Universidade de São Paulo
Instituto de Física de São Carlos

Projeto 3: Introdução à física computacional

Julia Martins Simão - 13694997

São Carlos
2025

Sumário

1 Introdução

As tarefas descritas no presente projeto abrangem diferentes ferramentas do cálculo númerico: derivação, integração e busca por raizes de uma função. Para que fosse feita uma análise mais profunda da precisão de cada método estudado, foi necessário que todos os programas fossem de dupla precisão, de modo que o erro da máquina fosse da ordem de 10^{-12}.

Tarefa 1

Derivação numérica

Escreva um código FORTRAN que forneça os dados da tabela I para as derivadas da função

$$f(x) = e^{2x^2} \tanh 2x$$

para $x = \frac{1}{2}$. Na última linha escreva os valores numéricos exatos com precisão 10^{-11} obtidos mediante a expressão analítica que você deve derivar. Diga em cada caso qual o valor de h mais apropriado para uso. Explique seus resultados.

Derivação numérica

A derivação numérica é fundamentada na interpolação de conjuntos de pontos. Sabe-se, do cálculo, que a expansão de Taylor para uma função $f(x+h)$ com h pequeno é

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2!}f''(x)h^2 + ...$$

A partir disso, é possível obter a definição da derivada de uma função, sendo ela

$$f'(x) \equiv \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

Para um conjunto de 2 pontos "à frente", tem-se que

$$f'_{2f} = f'(x) + \epsilon \tag{1}$$

Aqui, $\epsilon \propto h$. Seguindo a mesma lógica, a derivada de um conjunto de dois ponto "para trás"é

$$f'_{2t} = -f'(x) + \epsilon \tag{2}$$

Como qualquer aproximação numérica, a quantidade de pontos considerados é importante, de modo que o erro intrínseco dos valores obtidos sejam cada vez menores. Combinando f'_{2f} e f'_{2t}, é possível obter a derivada simétrica de 3 pontos, $\epsilon \propto h^2$:

$$f'_{3s} = \frac{1}{2}(f'_{2f} + f'_{2t}) + \epsilon \tag{3}$$

Extendendo o conceito para uma quantidade maior de pontos, tem-se que

$$f'_{5s} = \frac{1}{12h}[8(f'_{2f} + f'_{2t}) - f(x+2h) + f(x-2h)] + \epsilon \tag{4}$$

que possui $\epsilon \propto h^4$. É possível, também, obter expressões aproximadas para a derivada de segunda ordem de $f(x)$ utilizando expansão de Taylor, mantendo as mesmas ordens de erro intrínseco de suas respectivas derivadas de primeira ordem. As expressões para derivadas de segunda ordem que serão utilizadas nessa tarefa são:

$$f''_{3s} = \frac{f(x+h) + 2f(x) + f(x-h)}{h^2} + \epsilon \tag{5}$$

$$f''_{5s} = \frac{-f(x-2h) + 16f(x-h) - 30f(x) + 16f(x+h) - f(x+2h)}{12h^2} \tag{6}$$

Resultados:

h	f'_2f	f'_2t	f'_3s	f'_5s	f''_3s	f''_5s
2.0000E-01	5.516621203949	3.063457090490	4.290039147220	3.810496475878	12.265820567294	11.199540147581
4.0000E-02	4.139205906346	3.683188545531	3.911197225938	3.896039003775	11.400434020386	11.365635319811
8.0000E-03	3.942223846331	3.851285852349	3.896754849340	3.896154046352	11.367249247673	11.365868502851
1.6000E-03	3.905270991771	3.887085513233	3.896178252502	3.896154229172	11.365924086192	11.365868874216
3.2000E-04	3.897973729756	3.894336651011	3.896155190384	3.896154229465	11.365871080676	11.365868869084
6.4000E-05	3.896517975707	3.895790560097	3.896154267902	3.896154229465	11.365868912271	11.365868772229
1.2800E-05	3.896226972566	3.896081489426	3.896154230996	3.896154229463	11.365870267524	11.365869364022
2.5600E-06	3.896168777797	3.896139681280	3.896154229539	3.896154229474	11.365826899437	11.365790194676
5.1200E-07	3.896157138670	3.896151320407	3.896154229539	3.896154229575	11.363794020364	11.363017573495
1.0240E-07	3.896154813707	3.896153647105	3.896154230406	3.896154230587	11.392593140570	11.401416400438
2.0480E-08	3.896154351837	3.896154113312	3.896154232574	3.896154235285	11.646703024747	11.426121528066
4.0960E-09	3.896154178364	3.896154232574	3.896154205469	3.896154209987	-13.234889800848	-23.161057151485
Exato	3.896154229465	11.365868874671	-	-	-	-

Figura 1: Tabela dos resultados obtidos para as diferentes derivações de $f(x)$ para cada h.

