Universidade de São Paulo

Introdução à Física Computacional - Projeto 3

João Victor Dell Agli Floriano - 10799783

1. Tarefa A

Esta tarefa tem como objetivo, por meio de algoritmos conhecidos de derivação numérica, implementar e calcular os valores das derivadas da função:

$$f(x) = cosh(3x)sin(\frac{x}{4}) \tag{1}$$

Que tem como derivadas analíticas:

$$f'(x) = 3sin(\frac{x}{4})sinh(3x) + \frac{1}{4}cos(\frac{x}{4})cosh(3x) \tag{2}$$

$$f''(x) = \frac{3}{4}cos(\frac{x}{4})sinh(3x) + \frac{143}{16}sin(\frac{x}{4})cosh(3x) \tag{3}$$

$$f'''(x) = \frac{1}{64}(1692sin(\frac{x}{4})sinh(3x) + 431cos(\frac{x}{4})cosh(3x)) \tag{4}$$

Que, em x = 0.5, têm os valores:

$$\begin{aligned} f'(0.5) &= 1.379915878252816 \\ f''(0.5) &= 5.790243159076722 \\ f'''(0.5) &= 22.736696903001942 \end{aligned} \tag{5}$$

Dentre as várias implementações do cálculo das derivadas, decorrentes da expansão de Taylor, usaremos as seguintes para este projeto:

Derivada para frente 2 pontos:

$$f'(x) = \frac{(f_1 - f_0)}{h} \tag{6}$$

Derivada para trás 2 pontos:

$$f'(x) = \frac{(f_0 - f_{-1})}{h} \tag{7}$$

Derivada simétrica de 3 pontos:

$$f'(x) = \frac{(f_1 - f_{-1})}{2h} \tag{8}$$

Derivada simétrica de 5 pontos:

$$f'(x) = \frac{(f_{-2} - 8f_{-1} + 8f_1 - f_2)}{12h} \tag{9}$$

Derivada segunda simétrica de 5 pontos:

$$f''(x) = \frac{(-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2)}{12h^2} \tag{10}$$

Derivada terceira anti-simétrica de 5 pontos:

$$f'''(x) = \frac{(-f_{-2} + 2f_{-1} - 2f_1 + f_2)}{2h^3} \tag{11}$$

Sendo $f_n = f(x_0 + nh)$. As derivadas serão calculadas para diferentes valores de h, que vão de $5 \cdot 10^{-1} \leq h \leq 1 \cdot 10^{-8}$, e comparadas aos resultados analíticos calculados previamente em (5), gerando um erro a ser analisado. A seguir, o programa que implementa todo esse algoritmo:

```
      !Declaração das funções para ajudar na simplicidade do código

      !função a ser usada
      double precision function dfx(x)
      implicit double precision (a-h, o-z)


      dfx = dcosh(3d0*x)*dsin((25d-2)*x)

      return
      end function dfx


      !função derivada primeira analitica
      double precision function dref(x)
      implicit double precision (a-h, o-z)


      dref = 3d0*dsin(x/4d0)*dsinh(3d0*x)
      & + dcos(x/4d0)*dcosh(x*3d0)/4d0


      return
      end function dref

      !função derivada segunda analítica
      double precision function d2ref(x)
      implicit double precision (a-h, o-z)


      d2ref = (3d0/2d0)*dcos(x/4d0)*dsinh(3d0*x)
      & + (143d0/16d0)*dsin(x/4d0)*dcosh(x*3d0)


      return
```

```
      end function d2ref

      !função derivada terceira analítica
      double precision function d3ref(x)
      implicit double precision (a-h, o-z)


      d3ref = (1d0/64d0)*(1692d0*dsin(x/4d0)*dsinh(3d0*x)
      & + 431d0*dcos(x/4d0)*dcosh(x*3d0))


      return
      end function d3ref




      !programa geral
      program main

      implicit double precision (a-h, o-z)
      double precision hvalues(14)
      double precision rvalues(14, 6)




      open(7, file = "saida-A-10799783.dat")




      x0 = 5d-1
      h = 0d0
      raux = 0d0

      hvalues(1) = 5d-1
      hvalues(2) = 2d-1
      hvalues(3) = 1d-1
      hvalues(4) = 5d-2
      hvalues(5) = 1d-2
      hvalues(6) = 5d-3
      hvalues(7) = 1d-3
      hvalues(8) = 5d-4
      hvalues(9) = 1d-4
      hvalues(10)= 5d-5
      hvalues(11)= 1d-5
      hvalues(12)= 1d-6
      hvalues(13)= 1d-7
      hvalues(14)= 1d-8



      !valores de referência:
```

```
!primeira derivada:        1.37992
!segunda derivada:         5.79024
!terceira derivada:       22.7367


!começo da tabela
write(7, 5) "h", "sim 3 pontos", "frente 2 pontos",
&"tras 2 pontos", "sim 5 pontos", "seg sim 5 pontos",
&"terc antisim"

!loop para calcular as derivadas para cada um dos 14 hs
do i = 1, 14

      h = hvalues(i)
      er = 0d0

      !derivada simetrica de 3 pontos
      raux = (dfx(x0 + h) -dfx(x0 - h))/(2*h)

      rvalues(i, 1) = dabs(dref(x0) - raux)


      write(*,*) "Derivada simetrica de 3 pontos: ", raux


      !derivada pra frente de 2 pontos
      raux = (dfx(x0 + h) - dfx(x0))/h

      rvalues(i, 2) = dabs(dref(x0) - raux)


      write(*,*) "Derivada pra frente de 2 pontos: ", raux


      !derivada pra tras de 2 pontos
      raux = (dfx(x0) - dfx(x0 - h))/h

      rvalues(i, 3) = dabs(dref(x0) - raux)


      write(*,*) "Derivada pra tras de 2 pontos: ", raux


      !derivada simetrica de 5 pontos
      raux = (dfx(x0 -2d0*h) - 8d0*dfx(x0 -h) + 8d0*dfx(x0 + h)
&-dfx(x0 +2d0*h))/(12d0*h)

      rvalues(i, 4) = dabs(dref(x0) - raux)

      write(*,*) "Derivada simetrica de 5 pontos: ", raux


      !derivada segunda simetrica de 5 pontos
      raux = (-dfx(x0 -2d0*h) +16d0*dfx(x0 -h) -30d0*dfx(x0)
& +16d0*dfx(x0 + h) -dfx(x0 + 2d0*h))/(12d0*(h**2))
```

```
            rvalues(i, 5) = dabs(d2ref(x0) - raux)

            write(*,*) "Derivada segunda simetrica de 5 pontos: ",raux

            !derivada terceira anti-simetrica de 5 pontos
            raux = (-dfx(x0 -2d0*h) +2d0*dfx(x0 -h) -2d0*dfx(x0 +h)
      & +dfx(x0 +2d0*h))/(2d0*(h**3))

            rvalues(i, 6) = dabs(d3ref(x0) - raux)

            write(*,*) "Derivada terceira anti-sim de 5 pontos: ",raux


            write(*,*) " "

            !escrita dos valores na tabela
            write(7, 4) hvalues(i), (rvalues(i, k), k =1, 6)
            write(7, *)

      end do

      !escrita dos valores exatos na tabela
      write(7, 6) "exatos", dref(x0), dref(x0), dref(x0), dref(x0),
      &d2ref(x0), d3ref(x0)

      !formatos de escrita
4     format(F22.11 , F22.11)
5     format(A22)
6     format(A22, F22.11)


      close(7)




      stop
      end program main
```

