MÉTODOS NUMÉRICOS PARA CALCULAR ZEROS DE FUNÇÕES

Evandro Pedro Alves de Mendonça[a], **Marcelino José de Lima Andrade**[a].

[a] *Núcleo de Tecnologia (NTI), Universidade Federal de Pernambuco (UFPE), Campus Acadêmico do Agreste (CAA), Rodovia BR-104, km 59, S/N, Nova Caruaru, CEP. 55.014-900, Caruaru-PE, Brasil, http://www.ufpe.br/caa*

Palavras Chave: raízes de funções, zeros de funções, métodos numéricos, convergência, gráficos, aproximações.

Resumo. Na ciência, é muito comum encontrar valores para os quais uma determinada função é igual a zero. Estes valores são chamados de zeros da função, ou raízes da função. Muitas vezes, é complicado e dispendioso demais calculá-los analiticamente e, por isso, existem métodos numéricos que fornecem valores aproximados das raízes e que são aceitáveis, pois estão dentro de um limite tolerável de erro. Este trabalho se destina, portanto, a resolver alguns problemas utilizando, para isso, métodos numéricos para determinação de zeros de funções. Ao longo do trabalho será feita uma discussão em torno desses métodos, abordando vantagens e desvantagens de cada um e como é feita a implementação deles utilizando o MATLAB. Também serão feitas comparações entre os métodos implementados neste trabalhos e os que já são nativos do MATLAB.

1 INTRODUÇÃO

Em várias áreas das ciências exatas encontram-se constantemente situações onde é necessário resolver um problema com a seguinte forma:

$$f(x) = 0 \tag{1}$$

Algumas vezes, a resolução de um problema desse tipo pode ser feita de forma analítica, como é o caso das equações polinomiais de 1º e 2º grau. Porém, ao lidar com equações polinomiais de 3º grau em diante, ou quando trabalha-se com funções mais complexas, como seno, cosseno, a função logarítmica, entre outras, torna-se mais complicada a resolução do problema.

Graficamente, a resolução do problema representado na equação (1) acima é ponto onde a função *f(x)* intercepta o eixo das abcissas, ou eixo x. Esses pontos são as chamadas raízes da função. Isso é mostrado na imagem abaixo, onde x' e x'' são as raízes da função.

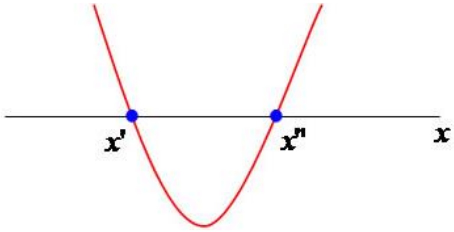

Figura 1: raízes de um polinômio de segundo grau.
Adaptado de http://mundoeducacao.bol.uol.com.br/upload/conteudo/Untitled-7(12).jpg

Como já dito, em alguns casos, o cálculo analítico das raízes de uma função é inviável. Uma saída para essa problema é a utilização de métodos numéricos para a determinação das raízes. Inicialmente, procura-se dar uma estimativa para a raiz da função. Em seguida, é aplicado um método para aperfeiçoar essa estimativa inicial. É importante observar que, nos métodos numéricos, nem sempre é possível fazer uma estimativa exata para a raiz, diferente do cálculo analítico. Muitas vezes apenas será possível realizar uma boa estimativa da raiz.

Os métodos de estimativa de raízes dividem-se em dois grupos, são eles: os métodos de confinamento, e os métodos abertos. Os métodos de confinamento requerem um intervalo inicial em que a raiz da função se encontra em seu interior. Os métodos abertos consistem em, a partir de um palpite de solução inicial, são feitas manipulações numéricas para o cálculo da raiz. No método aberto, essa estimativa inicial deve ser próxima da raiz exata.

Independentemente do método utilizado, é necessário estabelecer uma tolerância no cálculo da raiz, visto que, em alguns casos, o cálculo exato é praticamente impossível, além de que, o custo computacional é extremamente alto. Em termos simples, a tolerância seria o quanto a solução encontrada pode desviar-se da solução exata. Obviamente, quanto menor for a tolerância permitida, menor será o erro associado. Porém, uma boa escolha do método a ser utilizado leva a um menor custo computacional, e a um menor erro associado. Alguns métodos são listados a seguir:

- Bisseção – método de confinamento
- Falsa posição – método de confinamento
- Ponto fixo – método aberto
- Newton-Raphson – método aberto
- Secante – método aberto

A escolha do método a ser utilizado depende do problema a ser resolvido e da precisão e exatidão necessárias. Além disso, existem casos específicos onde alguns desses métodos não funcionam, ou se distanciam muito da solução exata. Uma análise do procedimento usado por cada método leva a uma boa escolha de qual deve ser utilizado. O detalhamento de cada método é feito nos exercícios propostos deste trabalho.

2 EXERCÍCIOS PROPOSTOS

Segue, abaixo, a solução dos exercícios propostos sobre o tema.

2.1 1ª questão

Utilizando o algoritmo que consta no Anexo 1, foram encontrados os seguintes resultados.

- 1º intervalo [0,2]: raiz = 3,675460815429688 x 10^{-1};
- 2º intervalo [2,4]: raiz = 3,767829895019531;
- 3º intervalo [4,6]: raiz = 5,954948425292969;

Detalhes no Anexo 6.

2.2 2ª questão

Utilizando o algoritmo que consta no Anexo 1, foram encontrados os seguintes resultados.

- Letra a.
  - Primeiro intervalo [-3/2, 5/2]: raiz = 0;
  - Segundo intervalo [-1/2, 12/5]: raiz = 2,861022949203437 x 10^{-6};
  - Terceiro intervalo [-1/2, 3]: raiz = 1,999997138977051;
  - Quarto intervalo [-3, -1/2]: raiz = -2,000005722045898;

Detalhes no Anexo 7.

- Letra b.

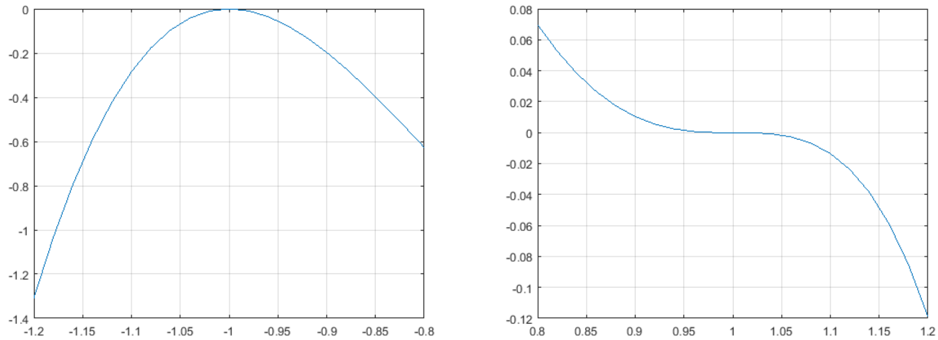

Figura 2: à esquerda, gráfico da função na vizinhança da raiz x = -1; à direita, da raiz x = 1.

Analisando o gráfico da função mostrado na Figura 2, não fica muito claro se é possível ou não aplicar o método da bisseção para as raízes -1 e 1. Portanto, foram gerados outros dois gráficos, dispostos na Figura 2, que mostram o que acontece na imagem da função na vizinhança dessas duas raízes. Dessa forma, é possível ver que na raiz x = -1 qualquer par de pontos tomados em sua vizinhança terá imagens negativas em ambos os pontos, e não será possível aplicar o método da bisseção, uma vez que ele exige um par de pontos com imagens de sinais contrários. Já na raiz x = 1, observa-se que as imagens terão sinais contrários, sendo, assim, possível aplicar o método da bisseção para encontrá-la.

2.3 3ª questão

Primeiramente, vejamos o gráfico da função.

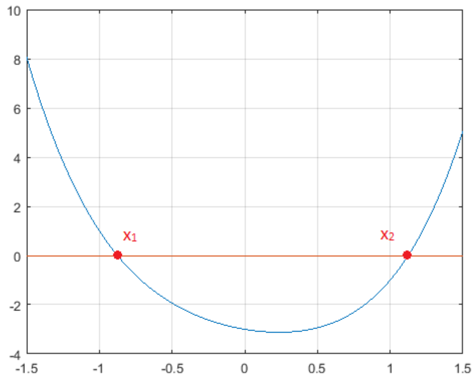

Figura 3: gráfico da função dada na questão entre os pontos x = -1,5 e x = 1,5. A raiz próxima ao ponto x = -1 será chamada de x_1 e a raiz próxima ao ponto x = 1 será chamada de x_2.

Para determinar a convergência da função de iteração em cada caso, fazemos uma plotagem dos gráficos de cada função em contraste com a função identidade (y = x) e observamos a interseção entre elas. O ponto de interseção determina o valor da raiz.

