até 3 raízes reais, dada por:

$$x^3 - 14x - 20.$$

Além disso, destaca-se que os 3 métodos estudados foram a busca direta, o de Newton-Raphson e o da secante, de forma que foi possível comparar quantas iterações são necessárias para obter uma dada raíz da equação para uma mesma precisão em cada método, e assim inferir qual deles foi o mais eficiente nesse sentido.

No código abaixo, cada um dos métodos citados estão implementados em sub-rotinas diferentes que são chamadas no programa principal, além da precisão que é definida com o auxílio do comando *common* e passada para as sub-rotinas. Os chutes iniciais para todos os casos é -10 (no da secante foi utilizado -9 também porque são necessários dois chutes) e, diferentemente das tarefas anteriores, não há arquivo de entrada, mas em cada sub-rotina um arquivo de saída contendo os valores das 3 raízes após cada iteração do método de obtenção e o valor delas calculado diretamente é gerado. Observe ainda que em cada sub-rotina os métodos para obter uma raíz são repetidos 3 vezes, uma para cada raíz, e, antes de repetí-los, as variáveis utilizadas são resetadas, e um novo chute inicial arbitrário para nova raíz é dado; no caso da busca direta, esses novos chutes são $-2, 2$ e 0, no Newton-Raphson são 0 e 10, e no da secante são -1 e 0, além de 9 e 10.