Código:

```
      implicit real*8 (a-h, o-z)

      external f

      x=0.5d0

      open(unit=3, file='derivadas.dat')

      t=tanh(2.0d0*x)
      c=1.0d0/cosh(2.0d0*x)

c derivadas calculadas
      df1=2.0d0*exp(2.0d0*x**2)*(2.0d0*x*t+c**2)

      df2=-4.0d0*exp(2.0d0*x**2)*
     &     (((2.0d0*c**2-4.0d0*x**2-1.0d0)*t)-4.0d0*x*c**2)

      write(3,10)
 10   format(1x,13('-'))
      write(3,11)
 11   format(5x,'h',10x,"f'_2f",15x,"f'_2t",15x,
     &        "f'_3s",15x,"f'_5s",15x,
     &        "f''_3s",15x,"f''_5s")
      write(3,10)

c derivadas numericas
      do i=1,12
         h=5.0d0**(-i)
```

```
c funcao f(X) calculada para diferentes valores
c que serao utilizados nos calc de derivadas
        ff=f(x+h)
        ft=f(x-h)
        f2f=f(x+2.0d0*h)
        f2t=f(x-2.0d0*h)

c derivada 2 pts para frente
        dff=(ff-f(x))/h
c derivada 2 pts para tras
        dft=(f(x)-ft)/h

c derivada simetrica de 3 pts
        df3s=(ff-ft)/(2.0d0*h)
c derivada segunda simetrica de 3pts
        df3s2=(ff-2.0d0*f(x)+ft)/(h*h)

c derivada simetrica de 5 pts
        df5s=(f2t-8.0d0*ft+8.0d0*ff-f2f)/(12.0d0*h)
c derivada segunda simetrica de 5 pts
        df5s2=(-f2t+16.0d0*ft-30.0d0*f(x)+16.0d0*ff-f2f)
    \&           /(12.0d0*h*h)

        write(3,12) h, dff, dft, df3s, df5s, df3s2, df5s2
 12     format(1x,es12.4,6(2x,f18.12))
     end do

     write(3,10)
     write(3,13) df1, df2
 13  format(1x,'Exato',7x,2(2x,f18.12),4(2x,'-'))
     write(3,10)

     close(3)
     end

     double precision function f(x)
     double precision x
     f=exp(2.0d0*x**2)*tanh(2.0d0*x)
     return
     end
```

O código consiste fundamentalmente em uma função que calcula $f(x)$ e um loop que percorre valores de $i = 1, ..., 12$. Os valores de i são utilizados na obtenção de h, tal que $h = 5^{-i}$, e as diferentes derivadas são calculadas por meio das relações 1, 2, 3, 5, 4 e 6. Os resultados foram formatados e armazenados no arquivo *derivadas.dat*.

As derivadas exatas foram calculadas manualmente:

$$f(x) = e^{2x^2}\tanh(2x) \rightarrow f'(x) = 4x \cdot e^{2x^2}\tanh(2x) + 2x \cdot \left(\frac{1}{\cosh(2x)}\right) \rightarrow$$

$$f''(x) = e^{2x^2}\left[4x\left(4x\tanh(2x) + \frac{2}{\cosh^2(2x)}\right) + 4\tanh(2x) + \frac{8x}{\cosh^2(2x)} - \frac{8\tanh(2x)}{\cosh^2(2x)}\right] =$$

$$e^{2x^2}\left(4x(4x\tanh(2x) + \frac{2}{\cosh^2(2x)}\right) + 4\tanh(2x) + \frac{8x}{\cosh^2(2x)} - 8\frac{\tanh(2x)}{\cosh^2(2x)}\Big)$$

Os resultados obtidos pelo programa são estão na figura 1e mostram que os valores de h mais apropriados para cada caso foram:

- f'_{2f} e f'_{2t}: $h \approx 5^{-12}$;
- f'_{3s}: $h \approx 5^{-8}$;
- f'_{5s}: $h \approx 5^{-5}$;
- f''_{3s}: $h \approx 5^{-6}$;
- f''_{5s}: $h \approx 5^{-4}$.