Primeiramente, a função da letra (A):

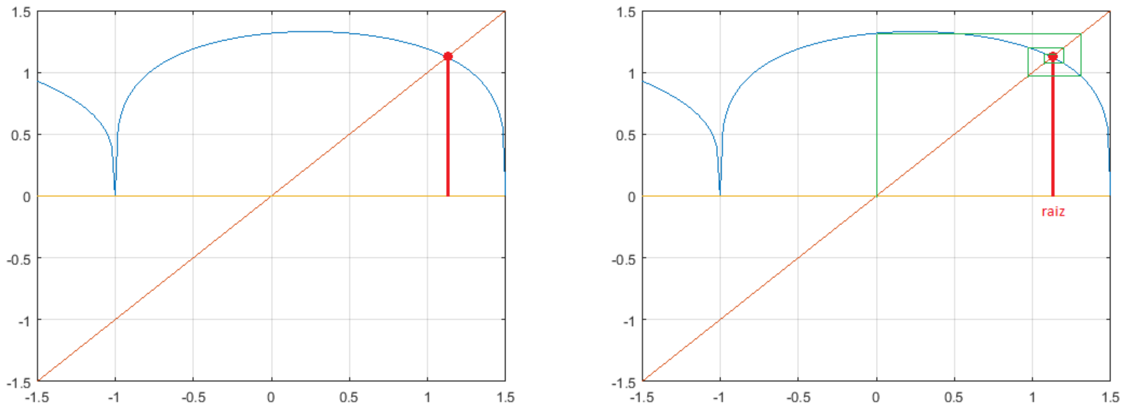

Figura 4: à esquerda, vemos o ponto de interseção entre a função de iteração e a reta y = x; à direita, o procedimento gráfico do método do ponto fixo para encontrar a raiz a partir de retas horizontais e verticais tendo como estimativa inicial o ponto $x_e = 0$.

Com a análise dos gráficos, vemos que a raiz dessa função de iteração converge apenas para o ponto próximo a x = 1. Vejamos agora o que acontece na função de iteração da letra (B):

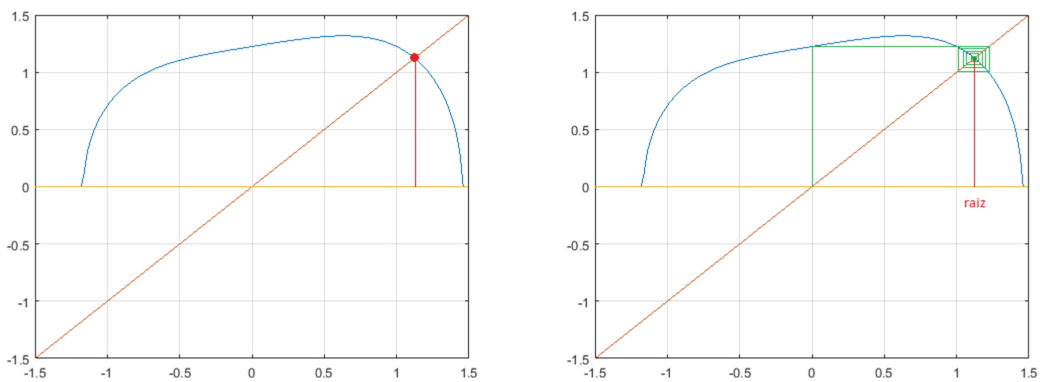

Figura 5: à esquerda, vemos o ponto de interseção entre a função de iteração e a reta y = x; à direita, o procedimento gráfico do método do ponto fixo para encontrar a raiz a partir de retas horizontais e verticais tendo como estimativa inicial o ponto $x_e = 0$.

Percebemos que as duas funções de iteração têm comportamento semelhante. Ambas convergem apenas para a raiz próxima ao ponto x = 1. Assim, respondendo à pergunta do item 3.a), não utilizaria nenhuma delas para encontrar a raiz x_1, pois nenhuma delas converge para esse ponto; já para a raiz x_2, ambas convergem, mas a função de iteração da letra (A) converge muito mais rápido, então essa seria a escolha para encontrar a raiz x_2.

2.4 4ª questão

Para esta questão, foi utilizado o algoritmo que consta no Anexo 4.

- 4.a) $f(x) = 2xcos(2x) - (x-2)^2 = 0 \therefore f'(x) = 2\cos(2x) - 4xsen(2x) - 2(x-2)$
    - Intervalo: $2 \leq x \leq 3$

```
>> newton_raphson(f,df,2.5,1e-6,50)
iteração          Xe         Solução(Xi)         f(Xi)          tolerância
   1          2.500000         2.372407     0.0151395834     0.0510370715
   2          2.372407         2.370688     0.0000079869     0.0007247893
   3          2.370688         2.370687     0.0000000000     0.0000003830

Número total de iterações: 3
ans =
     2.370686917662517e+00
```

Figura 6: resultado para cálculo de raiz de f(x) utilizando método de Newton-Raphson com estimativa de 2,5.

- Intervalo: $3 \leq x \leq 4$

```
>> newton_raphson(f,df,3.5,1e-6,50)
iteração          Xe         Solução(Xi)         f(Xi)          tolerância
   1          3.500000         3.783191    -1.0335591821     0.0809117613
   2          3.783191         3.724165    -0.0335094378     0.0156021096
   3          3.724165         3.722115    -0.0000444134     0.0005504332
   4          3.722115         3.722113    -0.0000000001     0.0000007319

Número total de iterações: 4
ans =
     3.722112773106613e+00
```

Figura 7: resultado para cálculo de raiz de f(x) utilizando método de Newton-Raphson com estimativa de 3,5.

- 4.b) $g(x) = sen(x) - e^{-x} = 0 \therefore g'(x) = \cos(x) + e^{-x}$
    - Intervalo: $0 \leq x \leq 1$

```
>> newton_raphson(g,dg,0.5,1e-6,50)
iteração          Xe         Solução(Xi)         f(Xi)          tolerância
   1          0.500000         0.585644    -0.0040112772     0.1712876339
   2          0.585644         0.588529    -0.0000046203     0.0049272195
   3          0.588529         0.588533    -0.0000000000     0.0000056605
   4          0.588533         0.588533     0.0000000000     0.0000000000

Número total de iterações: 4
ans =
     5.885327439818611e-01
```

Figura 8: resultado para cálculo de raiz de g(x) utilizando método de Newton-Raphson com estimativa de 0,5.

- Intervalo: $3 \leq x \leq 4$

```
>> newton_raphson(g,dg,3.5,1e-6,50)
iteração         Xe        Solução(Xi)        f(Xi)         tolerância
   1         3.500000        3.079612     0.0159639634    0.1201108807
   2         3.079612        3.096379    -0.0000143495    0.0054445367
   3         3.096379        3.096364    -0.0000000000    0.0000048589
   4         3.096364        3.096364    -0.0000000000    0.0000000000

Número total de iterações: 4
ans =
     3.096363932410646e+00
```

Figura 9: resultado para cálculo de raiz de g(x) utilizando método de Newton-Raphson com estimativa de 3,5.

- 4.c) $h(x) = \ln(x-1) + \cos(x-1) = 0 \therefore h'(x) = \frac{1}{x-1} - sen(x-1)$
  - Intervalo: $1,3 \leq x \leq 2$

```
>> newton_raphson(h,dh,1.5,1e-6,50)
iteração         Xe        Solução(Xi)        f(Xi)         tolerância
   1         1.500000        1.378707    -0.0418495132    0.0808621505
   2         1.378707        1.397136    -0.0013043765    0.0133669025
   3         1.397136        1.397748    -0.0000013590    0.0004380560
   4         1.397748        1.397748    -0.0000000000    0.0000004571

Número total de iterações: 4
ans =
     1.397748475958052e+00
```

Figura 10: resultado para cálculo de raiz de h(x) utilizando método de Newton-Raphson com estimativa de 1,5.

A ordem de convergência do método de Newton-Raphson é quadrática, como pode ser observado nos gráficos que constam no Anexo 8, juntamente com outros detalhes mais.

2.5 5ª questão

Para esta questão, foi utilizado o algoritmo que consta no Anexo 3. Mais detalhes, vide Anexo 9.

- 5.a) $f(x) = 2x\cos(2x) - (x-2)^2 = 0$ Intervalos: $2 \leq x \leq 3$ e $3 \leq x \leq 4$

```
>> secante(f,2,3,1e-6,50)
iteração         x1           x2         Solução(x3)       f(x3)         tolerância
   1         2.000000     3.000000       2.354490     -0.1417166747    0.2151700278
   2         3.000000     2.354490       2.373149     0.0216693873     0.0079248022
   3         2.354490     2.373149       2.370674     -0.0001125971    0.0010427782
   4         2.373149     2.370674       2.370687     -0.0000000853    0.0000053960
   5         2.370674     2.370687       2.370687     0.0000000000     0.0000000041

Número total de iterações: 5
ans =
     2.370686917662300e+00
```

Figura 11: resultado para cálculo de raiz de f(x) utilizando método da Secante com pontos 2 e 3.

```
>> secante(f,3,4,1e-6,50)
iteração         x1          x2          Solução(x3)      f(x3)          tolerância
   1          3.000000    4.000000        3.479699    3.2384648091    0.1300752854
   2          4.000000    3.479699        3.680233    0.6636609839    0.0576295972
   3          3.479699    3.680233        3.731920    -0.1609112776   0.0140447538
   4          3.680233    3.731920        3.721834    0.0045472782    0.0027028053
   5          3.731920    3.721834        3.722111    0.0000286293    0.0000744822
   6          3.721834    3.722111        3.722113    -0.0000000052   0.0000004719

Número total de iterações: 6
ans =
     3.722112773420417e+00
```

Figura 12: resultado para cálculo de raiz de f(x) utilizando método da Secante com pontos 2 e 3.