tarefa-C-11212550.f:

```
      program tarefaC

      real*8 precisao, a, b, c, d
c     definindo a precisão como uma variável comum a ser utilizada nas demais funções
      common /cte/precisao
      precisao = 1.d-6
      a = -10.d0
      b = -10.d0
      c = -9.d0
      d = -10.d0

c     os métodos foram definidos por meio de sub-rotinas
      call procura(a)
      call newton(b)
      call secante(c, d)
```

```

      end

c     sub-rotina que aplica o método de busca direta
      subroutine procura (r)
        real*8 r, raux, h, precisao
        common /cte/precisao
c       i marca o número de iterações
        i = 0
c       h é o intervalo inicial utilizado
        h = 0.3d0
c       raux marca o primeiro r que muda o sinal do polinômio
        raux = r + h
        open(1, file='1saida-C-11212550')

        write(1,*)"Método␣de␣busca␣direta"
        write(1,100)"Iteração", "|", "r1", "|"
c       itera até o módulo do valor do polinômio estar dentro da precisão
 15     if (abs((((r+raux)/2)**3 - 14*((r+raux)/2) - 20)).gt.precisao)
     +    then
c         escreve o valor da raíz para cada iteração
          write(1,200)i, "|", (raux+r)/2, "|"
c         muda os valores de r e raux antes de "refinar" a raíz
          if (i.ne.0) then
            r = raux
            raux = raux + h
          end if
c         procura o primeiro raux que inverte o sinal do polinômio
 10       if ((r**3 - 14*r - 20)*(raux**3 - 14*raux - 20).gt.0) then
            raux = raux + h
            goto 10
          end if
c         divide pela metade o tamanho do intervalo a cada iteração
          r = raux - h
          h = -h/2
          i = i + 1
          goto 15
        end if
        write(1,200)i, "|", (raux+r)/2, "|"
        write(1,300)"exato:␣", 1 - dsqrt(11.d0)

c       o processo é repetido mais duas vezes para encontrar o valor das outras raízes
c       r assume um novo valor e i e h são "resetados"
        r = -2.2d0
```

```
      i = 0
      h = 0.3d0
      raux = r + h
      write(1,*)"␣"
      write(1,100)"Iteração", "|", "r2", "|"
 25   if (abs((((r+raux)/2)**3 - 14*((r+raux)/2) - 20)).gt.precisao)
     +    then
        write(1,200)i, "|", (raux+r)/2, "|"
        if (i.ne.0) then
          r = raux
          raux = raux + h
        end if
 20     if ((r**3 - 14*r - 20)*(raux**3 - 14*raux - 20).gt.0) then
          raux = raux + h
          goto 20
        end if
        r = raux - h
        h = -h/2
        i = i + 1
        goto 25
      end if
      write(1,200)i, "|", (raux+r)/2, "|"
      write(1,300)"exato:␣", -2.d0

      r = 0.d0
      i = 0
      h = 0.3d0
      raux = r + h
      write(1,*)"␣"
      write(1,100)"Iteração", "|", "r3", "|"
 35   if (abs((((r+raux)/2)**3 - 14*((r+raux)/2) - 20)).gt.precisao)
     +    then
        write(1,200)i, "|", (raux+r)/2, "|"
        if (i.ne.0) then
          r = raux
          raux = raux + h
        end if
 30     if ((r**3 - 14*r - 20)*(raux**3 - 14*raux - 20).gt.0) then
          raux = raux + h
          goto 30
        end if
        r = raux - h
        h = -h/2
        i = i + 1
        goto 35
      end if
```

```
        write(1,200)i, "|", (raux+r)/2, "|"
        write(1,300)"exato:␣", 1 + dsqrt(11.d0)

c       formatação esperada no arquivo de saída
 100    format(A10, A, A11, A)
 200    format(I8, A, F11.7, A)
 300    format(A, F10.7)
        close(1)
        return
      end

c     sub-rotina que aplica o método de Newton-Raphson
      subroutine newton (r)
        real*8 r, precisao
        common /cte/precisao
c       j marca o número de iterações
        j = 0
        open(2, file='2saida-C-11212550')

        write(2,*)"Método␣de␣Newton-Raphson"
        write(2,100)"Iteração", "|", "r1", "|"
c       itera até o módulo do valor do polinômio estar dentro
    da precisão
 40     if (abs(r**3 - 14*r - 20).gt.precisao) then
          write(2,200)j, "|", r, "|"
          r = r - (r**3 - 14*r - 20)/(3*r**2 - 14)
          j = j + 1
          goto 40
        end if
        write(2,200)j, "|", r, "|"
        write(2,300)"exato:␣", 1 - dsqrt(11.d0)

c       os valores de r e j são "resetados"
c       o processo é repetido duas vezes para encontrar as
    outras raízes
        r = 0.d0
        j = 0
        write(2,*)"␣"
        write(2,100)"Iteração", "|", "r2", "|"
 50     if (abs(r**3 - 14*r - 20).gt.precisao) then
          write(2,200)j, "|", r, "|"
          r = r - (r**3 - 14*r - 20)/(3*r**2 - 14)
          j = j + 1
          goto 50
        end if
        write(2,200)j, "|", r, "|"
        write(2,300)"exato:␣", -2.d0

        r = 10.d0
        j = 0
```

```
        write(2,*)"␣"
        write(2,100)"Iteração", "|", "r3", "|"
 60     if (abs(r**3 - 14*r - 20).gt.precisao) then
          write(2,200)j, "|", r, "|"
          r = r - (r**3 - 14*r - 20)/(3*r**2 - 14)
          j = j + 1
          goto 60
        end if
        write(2,200)j, "|", r, "|"
        write(2,300)"exato:␣", 1 + dsqrt(11.d0)

 100    format(A10, A, A11, A)
 200    format(I8, A, F11.7, A)
 300    format(A, F10.7)
        close(2)
        return
      end

c     sub-rotina que aplica o método da secante
      subroutine secante (r1, r2)
c       r1 e r2 são os chutes iniciais
c       tmp é uma variável auxiliar necessária na hora de atualizar r1 e r2
        real*8 r1, r2, tmp, precisao
        common /cte/precisao
c       k marca o número de iterações
        k = 0
        open(3, file='3saida-C-11212550')

        write(3,*)"Método␣da␣secante"
        write(3,100)"Iteração", "|", "r1", "|"
c       itera até o módulo do valor do polinômio estar dentro da precisão
 70     if (abs(r2**3 - 14*r2 - 20).gt.precisao) then
          write(3,200)k, "|", r2, "|"
          tmp = r2
          r2 = r2 - (r2**3 - 14*r2 - 20)*(r2 - r1)/
     +              (r2**3 - 14*r2 - r1**3 + 14*r1)
          r1 = tmp
          k = k + 1
          goto 70
        end if
        write(3,200)k, "|", r2, "|"
        write(3,300)"exato:␣", 1 - dsqrt(11.d0)

c       os valores de r1, r2 e k são "resetados"
c       o processo é repetido duas vezes para encontrar as outras raízes
        r1 = -1.d0
        r2 = 0.d0
```

```
      k = 0
      write(3,*)"␣"
      write(3,100)"Iteração", "|", "r2", "|"
80    if (abs(r2**3 - 14*r2 - 20).gt.precisao) then
        write(3,200)k, "|", r2, "|"
        tmp = r2
        r2 = r2 - (r2**3 - 14*r2 - 20)*(r2 - r1)/
     +           (r2**3 - 14*r2 - r1**3 + 14*r1)
        r1 = tmp
        k = k + 1
        goto 80
      end if
      write(3,200)k, "|", r2, "|"
      write(3,300)"exato:␣", -2.d0

      r1 = 9.d0
      r2 = 10.d0
      k = 0
      write(3,*)"␣"
      write(3,100)"Iteração", "|", "r3", "|"
90    if (abs(r2**3 - 14*r2 - 20).gt.precisao) then
        write(3,200)k, "|", r2, "|"
        tmp = r2
        r2 = r2 - (r2**3 - 14*r2 - 20)*(r2 - r1)/
     +           (r2**3 - 14*r2 - r1**3 + 14*r1)
        r1 = tmp
        k = k + 1
        goto 90
      end if
      write(3,200)k, "|", r2, "|"
      write(3,300)"exato:␣", 1 + dsqrt(11.d0)

100   format(A10, A, A11, A)
200   format(I8, A, F11.7, A)
300   format(A, F10.7)
      close(3)
      return
    end
```

No método da busca direta, cada iteração é marcada pelo ponto em que o polinômio muda de sinal, e é suposto que a raíz foi encontrada quando o módulo do valor do polinômio no ponto ficar menor que 10^{-6}. Esse método foi o que requisitou mais iterações para encontrar as raízes dentro da precisão, mesmo que, após apenas uma iteração, os valores obtidos estão relativamente mais próximos da raíz do que os correspondentes dos outros métodos.

