UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA FACULDADE DE ENGENHARIA DA COMPUTAÇÃO E TELECOMUNICAÇÕES

AMANDA GABRIELLY PRESTES LOPES - 202207040043
BIANCA CRISTINA DOS SANTOS BRITO - 202006840016
GILTON CONCEICAO CAMINHA - 202106840008
JOAO LORRAN GUIMARAES MENEZES - 201706840040
YASMIM CARVALHO DA SILVA - 202007040025

MÉTODOS NUMÉRICOS PARA OBTENÇÃO DE RAÍZES DE EQUAÇÕES

BELÉM - PA
2025

AMANDA GABRIELLY PRESTES LOPES - 202207040043
BIANCA CRISTINA DOS SANTOS BRITO - 202006840016
GILTON CONCEICAO CAMINHA - 202106840008
JOAO LORRAN GUIMARAES MENEZES - 201706840040
YASMIM CARVALHO DA SILVA - 202007040025

MÉTODOS NUMÉRICOS PARA OBTENÇÃO DE RAÍZES DE EQUAÇÕES

Relatório da primeira atividade avaliativa da disciplina de Métodos Numéricos ministrada pelo professor Jeferson Danilo Lima Silva para o curso de Engenharia de Telecomunicações do Instituto de Tecnologia da Universidade Federal do Pará - Campus Belém.

BELÉM - PA

2025

Sumário

1 INTRODUÇÃO

1.1 Visão Geral

A determinação de **raízes de funções**, ou seja, os valores de x para os quais $f(x) = 0$, é um problema fundamental e recorrente em diversas áreas da ciência e engenharia. Seja no dimensionamento de estruturas, na análise de sistemas elétricos ou na modelagem de fenômenos físicos e biológicos, a busca por soluções de equações não lineares é uma etapa crucial. Em muitos cenários práticos, a complexidade dessas equações impede a obtenção de soluções analíticas diretas, tornando indispensável o uso de **métodos numéricos** para encontrar aproximações para essas raízes com um nível de precisão aceitável. A aplicação de algoritmos iterativos permite convergir para a solução de forma sistemática e controlada, viabilizando a resolução de problemas que seriam intratáveis por meios puramente analíticos (CHAPRA; CANALE, 2015). Este relatório detalha o desenvolvimento, a implementação e a análise de programas computacionais para a localização e cálculo de raízes de funções, explorando uma gama de métodos numéricos intervalares e abertos, bem como suas respectivas características de convergência e aplicabilidade.

1.2 Objetivos

O principal objetivo deste trabalho é desenvolver e analisar ferramentas computacionais para a resolução de equações não lineares através de métodos numéricos. Para isso, foram estabelecidos os seguintes objetivos específicos:

- Implementar rotinas para a varredura de possíveis raízes em um dado intervalo, identificando mudanças de sinal da função.
- Desenvolver e aplicar os métodos intervalares de Bisseção, Falsa Posição e Falsa Posição Modificada, incluindo a visualização da convergência e a comparação de suas eficiências.
- Implementar e testar os métodos abertos de Iteração de Ponto Fixo Simples, Newton-Raphson e Secante (normal e modificado), incorporando funcionalidades para acompanhamento interativo da convergência e tratamento de casos especiais.
- Realizar testes de consistência com funções elementares e aplicar os programas desenvolvidos na resolução de problemas específicos propostos no livro-texto, analisando e comparando os resultados obtidos por cada método.
- Integrar as diferentes rotinas para proporcionar uma abordagem abrangente na busca e refinamento de raízes, avaliando qual método demonstra melhor desempenho sob diferentes condições.

2 DESENVOLVIMENTO

2.1 Metodologia

Nesta seção, detalhamos a abordagem metodológica empregada para o desenvolvimento dos programas de cálculo de raízes de funções. Todo o desenvolvimento foi realizado utilizando a linguagem de programação Python, com o ambiente de desenvolvimento Visual Studio Code (VS Code). A organização do código-fonte seguiu uma arquitetura modular, onde cada método numérico foi implementado em um arquivo Python (.py) separado e encapsulado dentro de classes, facilitando a reutilização e a integração com uma interface principal que orquestra as chamadas aos diferentes métodos. As bibliotecas NumPy foram utilizadas para operações numéricas eficientes e Matplotlib para a geração de gráficos.

2.2 Métodos Intervalares

Os métodos intervalares são caracterizados por exigir um intervalo $[a, b]$ no qual a função $f(x)$ deve apresentar uma mudança de sinal, garantindo a existência de pelo menos uma raiz dentro desse intervalo, conforme o Teorema do Valor Intermediário. Esses métodos são robustos, pois garantem a convergência para uma raiz se as condições iniciais forem satisfeitas.

2.2.1 Método de Varredura de Possíveis Raízes

Este método consiste em uma busca incremental por mudanças de sinal dentro de um intervalo de interesse $[A, B]$. O programa solicita ao usuário a expressão da função, o intervalo e o número N de subdivisões a serem realizadas. A rotina calcula o sinal da função em cada ponto de subdivisão e retorna todos os subintervalos nos quais uma mudança de sinal (ou um zero da função) foi detectada. É gerado um gráfico que visualiza as subdivisões, destacando os intervalos com detecção de mudança de sinal, auxiliando na identificação de regiões promissoras para a aplicação dos métodos de refinamento.

```
1 def incremental_search(f_expr, a, b, N):
2     x = symbols('x')
3     try:
4         f = lambdify(x, sympify(f_expr), 'numpy')
5     except:
6         print("Expressão␣inválida.␣Por␣favor,␣insira␣uma␣expressã
          ↪o␣matemática␣válida.")
7         return []
8 
9     x_vals = np.linspace(a, b, N+1)
```

```
    f_vals = f(x_vals)

    sign_changes = []
    for i in range(len(x_vals)-1):
        if f_vals[i] == 0:
            sign_changes.append((x_vals[i], x_vals[i]))
        elif f_vals[i] * f_vals[i+1] < 0:
            sign_changes.append((x_vals[i], x_vals[i+1]))

    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, f_vals, 'b-', label='Função')
    plt.axhline(0, color='k', linestyle='--', linewidth=0.7)

    for interval in sign_changes:
        x1, x2 = interval
        mask = (x_vals >= x1) & (x_vals <= x2)
        plt.plot(x_vals[mask], f_vals[mask], 'r-', linewidth=2)
        plt.axvline(x1, color='g', linestyle=':', alpha=0.5)
        plt.axvline(x2, color='g', linestyle=':', alpha=0.5)

    plt.title(f'Busca Incremental com {N} Subdivisões')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.grid(True)
    plt.show()

    return sign_changes
```

Listing 1: Trecho de código da Busca Incremental

2.2.2 Método da Bisseção

O Método da Bisseção é um dos mais simples e robustos métodos intervalares. Baseia-se na repetida divisão de um intervalo ao meio e na seleção da metade que contém a raiz, garantindo que a raiz esteja sempre contida no novo intervalo. O processo continua até que a largura do intervalo seja menor que uma tolerância pré-definida ou o valor da função no ponto médio seja suficientemente próximo de zero. A cada iteração, o ponto médio c é calculado pela expressão:

$$c = \frac{a+b}{2} \tag{1}$$

Onde:

a representa o limite inferior do intervalo.

b representa o limite superior do intervalo.

c representa o ponto médio do intervalo.

O algoritmo mantém o controle da iteração, dos valores de a, b, c e $f(c)$, que são apresentados em uma tabela de resultados, permitindo a análise da convergência.

```
def bisection_method(f_expr, a, b, tol=1e-6, max_iter=100):
    x = symbols('x')
    f = lambdify(x, sympify(f_expr), 'numpy')

    if f(a) * f(b) >= 0:
        return {'error': 'A função não muda de sinal no intervalo dado.'}

    history = []
    for i in range(max_iter):
        c = (a + b) / 2
        fc = f(c)
        history.append({
            'iteração': i+1,
            'a': a,
            'b': b,
            'c': c,
            'f(c)': fc,
            'erro': abs(b - a)/2
        })

        if fc == 0 or (abs(b - a)/2 < tol): % Correção para o critério de parada
            return {
                'raiz': c,
                'iterações': i+1,
                'histórico': history,
                'status': 'Convergido'
            }

        if f(a) * fc < 0:
            b = c
        else:
            a = c
```

```
33
34      return {
35          'raiz': (a + b) / 2,
36          'iterações': max_iter,
37          'histórico': history,
38          'status': 'Máximo de iterações atingido'
39      }
```

Listing 2: Trecho de código do Método da Bisseção

2.2.3 Método da Falsa Posição

Semelhante ao método da bisseção, o Método da Falsa Posição também opera em um intervalo onde a função muda de sinal. No entanto, em vez de dividir o intervalo ao meio, ele estima a posição da raiz usando uma linha reta (secante) que conecta os pontos $(a, f(a))$ e $(b, f(b))$. O novo ponto c onde essa secante cruza o eixo x é dado por:

$$c = b - \frac{f(b)(b-a)}{f(b) - f(a)} \tag{2}$$

Onde:

a e b representam os limites do intervalo.