- 5.b) $g(x) = sen(x) - e^{-x} = 0$ Intervalos: $0 \leq x \leq 1$ e $3 \leq x \leq 4$

```
>> secante(g,0,1,1e-6,50)
iteração         x1          x2          Solução(x3)      f(x3)          tolerância
   1          0.000000    1.000000        0.678614    0.1203951836    0.3213858994
   2          1.000000    0.678614        0.569062    -0.0272136880   0.1614346785
   3          0.678614    0.569062        0.589260    0.0010078002    0.0354923598
   4          0.569062    0.589260        0.588538    0.0000077861    0.0012240031
   5          0.589260    0.588538        0.588533    -0.0000000023   0.0000095417
   6          0.588538    0.588533        0.588533    0.0000000000    0.0000000028

Número total de iterações: 6
ans =
     5.885327439818647e-01
```

Figura 13: resultado para cálculo de raiz de g(x) utilizando método da Secante com pontos 0 e 1.

```
>> secante(g,3,4,1e-6,50)
iteração         x1          x2          Solução(x3)      f(x3)          tolerância
   1          3.000000    4.000000        3.105410    -0.0086317509   0.2236474042
   2          4.000000    3.105410        3.095336    0.0009803535    0.0032441379
   3          3.105410    3.095336        3.096364    0.0000004073    0.0003319516
   4          3.095336    3.096364        3.096364    -0.0000000000   0.0000001379

Número total de iterações: 4
ans =
     3.096363932431540e+00
```

Figura 14: resultado para cálculo de raiz de g(x) utilizando método da Secante com pontos 3 e 4.

- 5.c) $h(x) = \ln(x-1) + \cos(x-1) = 0$ Intervalo: $1{,}3 \leq x \leq 2$

```
>> secante(h,1.3,2,1e-6,50)
iteração         x1          x2          Solução(x3)      f(x3)          tolerância
   1          1.300000    2.000000        1.520607    0.2147576421    0.2396964758
   2          2.000000    1.520607        1.204358    -0.6086920298   0.2079757463
   3          1.520607    1.204358        1.438128    0.0803041041    0.1941041249
   4          1.204358    1.438128        1.410882    0.0273195257    0.0189458250
   5          1.438128    1.410882        1.396833    -0.0019495179   0.0099573599
   6          1.410882    1.396833        1.397769    0.0000436500    0.0006698985
   7          1.396833    1.397769        1.397749    0.0000000680    0.0000146608
   8          1.397769    1.397749        1.397748    -0.0000000000   0.0000000229

Número total de iterações: 8
ans =
     1.397748475957629e+00
```

Figura 15: resultado para cálculo de raiz de h(x) utilizando método da Secante com pontos 1,3 e 2.

2.6 6ª questão

Para esta questão, foram utilizados os algoritmos que constam nos Anexos 3 e 4, por terem o melhor desempenho na solução de funções não lineares. Mais detalhes, como gráfico e execução dos comandos no MATLAB, vide Anexo 10.

Função: $f(x) = \ln(x^2 + 1) - e^{0,4x}\cos(\pi x)$

Derivada: $f'(x) = \frac{2x}{x^2+1} - 0,4e^{0,4x}\cos(\pi x) + \pi e^{0,4x} sen(\pi x)$

- 2.a)
    - 1ª raiz com o método de Newton-Raphson:
        - Raiz: $4,506567478899358 \times 10^{-1}$;
        - Tempo de processamento: 0,032329s;
        - Iterações: 4.
    - 1ª raiz com o método da Secante:
        - Raiz: $4,506567478888424 \times 10^{-1}$;
        - Tempo de processamento: 0,025429s;
        - Iterações: 5.
    - 2ª raiz com o método de Newton-Raphson:
        - Raiz: 1,744738053368827;
        - Tempo de processamento: 0,008068s;
        - Iterações: 3.
    - 2ª raiz com o método da Secante:
        - Raiz: 1,744738053368886;
        - Tempo de processamento: 0,020020s;
        - Iterações: 7.
    - 3ª raiz com o método de Newton-Raphson:
        - Raiz: 2,238319795074177;
        - Tempo de processamento: 0,004335s;
        - Iterações: 3.
    - 3ª raiz com o método da Secante:
        - Raiz: 2,238319795040133;
        - Tempo de processamento: 0,009471s;
        - Iterações: 7.
- 2.b) Para encontrar raízes de funções não lineares, o MATLAB fornece a função *fzero*, que utiliza o método da Bisseção.
    - 1ª raiz com a função *fzero*:
        - Raiz: $4,506567478899357 \times 10^{-1}$;
        - Tempo de processamento: 0,010251s.
    - 2ª raiz com a função *fzero*:
        - Raiz: 1,744738053368827;
        - Tempo de processamento: 0,037282s.
    - 3ª raiz com a função *fzero*:
        - Raiz: 2,238319795074138;
        - Tempo de processamento: 0,006313s.
- 2.c)

Para fazer a comparação entre a letra A e B desse exercício, vimos que a função fzero só fornece a resposta final, sem mostrar o número de iterações nem informar os valores de cada iteração. Dessa forma, nossas funções fornecem mais informações e, por isso, tem um tempo de execução muito maior. Sendo assim, eliminamos as informações de cada iteração e deixamos apenas a apresentação do número de iterações e, com isso, o tempo de execução se tornou menor (para detalhes, vide Anexo 10). Portanto, temos a seguinte tabela de valores para comparação:

Função	Parâmetros	Raiz 1	Raiz 2	Raiz 3
Newton-Raphson (antes da alteração)	Valor	4,506567478899358 x 10^{-1}	1,744738053368827	2,238319795074177
	Iterações	4	3	3
	Tempo de processamento	0,032329s	0,008068s	0,004335s
Newton-Raphson (depois da alteração)	Valor	4,506567478899358 x 10^{-1}	1,744738053368827	2,238319795074177
	Iterações	4	3	3
	Tempo de processamento	0,004730s	0,009042s	0,002354s
Secante (antes da alteração)	Valor	4,506567478888424 x 10-1	1,744738053368827	2,238319795040133
	Iterações	5	7	7
	Tempo de processamento	0,025429s	0,020020s	0,009471s
Secante (depois da alteração)	Valor	4,506567478888424 x 10-1	1,744738053368827	2,238319795040133
	Iterações	5	7	7
	Tempo de processamento	0,011602s	0,002466s	0,001808s
fzero	Valor	4,506567478899357 x 10-1	1,744738053368827	2,238319795074138
	Iterações	Não informado.	Não informado.	Não informado.
	Tempo de processamento	0,010251s	0,037282s	0,006313s

Tabela 1: comparação dos resultados obtidos para os 3 métodos testados.

Visualizando os dados da Tabela 1, vemos que, na maioria das vezes, as funções *newton_raphson* e *secante* tiveram tempo de processamento menor que o de *fzero*. Este comportamento é ainda mais notável quando as funções *newton_raphson* e *secante* foram modificadas (removendo a impressão de valores de cada iteração) Isto se deve ao fato de que os métodos da Secante e NR (Newton-Raphson) têm taxas de convergência muito maiores que a do método da Bisseção. Comparando agora NR à Secante, vemos que NR geralmente tem tempo de processamento menor que Secante, mas o que chama mais a atenção é que o número de iterações é muito menor. Para esse caso, não faz tanta diferença, mas para situações mais adversas, o número de iterações pode crescer muito e o método NR tende a ser mais vantajoso que o da Secante. A desvantagem, porém, do método NR é que o usuário deve fornecer além da função, sua derivada. O método da Secante não precisa da derivada.

3 CONCLUSÃO

Após implementação dos códigos propostos e análise dos resultados obtidos, conclui-se que há uma variedade considerável de métodos numéricos disponíveis para se chegar ao mesmo resultado: calcular raízes de funções. Porém, para cada caso, haverá sempre um método melhor que outro, e as situações variam muito, pois isso depende do tipo de função que está sendo usada. É importante estar ciente de questões como tempo de processamento, número de iterações, convergência e precisão. Todos esses parâmetros podem ser ajustados conforme especificações do projetos com o qual se está trabalhando.

REFERÊNCIAS

Chapra, S. C., e Canale, R. P. *Métodos Numéricos para Engenharia*. 5ª edição. Porto Alegre: AMGH, 2011.

Gilat, A., e Subramaniam, V. *Métodos Numéricos para Engenheiros e Cientistas: uma introdução com aplicações usando o MATLAB*. Porto Alegre: Bookman, 2008.