Ou seja, nos casos em que o erro intrínseco da aproximação é grande (da ordem de h), a precisão fica muito dependente do erro da máquina, que é em torno de 10^{-12}. Nos casos em que o erro intrínseco é menor, a precisão atinge seu pico antes de se aproximar da precisão da máquina, e depois o valor fica gradualmente ficando menos preciso.

Tarefa 2

Quadratura numérica

Escreva um código FORTRAN que calcule uma aproximação da integral

$$I = \int_0^{2\pi} e^{-x} \sin(x) dx$$

usando os métodos e intervalos para preencher a tabela (mostrada na figura 2). Na última linha da tabela escreva o valor numérico exato com precisão 10^{-11} obtido pela expressão analítica que você deve calcular. Aponte o valor ótimo de h em cada um dos casos e discuta seus resultados.

Integração numérica

Do cálculo, sabe-se que uma integral definida pode ser entendida geometricamente como a área embaixo de um gráfico. Dessa forma, tem-se que, para uma função $f(x)$ no intervalo $[a, b]$,

$$\int_a^b f(x) \equiv \sum_i^{n-1} f(x_i) \cdot h$$

sendo $h \equiv dx$ pequenos "pedaços"da área total desse gráfico. A integração numérica resulta em uma aproximação do valor da integral de interesse, visto que os "pedaços"considerados nos cálculos não são formalmente infinitesimais. O objetivo dessa tarefa é observar quão precisa é essa aproximação; para isso, serão implementados 3 diferentes métodos:

1. **Regra do trapézio:** Como o nome já sugere, a regra do trapézio consiste na aproximação do valor da integral em pequenos trapézios, que são nada mais do que a junção de retângulos e triângulos. Dessa maneira, é fácil notar que

$$\int_a^b f(x)dx \approx A_{tri} + A_{ret} = \frac{1}{2}(b-a)[f(b) - f(a)] + (b-a)[f(a)] = \frac{1}{2}(b-a)[f(b) + f(a)]$$

Todavia, para uma aproximação mais precisa numericamente, é necessário, também, que seja feita uma análise de pontos intermediários. Assim, sendo n a quantidadde de pontos a serem analisados e $h = \frac{(b-a)}{n}$:

$$\int_a^b f(x)dx \approx h \cdot [\frac{1}{2}(f(b) + f(a)) + \sum_{i=1}^{n-1} f(x_i)] \tag{7}$$

O erro global ϵ desse método é $\epsilon \propto h^2$, enquanto que o erro local δ é $\delta \propto h^3$.

2. **Regra de Simpson:** Utilizando o mesmo raciocínio, é possível chegar ao método de Simpson, que consiste na expansão da regra anterior até ordem quadrática e, por conta disso, surgem "termos cruzados"na sua expressão final, fazendo com que seja necessária a separação entre casos ímpares e pares no somatório de $\sum_i^{n-1} f(x_i)$. Assim, tem-se que

$$\sum_{i}^{n-1} f(x_i) = 2 \cdot \sum_{i}^{\text{n par}} f(x_i) + 4 \cdot \sum_{i}^{\text{n ímpar}} f(x_i)$$

Além disso, a regra utiliza 3 pontos no intervalo de integração em cada subintervalo, de modo que a expressão final fica:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(a) + 2 \cdot \sum_{i}^{\text{n par}} f(x_i) + 4 \cdot \sum_{i}^{\text{n ímpar}} f(x_i) + f(b)] \tag{8}$$

Que possui erro global $\epsilon \propto h^4$. Por isso, é possível considerar que a aproximação torna-se exata para polinômios de até terceiro grau.

3. **Regra de Boole:** Considerando pontos no espaço previamente determinados:

$$x_0 = a$$
$$x_1 = x_0 + h$$
$$x_2 = x_0 + 2h$$
$$x_3 = x_0 + 3h$$
$$x_4 = x_0 + 4h = b$$

tem-se que a aproximação pela Regra de Boole é da forma:

$$\int_a^b f(x)dx \approx \frac{2h}{45}[7(f(x_0) + f(x_4)) + 12f(x_2) + 32(f(x_1) + f(x_3))] \tag{9}$$

Note que a expressão separa valores de x_n ímpares e pares como a (8), mas o ponto intermediário do intervalo x_2 é deixado de fora. No total, são utilizados 5 pontos por intervalo, e o erro global é $\epsilon \propto h^7$.