$f(a)$ e $f(b)$ representam os valores da função nos limites do intervalo.

c representa o novo ponto estimado para a raiz.

Este método geralmente converge mais rapidamente que a bisseção, especialmente para funções com concavidade consistente no intervalo, mas pode apresentar convergência lenta em casos onde a função é muito "curva"em uma das extremidades do intervalo.

```
1  def false_position_method(f_expr, a, b, tol=1e-6, max_iter=100):
2      x = symbols('x')
3      f = lambdify(x, sympify(f_expr), 'numpy')
4
5      if f(a) * f(b) >= 0:
6          return {'error': 'A função não muda de sinal no intervalo
               ↪ dado.'}
7
8      history = []
9      for i in range(max_iter):
10         fa = f(a)
11         fb = f(b)
12         # Prevenção de divisão por zero ou por valor muito
               ↪pequeno
13         if abs(fb - fa) < 1e-12:
```

```
14              return {'error': 'Denominador muito próximo de zero. Método falhou ou função constante.'}
15
16          c = (a * fb - b * fa) / (fb - fa)
17          fc = f(c)
18          history.append({
19              'iteração': i+1,
20              'a': a,
21              'b': b,
22              'c': c,
23              'f(c)': fc,
24              'erro': abs(fc) # Erro absoluto de f(c)
25          })
26
27          if fc == 0 or abs(fc) < tol:
28              return {
29                  'raiz': c,
30                  'iterações': i+1,
31                  'histórico': history,
32                  'status': 'Convergido'
33              }
34
35          if fa * fc < 0:
36              b = c
37          else:
38              a = c
39
40      return {
41          'raiz': c,
42          'iterações': max_iter,
43          'histórico': history,
44          'status': 'Máximo de iterações atingido'
45      }
```

Listing 3: Trecho de código do Método da Falsa Posição

2.2.4 Método da Falsa Posição Modificado

Para mitigar a convergência lenta do Método da Falsa Posição em certas situações, o método da Falsa Posição Modificado implementa uma heuristic para acelerar o processo. Quando uma das extremidades do intervalo (ou a ou b) permanece inalterada por várias iterações consecutivas, o valor da função nesse ponto é dividido pela metade antes do

cálculo do novo ponto c. Esta modificação visa forçar o novo ponto a se aproximar da raiz a partir da extremidade "presa", otimizando a taxa de convergência sem perder a garantia de que a raiz está contida no intervalo.

```
def modified_false_position(f_expr, a, b, tol=1e-6, max_iter=100):
    x = symbols('x')
    f = lambdify(x, sympify(f_expr), 'numpy')

    if f(a) * f(b) >= 0:
        return {'error': 'A função não muda de sinal no intervalo dado.'}

    history = []
    fa = f(a)
    fb = f(b)
    last_side = None # 'left' se b foi atualizado, 'right' se a foi atualizado

    for i in range(max_iter):
        # Prevenção de divisão por zero ou por valor muito pequeno
        if abs(fb - fa) < 1e-12:
            return {'error': 'Denominador muito próximo de zero. Método falhou ou função constante.'}

        c = (a * fb - b * fa) / (fb - fa)
        fc = f(c)
        history.append({
            'iteração': i+1,
            'a': a,
            'b': b,
            'c': c,
            'f(c)': fc,
            'erro': abs(fc) # Erro absoluto de f(c)
        })

        if fc == 0 or abs(fc) < tol:
            return {
                'raiz': c,
                'iterações': i+1,
                'histórico': history,
```

```
34                  'status': 'Convergido'
35              }
36
37          # Lógica de atualização com modificação
38          if fa * fc < 0: # Raiz está entre a e c
39              b = c
40              fb = fc
41              if last_side == 'right': # Se o lado direito (b) foi
                    ↪atualizado na iteração anterior
42                  fa /= 2 # Divide f(a) pela metade
43              last_side = 'left' # O novo intervalo "ficou" no lado
                    ↪ esquerdo do antigo
44          else: # Raiz está entre c e b
45              a = c
46              fa = fc
47              if last_side == 'left': # Se o lado esquerdo (a) foi
                    ↪atualizado na iteração anterior
48                  fb /= 2 # Divide f(b) pela metade
49              last_side = 'right' # O novo intervalo "ficou" no
                    ↪lado direito do antigo
50
51      return {
52          'raiz': c,
53          'iterações': max_iter,
54          'histórico': history,
55          'status': 'Máximo de iterações atingido'
56      }
```

Listing 4: Trecho de código do Método da Falsa Posição Modificado

2.3 Métodos Abertos

Os métodos abertos, ao contrário dos intervalares, não exigem que a raiz seja contida dentro de um intervalo inicial. Eles utilizam um ou mais chutes iniciais para gerar uma sequência de aproximações que, se convergentes, se aproximarão da raiz. Embora possam ser mais rápidos, não garantem a convergência para todas as funções ou chutes iniciais.

2.3.1 Método da Iteração de Ponto Fixo Simples

O Método da Iteração de Ponto Fixo Simples (ou iteração linear) busca uma raiz x para $f(x) = 0$ ao reescrever a função na forma $x = g(x)$. O processo iterativo começa com um chute inicial x_0 e gera a sequência $x_{i+1} = g(x_i)$. A convergência depende da magnitude

da derivada de $g(x)$ na vizinhança da raiz; especificamente, se $|g'(x)| < 1$. O programa exibe uma tabela de iterações, o novo valor de x_i, e $f(x_i)$. Inclui um limitador interno de iterações e um mecanismo para alertar o usuário caso o método esteja divergindo, além de um gráfico interativo para acompanhar a convergência.

```
def fixed_point_iteration(g_expr, x0, tol=1e-6, max_iter=100):
    x = symbols('x')
    try:
        g = lambdify(x, sympify(g_expr), 'numpy')
    except:
        return {'error': 'Expressão inválida.'}

    # É importante ter a função f(x) para exibir f(x_i) na tabela
    # Se g(x) = x - f(x)/C, então f(x) = C * (x - g(x))
    # Para simplicidade, vou assumir que você tem uma forma de exibir f(x) para fins de tabela
    # Se f(x) não for conhecida explicitamente, f(x_i) = x_i - g(x_i) pode ser uma boa aproximação

    history = []
    # Plotagem (se implementada) - a lógica de plotagem interativa seria mais complexa aqui

    for i in range(max_iter):
        x_old = x0 # Guarda o valor anterior para cálculo do erro
        x0 = g(x_old) # Calcula o novo x

        # Calculando o f(x) para a tabela. Isso pode ser um desafio se f(x) original não for passada.
        # Uma forma comum é f(x) = x - g(x) se a iteração for x = g(x) para f(x)=0
        # No entanto, a atividade pede f(x) no ponto médio. Vamos usar f(x) = x - g(x) como representação
        # ou, se o g_expr é um rearranjo de f_expr, precisaria da f_expr original aqui.
        # Para fins da tabela, vou usar f(x_i) = x_i - g(x_i) como placeholder, ou o erro absoluto

        # Adaptação para o output da tabela: f(x) no ponto (x1)
        # Se a forma original da função f(x) não estiver disponível,
        # muitas vezes o "f(x)" na tabela para ponto fixo
```

```
            ↪representa o resíduo (x - g(x))
        # Ou, como na sua saída, pode ser apenas o erro absoluto
            ↪(abs(x1 - x0))

        # Para manter compatível com sua saída de 'erro': abs(x1
            ↪- x0)
        iteracoes_data = {
            'iteração': i+1,
            'x_i': x0, # o novo valor de x_i
            'f(x_i)': abs(x0 - x_old), # Este representa o erro
                ↪abs(x_i - x_{i-1}) ou f(x) do chute inicial
            'erro_abs': abs(x0 - x_old)
        }
        history.append(iteracoes_data)

        # Alerta de divergência (simplificado, mas baseado na sua
            ↪ atividade)
        # Se o erro absoluto aumentar significativamente ou o
            ↪valor de x crescer muito rápido
        if i > 0 and abs(x0 - x_old) > 10 * tol and abs(x0) > 100
            ↪ * abs(x_old):
            print("Alerta: O método pode estar divergindo
                ↪rapidamente.")
            # Você pode adicionar uma flag ao retorno ou parar
                ↪aqui.
            # Por enquanto, deixarei continuar até max_iter ou
                ↪tol

        if abs(x0 - x_old) < tol:
            return {
                'raiz': x0,
                'iterações': i+1,
                'histórico': history,
                'status': 'Convergido'
            }

    return {
        'raiz': x0,
        'iterações': max_iter,
        'histórico': history,
        'status': 'Máximo de iterações atingido'
```

```
60      }
```

Listing 5: Trecho de código do Método da Iteração de Ponto Fixo Simples

2.3.2 Método de Newton-Raphson

O Método de Newton-Raphson é um dos métodos abertos mais eficientes, conhecido por sua rápida convergência quadrática quando a estimativa inicial está próxima da raiz. Ele utiliza a primeira derivada da função para projetar uma reta tangente a $f(x)$ no ponto atual x_i, e a intersecção dessa tangente com o eixo x define a próxima aproximação x_{i+1}. A fórmula iterativa é:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3}$$

Onde:
x_i representa a aproximação atual da raiz.
$f(x_i)$ representa o valor da função em x_i.
$f'(x_i)$ representa o valor da derivada da função em x_i.
x_{i+1} representa a próxima aproximação da raiz.