ANEXO 1

```
function [ raiz ] = bissecao(f,x1,x2,tol,imax)
%A função bissecao calcula a raiz de uma função qualquer de uma variável
%dentro de um intervalo fornecido utilizando o método da Bisseção.
%Parâmetros:
%f: nome da função ou vetor de caracteres com a expressão da função.
%x1: extremidade inferior do intervalo.
%x2: extremidade superior do intervalo.
%tol: tolerância.
%imax: número máximo de iterações.
%raiz: variável de saída.

    %validação de parâmetros
    if ischar(f)==true %f é um vetor de caracteres
        f = str2func(strcat('@(x)',f));%transforma em function_handle
    elseif isa(f,'function_handle')==false&& ...
            isa(f,'inline')==false%f não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro função (f).');
        return;
    end
    if isnumeric(x1)==false %x1 não é um tipo numérico
        disp('Erro. O parâmetro x1 deve ser numérico.');
        return;
    end
    if isnumeric(x2)==false %x2 não é um tipo numérico
        disp('Erro. O parâmetro x2 deve ser numérico.');
        return;
    end
    if isnumeric(tol)==false %tolerância não é um tipo numérico
        disp('Erro. O parâmetro tol deve ser numérico.');
        return;
    end
    if isnumeric(imax)==false %imax não é um tipo numérico
        disp('Erro. O parâmetro imax deve ser numérico.');
        return;
    end
    if x1>x2 % x1 deve ser menor que x2
        disp('Erro. x1 deve ser menor que x2.');
        return;
    elseif x1==x2
        disp('Erro. x1 e x2 devem ser diferentes.');
        return;
    end
    if f(x1)*f(x2)>0 %deve ser menor que zero
        disp('Erro. Não há raiz entre x1 e x2.');
        return;
    end
```

Figura 16: função criada em MATLAB que calcula uma raiz pelo método da Bisseção (parte 1).

```matlab

    %após verificadas todas as situações problema, prosseguimos com o
    %método da bisseção

    %processamento
    format longe;%formato longo com notação exponencial
    fprintf('iteração\t\tx1\t\t\tx2\t\t  Solução(Xi)\t\tf(Xi)\t\ttolerância\n');
    for i=1:imax
       Xi = (x1 + x2)/2;%solução da iteração atual
       Ti = (x2 - x1)/2;%tolerância da iteração atual
       Fi = f(Xi);%imagem da função em Xi na iteração atual
       %exibição de resultados da iteração
       fprintf('%4d\t\t%.6f\t%.6f\t\t%.6f\t%.6f\t\t%.10f\n',i,x1,x2,Xi,Fi,Ti);
       if Fi==0 %raiz exata
          fprintf('Solução exata encontrada: %.6f',Xi);
          break;
       end
       if Ti<tol %critério da tolerância foi atingido
           break;
       end

       if f(x1)*f(Xi)<0 %a raiz está entre x1 e Xi
           x2 = Xi;%atualiza extremidade superior
       else %a raiz está entre Xi e x2
           x1 = Xi;%atualiza extremidade inferior
       end
    end
    if i==imax %núemro máximo de iterações atingido
        disp('Função finalizada, pois o número máximo de iterações foi atingido.');
    end
    raiz = Xi;%atualiza variável de saída
    fprintf('\nNúmero total de iterações: %d\n',i);
end
```

Figura 17: função criada em MATLAB que calcula uma raiz pelo método da Bisseção (parte 2).

ANEXO 2

```
function [ raiz ] = falsa_posicao(f,x1,x2,tol,imax)
%A função falsa_posicao calcula a raíz de uma função qualquer de uma variável
%dentro de um intervalo fornecido utilizando o método da Falsa Posição.
%Parâmetros:
%f: nome da função ou vetor de caracteres com a expressão da função.
%x1: extremidade inferior do intervalo.
%x2: extremidade superior do intervalo.
%tol: tolerância.
%imax: número máximo de iterações.
%raiz: variável de saída.

    %validação de parâmetros
    if ischar(f)==true %f é um vetor de caracteres
        f = str2func(strcat('@(x)',f));%transforma em function_handle
    elseif isa(f,'function_handle')==false&& ...
            isa(f,'inline')==false%f não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro função (f).');
        return;
    end
    if isnumeric(x1)==false %x1 não é um tipo numérico
        disp('Erro. O parâmetro x1 deve ser numérico.');
        return;
    end
    if isnumeric(x2)==false %x2 não é um tipo numérico
        disp('Erro. O parâmetro x2 deve ser numérico.');
        return;
    end
    if isnumeric(tol)==false %tolerância não é um tipo numérico
        disp('Erro. O parâmetro tol deve ser numérico.');
        return;
    end
    if isnumeric(imax)==false %imax não é um tipo numérico
        disp('Erro. O parâmetro imax deve ser numérico.');
        return;
    end
    if x1>x2 % x1 deve ser menor que x2
        disp('Erro. x1 deve ser menor que x2.');
        return;
    elseif x1==x2
        disp('Erro. x1 e x2 devem ser diferentes.');
        return;
    end
    if f(x1)*f(x2)>0 %deve ser menor que zero
        disp('Erro. Não há raiz entre x1 e x2.');
        return;
    end
```

Figura 18: função criada em MATLAB que calcula uma raiz pelo método da Falsa Posição (parte 1).

```
    %após verificadas todas as situações problema, prosseguimos com o
    %método da falsa posição

    %processamento
    format longe;%formato longo com notação exponencial
    fprintf('iteração\t\tx1\t\t\tx2\t\t  Solução(Xi)\tf(Xi)\t\t\ttolerância\n');
    for i=1:imax
       Xi = (x1*f(x2) - x2*f(x1))/(f(x2)-f(x1));%solução da iteração atual
       Ti = (x2 - x1)/2;%tolerância da iteração atual
       Fi = f(Xi);%imagem da função em Xi na iteração atual

       %exibição de resultados da iteração
       fprintf('%4d\t\t%.6f\t%.6f\t\t%.6f\t%.6f\t\t%.10f\n',i,x1,x2,Xi,Fi,Ti);
       if Fi==0 %raiz exata
          fprintf('Solução exata encontrada: %.6f',Xi);
          break;
       end
       if Ti<tol %critério da tolerância foi atingido
           break;
       end

       if f(x1)*f(Xi)<0 %a raiz está entre x1 e Xi
           x2 = Xi;%atualiza extremidade superior
       else %a raiz está entre Xi e x2
           x1 = Xi;%atualiza extremidade inferior
       end
    end
    if i==imax %núemro máximo de iterações atingido
        disp('Função finalizada, pois o número máximo de iterações foi atingido.');
    end
    raiz = Xi;%atualiza variável de saída
    fprintf('\nNúmero total de iterações: %d\n',i);
end
```

Figura 19: função criada em MATLAB que calcula uma raiz pelo método da Falsa Posição (parte 2).

ANEXO 3

```
function [ raiz ] = secante(f,x1,x2,tol,imax)
%A função secante calcula a raiz de uma função qualquer de uma
%variável em torno de um valor de estimativa dado utilizando o método de
%Newton-Raphson.
%Parâmetros:
%f: nome da função ou vetor de caracteres com a expressão da função.
%x1 e x2: pontos na vizinhança da solução
%tol: tolerância
%imax: número máximo de iterações.
%raiz: variável de saída.

    %validação de parâmetros
    if ischar(f)==true %função é um vetor de caracteres
        f = str2func(strcat('@(x)',f));%transforma em function_handle
    elseif isa(f,'function_handle')==false&& ...
            isa(f,'inline')==false%f não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro funcao (f).');
        return;
    end
    if isnumeric(x1)==false %x1 não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro x1.');
        return;
    end
    if isnumeric(x2)==false %x2 não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro x2.');
        return;
    end
    if isnumeric(tol)==false %tolerância não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro tol.');
        return;
    end
    if isnumeric(imax)==false %imax não é um tipo numérico
        disp('Erro. O parâmetro imax deve ser numérico.');
        return;
    end

    %processamento
    format longe;%formato longo com notação exponencial
    fprintf('iteração\t\tx1\t\t\tx2\t\t  Solução(x3)\t\tf(x3)\t\ttolerância\n');
    for i=1:imax
        %cálculo da estimativa atual
        x3 = x2 - ((f(x2)*(x1-x2))/(f(x1)-f(x2)));
        Ti = abs((x3 - x2)/x2);
        fprintf('%4d\t\t%.6f\t%.6f\t\t%.6f\t%.10f\t%.10f\n',i,x1,x2,x3,f(x3),Ti);
        %quando o erro relativo da estimativa atual com a anterior é menor
        %que a tolerância, temos o resultado da raiz.
        if Ti<tol
            raiz = x3; %valor é atribuído à variável de saída
            break; %finaliza laço de repetição
        end
        %atualização das estimativas
        x1 = x2;
        x2 = x3;
    end
    if i==imax %núemro máximo de iterações atingido
        disp('Função finalizada, pois o número máximo de iterações foi atingido.');
        raiz = x3;
    end
    fprintf('\nNúmero total de iterações: %d\n',i);
end
```

Figura 20: função criada em MATLAB que calcula uma raiz pelo método da Secante.