Rodando este programa, temos:

```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Terceiro ...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Terc
iro Projeto/tarefaA$ f77 derivada.f -o derivada
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Terc
iro Projeto/tarefaA$ ./derivada
 Derivada simetrica de 3 pontos:    2.4907794381917112
 Derivada pra frente de 2 pontos:    4.3949867931968267
 Derivada pra tras de 2 pontos:   0.58657208318659582
 Derivada simetrica de 5 pontos:   0.52425230542584822
 Derivada segunda simetrica de 5 pontos:    4.9532467244921463
 Derivada terceira anti-sim de 5 pontos:    47.196651186380713

 Derivada simetrica de 3 pontos:    1.5354447608663040
 Derivada pra frente de 2 pontos:    2.1413630305572640
 Derivada pra tras de 2 pontos:   0.92952649117534425
 Derivada simetrica de 5 pontos:    1.3634064914279656
 Derivada segunda simetrica de 5 pontos:    5.7727914009552723
 Derivada terceira anti-sim de 5 pontos:    25.805740415750797

 Derivada simetrica de 3 pontos:    1.4180552826203119
 Derivada pra frente de 2 pontos:    1.7108894658735778
 Derivada pra tras de 2 pontos:    1.1252210993670455
 Derivada simetrica de 5 pontos:    1.3789254565383140
 Derivada segunda simetrica de 5 pontos:    5.7891839877838915
 Derivada terceira anti-sim de 5 pontos:    23.477895649198494

 Derivada simetrica de 3 pontos:    1.3894047773915965
 Derivada pra frente de 2 pontos:    1.5345748773950396
 Derivada pra tras de 2 pontos:    1.2442346773881534
```

Que, por sua vez, rende o arquivo de saída:

h	sim 3 pontos	frente 2 pontos	tras 2 pontos	sim 5 pontos	seg sim 5 pontos	terc antisim
0.50000000000	1.11086355994	3.01507091494	0.79334379507	0.85566357283	0.83699643458	24.45995428338
0.20000000000	0.15552888261	0.76144715230	0.45038938708	0.01650938682	0.01745175812	3.06904351275
0.10000000000	0.03813940437	0.33097358762	0.25469477889	0.00099042171	0.00105917129	0.74119874620
0.05000000000	0.00948889914	0.15465899914	0.13568120086	0.00006126927	0.00006571391	0.18370727997
0.01000000000	0.00037896937	0.02933349419	0.02857555544	0.00000009771	0.00000010490	0.00732802977
0.00500000000	0.00009473776	0.01457075928	0.01438128375	0.00000000611	0.00000000655	0.00183184934
0.00100000000	0.00000378945	0.00289891434	0.00289133544	0.00000000001	0.00000000000	0.00007323031
0.00050000000	0.00000094736	0.00144850857	0.00144661384	0.00000000000	0.00000000047	0.00001916245
0.00010000000	0.00000003789	0.00028955006	0.00028947427	0.00000000000	0.00000000098	0.00002324807
0.00005000000	0.00000000947	0.00014476555	0.00014474661	0.00000000000	0.00000002041	0.00043958170
0.00001000000	0.00000000038	0.00002895159	0.00002895084	0.00000000000	0.00000047561	0.05063067743
0.00000100000	0.00000000002	0.00000289511	0.00000289515	0.00000000003	0.00000937762	32.77445432826
0.00000010000	0.00000000030	0.00000028919	0.00000028979	0.00000000049	0.00320564322	22.73669690300
0.00000001000	0.00000000331	0.00000003661	0.00000003000	0.00000000285	1.19491003752	55511173.96795472503
exatos	1.37991587825	1.37991587825	1.37991587825	1.37991587825	5.79024315908	22.73669690300