1saida-C-11212550:

```
 Método de busca direta
Iteração|         r1|
```

```
        0|  -9.8500000|
        1|  -2.3500000|
        2|  -2.2750000|
        3|  -2.3125000|
        4|  -2.3312500|
        5|  -2.3218750|
        6|  -2.3171875|
        7|  -2.3148437|
        8|  -2.3160156|
        9|  -2.3166016|
       10|  -2.3168945|
       11|  -2.3167480|
       12|  -2.3166748|
       13|  -2.3166382|
       14|  -2.3166199|
       15|  -2.3166290|
       16|  -2.3166245|
exato:  -2.3166248

Iteração|          r2|
        0|  -2.0500000|
        1|  -2.0500000|
        2|  -1.9750000|
        3|  -2.0125000|
        4|  -1.9937500|
        5|  -2.0031250|
        6|  -1.9984375|
        7|  -2.0007813|
        8|  -1.9996094|
        9|  -2.0001953|
       10|  -1.9999023|
       11|  -2.0000488|
       12|  -1.9999756|
       13|  -2.0000122|
       14|  -1.9999939|
       15|  -2.0000031|
       16|  -1.9999985|
       17|  -2.0000008|
       18|  -1.9999996|
exato:  -2.0000000

Iteração|          r3|
        0|   0.1500000|
        1|   4.3500000|
        2|   4.2750000|
        3|   4.3125000|
        4|   4.3312500|
        5|   4.3218750|
        6|   4.3171875|
        7|   4.3148437|
```

```
        8|  4.3160156|
        9|  4.3166016|
       10|  4.3168945|
       11|  4.3167480|
       12|  4.3166748|
       13|  4.3166382|
       14|  4.3166199|
       15|  4.3166290|
       16|  4.3166245|
       17|  4.3166267|
       18|  4.3166256|
       19|  4.3166250|
       20|  4.3166247|
       21|  4.3166249|
       22|  4.3166248|
exato:  4.3166248
```

Já no método de Newton-Raphson, que é baseado em encontrar uma raíz a partir da seguinte fórmula iterativa:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

foi observado que ele foi o que necessitou de um menor número de iterações para encontrar as raízes desejadas, além de ter sido o único a dar todas elas iguais ao respectivo valor exato mostrado nos arquivos.

2saida-C-11212550:

```
 Método de Newton-Raphson
Iteração|         r1|
       0|-10.0000000|
       1| -6.9230769|
       2| -4.9591432|
       3| -3.7458017|
       4| -3.0297455|
       5| -2.6312633|
       6| -2.4274439|
       7| -2.3405741|
       8| -2.3182732|
       9| -2.3166337|
      10| -2.3166248|
exato: -2.3166248

Iteração|         r2|
       0|  0.0000000|
       1| -1.4285714|
       2| -1.7986677|
       3| -1.9471667|
       4| -1.9937336|
       5| -1.9998867|
```

```
        6|  -2.0000000|
exato:  -2.0000000

Iteração|          r3|
        0|  10.0000000|
        1|   7.0629371|
        2|   5.3420005|
        3|   4.5368648|
        4|   4.3302272|
        5|   4.3166816|
        6|   4.3166248|
exato:   4.3166248
```

Por fim, no método da secante, que é bastante similar ao método de Newton-Raphson, se baseando também em uma fórmula iterativa para encontrar as raízes, dada por:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

foi observado um número de iterações necessárias um pouco maior do que aquele visto no método anterior, embora ainda seja bem mais eficiente que o da busca direta nesse sentido. O ganho em número de operações para encontrar uma raíz com mesma precisão se dá pelo fato de que, no método da secante, a derivada observada no método de Newton-Raphson é calculada de forma aproximada aqui, por isso são necessários dois chutes ao invés de um em cada iteração.

3saida-C-11212550:

```
 Método da secante
Iteração|          r1|
        0|-10.0000000|
        1|  -6.5758755|
        2|  -5.4872002|
        3|  -4.3513935|
        4|  -3.6475268|
        5|  -3.1355799|
        6|  -2.7987149|
        7|  -2.5784800|
        8|  -2.4427974|
        9|  -2.3659554|
       10|  -2.3298850|
       11|  -2.3184318|
       12|  -2.3167005|
       13|  -2.3166252|
exato:  -2.3166248

Iteração|          r2|
        0|   0.0000000|
        1|  -1.5384615|
        2|  -1.7192269|
```

```
       3|  -1.8870553|
       4|  -1.9580596|
       5|  -1.9904854|
       6|  -1.9989708|
       7|  -1.9999716|
       8|  -1.9999999|
exato:  -2.0000000

Iteração|          r3|
       0|  10.0000000|
       1|   6.7315175|
       2|   5.7709997|
       3|   4.8875551|
       4|   4.4907819|
       5|   4.3428094|
       6|   4.3179715|
       7|   4.3166356|
       8|   4.3166248|
exato:   4.3166248
```

4 Referências

[1] Justo, D. A. R. et al, Cálculo Numérico - Versão GNU Octave, Versão de 19 de agosto de 2020.