ANEXO 4

```matlab
function [ raiz ] = newton_raphson(f, dfdx, Xe, tol, imax)
%A função newton_raphson calcula a raíz de uma função qualquer de uma
%variável em torno de um valor de estimativa dado utilizando o método de
%Newton-Raphson.
%Parâmetros:
%f: nome da função ou vetor de caracteres com a expressão da função.
%dfdx: nome da função derivada ou vetor de caracteres com a expressão
%analítica da derivada.
%Xe: estimativa inicial para a raíz.
%tol: tolerância
%imax: número máximo de iterações.
%raiz: variável de saída.

    %validação de parâmetros
    if ischar(f)==true %função é um vetor de caracteres
        f = str2func(strcat('@(x)',f));%transforma em function_handle
    elseif isa(f,'function_handle')==false&& ...
            isa(f,'inline')==false%f não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro funcao.');
        return;
    end
    if ischar(dfdx)==true%derivada é um vetor de caracteres
        dfdx = str2func(strcat('@(x)',dfdx));%transforma em function_handle
    elseif isa(dfdx,'function_handle')==false&& ...
            isa(dfdx,'inline')==false%dfdx não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro derivada.');
        return;
    end
    if isnumeric(Xe)==false %estimativa não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro Xe.');
        return;
    end
    if isnumeric(tol)==false %tolerância não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro tol.');
        return;
    end
    if isnumeric(imax)==false %imax não é um tipo numérico
        disp('Erro. O parâmetro imax deve ser numérico.');
        return;
    end

    %processamento
    format longe;%formato longo com notação exponencial
    fprintf('iteração\t\tXe\t\t Solução(Xi)\t\tf(Xi)\t\ttolerância\n');
    for i=1:imax
        %cálculo da estimativa atual
        Xi = Xe - f(Xe)/dfdx(Xe);
        Ti = abs((Xi - Xe)/Xe);
        fprintf('%4d\t\t%.6f\t\t%.6f\t%.10f\t%.10f \n',i,Xe,Xi,f(Xi),Ti);
        %quando o erro relativo da estimativa atual com a anterior é menor
        %que a tolerância, temos o resultado da raiz.
        if Ti<tol
            raiz = Xi; %valor é atribuído à variável de saída
            break; %finaliza laço de repetição
        end

        Xe = Xi; %atualiza o valor da estimativa anterior
    end
    if i==imax %núemro máximo de iterações atingido
        disp('Função finalizada, pois o número máximo de iterações foi atingido.');
        raiz = Xi;
    end
    fprintf('\nNúmero total de iterações: %d\n',i);
end
```

Figura 21: função criada em MATLAB que calcula uma raiz pelo método de Newton-Raphson.

ANEXO 5

```
function [ raiz ] = ponto_fixo(f,Xe,tol,imax)
%A função ponto_fixo calcula a raíz de uma função qualquer de uma
%variável em torno de um valor de estimativa dado utilizando o método do
%Ponto Fixo.
%Parâmetros:
%f: nome da função ou vetor de caracteres com a expressão da função de
%iteração.
%Xe: estimativa inicial para a raiz.
%tol: tolerância
%imax: número máximo de iterações.
%raiz: variável de saída.

    %validação de parâmetros
    if ischar(f)==true %função é um vetor de caracteres
        f = str2func(strcat('@(x)',f));%transforma em function_handle
    elseif isa(f,'function_handle')==false&& ...
            isa(f,'inline')==false%f não é do tipo function_handle ou inline
        disp('Tipo de variável inválido para o parâmetro funcao.');
        return;
    end
    if isnumeric(Xe)==false %estimativa não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro Xe.');
        return;
    end
    if isnumeric(tol)==false %tolerância não é um tipo numérico
        disp('Tipo de variável inválido para o parâmetro tol.');
        return;
    end
    if isnumeric(imax)==false %imax não é um tipo numérico
        disp('Erro. O parâmetro imax deve ser numérico.');
        return;
    end

    %processamento
    format longe;%formato longo com notação exponencial
    fprintf('iteração\t\tXe\t\t Solução(Xi)\t\tf(Xi)\t\ttolerância\n');
    for i=1:imax
        %cálculo da estimativa atual
        Xi = f(Xe); %Xi é a imagem de f no ponto Xe
        Ti = abs((Xi - Xe)/Xe);
        fprintf('%4d\t\t%.6f\t\t%.6f\t%.10f\t%.10f \n',i,Xe,Xi,f(Xi),Ti);
        %quando o erro relativo da estimativa atual com a anterior é menor
        %que a tolerância, temos o resultado da raiz.
        if Ti<tol
            raiz = Xi; %valor é atribuído à variável de saída
            break; %finaliza laço de repetição
        end

        Xe = Xi; %atualiza o valor da estimativa anterior
    end
    if i==imax %núemro máximo de iterações atingido
        disp('Função finalizada, pois o número máximo de iterações foi atingido.');
        raiz = Xi;
    end
    fprintf('\nNúmero total de iterações: %d\n',i);
end
```

Figura 22: função criada em MATLAB que calcula uma raiz pelo método de Iterações de Ponto Fixo.

ANEXO 6 – DETALHES SOBRE A 1ª QUESTÃO.

```
>> f = @(x) sqrt(x+15)-cos(x-5)-4

f =

  function_handle with value:

    @(x)sqrt(x+15)-cos(x-5)-4
```

Figura 23: declaração da função conforme enunciado da questão.

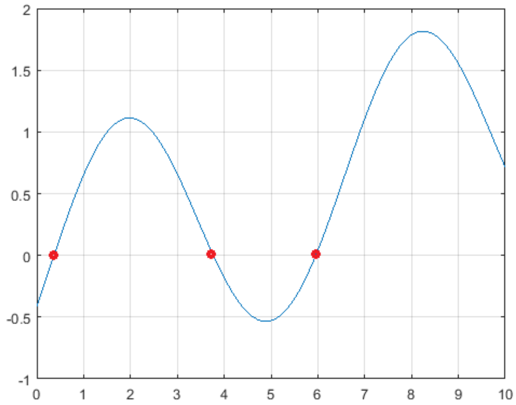

Figura 24: Gráfico da função evidenciando as raízes nos intervalos fornecidos.

```
>> bissecao(f,0,2,1e-5,100)
iteração        x1           x2          Solução(Xi)       f(Xi)          tolerância
   1         0.000000     2.000000      1.000000     0.653644       1.0000000000
   2         0.000000     1.000000      0.500000     0.147800       0.5000000000
   3         0.000000     0.500000      0.250000     -0.132477      0.2500000000
   4         0.250000     0.500000      0.375000     0.008375       0.1250000000
   5         0.250000     0.375000      0.312500     -0.061995      0.0625000000
   6         0.312500     0.375000      0.343750     -0.026781      0.0312500000
   7         0.343750     0.375000      0.359375     -0.009194      0.0156250000
   8         0.359375     0.375000      0.367188     -0.000407      0.0078125000
   9         0.367188     0.375000      0.371094     0.003984       0.0039062500
  10         0.367188     0.371094      0.369141     0.001789       0.0019531250
  11         0.367188     0.369141      0.368164     0.000691       0.0009765625
  12         0.367188     0.368164      0.367676     0.000142       0.0004882813
  13         0.367188     0.367676      0.367432     -0.000133      0.0002441406
  14         0.367432     0.367676      0.367554     0.000005       0.0001220703
  15         0.367432     0.367554      0.367493     -0.000064      0.0000610352
  16         0.367493     0.367554      0.367523     -0.000030      0.0000305176
  17         0.367523     0.367554      0.367538     -0.000012      0.0000152588
  18         0.367538     0.367554      0.367546     -0.000004      0.0000076294

Número total de iterações: 18

ans =

     3.675460815429688e-01
```

Figura 25: cálculo da primeira raiz utilizando o método da Bisseção.

```
>> bissecao(f,2,4,1e-5,100)
iteração         x1           x2          Solução(Xi)      f(Xi)          tolerância
   1          2.000000     4.000000       3.000000     0.658788         1.0000000000
   2          3.000000     4.000000       3.500000     0.230425         0.5000000000
   3          3.500000     4.000000       3.750000     0.014805         0.2500000000
   4          3.750000     4.000000       3.875000     -0.086640        0.1250000000
   5          3.750000     3.875000       3.812500     -0.036642        0.0625000000
   6          3.750000     3.812500       3.781250     -0.011085        0.0312500000
   7          3.750000     3.781250       3.765625     0.001820         0.0156250000
   8          3.765625     3.781250       3.773438     -0.004643        0.0078125000
   9          3.765625     3.773438       3.769531     -0.001414        0.0039062500
  10          3.765625     3.769531       3.767578     0.000202         0.0019531250
  11          3.767578     3.769531       3.768555     -0.000606        0.0009765625
  12          3.767578     3.768555       3.768066     -0.000202        0.0004882813
  13          3.767578     3.768066       3.767822     0.000000         0.0002441406
  14          3.767822     3.768066       3.767944     -0.000101        0.0001220703
  15          3.767822     3.767944       3.767883     -0.000051        0.0000610352
  16          3.767822     3.767883       3.767853     -0.000025        0.0000305176
  17          3.767822     3.767853       3.767838     -0.000013        0.0000152588
  18          3.767822     3.767838       3.767830     -0.000006        0.0000076294

Número total de iterações: 18

ans =

     3.767829895019531e+00
```

Figura 26: cálculo da segunda raiz utilizando o método da Bisseção.

```
>> bissecao(f,4,6,1e-5,100)
iteração         x1           x2          Solução(Xi)      f(Xi)          tolerância
   1          4.000000     6.000000       5.000000     -0.527864        1.0000000000
   2          5.000000     6.000000       5.500000     -0.349890        0.5000000000
   3          5.500000     6.000000       5.750000     -0.176472        0.2500000000
   4          5.750000     6.000000       5.875000     -0.072080        0.1250000000
   5          5.875000     6.000000       5.937500     -0.016054        0.0625000000
   6          5.937500     6.000000       5.968750     0.012835         0.0312500000
   7          5.937500     5.968750       5.953125     -0.001680        0.0156250000
   8          5.953125     5.968750       5.960938     0.005560         0.0078125000
   9          5.953125     5.960938       5.957031     0.001936         0.0039062500
  10          5.953125     5.957031       5.955078     0.000127         0.0019531250
  11          5.953125     5.955078       5.954102     -0.000777        0.0009765625
  12          5.954102     5.955078       5.954590     -0.000325        0.0004882813
  13          5.954590     5.955078       5.954834     -0.000099        0.0002441406
  14          5.954834     5.955078       5.954956     0.000014         0.0001220703
  15          5.954834     5.954956       5.954895     -0.000043        0.0000610352
  16          5.954895     5.954956       5.954926     -0.000015        0.0000305176
  17          5.954926     5.954956       5.954941     -0.000000        0.0000152588
  18          5.954941     5.954956       5.954948     0.000007         0.0000076294

Número total de iterações: 18

ans =

     5.954948425292969e+00
```

Figura 27: cálculo da terceira raiz utilizando o método da Bisseção.