De maneira mais organizada:

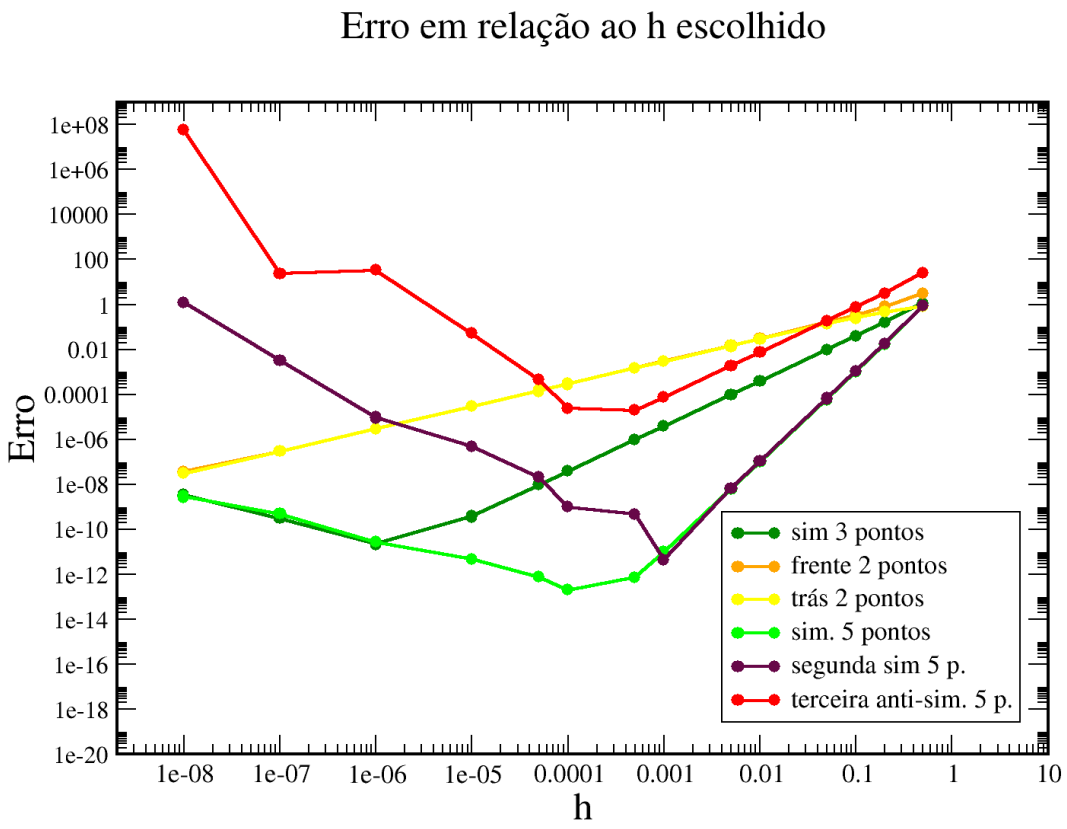
h	**sim. 3 pontos**	**frente 2 pontos**	**tras 2 pontos**	**sim. 5 pontos**
0.5	1.11086355994	3.01507091494	0.79334379507	0.85566357283
0.2	0.15552888261	0.76144715230	0.45038938708	0.01650938682
0.1	0.03813940437	0.33097358762	0.25469477889	0.00099042171
0.05	0.00948889914	0.15465899914	0.13568120086	0.00006126927

0.01	0.00037896937	0.02933349419	0.02857555544	0.00000009771
0.005	0.00009473776	0.01457075928	0.01438128375	0.00000000611
0.001	0.00000378945	0.00289891434	0.00289133544	0.00000000001
0.0005	0.00000094736	0.00144850857	0.00144661384	0.00000000000
0.0001	0.00000003789	0.00028955006	0.00028947427	0.00000000000
0.00005	0.00000000947	0.00014476555	0.00014474661	0.00000000000
0.00001	0.00000000038	0.00002895159	0.00002895084	0.00000000000
0.000001	0.00000000002	0.00000289511	0.00000289515	0.00000000003
0.0000001	0.00000000030	0.00000028919	0.00000028979	0.00000000049
0.00000001	0.00000000331	0.00000003661	0.00000003000	0.00000000285
Exatos	**1.37991587825**	**1.37991587825**	**1.37991587825**	**1.37991587825**

h	**seg. sim. 5 pontos**	**terc. anti-sim. 5 pontos**
0.5	0.83699643458	24.45995428338
0.2	0.01745175812	3.06904351275
0.1	0.00105917129	0.74119874620
0.05	0.00006571391	0.18370727997
0.01	0.00000010490	0.00732802977
0.005	0.00000000655	0.00183184934
0.001	0.00000000000	0.00007323031
0.0005	0.00000000047	0.00001916245
0.0001	0.00000000098	0.00002324807
0.00005	0.00000002041	0.00043958170
0.00001	0.00000047561	0.05063067743
0.000001	0.00000937762	32.77445432826
0.0000001	0.00320564322	22.73669690300
0.00000001	1.19491003752	55511173.96795472503
Exatos	**5.79024315908**	**22.73669690300**