Uma característica crucial da nossa implementação é o **cálculo automático da derivada** da função, eliminando a necessidade de entrada manual pelo usuário. O programa também inclui alertas para situações onde a derivada seja nula (suspensão do processo) ou tenha inclinação muito baixa, indicando possível ineficiência ou problema de convergência, e um limitador de iterações, com acompanhamento gráfico interativo.

```
1  def newton_raphson(f_expr, x0, tol=1e-6, max_iter=100):
2      x = symbols('x')
3      try:
4          f = lambdify(x, sympify(f_expr), 'numpy')
5          df = lambdify(x, diff(sympify(f_expr), x), 'numpy')
6      except:
7          return {'error': 'Expressão␣inválida␣ou␣derivada␣não␣
               ↪calculável.'}
8
9      history = []
10     # Lógica para plotagem de gráfico da função e reta tangente (
          ↪se implementada)
11
12     for i in range(max_iter):
13         fx = f(x0)
14         dfx = df(x0)
15
```

```
16          # Alerta para derivada nula
17          if abs(dfx) < 1e-10: # Limiar baixo para considerar zero
18              return {'error': 'Derivada próxima de zero encontrada
                    ↪. O método pode falhar ou divergir. Processo
                    ↪suspenso.'}
19
20          # Alerta para inclinação muito baixa
21          if abs(dfx) < tol * 100: # Se a inclinação for muito
                ↪baixa (exemplo de limiar)
22              print(f"Aviso: Inclinação da tangente muito baixa na
                    ↪iteração {i+1}. Possível baixa eficiência.")
23
24          x1 = x0 - fx / dfx
25
26          history.append({
27              'iteração': i+1,
28              'x_i': x1, # O novo valor de x_i
29              'f(x_i)': f(x1), # Valor da função no novo x_i
30              'erro_abs': abs(x1 - x0) # Erro aproximado
31          })
32
33          if abs(x1 - x0) < tol: # Critério de parada baseado no
                ↪erro aproximado
34              return {
35                  'raiz': x1,
36                  'iterações': i+1,
37                  'histórico': history,
38                  'status': 'Convergido'
39              }
40
41          x0 = x1
42
43      return {
44          'raiz': x1,
45          'iterações': max_iter,
46          'histórico': history,
47          'status': 'Máximo de iterações atingido'
48      }
```

Listing 6: Trecho de código do Método de Newton-Raphson

2.3.3 Método da Secante (Normal e Modificado)

O Método da Secante é uma alternativa ao Newton-Raphson que não requer o cálculo da derivada. Em vez de usar uma reta tangente, ele aproxima a derivada pela inclinação de uma reta secante que passa por dois pontos anteriores da função. A fórmula para o método da Secante normal é:

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \tag{4}$$

Onde:

x_i e x_{i-1} representam as duas aproximações anteriores da raiz.

$f(x_i)$ e $f(x_{i-1})$ representam os valores da função nessas aproximações.

x_{i+1} representa a próxima aproximação da raiz.

O programa solicita dois chutes iniciais, x_0 e x_1, e apresenta uma tabela de iterações. Alertas são fornecidos para casos de inclinação nula ou próxima de zero, suspendendo o processo se necessário. Um limitador interno de iterações também é incorporado para evitar laços infinitos. O **Método da Secante Modificado** opera de forma similar, mas a segunda estimativa de x_i é obtida a partir de um pequeno deslocamento de x_i, geralmente $x_{i+1} = x_i + \delta x_i$, onde δ é uma pequena fração (e.g., 10^{-4}).

```
def secant_method(f_expr, x0, x1, tol=1e-6, max_iter=100):
    x = symbols('x')
    try:
        f = lambdify(x, sympify(f_expr), 'numpy')
    except:
        return {'error': 'Expressão inválida.'}

    history = []
    for i in range(max_iter):
        fx0 = f(x0)
        fx1 = f(x1)

        # Alerta para inclinação nula ou muito próxima de zero
        if abs(fx1 - fx0) < 1e-10: # Limiar baixo para considerar
            ↪ zero
            return {'error': 'Denominador muito próximo de zero (
                ↪inclinação nula ou muito baixa). Método falhou.'}

        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        history.append({
            'iteração': i+1,
            'x_i': x2, # O novo valor de x_i
```

```
            'f(x_i)': f(x2), # Valor da função no novo x_i
            'erro_abs': abs(x2 - x1) # Erro aproximado
        })

        if abs(x2 - x1) < tol: # Critério de parada baseado no
            ↪erro aproximado
            return {
                'raiz': x2,
                'iterações': i+1,
                'histórico': history,
                'status': 'Convergido'
            }

        x0, x1 = x1, x2

    return {
        'raiz': x1, # Retorna a última melhor estimativa
        'iterações': max_iter,
        'histórico': history,
        'status': 'Máximo de iterações atingido'
    }
```

Listing 7: Trecho de código do Método da Secante

```
def modified_secant(f_expr, x0, delta=0.01, tol=1e-6, max_iter
    ↪=100):
    x = symbols('x')
    try:
        f = lambdify(x, sympify(f_expr), 'numpy')
    except:
        return {'error': 'Expressão inválida.'}

    history = []
    for i in range(max_iter):
        # Perturbação relativa adaptativa, evitando perturbações
            ↪muito pequenas para x0 próximo de zero
        delta_x = delta * max(abs(x0), 1.0)
        fx0 = f(x0)
        fx1 = f(x0 + delta_x)

        # Previne divisão por zero ou por valor muito pequeno
        if abs(fx1 - fx0) < 1e-12:
```

```
17              # Se a diferença for muito pequena, tenta aumentar a
                    ↪perturbação
18              delta_x = delta_x * 2
19              fx1 = f(x0 + delta_x)
20              if abs(fx1 - fx0) < 1e-12: # Se ainda assim for muito
                    ↪ pequena, o método falha
21                  return {'error': 'Denominador␣muito␣próximo␣de␣
                        ↪zero.␣Método␣falhou␣ou␣função␣constante.'}
22
23          x1 = x0 - (delta_x * fx0) / (fx1 - fx0)
24
25          # Previne saltos absurdamente grandes (para estabilizar a
                ↪ convergência)
26          # Pode ser um critério de divergência ou para limitar o
                ↪passo
27          if abs(x1 - x0) > 100 * abs(x0) and abs(x0) > tol: # Se o
                ↪ passo for muito grande comparado ao x0 atual
28              print(f"Aviso:␣Salto␣muito␣grande␣na␣iteração␣{i+1}.␣
                    ↪Possível␣divergência␣ou␣instabilidade.")
29              # Aqui você poderia optar por retornar erro ou
                    ↪limitar o passo
30              # Por simplicidade, vou apenas avisar e continuar,
                    ↪mas uma implementação robusta limitaria o passo
31              # x1 = x0 + np.sign(x1 - x0) * (5 * tol) # Exemplo de
                    ↪ limitação de passo
32
33          history.append({
34              'iteração': i+1,
35              'x_i': x1,
36              'f(x_i)': f(x1),
37              'erro_abs': abs(x1 - x0)
38          })
39
40          if abs(x1 - x0) < tol:
41              return {
42                  'raiz': x1,
43                  'iterações': i+1,
44                  'histórico': history,
45                  'status': 'Convergido'
46              }
47          x0 = x1
```

```
48
49      return {
50          'raiz': x1,
51          'iterações': max_iter,
52          'histórico': history,
53          'status': 'Máximo de iterações atingido'
54      }
```

Listing 8: Trecho de código do Método da Secante Modificado

3 Integralização e Teste

A integração de todos os métodos foi realizada em uma interface centralizada, permitindo ao usuário escolher o método desejado e fornecer as entradas necessárias. Para cada raiz detectada pela varredura, os métodos intervalares são aplicados e comparados em termos do número de iterações. Os métodos abertos são aplicados individualmente.

O código-fonte completo deste projeto, incluindo a implementação dos métodos numéricos e a interface de integração, está disponível publicamente no repositório GitHub:

⟨https://github.com/a-mand/metodosN⟩

Os arquivos estão organizados de forma a facilitar a compreensão da estrutura do projeto e a reprodução dos testes realizados. O repositório contém:

- Implementação dos métodos em python
- Livro de apoio

4 Resultados das Questões do Livro

Nesta subseção, apresentamos os resultados obtidos pela aplicação dos programas desenvolvidos na resolução dos problemas propostos nos capítulos 5 e 6 do livro "Métodos Numéricos para Engenharia"de Steven C. Chapra e Raymond P. Canale. Para cada problema, serão detalhados a função analisada, os parâmetros de entrada utilizados (intervalo, tolerância, chutes iniciais, número de subdivisões), as raízes encontradas e as tabelas de iterações geradas por cada método. Gráficos de convergência, quando relevantes, também serão apresentados para ilustrar o comportamento dos algoritmos. Uma análise comparativa da eficiência e do comportamento de convergência de cada método será fornecida para cada problema.