Resultado:

i	n	trapezio	simpson	boole
2	4	0.312425554409	0.416567405879	0.444338566270
3	8	0.448843070178	0.494315575435	0.499498786738
4	16	0.486305646676	0.498793172175	0.499091678624
5	32	0.495863645010	0.499049644455	0.499066742607
6	64	0.498264845766	0.499065246018	0.499066286123
7	128	0.498865872096	0.499066214206	0.499066278752
8	256	0.499016173981	0.499066274609	0.499066278636
9	512	0.499053752282	0.499066278383	0.499066278634
10	1024	0.499063147034	0.499066278618	0.499066278634
11	2048	0.499065495733	0.499066278633	0.499066278634
12	4096	0.499066082909	0.499066278634	0.499066278634
13	8192	0.499066229703	0.499066278634	0.499066278634

valor exato da integral= 0.499066278634

Figura 2: Tabela de resultados obtidos para os 3 métodos a cada valor de h.

Código:

```
              implicit real*8 (a-h, o-z)
              parameter (pi=acos(-1d0))
              external f

              a=0.0d0
              b=2.0d0*pi

      c     valor exato da integral
              rint=(1.0d0-dexp(-b))/2.0d0

              open(unit=1, file='integrais.dat')

              write(1,10)
         10   format(1x,' i',3x,'   n',6x,'   trapezio',13x,'simpson',
           &      15x,'boole')
              write(1,11)
         11   format(1x,70('-'))

              do i=2,13
                      n=2**i
                      h=(b-a)/dfloat(n)

                      trap=(int(1/2d0))*(f(a)+f(b))
                      simp=f(a)+f(b)
                      boole=0.0d0

      c regra do trapezio

      c o loop vai ate b-h
               do j=1,n-1
                      x=a+j*h

                      trap=trap+f(x)
               end do
      c multiplico por h fora do loop
               trap=trap*h

      c regra de simpson
               do j=1,n-1
                      x=a+j*h
      c crio um caso para funcoes com j's pares
      c e outro para j's impares, por conta da
      c definicao da regra de simpson
                      if(mod(j,2).eq.0) then
                              simp=simp+2.0d0*f(x)
                      else
                              simp=simp+4.0d0*f(x)
                      end if
               end do

               simp=(h*simp)/3.0d0
```

```
c regra de boole, que considera intervalos de 4 pts
        do j=0,n-4,4
               x=a+j*h

               f1=f(x+h)
               f2=f(x+2*h)
               f3=f(x+3*h)
               f4=f(x+4*h)

               boole=boole+7.0d0*(f4+f(x))
     &         +32.0d0*(f3+f1)+12.0d0*f2
        end do
        boole=(2.0d0*h/45.0d0)*boole

        write(1,12) i, n, trap, simp, boole
 12     format(1x,i2,3x,i6,3(2x,f18.12))

      end do

      write(1,*)
      write(1,13) rint
 13   format('valor exato da integral=',f18.12)

      close(1)
      end

      double precision function f(x)
      double precision x
      f=dexp(-x)*dsin(x)
      return
      end
```

O código consiste em um loop externo que percorre $i = 1, ..., 13$, que vai definir a ordem de h, tal que $h = \frac{b-a}{2^i}$. No interior desse loop são implementados outros 3 loops, cada um deles correspondendo a um dos 3 métodos aqui citados. O objetivos dos loops internos é calcular as somas mostradas nas equações 7, 8 e 9, de modo que a multiplicação do valor total das somas por h é colocada fora dos mesmos. Foi criada uma função que calcula $f(x) = e^{-x}\sin(x)$ para que o código ficasse mais fluido. Além disso, os resultados foram formatados por meio do comando *format* fortran77 e guardados no arquivo *integrais.dat*. A função foi integrada manualmente para obter-se o valor exato que está no código:

$$\int_0^{2\pi} e^{-x}\sin x dx = \frac{1-e^{-2\pi}}{2}$$

Os resultados obtidos e representados na figura 2 mostram que, para cada um dos métodos, o melhor valor de h é sempre próximo da ordem do erro da máquina. Todavia, o método que atinge o valor exato mais rapidamente é o de Boole, com apenas 9 iterações.