Para uma melhor análise dos resultados, é possível graficar, em escala log X log os valores dos erros, agora no número de casas máxima permitida pela dupla precisão,

de cada método em relação ao h escolhido:



Analisando o gráfico, percebemos alguns pontos curiosos. O valor mais exato possível dados estes métodos para estes valores de h escolhidos foi o de $1 \cdot 10^{-4}$ na primeira derivada simétrica de 5 pontos, rendendo um erro abaixo de $1 \cdot 10^{-12}$. Para todos os métodos:

Método	**Melhor h**	**Erro**
Simétrica 3 pontos	0.0000010000000	0.0000000000227
Frente 2 pontos	0.0000000100000	0.0000000366147
Trás 2 pontos	0.0000000100000	0.0000000299987
Simétrica 5 pontos	0.0001000000000	0.0000000000002
Segunda simétrica 5 pontos	0.0010000000000	0.0000000000043
Terceira anti-simétrica 5 pontos	0.0005000000000	0.0000191624504

Outro ponto importante a ser destacado do gráfico é que, de todos os métodos, a derivada terceira anti-simétrica de 5 pontos é a menos precisa, chegando a explodir conforme h diminui, possivelmente por conta da divisão por um h^3.

2. Tarefa B

Para esta tarefa, o objetivo foi de implementar métodos de integração numérica. Para isso, usou-se os métodos conhecidos de integração numérica, baseados na seguinte segmentação:

$$\int_a^b f(x)dx = \int_a^{a+2h} f(x)dx + \int_{a+2h}^{a+4h} f(x)dx + \cdots + \int_{b-2h}^{b} f(x)dx \tag{12}$$

Tal que, um intervalo de integração será calculado de três maneiras diferentes para este projeto:

Regra do Trapézio:

$$\int_{-h}^{h} f(x)dx = \frac{h}{2}(f_{-1} + 2f_0 + f_1) + O(h^3) \tag{13}$$

Regra de Simpson:

$$\int_{-h}^{h} f(x)dx = \frac{h}{3}(f_1 + 4f_0 + f_{-1}) + O(h^5) \tag{14}$$

Regra de Simpson 3/8:

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3) + O(h^5) \tag{15}$$

Regra de Boole:

$$\int_{x_0} x_4 f(x)dx = \frac{2h}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) + O(h^7) \tag{16}$$

Sendo $f_n = f(a + nh)$ e $h = (a - b)/N$. Para o cálculo no intervalo [0, 1], escolhemos alguns valores diferentes de N, para avaliar a eficiência de três destes métodos: trapézio, simpson e boole. A seguir, o código que implementa este algoritmo:

```
!declaração de funções para ajudar na simplicidade do código

!função a ser usada
double precision function dfx(x)
implicit double precision(a-h, o-z)
```