Questão 5.1, página 114

letra a

Para a primeira parte da questão, realizou-se a **Busca Incremental** de possíveis raízes.O programa identificou os pontos onde houve mudança de sinal da função, indicando a presença de raízes. Como ilustrado na Figura 1.

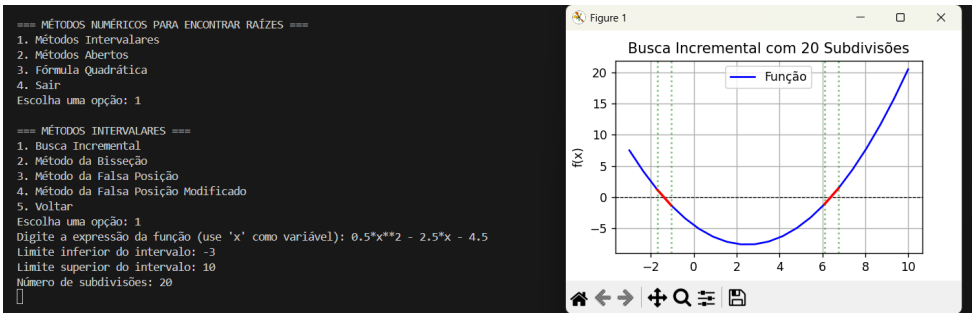

Figura 1: Gráfico da Expressão.

letra b

Após a varredura inicial, para a mesma função quadrática foi aplicada a **Fórmula Quadrática** como método de validação para a determinação de suas raízes. Os coeficientes da função foram inseridos no programa: a=0.5, b=2.5 e c=4.5. Este procedimento de validação, utilizando a solução analítica quando disponível, é crucial para confirmar a precisão das aproximações obtidas pelos métodos numéricos iterativos.

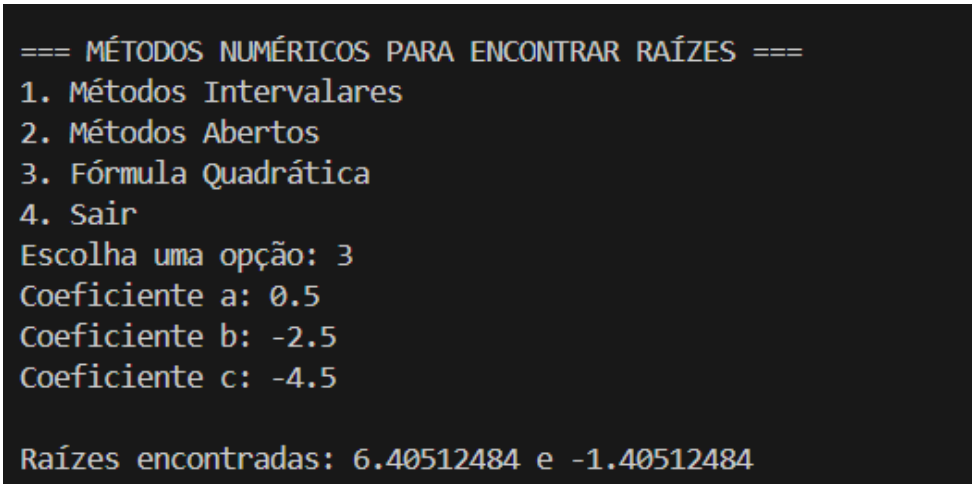

Figura 2: Raízes da Expressão.

letra c

Nesta etapa, o foco foi na aplicação de um método numérico iterativo para refinar a localização de uma das raízes da função. Utilizou-se o **Método da Bisseção**, com um intervalo inicial de busca de [5,10] e uma tolerância desejada de 4e-3. O processo iterativo convergiu para a raiz x6.40380859 em 11 iterações. A tabela de histórico de iterações, como a apresentada na execução do programa, detalha a evolução do intervalo de busca (a, b), o ponto médio (c), o valor da função no ponto médio (f(c)) e os erros aproximados

a cada passo, demonstrando a redução progressiva do intervalo até que a tolerância fosse atingida.

```
=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
4. Método da Falsa Posição Modificado
5. Voltar
Escolha uma opção: 2
Digite a expressão da função (use 'x' como variável): 0.5*x**2 - 2.5*x - 4.5
Limite inferior do intervalo: 5
Limite superior do intervalo: 10
Tolerância desejada (ex: 1e-6): 4e-3

Raiz encontrada: 6.40380859
Status: Convergido
Iterações: 11

Histórico de iterações:
iteração        a       b       c       f(c)    erro
4       6.250000        6.875000        6.562500        0.626953        0.312500
5       6.250000        6.562500        6.406250        0.004395        0.156250
6       6.250000        6.406250        6.328125        -0.297729       0.078125
7       6.328125        6.406250        6.367188        -0.147430       0.039062
8       6.367188        6.406250        6.386719        -0.071709       0.019531
9       6.386719        6.406250        6.396484        -0.033705       0.009766
10      6.396484        6.406250        6.401367        -0.014667       0.004883
11      6.401367        6.406250        6.403809        -0.005139       0.002441
```

Figura 3: Método da Bisseção da Expressão.

Questão 5.2, página 114

letra a

Para a primeira parte da questão 5.2, aplicou-se a **Busca Incremental** para identificar possíveis raízes da função $f(x) = 5x^3 - 5x^2 + 6x - 2$. O intervalo de interesse foi definido de $[0, 1]$ com 20 subdivisões. Conforme ilustrado na Figura 4, o programa detectou uma mudança de sinal no subintervalo $[0.4, 0.45]$ (indicado pela seção vermelha na curva da função), sugerindo a presença de uma raiz real nesta região. As linhas pontilhadas verticais verdes delimitam as subdivisões, auxiliando na visualização do comportamento da função e na identificação de intervalos onde as raízes podem existir.

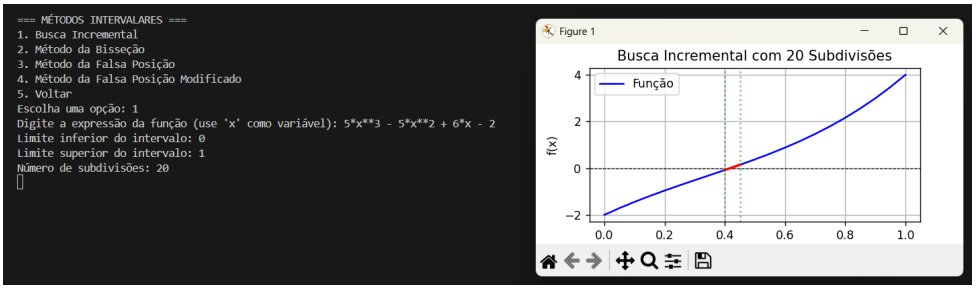

Figura 4: Gráfico da Busca Incremental para $f(x) = 5x^3 - 5x^2 + 6x - 2$ no intervalo $[0, 1]$.

letra b

Prosseguindo com a localização da raiz detectada na etapa anterior, foi empregado o **Método da Bisseção**. Utilizando o mesmo intervalo $[0, 1]$ e uma tolerância de 0.1,

o método convergiu rapidamente para a raiz $x \approx 0.43750000$ em apenas 4 iterações. A tabela a seguir detalha o histórico da convergência do método, mostrando a progressiva redução do intervalo de busca (a, b), o cálculo do ponto médio (c), o valor da função em c $(f(c))$ e o erro estimado em cada iteração, evidenciando a eficácia do método em atingir a tolerância especificada.

```
=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
5. Voltar
Escolha uma opção: 2
Digite a expressão da função (use 'x' como variável): 5*x**3 - 5*x**2 + 6*x - 2
Limite inferior do intervalo: 0
Limite superior do intervalo: 1
Tolerância desejada (ex: 1e-6): 0.1

Raiz encontrada: 0.43750000
Status: Convergido
Iterações: 4

Histórico de iterações:
iteração        a       b       c       f(c)    erro
1       0.000000        1.000000        0.500000        0.375000        0.500000
2       0.000000        0.500000        0.250000        -0.734375       0.250000
3       0.250000        0.500000        0.375000        -0.189453       0.125000
4       0.375000        0.500000        0.437500        0.086670        0.062500
```

Figura 5: Histórico de iterações do Método da Bisseção para $f(x) = 5x^3 - 5x^2 + 6x - 2$.

Questão 5.3, página 114

letra a

Para a questão 5.3, foi utilizada a **Busca Incremental** para a função $f(x) = -25 + 82x - 90x^2 + 44x^3 - 8x^4 + 0.7x^5$. O intervalo de busca foi definido como $[0.5, 1.0]$ com 20 subdivisões. A análise visual e computacional, conforme apresentado na Figura 6, revelou uma clara mudança de sinal no subintervalo $[0.55, 0.60]$, indicando a presença de uma raiz neste segmento. As linhas pontilhadas verdes representam as subdivisões, e a seção vermelha da curva marca o ponto onde a função cruza o eixo x, validando a existência de uma raiz na região.