ANEXO 7 – DETALHES SOBRE A 2ª QUESTÃO.

```
>> f = @(x)  (x+2)*(x+1)^2*x*(x-1)^3*(x-2)

f =

  function_handle with value:

    @(x)(x+2)*(x+1)^2*x*(x-1)^3*(x-2)
```

Figura 28

```
>> a1 = bissecao(f,-(3/2),(5/2),1e-5,100)
iteração         x1          x2         Solução(Xi)        f(Xi)          tolerância
   1         -1.500000   2.500000      0.500000     0.527344         2.0000000000
   2         -1.500000   0.500000      -0.500000    -1.582031        1.0000000000
   3         -0.500000   0.500000      0.000000     0.000000         0.5000000000
Solução exata encontrada: 0.000000
Número total de iterações: 3

a1 =

     0
```

Figura 29: cálculo da primeira raiz utilizando o método da Bisseção.

```
>> a2 = bissecao(f,-(1/2),(12/5),1e-5,100)
iteração         x1          x2          Solução(Xi)        f(Xi)          tolerância
   1         -0.500000   2.400000      0.950000     0.001399          1.4500000000
   2         -0.500000   0.950000      0.225000     0.620709          0.7250000000
   3         -0.500000   0.225000     -0.137500    -0.599346          0.3625000000
   4         -0.137500   0.225000      0.043750     0.166624          0.1812500000
   5         -0.137500   0.043750     -0.046875    -0.195320          0.0906250000
   6         -0.046875   0.043750     -0.001563    -0.006260          0.0453125000
   7         -0.001563   0.043750      0.021094     0.082513          0.0226562500
   8         -0.001563   0.021094      0.009766     0.038673          0.0113281250
   9         -0.001563   0.009766      0.004102     0.016338          0.0056640625
  10         -0.001563   0.004102      0.001270     0.005072          0.0028320312
  11         -0.001563   0.001270     -0.000146    -0.000586          0.0014160156
  12         -0.000146   0.001270      0.000562     0.002245          0.0007080078
  13         -0.000146   0.000562      0.000208     0.000830          0.0003540039
  14         -0.000146   0.000208      0.000031     0.000122          0.0001770020
  15         -0.000146   0.000031     -0.000058    -0.000232          0.0000885010
  16         -0.000058   0.000031     -0.000014    -0.000055          0.0000442505
  17         -0.000014   0.000031      0.000008     0.000034          0.0000221252
  18         -0.000014   0.000008     -0.000003    -0.000011          0.0000110626
  19         -0.000003   0.000008      0.000003     0.000011          0.0000055313

Número total de iterações: 19
a2 =
     2.861022949203437e-06
```

Figura 30: cálculo da segunda raiz utilizando o método da Bisseção.

```
>> a3 = bissecao(f,-(1/2),3,1e-5,100)
iteração        x1          x2          Solução(Xi)     f(Xi)          tolerância
   1         -0.500000   3.000000     1.250000     -0.241013      1.7500000000
   2         1.250000    3.000000     2.125000     15.235282      0.8750000000
   3         1.250000    2.125000     1.687500     -4.563950      0.4375000000
   4         1.687500    2.125000     1.906250     -4.388551      0.2187500000
   5         1.906250    2.125000     2.015625     1.204863       0.1093750000
   6         1.906250    2.015625     1.960938     -2.360292      0.0546875000
   7         1.960938    2.015625     1.988281     -0.800994      0.0273437500
   8         1.988281    2.015625     2.001953     0.141842       0.0136718750
   9         1.988281    2.001953     1.995117     -0.344047      0.0068359375
  10         1.995117    2.001953     1.998535     -0.104788      0.0034179688
  11         1.998535    2.001953     2.000244     0.017597       0.0017089844
  12         1.998535    2.000244     1.999390     -0.043827      0.0008544922
  13         1.999390    2.000244     1.999817     -0.013173      0.0004272461
  14         1.999817    2.000244     2.000031     0.002198       0.0002136230
  15         1.999817    2.000031     1.999924     -0.005491      0.0001068115
  16         1.999924    2.000031     1.999977     -0.001648      0.0000534058
  17         1.999977    2.000031     2.000004     0.000275       0.0000267029
  18         1.999977    2.000004     1.999990     -0.000687      0.0000133514
  19         1.999990    2.000004     1.999997     -0.000206      0.0000066757

Número total de iterações: 19
a3 =
     1.999997138977051e+00
```

Figura 31: cálculo da terceira raiz utilizando o método da Bisseção.

```
>> a4 = bissecao(f,-3,-(1/2),1e-5,100)
iteração        x1           x2           Solução(Xi)      f(Xi)           tolerância
   1         -3.000000    -0.500000     -1.750000     -19.192429      1.2500000000
   2         -3.000000    -1.750000     -2.375000     283.204185      0.6250000000
   3         -2.375000    -1.750000     -2.062500     16.980619       0.3125000000
   4         -2.062500    -1.750000     -1.906250     -14.073630      0.1562500000
   5         -2.062500    -1.906250     -1.984375     -3.181891       0.0781250000
   6         -2.062500    -1.984375     -2.023438     5.523622        0.0390625000
   7         -2.023438    -1.984375     -2.003906     0.856183        0.0195312500
   8         -2.003906    -1.984375     -1.994141     -1.238063       0.0097656250
   9         -2.003906    -1.994141     -1.999023     -0.210166       0.0048828125
  10         -2.003906    -1.999023     -2.001465     0.318148        0.0024414063
  11         -2.001465    -1.999023     -2.000244     0.052783        0.0012207031
  12         -2.000244    -1.999023     -1.999634     -0.078993       0.0006103516
  13         -2.000244    -1.999634     -1.999939     -0.013181       0.0003051758
  14         -2.000244    -1.999939     -2.000092     0.019782        0.0001525879
  15         -2.000092    -1.999939     -2.000015     0.003296        0.0000762939
  16         -2.000015    -1.999939     -1.999977     -0.004943       0.0000381470
  17         -2.000015    -1.999977     -1.999996     -0.000824       0.0000190735
  18         -2.000015    -1.999996     -2.000006     0.001236        0.0000095367

Número total de iterações: 18
a4 =
    -2.000005722045898e+00
```

Figura 32: cálculo da quarta raiz utilizando o método da Bisseção.

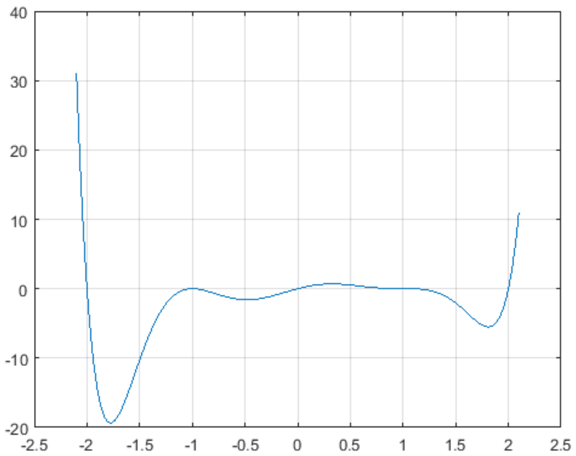

Figura 33: gráfico da função envolvendo todas as raízes apresentadas na questão.

ANEXO 8 – DETALHES SOBRE A 4ª QUESTÃO.

```
>> f = @(x) 2*x*cos(2*x)-(x-2)^2
f =
  function_handle with value:
    @(x)2*x*cos(2*x)-(x-2)^2
>> g = @(x) sin(x)-exp(-x)
g =
  function_handle with value:
    @(x)sin(x)-exp(-x)
>> h = @(x) log(x-1)+cos(x-1)
h =
  function_handle with value:
    @(x)log(x-1)+cos(x-1)
>> df = @(x) 2*cos(2*x)-4*x*sin(2*x)-2*(x-2)
df =
  function_handle with value:
    @(x)2*cos(2*x)-4*x*sin(2*x)-2*(x-2)
>> dg = @(x) cos(x)+exp(-x)
dg =
  function_handle with value:
    @(x)cos(x)+exp(-x)
>> dh = @(x) (1/(x-1))-sin(x-1)
dh =
  function_handle with value:
    @(x)(1/(x-1))-sin(x-1)
```

Figura 34: declaração das funções e de suas derivadas, conforme enunciado da questão.