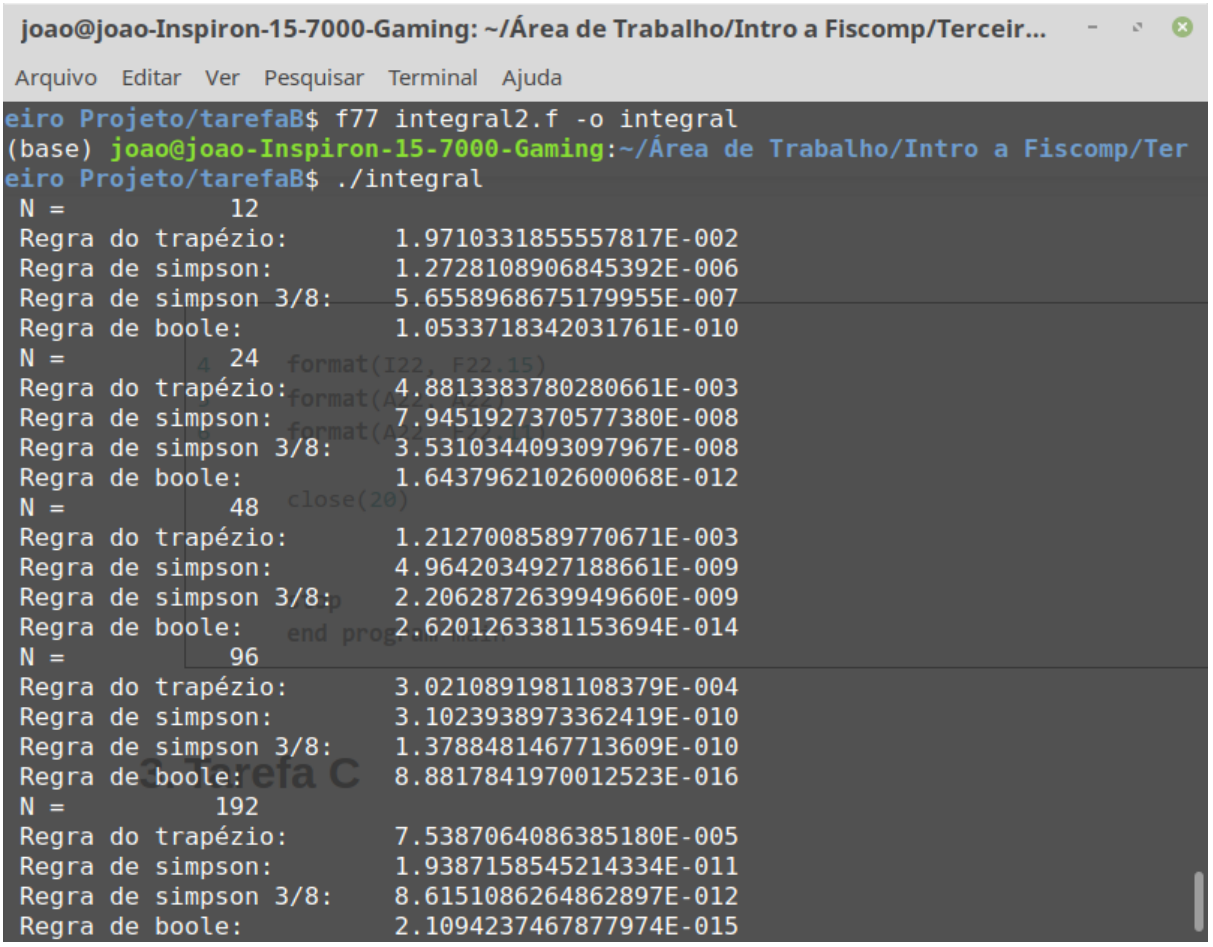
```
      pi = 4d0*datan(1d0)

      dfx = dexp(x/2)*dsin(pi*x)

      return
      end function dfx


      !integral analítica da função a ser usada
      double precision function drefx(a, b)
      implicit double precision(a-h, o-z)

      pi = 4d0*datan(1d0)

      drefx = -(2d0*dexp(b/2d0)*(2d0*pi*dcos(pi*b) - dsin(pi*b)))
     &/(1d0 + 4d0*(pi**2d0)) +(2d0*dexp(a/2d0)*(2d0*pi*dcos(pi*a)
     &- dsin(pi*a)))/(1d0 + 4d0*(pi**2d0))

      return
      end function drefx




      !programa
      program main

      implicit double precision(a-h, o-z)
      integer rnvalues(10)

      double precision evalues(10,3)

      rnvalues(1)= 12
      rnvalues(2)= 24
      rnvalues(3)= 48
      rnvalues(4)= 96
      rnvalues(5)= 192
      rnvalues(6)= 384
      rnvalues(7)= 768
      rnvalues(8)= 1536
      rnvalues(9)= 3072
      rnvalues(10)= 6144

      open(20, file = "saida-B-10799783.dat")

      write(20, 5) "N", "trapezio", "simpson", "boole"

      do j = 1, 10


      a = 0d0
      b = 1d0
      rint = 0d0
      iseg = rnvalues(j)
      dh = 1d0/iseg
```

```
write(*,*) "N = ", iseg


!regra do trapezio
do i = 1, iseg

      a = a + dh
      rint = rint + (dh/2)*(dfx(a - dh) + 2*dfx(a) +dfx(a + dh))

end do

a = 0d0
b = 1d0

x = dabs(rint/2 - drefx(a, b))
evalues(j, 1) = x

write(*,*) "Regra do trapézio:   ", x




!regra de simpson
rint = 0d0
a = 0d0
dh = 1d0/(2*iseg)
b = 1d0

do i = 1, iseg

      rint = rint + (dh/3)*(dfx(a) + 4*dfx(a + dh)
& + dfx(a + 2*dh))
      a = a + 2*dh
      !a = a + 2*dh

end do

a = 0d0
b = 1d0

x = dabs(rint - drefx(a, b))
evalues(j, 2) = x

write(*,*) "Regra de simpson:    ", x




!regra de simpson 3/8
rint = 0d0
a = 0d0
b = 1d0
dh = 1d0/(3*iseg)
```

```
      do i = 1, iseg

            rint = rint + (3*dh/8)*(dfx(a) + 3*dfx(a + dh)
     & + 3*dfx(a + 2*dh) + dfx(a + 3*dh))
            a = a + 3*dh

      end do
      a = 0d0
      b = 1d0

      x = dabs(rint - drefx(a, b))

      write(*,*) "Regra de simpson 3/8: ", x




      !regra de Boole
      rint = 0d0
      a = 0d0
      b = 1d0
      dh = 1d0/(4*iseg)

      do i = 1, iseg

            rint  = rint + (2*dh/45)*(7*dfx(a) + 32*dfx(a + dh)
     &+ 12*dfx(a + 2*dh) + 32*dfx(a + 3*dh) + 7*dfx(a + 4*dh))


            a = a + 4*dh

      end do
      a = 0d0
      b = 1d0

      x = dabs(rint - drefx(a, b))
      evalues(j, 3) = x

      write(*,*) "Regra de boole:      ", x

      write(20, 4) rnvalues(j) ,(evalues(j, k), k = 1, 3)
      write(20, *) " "




      end do

      write(20, 6) "exatos", drefx(a, b), drefx(a, b), drefx(a, b)


4     format(I22, 3F22.15)
5     format(A22, 3A22)
6     format(A22, 3F22.11)
```

```
      close(20)


      stop
      end program main
```