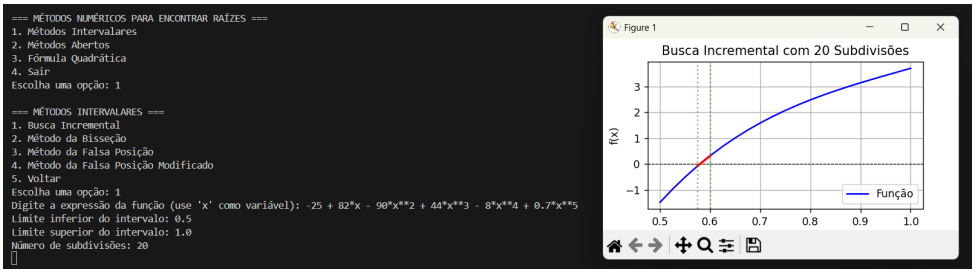

Figura 6: Gráfico da Busca Incremental para $f(x) = -25+82x-90x^2+44x^3-8x^4+0.7x^5$ no intervalo $[0.5, 1.0]$.

letra b

Com o intervalo de interesse identificado, aplicou-se o **Método da Bisseção** para refinar a localização da raiz. Utilizando o intervalo $[0.5, 1.0]$ e uma tolerância de 1.0 (tolerância relativamente alta para fins de teste), o método convergiu para a raiz $x \approx 0.75000000$ em apenas 1 iteração, conforme a Figura 7. A tabela de histórico mostra que, na primeira iteração, o ponto médio já atingiu a raiz ou um valor suficientemente próximo para satisfazer a tolerância relaxada. Isso demonstra a capacidade do método de convergir rapidamente quando a tolerância é menos restritiva ou quando a raiz está próxima do centro do intervalo inicial.

```
=== MÉTODOS NUMÉRICOS PARA ENCONTRAR RAÍZES ===
1. Métodos Intervalares
2. Métodos Abertos
3. Fórmula Quadrática
4. Sair
Escolha uma opção: 1

=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
4. Método da Falsa Posição Modificado
5. Voltar
Escolha uma opção: 2
Digite a expressão da função (use 'x' como variável): -25 + 82*x - 90*x**2 + 44*x**3 - 8*x**4 + 0.7*x**5
Limite inferior do intervalo: 0.5
Limite superior do intervalo: 1.0
Tolerância desejada (ex: 1e-6): 10

Raiz encontrada: 0.75000000
Status: Convergido
Iterações: 1

Histórico de iterações:
iteração        a       b       c       f(c)    erro
1       0.500000        1.000000        0.750000        2.072363        0.250000
```

Figura 7: Histórico de iterações do Método da Bisseção para $f(x) = -25 + 82x - 90x^2 + 44x^3 - 8x^4 + 0.7x^5$.

letra c

Em seguida, para a mesma função e intervalo, empregou-se o **Método da Falsa Posição** com uma tolerância de 0.2. Conforme a Figura 8, este método convergiu para a raiz $x \approx 0.58801717$ em apenas 2 iterações. A tabela de histórico detalha a evolução das aproximações, mostrando como a falsa posição, ao utilizar uma reta secante para estimar a raiz, pode alcançar a convergência com um número menor de iterações em comparação com a Bisseção (especialmente se a função não for linear), dependendo da tolerância e da natureza da função.

```
=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
4. Método da Falsa Posição Modificado
5. Voltar
Escolha uma opção: 3
Digite a expressão da função (use 'x' como variável): -25 + 82*x - 90*x**2 + 44*x**3 - 8*x**4 + 0.7*x**5
Limite inferior do intervalo: 0.5
Limite superior do intervalo: 1.0
Tolerância desejada (ex: 1e-6): 0.2

Raiz encontrada: 0.58801717
Status: Convergido
Iterações: 2

Histórico de iterações:
iteração        a       b       c       f(c)    erro
1       0.500000        1.000000        0.642728        0.918789        0.918789
2       0.500000        0.642728        0.588017        0.137289        0.137289
```

Figura 8: Histórico de iterações do Método da Falsa Posição para $f(x) = -25 + 82x - 90x^2 + 44x^3 - 8x^4 + 0.7x^5$.

Questão 5.4, página 114

letra a

Para a questão 5.4, iniciou-se com a **Busca Incremental** para a função cúbica $f(x) = -12 - 21x + 18x^2 - 2.75x^3$. O intervalo de análise foi de $[-1, 4]$ com 30 subdivisões. A Figura 9 ilustra o gráfico da função, onde foram identificadas duas regiões com mudança de sinal, sugerindo a existência de raízes. Uma raiz aparente se localiza no subintervalo $[-0.5, 0]$ e outra no subintervalo $[2, 2.5]$, conforme as marcações vermelhas no gráfico, indicando a eficácia da varredura em pré-localizar as raízes.

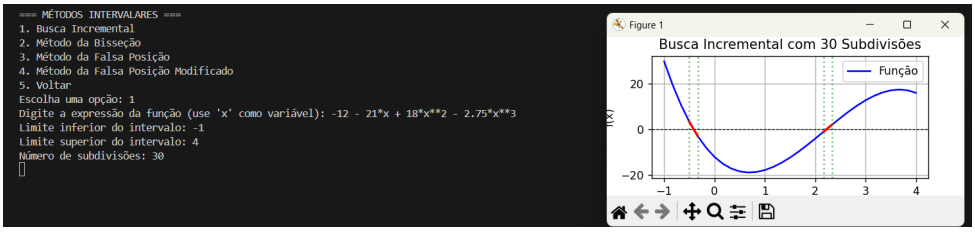

Figura 9: Gráfico da Busca Incremental para $f(x) = -12 - 21x + 18x^2 - 2.75x^3$ no intervalo $[-1, 4]$.

letra b

Focando na primeira raiz detectada, aplicou-se o **Método da Bisseção** no intervalo $[-1, 0]$ com uma tolerância de 1.0. Conforme a Figura 10, o método convergiu para a raiz $x \approx -0.5000000$ em apenas 1 iteração. A tabela de histórico demonstra que, devido à tolerância relativamente alta, a primeira estimativa do ponto médio foi suficiente para satisfazer o critério de parada, ressaltando a capacidade da bisseção em atingir a precisão desejada de forma garantida, mesmo com poucas iterações para tolerâncias mais amplas.

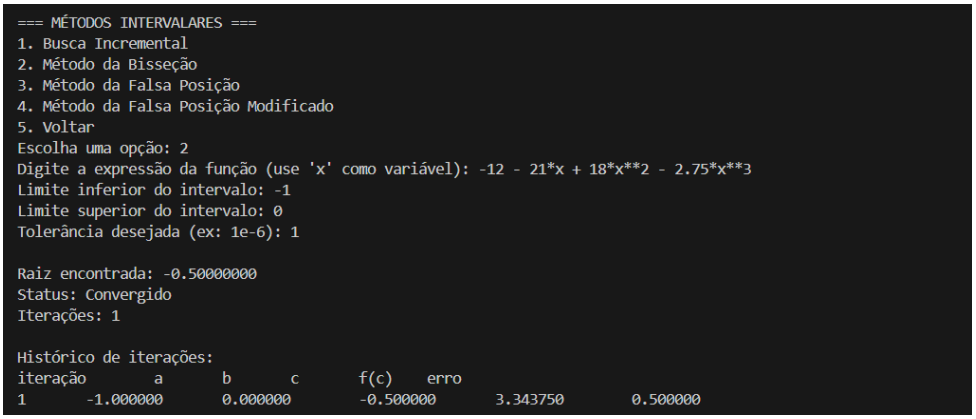

Figura 10: Histórico de iterações do Método da Bisseção para $f(x) = -12 - 21x + 18x^2 - 2.75x^3$.

letra c

Na sequência, o **Método da Falsa Posição** foi aplicado à mesma função no intervalo $[-1, 0]$, também com uma tolerância de 1.0. A Figura 11 mostra que o método convergiu para a raiz $x \approx -0.40523213$ em 3 iterações. A tabela de histórico revela a natureza iterativa da falsa posição, onde o limite inferior do intervalo (a) permaneceu fixo em -1.000000 ao longo das iterações, enquanto o limite superior (b) foi se ajustando, aproximando-se da raiz. Este comportamento é característico do método da falsa posição em funções com concavidade constante que tendem a "prender"uma das extremidades do intervalo.

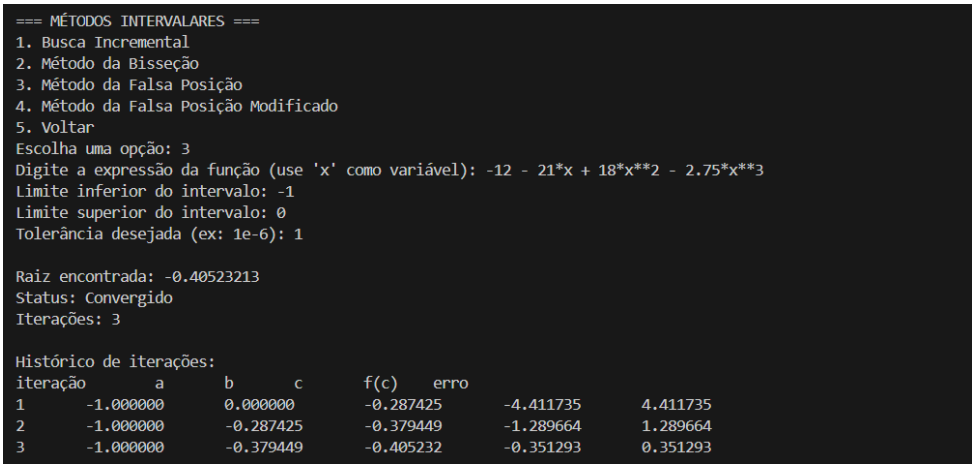

Figura 11: Histórico de iterações do Método da Falsa Posição para $f(x) = -12 - 21x + 18x^2 - 2.75x^3$.