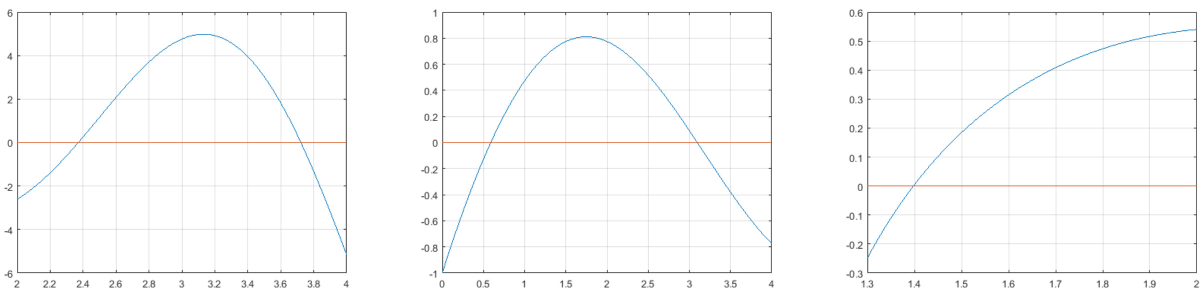

Figura 35: gráficos das funções envolvendo as raízes. À esquerda, f(x); no centro, g(x) e; à direita, h(x).

Aqui serão mostrados os gráficos das ordens de convergência das funções usando o método de Newton-Raphson.

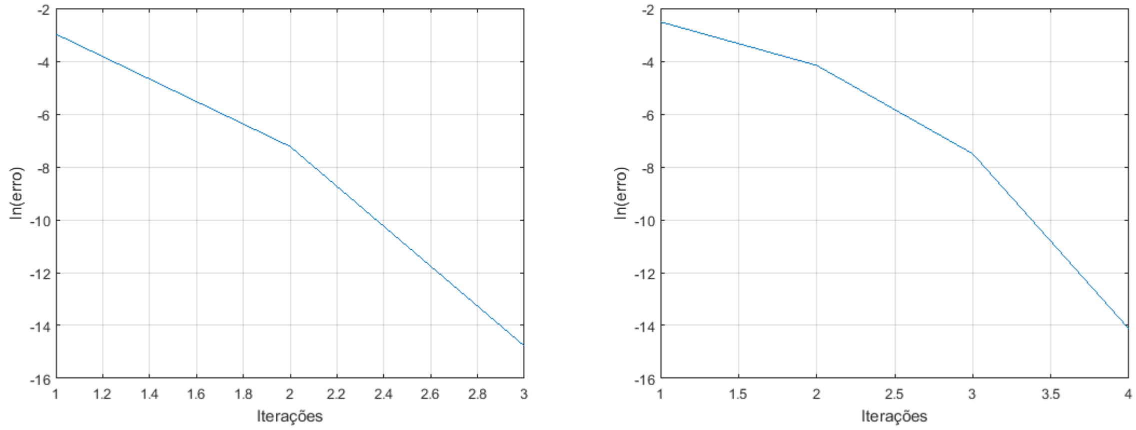

Figura 36: ordem de convergência da função f(x). À esquerda, intervalo 1; à direita, intervalo 2.

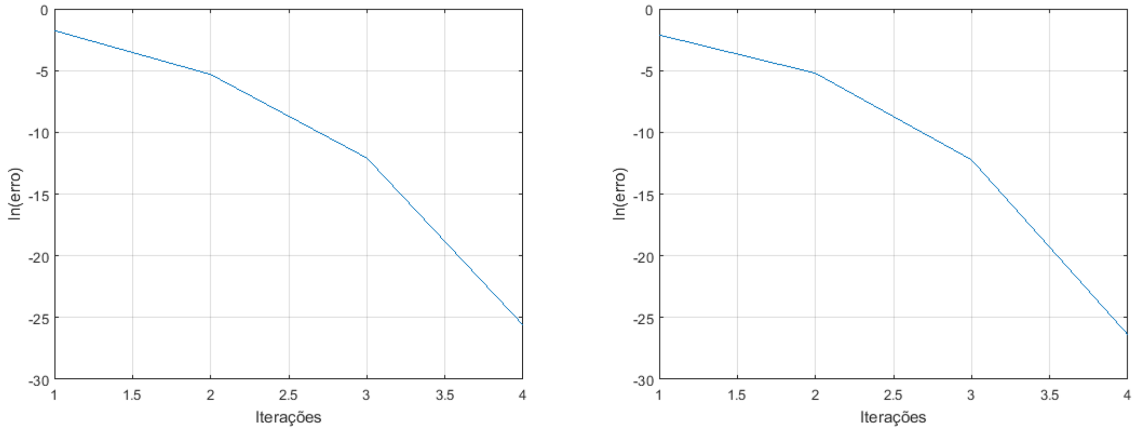

Figura 37: ordem de convergência da função g(x). À esquerda, intervalo 1; à direita, intervalo 2.

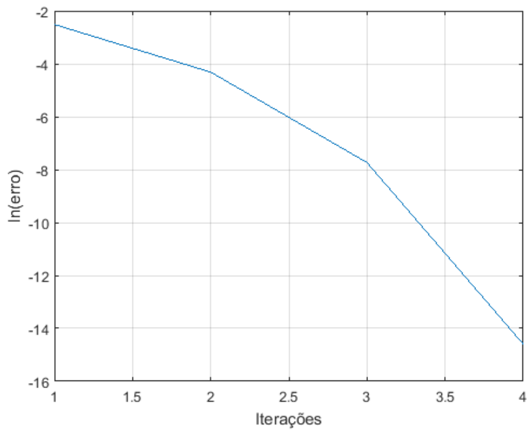

Figura 38: ordem de convergência da função h(x).

ANEXO 9 – DETALHES SOBRE A 5ª QUESTÃO.

Nesta questão, as funções, os gráficos e as declarações das funções no MATLAB são iguais aos da 4ª questão. Portanto, serão mostrados apenas os gráficos de ordem de convergência das funções usando o método da Secante.

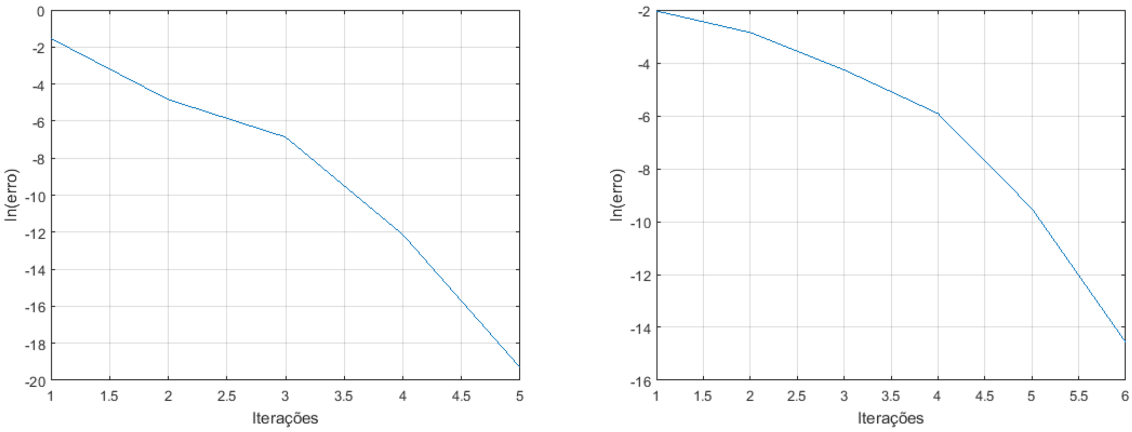

Figura 39: ordem de convergência da função f(x). À esquerda, intervalo 1; à direita, intervalo 2.

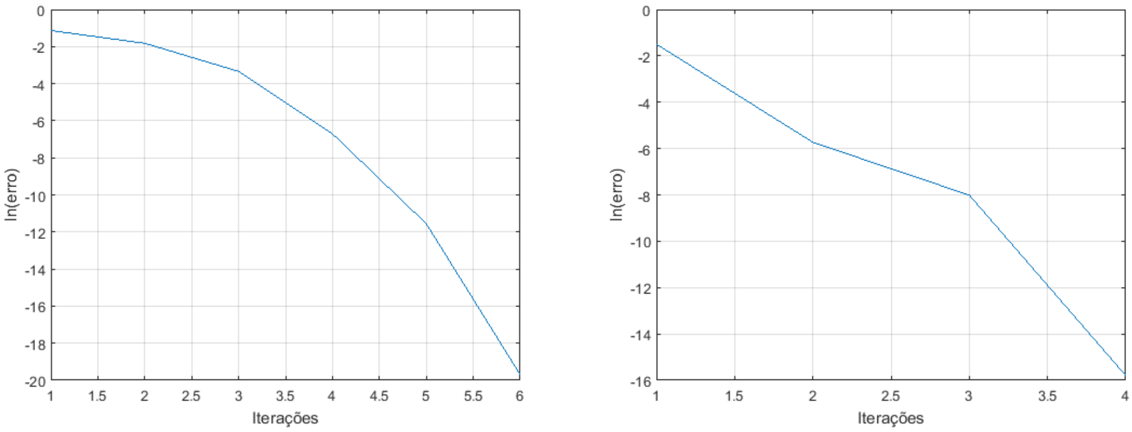

Figura 40: ordem de convergência da função g(x). À esquerda, intervalo 1; à direita, intervalo 2.