Rodando o programa: temos:



Que rende o arquivo de saída:

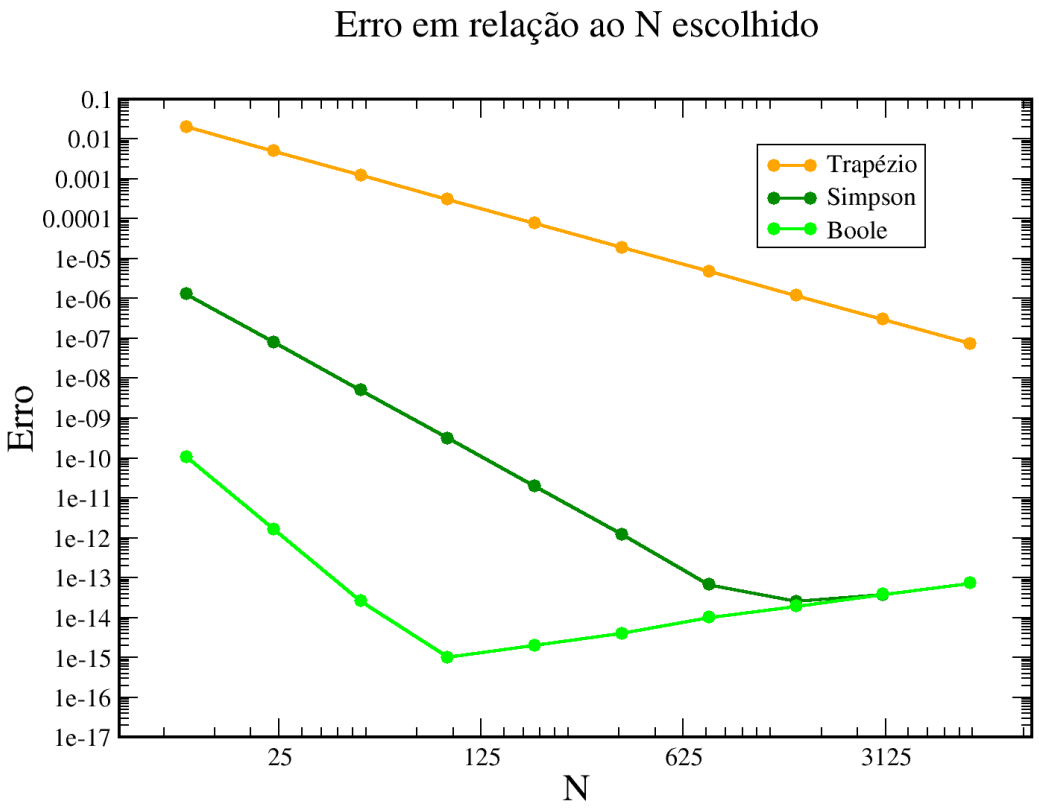
N	trapezio	simpson	boole
12	0.019710331855558	0.000001272810891	0.000000000105337
24	0.004881338378028	0.000000079451927	0.000000000001644
48	0.001212700858977	0.000000004964203	0.000000000000026
96	0.000302108919811	0.000000000310239	0.000000000000001
192	0.000075387064086	0.000000000019387	0.000000000000002
384	0.000018828819410	0.000000000001215	0.000000000000004
768	0.000004704935050	0.000000000000067	0.000000000000010
1536	0.000001175948364	0.000000000000025	0.000000000000019
3072	0.000000293951358	0.000000000000037	0.000000000000038
6144	0.000000073483274	0.000000000000073	0.000000000000073
exatos	0.82228543287	0.82228543287	0.82228543287

De maneira mais organizada:

N	**Trapézio**	**Simpson**	**Boole**
12	0.019710331855558	0.000001272810891	0.000000000105337
24	0.004881338378028	0.000000079451927	0.000000000001644
48	0.001212700858977	0.000000004964203	0.000000000000026
96	0.000302108919811	0.000000000310239	0.000000000000001
192	0.000075387064086	0.000000000019387	0.000000000000002
384	0.000018828819410	0.000000000001215	0.000000000000004
768	0.000004704935050	0.000000000000067	0.000000000000010
1536	0.000001175948364	0.000000000000025	0.000000000000019
3072	0.000000293951358	0.000000000000037	0.000000000000038
6144	0.000000073483274	0.000000000000073	0.000000000000073
Exatos	**0.82228543287**	**0.82228543287**	**0.82228543287**

Para eleger os melhores valores de N para cada método, grafica-se os dados

novamente em escala log X log, rendendo:



A partir do gráfico, é possível concluir que, para cada método, temos:

Método	**Melhor N**	**Erro**
Trapézio	6144	0.000000073483274
Simpson	1536	0.000000000000025
Boole	96	0.000000000000001

3. Tarefa C

Para esta última tarefa, o objetivo foi de encontrar as raízes do seguinte polinômio de terceiro grau:

$$f(x) = x^3 - 14x - 20 \tag{17}$$

Para isso, foram utilizadas três técnicas conhecidas de cálculo numérico de raízes de funções:

Busca direta:
A busca direta, como o próprio nome diz, é uma busca pelas raízes de funções diretamente, a qual a partir de um chute inicial de x, itera-se a partir dele por toda a reta real, verificando os valores da função. Caso entre o intervalo [x, x + passo] haja uma troca de sinal em f(x), naquele intervalo deve necessariamente existir pelo

menos uma raíz da função analisada. Tendo identificado troca de sinal, quebra-se o intervalo no meio com x = x + passo/2, analisando assim a primeira metade e a segunda metade, procurando novamente uma troca de sinal. Faz-se isso recursivamente até que uma precisão desejada seja atingida, e retorna-se o valor do x, que deve assim se aproximar do valor real da raíz da função.