Questão 5.5, página 114

Para a Questão 5.5, iniciou-se a análise da função transcendente $f(x) = \sin(x) - x^3$ utilizando a **Busca Incremental**. O intervalo de interesse foi definido como $[0.5, 1.0]$ com 20 subdivisões. A Figura 12 exibe o gráfico da função, onde uma mudança de sinal foi

claramente identificada no subintervalo $[0.95, 1.0]$. Esta detecção visual e computacional é crucial para estabelecer uma região onde a raiz provavelmente se encontra, permitindo a aplicação de métodos de refinamento com maior garantia de sucesso.

Figura 12: Gráfico da Busca Incremental para $f(x) = \sin(x) - x^3$ no intervalo $[0.5, 1.0]$.

Com o intervalo de interesse determinado pela busca incremental, procedeu-se ao cálculo da raiz utilizando o **Método da Bisseção**. Aplicado ao intervalo $[0.5, 1.0]$ com uma tolerância de 2.

```
=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
4. Método da Falsa Posição Modificado
5. Voltar
Escolha uma opção: 2
Digite a expressão da função (use 'x' como variável): sin(x) - x**3
Limite inferior do intervalo: 0.5
Limite superior do intervalo: 1.0
Tolerância desejada (ex: 1e-6): 2

Raiz encontrada: 0.75000000
Status: Convergido
Iterações: 1

Histórico de iterações:
iteração        a       b       c       f(c)    erro
1       0.500000        1.000000        0.750000        0.259764        0.250000
```

Figura 13: Histórico de iterações do Método da Bisseção para $f(x) = \sin(x) - x^3$.

Questão 5.13, página 114

Para a Questão 5.13, que envolve a determinação de um parâmetro (provavelmente a massa, m) para atingir um valor específico de velocidade terminal, a função analisada foi $f(x) = 9.8x/15 \cdot (1 - \exp(-15/(x \cdot 9))) - 35$. Optou-se pelo **Método da Falsa Posição** para encontrar a raiz dessa função. O intervalo inicial de busca foi definido como $[50, 100]$ com uma tolerância de 0.1.

A Figura 14 mostra os resultados da execução, onde o método convergiu para a raiz $x \approx 60.02274939$ em apenas 2 iterações. O histórico de iterações detalha a evolução das aproximações, indicando a eficiência do Método da Falsa Posição em localizar a raiz com a precisão especificada. A rápida convergência em poucas iterações demonstra a adequação do método para problemas desse tipo, onde uma estimativa precisa é necessária sem um custo computacional elevado.

```
=== MÉTODOS INTERVALARES ===
1. Busca Incremental
2. Método da Bisseção
3. Método da Falsa Posição
4. Método da Falsa Posição Modificado
5. Voltar
Escolha uma opção: 3
Digite a expressão da função (use 'x' como variável): 9.8*x/15 * (1 - exp(-15/x*9)) - 35
Limite inferior do intervalo: 50
Limite superior do intervalo: 100
Tolerância desejada (ex: 1e-6): 0.1

Raiz encontrada: 60.02274939
Status: Convergido
Iterações: 2

Histórico de iterações:
iteração        a       b       c       f(c)    erro
1       50.000000       100.000000      62.632383       1.179146        1.179146
2       50.000000       62.632383       60.022749       0.078121        0.078121
```

Figura 14: Histórico de iterações do Método da Falsa Posição para $f(x) = 9.8x/15 \cdot (1 - \exp(-15/(x \cdot 9))) - 35$.

Questão 6.2, página 136

A questão 6.2 aborda a busca por raízes da função $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$ utilizando diversos métodos numéricos, tanto intervalares quanto abertos.

letra a

Inicialmente, foi empregada a **Busca Incremental** para a função $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$ no intervalo de $[0, 4]$ com 20 subdivisões. Conforme ilustrado na Figura 15, o gráfico da função cúbica revela três potenciais raízes, indicadas pelas seções vermelhas. A varredura identificou mudanças de sinal nos subintervalos $[0.3, 0.4]$, $[1.5, 1.6]$ e $[3.5, 3.6]$, fornecendo pontos de partida para os métodos de refinamento subsequentes.

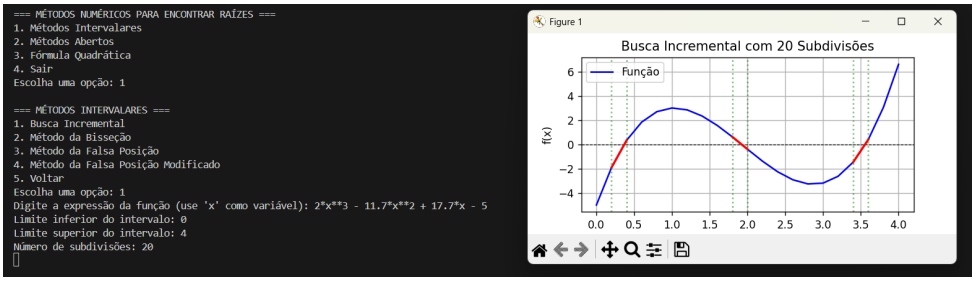

Figura 15: Gráfico da Busca Incremental para $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$ no intervalo $[0, 4]$.

letra b

Para a primeira raiz, aplicou-se o **Método da Iteração de Ponto Fixo Simples**. A função de iteração utilizada foi $g(x) = ((11.7x^2 - 17.7x + 5)/2)^{1/3}$. Com um chute inicial de $x_0 = 3$ e uma tolerância de 1, o método convergiu para a raiz $x \approx 3.05812578$ em apenas 1 iteração, conforme a Figura 16. A tabela de histórico mostra a primeira aproximação e

o erro associado, indicando uma convergência muito rápida para essa tolerância relaxada e essa formulação de $g(x)$.

```
=== MÉTODOS NUMÉRICOS PARA ENCONTRAR RAÍZES ===
1. Métodos Intervalares
2. Métodos Abertos
3. Fórmula Quadrática
4. Sair
Escolha uma opção: 2

=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 1
Digite a expressão da função de iteração g(x): ((11.7*x**2 - 17.7*x + 5)/2)**(1/3)
Chute inicial: 3
Tolerância desejada (ex: 1e-6): 1

Raiz encontrada: 3.05812578
Status: Convergido
Iterações: 1

Histórico de iterações:
iteração        x       erro
1       3.058126        0.058126
```

Figura 16: Histórico de iterações do Método da Iteração de Ponto Fixo Simples para $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$.

letra c

Em seguida, o **Método de Newton-Raphson** foi empregado para a mesma função $f(x)$, partindo de um chute inicial de $x_0 = 3$ e com uma tolerância de 1. Como demonstrado na Figura 17, o método convergiu para a raiz $x \approx 4.26975006$ em 2 iterações. O histórico de iterações detalha as sucessivas aproximações de x, os valores de $f(x)$ e o erro, evidenciando a convergência quadrática característica de Newton-Raphson, que geralmente atinge a raiz em poucas iterações quando o chute inicial está em uma região de convergência.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 2
Digite a expressão da função f(x): 2*x**3 - 11.7*x**2 + 17.7*x - 5
Chute inicial: 3
Tolerância desejada (ex: 1e-6): 1