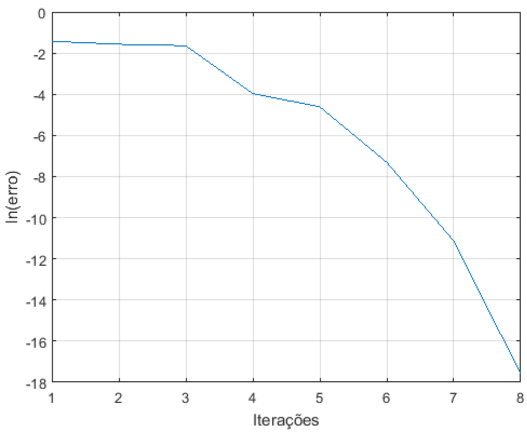

Figura 41: ordem de convergência da função h(x).

ANEXO 10 – DETALHES SOBRE A 6ª QUESTÃO.

```
>> f = @(x) log(x^2+1)-exp(0.4*x)*cos(pi*x)
f =
  function_handle with value:
    @(x)log(x^2+1)-exp(0.4*x)*cos(pi*x)
>> dfdx = @(x) ((2*x)/(x^2+1))-0.4*exp(0.4*x)*cos(pi*x)+pi*exp(0.4*x)*sin(pi*x)
dfdx =
  function_handle with value:
    @(x)((2*x)/(x^2+1))-0.4*exp(0.4*x)*cos(pi*x)+pi*exp(0.4*x)*sin(pi*x)
```

Figura 42: declaração da função e de sua derivada, conforme enunciado da questão.

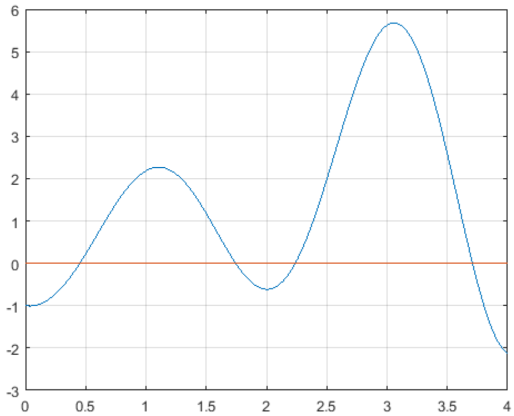

Figura 43: gráfico da função. Vemos que as três primeiras raízes positivas estão nos intervalos [0,1/2], [3/2,2] e [2,5/2]. Portanto, foram usados esses valores de intervalo para o método da Secante e o valor médio do intervalo para a estimativa do método de Newton-Raphson.

```
>> tic; newton_raphson(f,dfdx,0.5,1e-6,50), toc;
iteração        Xe        Solução(Xi)        f(Xi)        tolerância
   1         0.500000       0.451879     0.0053735158     0.0962416806
   2         0.451879       0.450658     0.0000043570     0.0027029789
   3         0.450658       0.450657     0.0000000000     0.0000022011
   4         0.450657       0.450657     0.0000000000     0.0000000000

Número total de iterações: 4
ans =
     4.506567478899358e-01
Elapsed time is 0.032329 seconds.
```

Figura 44: aplicando método NR (Newton-Raphson) para calcular a primeira raiz positiva (antes da alteração).

```
>> tic; newton_raphson(f,dfdx,1.75,1e-6,50), toc;
iteração        Xe        Solução(Xi)        f(Xi)        tolerância
   1         1.750000       1.744705     0.0001386740     0.0030255449
   2         1.744705       1.744738     0.0000000052     0.0000187744
   3         1.744738       1.744738     0.0000000000     0.0000000007

Número total de iterações: 3
ans =
     1.744738053368827e+00
Elapsed time is 0.008068 seconds.
```

Figura 45: aplicando método NR para calcular a segunda raiz positiva (antes da alteração).

```
>> tic; newton_raphson(f,dfdx,2.25,1e-6,50), toc;
iteração         Xe        Solução(Xi)        f(Xi)         tolerância
   1          2.250000       2.238582     0.0013788427     0.0050748184
   2          2.238582       2.238320     0.0000007321     0.0001169153
   3          2.238320       2.238320     0.0000000000     0.0000000621

Número total de iterações: 3
ans =
     2.238319795074177e+00
Elapsed time is 0.004335 seconds.
```

Figura 46: aplicando método NR para calcular a terceira raiz positiva (antes da alteração).

```
>> tic; newton_raphson(f,dfdx,0.5,1e-6,50), toc;

Número total de iterações: 4
ans =
     4.506567478899358e-01
Elapsed time is 0.004730 seconds.
>> tic; newton_raphson(f,dfdx,1.75,1e-6,50), toc;

Número total de iterações: 3
ans =
     1.744738053368827e+00
Elapsed time is 0.009042 seconds.
>> tic; newton_raphson(f,dfdx,2.25,1e-6,50), toc;

Número total de iterações: 3
ans =
     2.238319795074177e+00
Elapsed time is 0.002354 seconds.
```

Figura 47: aplicando o método NR para calcular as três primeiras raízes positivas da função. Aqui foi alterada a função newton_raphson para que não exibisse os dados de cada iteração e, assim, economizasse tempo de processamento.

```
>> tic; secante(f,0,0.5,1e-6,50), toc;
iteração         x1          x2        Solução(x3)       f(x3)         tolerância
   1          0.000000    0.500000       0.408783    -0.1783485473    0.1824344747
   2          0.500000    0.408783       0.449303    -0.0059416281    0.0991235652
   3          0.408783    0.449303       0.450699    0.0001864699     0.0031080011
   4          0.449303    0.450699       0.450657    -0.0000001693    0.0000942793
   5          0.450699    0.450657       0.450657    -0.0000000000    0.0000000855

Número total de iterações: 5
ans =
     4.506567478888424e-01
Elapsed time is 0.025429 seconds.
```

Figura 48: aplicando o método da Secante para calcular a primeira raiz positiva (antes da alteração).

```
>> tic; secante(f,1.5,2,1e-6,50), toc;
iteração       x1          x2         Solução(x3)       f(x3)         tolerância
   1        1.500000    2.000000       1.828360    -0.3144968732   0.0858197890
   2        2.000000    1.828360       1.649385    0.4390208980    0.0978885231
   3        1.828360    1.649385       1.753661    -0.0373811200   0.0632212768
   4        1.649385    1.753661       1.745479    -0.0031341928   0.0046657162
   5        1.753661    1.745479       1.744730    0.0000330257    0.0004289962
   6        1.745479    1.744730       1.744738    -0.0000000281   0.0000044752
   7        1.744730    1.744738       1.744738    -0.0000000000   0.0000000038

Número total de iterações: 7
ans =
     1.744738053368886e+00
Elapsed time is 0.020020 seconds.
```

Figura 49: aplicando o método da Secante para calcular a segunda raiz positiva (antes da alteração).

```
>> tic; secante(f,2,2.5,1e-6,50), toc;
iteração       x1          x2         Solução(x3)       f(x3)         tolerância
   1        2.000000    2.500000       2.118613    -0.4708609219   0.1525546223
   2        2.500000    2.118613       2.191856    -0.2210273711   0.0345708623
   3        2.118613    2.191856       2.256653    0.1000457850    0.0295627797
   4        2.191856    2.256653       2.236462    -0.0097382395   0.0089472018
   5        2.256653    2.236462       2.238253    -0.0003496523   0.0008008142
   6        2.236462    2.238253       2.238320    0.0000013257    0.0000298003
   7        2.238253    2.238320       2.238320    -0.0000000002   0.0000001126

Número total de iterações: 7
ans =
     2.238319795040133e+00
Elapsed time is 0.009471 seconds.
```

Figura 50: aplicando o método da Secante para calcular a terceira raiz positiva (antes da alteração).

```
>> tic; secante(f,0,0.5,1e-6,50), toc;

Número total de iterações: 5
ans =
     4.506567478888424e-01
Elapsed time is 0.011602 seconds.
>> tic; secante(f,1.5,2,1e-6,50), toc;

Número total de iterações: 7
ans =
     1.744738053368886e+00
Elapsed time is 0.002466 seconds.
>> tic; secante(f,2,2.5,1e-6,50), toc;

Número total de iterações: 7
ans =
     2.238319795040133e+00
Elapsed time is 0.001808 seconds.
```

Figura 51: aplicando o método da Secante para calcular as três primeiras raízes positivas da função. Aqui foi alterada a função secante para que não exibisse os dados de cada iteração e, assim, economizasse tempo de processamento.

```
>> tic; fzero(f,0.5), toc;
ans =
     4.506567478899357e-01
Elapsed time is 0.010251 seconds.
>> tic; fzero(f,1.75), toc;
ans =
     1.744738053368827e+00
Elapsed time is 0.037282 seconds.
>> tic; fzero(f,2.25), toc;
ans =
     2.238319795074138e+00
Elapsed time is 0.006313 seconds.
```

Figura 52: aplicando a função fzero do MATLAB, que utiliza o método da Bisseção, para calcular as três primeiras raízes positivas da função.