Tarefa 3

Raízes de funções

Faça um programa que calcule as raízes reais de

$$f(x) = 0.042 - 0.13x - 0.6x^2 + x^3$$

no intervalo $x \in [5, 5]$, preenchendo a tabela. Eleja uma tolerância de 10^6. Inicie fazendo uma busca direta usando como ponto inicial $x = 5$ e um espaçamento inicial de 0.01. Quando verificar a mudança de sinal em $f(x)$, use o intervalo correspondente $]x_{,x+}[$ para iniciar o método da bisseção e conte as iterações a partir daí. Para o método da secante, use os extremos desse intervalo como pontos iniciais x_1 e x_0. Para o método de Newton-Raphson, use um dos extremos como ponto inicial x_0. Finalmente, na última linha da tabela coloque os valores exatos. (Note que o número total de tabelas é igual ao número de raízes dentro do intervalo $x \in [5, 5]$.

1. **Busca direta:**

   Nessa tarefa, a busca direta é utilizada para encontrar 3 diferentes subintervalos em que existem as maiores posibilidades de se encontrar uma raiz da função. Para isso, é necessário que o sinal da função em cada ponto (em um passo de $h = 0.01$) seja analisado. O número de subintervalos varia de acordo com a quantidade de raizes da função em estudo e, aqui, como função é cúbica, serão 3 raizes e 3 subintervalos.

2. **Método da Bisseção:** Esse método percorre cada subintervalo previamente obitido e toma o ponto médio dos seus extremos. Em seguida, o ponto médio define juntamente com um dos antigos extremos um novo intervalo. O processo é repetido até que atinja-se a precisão desejada que, nessa tarefa, é $\epsilon = 10^{-6}$. Devido às contínuas subdivisões do subintervalo, o erro é reduzido pela metade a cada iteração.

3. **Método de Newton-Raphson:**

   Em posse dos subintervalos obtidos por busca direta, o Método de Newton-Raphson consiste em considerar algum dos extremos de cada intervalo como sendo um ponto inicial. A partir desse ponto, é possível calcular todos os seguintes por meio da relação

   $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{10}$$

   A implementação resulta em raizes cada vez mais precisas a cada iteração. Nessa tarefa, a precisão de interesse é, novamente, $\epsilon = 10^{-6}$, e a derivada da função polinomial dada foi calculada manualmente.

4. **Método da Secante:** O último método é semelhante ao anterior, mas mais abrangente, pois é possível obter resultados precisos mesmo para funções cuja derivada analítica é desconhecida ou não existente. É definido pela relação

   $$x_{n+1} = x_n - \frac{f(x_n) \cdot (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{11}$$

   Mais uma vez, o processo começa com um subintervalo previamente obtidos e o divide em intervalos cada vez menores, até que a precisão da raiz encontrada seja $\epsilon = 10^{-6}$.

   Resultados:

iteracao	bissecao	newton-raphson	secante
1	-0.29500000000	-0.30031468393	-0.29999999146
2	-0.29750000000	-0.30000028837	-0.29999999172
3	-0.29875000000	-0.29999999172	0.00000000000
4	-0.29937500000	0.00000000000	0.00000000000
5	-0.29968750000	0.00000000000	0.00000000000
6	-0.29984375000	0.00000000000	0.00000000000
7	-0.29992187500	0.00000000000	0.00000000000
8	-0.29996093750	0.00000000000	0.00000000000
9	-0.29998046875	0.00000000000	0.00000000000
10	-0.29999023438	0.00000000000	0.00000000000
11	-0.29999511719	0.00000000000	0.00000000000
12	-0.29999755859	0.00000000000	0.00000000000
13	-0.29999877930	0.00000000000	0.00000000000
14	-0.29999938965	0.00000000000	0.00000000000
exato		-0.30000000000	

Figura 3: Resultados obtidos para as raizes da função pelos 3 métodos - parte 1.

1	0.19500000000	0.19999999774	0.19999999773
2	0.19750000000	0.00000000000	0.00000000000
3	0.19875000000	0.00000000000	0.00000000000
4	0.19937500000	0.00000000000	0.00000000000
5	0.19968750000	0.00000000000	0.00000000000
6	0.19984375000	0.00000000000	0.00000000000
7	0.19992187500	0.00000000000	0.00000000000
8	0.19996093750	0.00000000000	0.00000000000
9	0.19998046875	0.00000000000	0.00000000000
10	0.19999023437	0.00000000000	0.00000000000
11	0.19999511719	0.00000000000	0.00000000000
12	0.19999755859	0.00000000000	0.00000000000
13	0.19999877930	0.00000000000	0.00000000000
14	0.19999938965	0.00000000000	0.00000000000
exato		0.20000000000	

Figura 4: Resultados obtidos para as raizes da função pelos 3 métodos - parte 2.