Método de Newton-Raphson:
Este método usa ferramentas diferentes, porém sendo construído com o auxílio do anterior, recorrendo-se à derivada da função em questão, que a partir de um chute inicial de x, se num intervalo [x, x + passo] há inversão de sinal, a seguinte operação iterativa é feita:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{18}$$

Realiza-se este processo recursivamente até que a diferença entre o x e seu valor anterior sejam menores que uma precisão desejada, retornando assim o valor de x.

Método da Secante:
Este método, ao contrário do anterior, não se utiliza da derivada da função, mas apenas da própria e de valores de x. Inicia-se a busca agora chutando dois valores iniciais de x, x0 = a, xm = a + passo, e realiza-se a busca, procurando pela troca de sinal, e ao identificá-la, o seguinte processo iterativo se inicia:

$$x_{i+1} = x - f(x_i)\frac{(x_i - x_{i-1})}{(f(x_i) - f(x_{i-1}))} \tag{19}$$

Realiza-se este processo recursivamente até que a diferença entre o x e seu valor anterior sejam menores que uma precisão desejada, retornando assim o valor de x.

Implementando os algoritmos acima, temos:

```
      !declaração de funções para ajudar na simplicidade do código

      !função a ser usada
      double precision function dfx(x)
      implicit double precision (a-h, o-z)

      dfx = x**3d0 - 14d0*x - 20d0

      return
      end function dfx


      !derivada da função a ser usada calculada por metodos anteriores
```

```
      double precision function deri(x)

      implicit double precision (a-h, o-z)

      double precision dfx


      n = 1

      h = 1d-2

      !derivada simetrica de 5 pontos
      deri = (dfx(x -2d0*n*h) - 8d0*dfx(x -n*h) + 8d0*dfx(x + n*h)
     &-dfx(x +2d0*n*h))/(12d0*h)

      return
      end function deri




      !programa
      program main


      implicit double precision (a-h, o-z)



      !valores de referencia: -2, 4.31662, -2.31662

      x0 = -10d0
      x = x0
      r0 = 5d-1
      rint = r0


      open(20, file = "saida-C-10799783.dat")



      !busca direta
      l = 1

      write(20,*) "Busca Direta"
      do i = 1, 50



            if( (dfx(x) .GE. 0d0 .AND. dfx(x + rint) .LT.0d0).OR.
     & ( dfx(x + rint) .GE. 0d0 .AND. dfx(x) .LT. 0d0) ) then
```

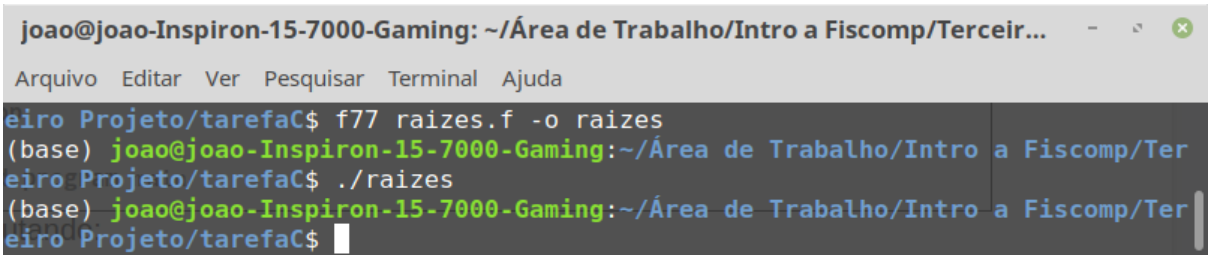
```
              x = x + rint/2d0
              rint = rint/2d0
              k = 1


              write(20,1) "r", l
              l = l + 1

              !buscar a raiz até que o erro seja menor que 1d-6
              do while( rint .GT. 1d-6)


              if( (dfx(x) .GE. 0d0 .AND. dfx(x + rint).LT.0d0)
& .OR.( dfx(x + rint) .GE. 0d0 .AND. dfx(x) .LT. 0d0) ) then



                    x = x + rint/2d0

              else if((dfx(x).GE.0d0.AND.dfx(x - rint).LT.0d0)
&.OR. ( dfx(x - rint) .GE. 0d0 .AND. dfx(x) .LT. 0d0)) then



                    x = x - rint/2d0

              end if



              rint= rint/2d0

              if (k .LE. 6) then
                    write(20,2) "i = ", k, x
              end if
              k = k + 1


              end do


       end if

       rint = r0

       x = x0 + i*r0

end do
```

```
!Newtown-Raphson
write(20,*) " "
write(20,*) "Newton-Raphson"

x0 = -10d0
x = x0
r0 = 5d-1


l = 1 !variavel para o arquivo de saída

do i = 1, 50

       k = 1


       if( (dfx(x0) .GE. 0d0 .AND. dfx(x0 + r0) .LT.0d0).OR.
& ( dfx(x0 + r0) .GE. 0d0 .AND. dfx(x0) .LT. 0d0) ) then

              write(20,1) "r", l
              l = l + 1
              do while(dabs(xant - x0) .GT. 1d-6)

              xant = x
              x = x0 - dfx(x0)/deri(x0)
              x0 = x

              if (k .LE. 6) then
                     write(20,2) "i = ", k, x
              end if
              k = k + 1


              end do


       end if


       xant = 0d0
       x0 = -10d0
       x0 = x0 + i*r0
end do




!Secante
write(20,*) " "
write(20,*) "Secante"
```

```
      x0 = -10d0
      r0 = 5d-1
      xm = x0 + r0

      l = 1 !variavel para o arquivo de saída


      do i = 1, 50

            k = 1


            if( (dfx(x0) .GE. 0d0 .AND. dfx(x0 + r0) .LT.0d0).OR.
     & ( dfx(x0 + r0) .GE. 0d0 .AND. dfx(x0) .LT. 0d0) ) then

                  write(20,1) "r", l
                  l = l + 1

                  do while(dabs(xant - x0) .GT. 1d-6)

                  if((dfx(x)- dfx(xm)) .EQ. 0d0) then
                        exit
                  end if

                  xant = x
                  x = x0 - dfx(x0)*(x0 - xm)/(dfx(x0) - dfx(xm))
                  xm = x0
                  x0 = x

                  if (k .LE. 6) then
                        write(20,2) "i = ", k, x
                  end if
                  k = k + 1


                  end do

            end if




            xant = 0d0
            x0 = -10d0
            x0 = x0 + i*r0
            xm = x0 - r0
      end do



1     format(A1, I1)
2     format(A4, I1, F20.15)

      close(20)
```

```
      stop

      end program main
```