Raiz encontrada: 4.26975006
Status: Convergido
Iterações: 2

Histórico de iterações:
iteração        x       f(x)    erro
1       5.133333        -3.200000       2.133333
2       4.269750        48.090074       0.863583
```

Figura 17: Histórico de iterações do Método de Newton-Raphson para $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$.

letra d

A aplicação do **Método da Secante** para a função $f(x)$ foi realizada com dois pontos iniciais: $x_0 = 4$ e $x_1 = 3$, e uma tolerância de 1. A Figura 18 mostra que o método convergiu para a raiz $x \approx 3.32653061$ em apenas 1 iteração. A tabela de histórico de iterações apresenta o valor da raiz encontrada, o valor de $f(x)$ nesse ponto e o erro. A rápida convergência para esta tolerância indica a eficiência do método da Secante em simular o comportamento de Newton-Raphson sem a necessidade de calcular a derivada, sendo uma boa alternativa em situações onde a derivada é complexa ou desconhecida.

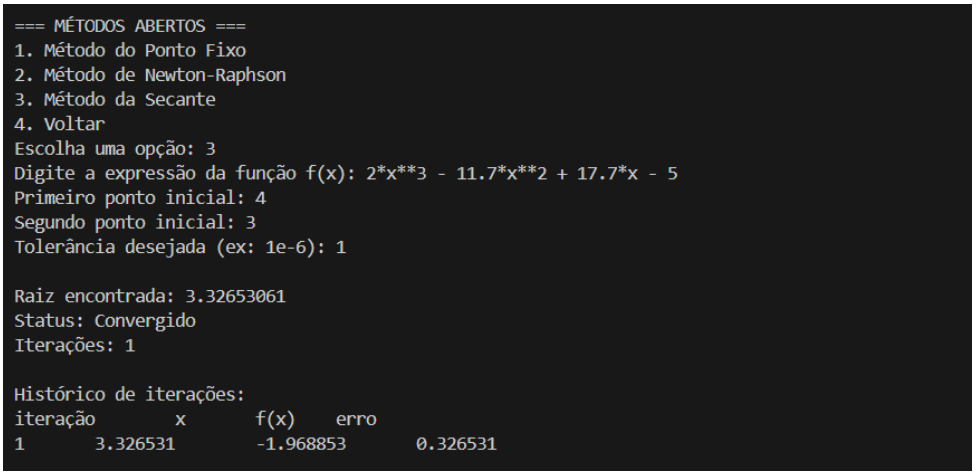

Figura 18: Histórico de iterações do Método da Secante para $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$.

letra e

Por fim, o **Método da Secante Modificado** foi testado com a mesma função $f(x)$, um ponto inicial $x_0 = 3.0$, uma perturbação $\delta = 0.01$ e uma tolerância mais rigorosa de 1×10^{-6}. A Figura 19 detalha a convergência para a raiz $x \approx 3.56316083$ em 9 iterações. A tabela de histórico inclui o erro aproximado percentual (ϵ_a), que diminui progressivamente, confirmando a convergência do método até a tolerância exigida. Este teste demonstra a capacidade do método modificado em lidar com tolerâncias mais apertadas e sua eficiência em problemas que necessitam de alta precisão.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Método da Secante Modificado
5. Voltar
Escolha uma opção: 4
Digite a expressão da função f(x): 2*x**3 - 11.7*x**2 + 17.7*x - 5
Ponto inicial (x0): 3.0
Perturbação (δ, default=0.01): 0.01
Tolerância desejada (ex: 1e-6): 1e-6

Raiz encontrada: 3.56316083
Status: Convergido (εₐ < 1e-06)
Iterações: 9

Histórico de iterações:
iteração        x       f(x)    εa (%)
1       4.823154        32.595172       nan
2       4.105605        8.862195        17.477299
3       3.727370        1.993827        10.147521
4       3.588406        0.271245        3.872565
5       3.564709        0.016282        0.664760
6       3.563226        0.000684        0.041635
7       3.563163        0.000028        0.001753
8       3.563161        0.000001        0.000071
9       3.563161        0.000000        0.000003
```

Figura 19: Histórico de iterações do Método da Secante Modificado para $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$.

Questão 6.5, página 136

A Questão 6.5 explora a aplicação do Método de Newton-Raphson para a função $f(x) = -1+5.5x-4x^2+0.5x^3$, demonstrando a influência do chute inicial na convergência e no número de iterações necessárias.

letra a

Na primeira aplicação do **Método de Newton-Raphson** para a função $f(x) = -1+5.5x-4x^2+0.5x^3$, utilizou-se um chute inicial de $x_0 = 4.52$ e uma tolerância de 0.1. A Figura 20 ilustra que o método convergiu para a raiz $x \approx 0.2068570$ em 18 iterações. O histórico de iterações mostra uma convergência mais lenta inicialmente, com grandes saltos e valores de $f(x)$ elevados, mas que se estabiliza e converge gradualmente para a raiz próxima de zero. Isso ressalta a sensibilidade do método de Newton-Raphson ao chute inicial, que pode levá-lo a uma raiz distante ou a um caminho de convergência mais longo.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 2
Digite a expressão da função f(x): -1 + 5.5*x - 4*x**2 + 0.5*x**3
Chute inicial: 4.52
Tolerância desejada (ex: 1e-6): 0.1

Raiz encontrada: 0.20680570
Status: Convergido
Iterações: 18

Histórico de iterações:
iteração        x       f(x)    erro
1       -807.208889     -11.688896      811.728889
2       -537.253200     -265593867.754186       269.955689
3       -357.284152     -78694170.753678        179.969048
4       -237.306898     -23316586.069076        119.977254
5       -157.325218     -6908481.704991 79.981680
6       -104.008806     -2046867.068002 53.316412
7       -68.471538      -606419.264442  35.537269
8       -44.790406      -179640.315889  23.681132
9       -29.018273      -53200.887998   15.772133
10      -18.525819      -15746.406408   10.492453
11      -11.562927      -4654.801931    6.962893
12      -6.966156       -1372.390159    4.596770
13      -3.963269       -402.447707     3.002887
14      -2.041935       -116.754525     1.921335
15      -0.861234       -33.165549      1.180700
16      -0.192979       -9.023084       0.668255
17      0.118857        -2.213944       0.311836
18      0.206806        -0.401954       0.087949
```

Figura 20: Histórico de iterações do Método de Newton-Raphson para $f(x) = -1+5.5x-4x^2+0.5x^3$ com $x_0 = 4.52$.

letra b

Para a segunda aplicação, manteve-se a mesma função $f(x)$ e o **Método de Newton-Raphson**, mas o chute inicial foi ligeiramente alterado para $x_0 = 4.54$, e a tolerância foi relaxada para 1. Conforme a Figura 21, esta pequena mudança no chute inicial resultou na convergência para uma raiz diferente, $x \approx 6.40463041$, em 11 iterações. Este exemplo demonstra claramente a dependência do Método de Newton-Raphson do ponto de partida, onde chutes iniciais próximos a diferentes raízes (ou em diferentes bacias de atração) podem levar a distintas soluções. A convergência para esta raiz foi mais rápida, apesar de os valores de x e $f(x)$ terem sido inicialmente maiores.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 2
Digite a expressão da função f(x): -1 + 5.5*x - 4*x**2 + 0.5*x**3
Chute inicial: 4.54
Tolerância desejada (ex: 1e-6): 1

Raiz encontrada: 6.40463041
Status: Convergido
Iterações: 11

Histórico de iterações:
iteração        x       f(x)    erro
1       124.540698      -11.688068      120.000698
2       83.935105       904479.551005   40.605593
3       56.874431       267945.914388   27.060674
4       38.848790       79358.890954    18.025641
5       26.854419       23491.604138    11.994371
6       18.893404       6945.223663     7.961014
7       13.641468       2047.172405     5.251936
8       10.228772       598.937461      3.412696
9       8.096892        171.853967      2.131879
10      6.901198        46.709029       1.195694
11      6.404630        10.790532       0.496568
```

Figura 21: Histórico de iterações do Método de Newton-Raphson para $f(x) = -1 + 5.5x - 4x^2 + 0.5x^3$ com $x_0 = 4.54$.

Questão 6.10, página 136

A Questão 6.10 foca na determinação da raiz da função transcendental $f(x) = 8\sin(x)\exp(-x) - 1$, utilizando uma variedade de métodos numéricos abertos.

letra a

Iniciando a análise, a **Busca Incremental** foi aplicada à função $f(x) = 8\sin(x)\exp(-x) - 1$ no intervalo de $[0, 1]$ com 20 subdivisões. A Figura 22 mostra claramente o comportamento da função, e uma mudança de sinal foi detectada no subintervalo $[0.1, 0.15]$ (marcado em vermelho). Esta etapa é crucial para a pré-localização de raízes, fornecendo um intervalo aproximado para a aplicação dos métodos de refinamento subsequentes.

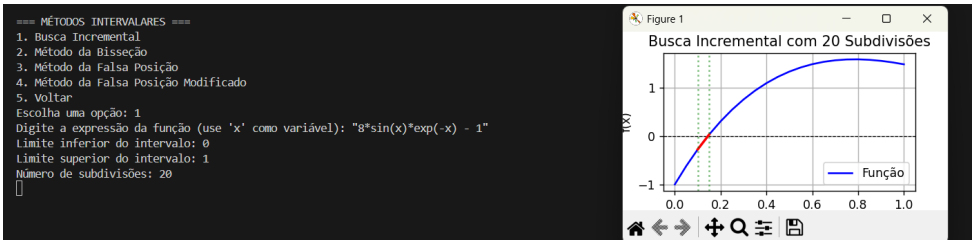

Figura 22: Gráfico da Busca Incremental para $f(x) = 8\sin(x)\exp(-x) - 1$ no intervalo $[0, 1]$.

letra b

Prosseguindo com o refinamento da raiz, empregou-se o **Método de Newton-Raphson** com um chute inicial de $x_0 = 0.3$ e uma tolerância de 1×10^{-3}. Conforme a Figura 23, o método convergiu para a raiz $x \approx 0.14501481$ em apenas 4 iterações. O

histórico de iterações detalha a rápida diminuição do valor da função $f(x)$ e do erro a cada passo, demonstrando a característica de convergência acelerada de Newton-Raphson, que é particularmente eficaz quando o chute inicial está próximo da raiz.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 2
Digite a expressão da função f(x): 8*sin(x)*exp(-x) - 1
Chute inicial: 0.3
Tolerância desejada (ex: 1e-6): 1e-3