1	0.70500000000	0.70028664702	0.70000001730
2	0.70250000000	0.70000026396	0.70000001781
3	0.70125000000	0.70000001782	0.00000000000
4	0.70062500000	0.00000000000	0.00000000000
5	0.70031250000	0.00000000000	0.00000000000
6	0.70015625000	0.00000000000	0.00000000000
7	0.70007812500	0.00000000000	0.00000000000
8	0.70003906250	0.00000000000	0.00000000000
9	0.70001953125	0.00000000000	0.00000000000
10	0.70000976562	0.00000000000	0.00000000000
11	0.70000488281	0.00000000000	0.00000000000
12	0.70000244141	0.00000000000	0.00000000000
13	0.70000122070	0.00000000000	0.00000000000
14	0.70000061035	0.00000000000	0.00000000000
exato		0.70000000000	

Figura 5: Resultados obtidos para as raizes da função pelos 3 métodos - parte 3.

Código:

```
implicit real*8 (a-h, o-z)
parameter (h=0.01d0, b=5, a=-5, eps=1e-6)
dimension xi(10), xf(10), biss(20), raph(20), sec(20)

external d, f

open(unit=1, file='raizes.dat')
```

```
        write(1,1)
1       format(9x,' ',10x, ' ',23x,"bissecao",23x
     & , ' ', 6x,"newton-raphson",8x, ' ', 23x,"secante",22x)

        write(1,2)
2       format(5x,'iteracao ',6x,' ',5x,F15.11, 3x,
     & F15.11,4x,' ',5x, F15.11, 4x,' ',5x, F15.11, 3x,
     & "xi :", F15.11,4x,' ')

3       format(' ',6x, I5 ,9x,' ',19x,F15.11,20x,' ',7x,F15.11,6x,' ',
     & 19x,F15.11, 18x,' ')

4       format(' ',6x, "exato" ,9x,' ',57X, F15.11, 64X,' ')

c busca direta
        xe=a
        xd=xe+h
        nraiz=0

c se houver mudanca de sinal entre dois valores de x
c encontrei um subintervalo
   11   if(f(xd)*f(xe) .lt. 0.0) then
                nraiz=nraiz+1
c o subintervalo sera [xe,xd]
                xi(nraiz)=xe
                xf(nraiz)=xd
        end if

c agora eu checo um intervalo a partir do xd
        xe=xd
        xd=xd+h

c a busca varre o intervalo [a,b]
        if(xd .le. b) then
        go to 11
        else
        go to 10
        end if

   10   do i=1,nraiz
                x1=xi(i)
                x2=xf(i)

c crio vetores de 0's
                do j=1,20
                        biss(j)=0
                        raph(j)=0
                        sec(j)=0
                end do

c metodo da bissecao
                ibiss=0
                do while(abs(x2-x1) .gt. eps)
                        xm=(x1+x2)/2d0

                        if(f(x2)*f(xm) .gt. 0.0) then
                                x2=xm
                        else
                                x1=xm
                        end if
```

```
                        ibiss=ibiss+1
                        biss(ibiss)=xm
                end do

c metodo de newton-raphson
                iraph=0
c preciso atualizar x1 e x2 a cada metodo
                x1=xi(i)
                x2=xf(i)
                dif=abs(x2-x1)
                do while(dif .gt. eps)
                        if(d(xn) .eq. 0.0) then
c evitando divisao por 0
                                xn=x2-f(x2)/d(x2+eps)
                        else
                                xn=x2-f(x2)/d(x2)
                        end if

                        iraph=iraph+1
c a diferenca eh atualizada para o novo interv
c e sera cada vez menor pois xn --> x2
                        dif=abs(xn-x2)
                        raph(iraph)=xn
                        x2=xn
                end do

c metodo da secante
                isec=0
                x1=xi(i)
                x2=xf(i)
                dif=abs(x2-x1)
                do while(dif .gt. eps)
                        if((f(x2)-f(x1)) .eq. 0.0) then
c evitando divisao por 0
                                xl=x2-((x2-x1)*f(x2))/(f(x2+eps)-f(x1))
                        else
                                xl=x2-((x2-x1)*f(x2))/(f(x2)-f(x1))
                        end if

                        isec=isec+1
                        dif=abs(xl-x2)
                        sec(isec)=xl
c o novo intevalo agora sera [x2,xl]
                        x1=x2
                        x2=xl
                end do

c impressao da tabela
                imax=max(max(ibiss, iraph), isec)
                do k=1,imax
                        write(1,3) k, biss(k), raph(k)
     &          , sec(k)
                end do
```

```
c linha de valores exatos
              if(i .eq. 1) then
                      write(1,4) -0.30d0
              else
                      if(i .eq. 2) then
                              write(1,4) 0.20d0
                      else
                              write(1,4) 0.70d0
                      end if
              end if
              write(1,*)
      end do
      close(1)
      end

      double precision function d(x)
      implicit real*8 (a-h, o-z)
      d=-0.13-1.2*x+3*x**2
      return
      end

      double precision function f(x)
      implicit real*8 (a-h, o-z)
      f=0.042-0.13*x-0.6*x**2+x**3
      return
      end
```