Que, executando:



Nos rende o seguinte arquivo:

```
 Busca Direta
r1
i = 1  -2.375000000000000
i = 2  -2.312500000000000
i = 3  -2.343750000000000
i = 4  -2.328125000000000
i = 5  -2.320312500000000
i = 6  -2.316406250000000
r2
i = 1  -1.875000000000000
i = 2  -1.937500000000000
i = 3  -1.968750000000000
i = 4  -1.984375000000000
i = 5  -1.992187500000000
i = 6  -1.996093750000000
r3
i = 1   4.375000000000000
i = 2   4.312500000000000
i = 3   4.343750000000000
i = 4   4.328125000000000
i = 5   4.320312500000000
i = 6   4.316406250000000

 Newton-Raphson
r1
i = 1  -2.368421052631554
i = 2  -2.323315634826530
i = 3  -2.316766911614408
i = 4  -2.316624857133375
i = 5  -2.316624790355415
r2
i = 1  -2.000000000000000
i = 2  -2.000000000000000
r3
i = 1   4.352941176470605
i = 2   4.317025663439338
i = 3   4.316624840013170
i = 4   4.316624790355401

 Secante
r1
i = 1  -2.428571428571428
i = 2  -2.351874244256349
i = 3  -2.325540980020492
i = 4  -2.317537332787099
i = 5  -2.316650901985038
i = 6  -2.316624868970207
r2
i = 1  -2.000000000000000
r3
i = 1   4.424778761061947
i = 2   4.305516232070152
```

Mais organizadamente:

BUSCA DIRETA

Iteração	r1	r2	r3

1	-2.375000000000000	-1.875000000000000	4.375000000000000
2	-2.312500000000000	-1.937500000000000	4.312500000000000
3	-2.343750000000000	-1.968750000000000	4.343750000000000
4	-2.328125000000000	-1.984375000000000	4.328125000000000
5	-2.320312500000000	-1.992187500000000	4.320312500000000
6	-2.316406250000000	-1.996093750000000	4.316406250000000
Exatos	**-2.3166247903553998**	**-2.000000000000000**	**4.3166247903553998**

NEWTON-RAPHSON

Iteração	r1	r2	r3
1	-2.368421052631554	-2.000000000000000	4.352941176470605
2	-2.323315634826530	-2.000000000000000	4.317025663439338
3	-2.316766911614408	-2.000000000000000	4.316624840013170
4	-2.316624857133375	-2.000000000000000	4.316624790355401
5	-2.316624790355415	-2.000000000000000	4.316624790355401
6	-2.316624790355415	-2.000000000000000	4.316624790355401
Exatos	**-2.3166247903553998**	**-2.000000000000000**	**4.3166247903553998**

SECANTE

Iteração	r1	r2	r3
1	-2.428571428571428	-2.000000000000000	4.424778761061947
2	-2.351874244256349	-2.000000000000000	4.305516232070152
3	-2.325540980020492	-2.000000000000000	4.316261666026671
4	-2.317537332787099	-2.000000000000000	4.316626040394230
5	-2.316650901985038	-2.000000000000000	4.316624790215096
6	-2.316624868970207	-2.000000000000000	4.316624790355400

Exatos	**-2.3166247903553998**	**-2.000000000000000**	**4.3166247903553998**

Sendo os valores em verde relativos às iterações em que a raiz alcançou a precisão de 1d-6. A partir dos valores obtidos, é possível perceber que, para este caso, o método de Newton-Raphson obteve sucesso em chegar nas raízes da equação desejada.