O programa começa com uma implementação da busca direta, varrendo o intervalo $[-5, 5]$ e armazenando os extremos dos subintervalos em dois vetores *xi* e *xf*. Os três subintervalos encontrados foram utilizados na implementação dos três métodos, sendo que os dois últimos foram construídos a partir das equações 10 e 11, respectivamente. Para isso, foi feito um loop que percorre cada subintervalo e dentro dele estão os loops de cada método. As raizes encontradas por meio de cada método foram armazenadas no arquivo *raizes.dat*, enquanto que as raizes exatas foram obtidas pela visualização gráfica da função obtida pela calculadora gráfica Desmos e mostrada na figura 6. O conteúdo do arquivo é mostrado nas figuras 3, 4 e 5.

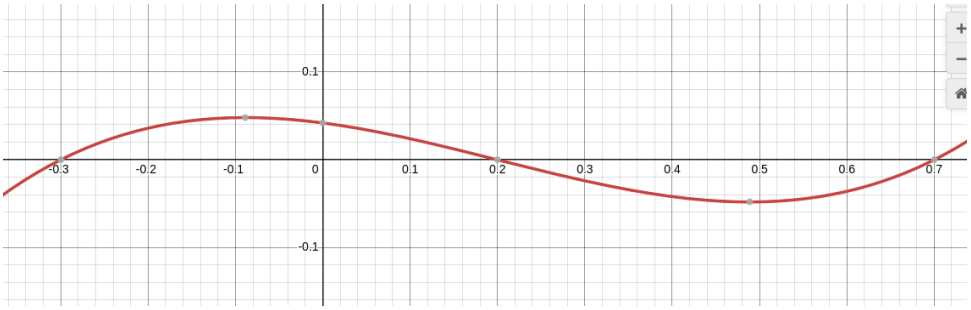

Figura 6: Gráfico da função $f(x)$ com raizes $x_1 = -0.3$, $x_2 = 0.2$ e $x_3 = 0.7$.

Referências

[1] Adérito Luís Martins Araujo. *Análise Numérica: Engenharia Mecânica e de Materiais*. F.C.T.U.C., 2002.

[2] Cuemath. Simpson's rule formula. Cuemath. URL: https://www.cuemath.com/simpsons-rule-formula/.

[3] LibreTexts. Numerical integration – midpoint, trapezoid, simpson's rule. LibreTexts. URL: https://math.libretexts.org/Courses/Mount_Royal_University/Calculus_for_Scientists_II/2%3A_Techniques_of_Integration/2.5%3A_Numerical_Integration_-_Midpoint%2C_Trapezoid%2C_Simpson%27s_rule.

[4] Wikipedia contributors. Método da bissecção. Wikipedia, The Free Encyclopedia. URL: https://pt.wikipedia.org/wiki/M%C3%A9todo_da_bisse%C3%A7%C3%A3o.

[5] Wikipedia contributors. Método das secantes. Wikipedia, The Free Encyclopedia. URL: https://pt.wikipedia.org/wiki/M%C3%A9todo_das_secantes.

[6] Wikipedia contributors. Numerical differentiation. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Numerical_differentiation.