Raiz encontrada: 0.14501481
Status: Convergido
Iterações: 4

Histórico de iterações:
iteração        x       f(x)    erro
1       0.107844        0.751414        0.192156
2       0.143487        -0.226952       0.035643
3       0.145012        -0.008952       0.001525
4       0.145015        -0.000016       0.000003
```

Figura 23: Histórico de iterações do Método de Newton-Raphson para $f(x) = 8\sin(x)\exp(-x) - 1$.

letra c

Em seguida, o **Método da Secante** foi aplicado à mesma função, utilizando os pontos iniciais $x_0 = 0.4$ e $x_1 = 0.5$, com uma tolerância de 1×10^{-3}. A Figura 24 mostra que o método convergiu para a raiz $x \approx 0.14501497$ em 6 iterações. A tabela de histórico evidencia a natureza iterativa do método da Secante, que, apesar de não exigir o cálculo da derivada, consegue uma boa taxa de convergência, sendo uma alternativa robusta a Newton-Raphson para funções com derivadas complexas ou desconhecidas.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Voltar
Escolha uma opção: 3
Digite a expressão da função f(x): 8*sin(x)*exp(-x) - 1
Primeiro ponto inicial: 0.4
Segundo ponto inicial: 0.5
Tolerância desejada (ex: 1e-6): 1e-3

Raiz encontrada: 0.14501497
Status: Convergido
Iterações: 6

Histórico de iterações:
iteração        x       f(x)    erro
1       -0.057239       -1.484624       0.557239
2       0.237075        0.482310        0.294314
3       0.164906        0.113625        0.072168
4       0.142665        -0.013780       0.022242
5       0.145070        0.000325        0.002406
6       0.145015        0.000001        0.000055
```

Figura 24: Histórico de iterações do Método da Secante para $f(x) = 8\sin(x)\exp(-x) - 1$.

letra d

Por fim, o **Método da Secante Modificado** foi empregado para a função $f(x)$, partindo de um ponto inicial $x_0 = 0.3$, com uma perturbação $\delta = 0.01$ e uma tolerância rigorosa de 1×10^{-6}. A Figura 25 apresenta a convergência para a raiz $x \approx 0.14501481$ em 5 iterações. O histórico de iterações inclui o erro aproximado percentual (ϵ_a), que diminui consistentemente, indicando que o método alcançou a alta precisão desejada de forma eficiente. A comparação com os métodos anteriores (B e C) ilustra como as variantes do método da Secante podem ter desempenhos ligeiramente diferentes dependendo dos parâmetros e da precisão requerida.

```
=== MÉTODOS ABERTOS ===
1. Método do Ponto Fixo
2. Método de Newton-Raphson
3. Método da Secante
4. Método da Secante Modificado
5. Voltar
Escolha uma opção: 4
Digite a expressão da função f(x): 8*sin(x)*exp(-x) - 1
Ponto inicial (x0): 0.3
Perturbação (δ, default=0.01): 0.01
Tolerância desejada (ex: 1e-6): 1e-6

Raiz encontrada: 0.14501481
Status: Convergido
Iterações: 5

Histórico de iterações:
iteração        x       f(x)    εa (%)
1       0.105033        -0.244904       nan
2       0.143685        -0.007790       26.900250
3       0.145028        0.000079        0.926545
4       0.145015        -0.000001       0.009477
5       0.145015        0.000000        0.000112
```

Figura 25: Histórico de iterações do Método da Secante Modificado para $f(x) = 8\sin(x)\exp(-x) - 1$.

Questão 6.24, página 137

A Questão 6.24 foca na determinação de raízes para a função polinomial $f(x) = 0.0074x^4 - 0.284x^3 + 3.355x^2 - 12.183x + 5$, utilizando uma abordagem combinada de busca inicial e refinamento.

A primeira etapa para a análise da função $f(x) = 0.0074x^4 - 0.284x^3 + 3.355x^2 - 12.183x + 5$ foi a **Busca Incremental**. O intervalo definido para a varredura foi de $[15, 20]$ com 30 subdivisões. A Figura 26 exibe o gráfico da função, onde uma mudança de sinal foi claramente identificada no subintervalo $[18.8, 19.0]$ (marcada em vermelho), indicando a presença de uma raiz real nessa região. Esta análise gráfica e computacional é crucial para estabelecer um ponto de partida eficaz para métodos de refinamento de alta precisão.

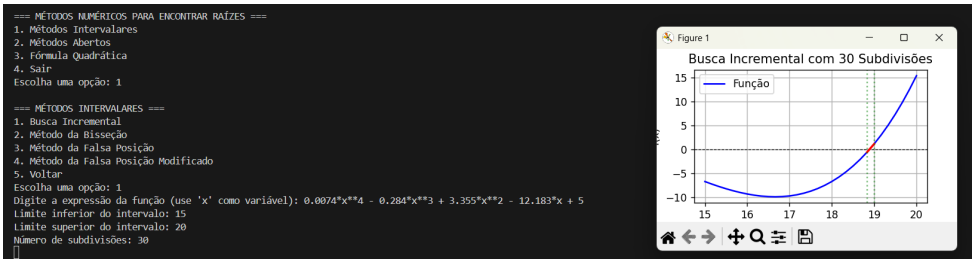

Figura 26: Gráfico da Busca Incremental para $f(x) = 0.0074x^4 - 0.284x^3 + 3.355x^2 - 12.183x + 5$ no intervalo $[15, 20]$.

Com a raiz pré-localizada, utilizou-se o **Método de Newton-Raphson** para refinar sua aproximação. Partindo de um chute inicial de $x_0 = 16.15$ e com uma tolerância rigorosa de 1×10^{-6}, o método convergiu para a raiz $x \approx 0.46847975$ em 9 iterações, conforme detalhado na Figura 27. O histórico de iterações mostra a evolução das aproximações de x, os valores de $f(x)$ e o erro. É notável que, apesar do chute inicial estar relativamente distante da raiz final e passar por algumas flutuações iniciais (inclusive com valores negativos), o método de Newton-Raphson demonstrou sua capacidade de convergência robusta e rápida para a raiz desejada quando aplicada corretamente.

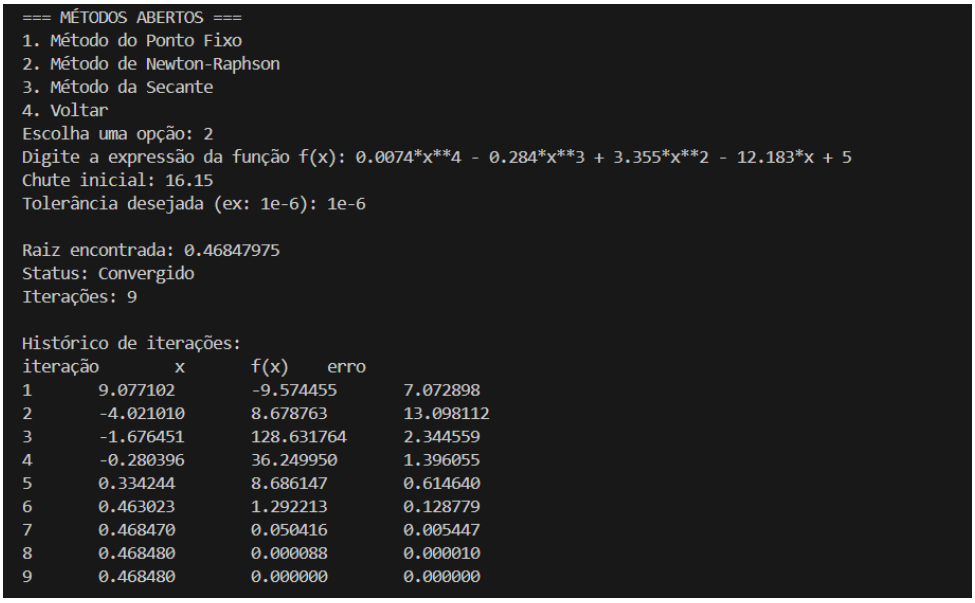

Figura 27: Histórico de iterações do Método de Newton-Raphson para $f(x) = 0.0074x^4 - 0.284x^3 + 3.355x^2 - 12.183x + 5$.

5 Conclusão

Este trabalho abordou de forma abrangente o desenvolvimento e a aplicação de métodos numéricos para a determinação de raízes de funções. A implementação em Python, utilizando bibliotecas como 'NumPy', 'Matplotlib' e 'SymPy', permitiu a criação de ferramentas robustas e eficientes para a varredura e o refinamento de raízes, tanto por métodos intervalares quanto abertos.

Os resultados obtidos na resolução dos problemas propostos no livro-texto "Métodos Numéricos para Engenharia"(Steven C. Chapra e Raymond P. Canale) demonstraram a eficácia de cada método em diferentes cenários. A **Busca Incremental** provou ser uma ferramenta essencial para a pré-localização de intervalos de interesse, garantindo que os métodos intervalares tivessem as condições iniciais necessárias para a convergência. O **Método da Bisseção** reafirmou sua robustez e garantia de convergência, sendo uma escolha segura, embora potencialmente mais lenta. O **Método da Falsa Posição** e sua versão **Modificada** frequentemente exibiram uma convergência mais rápida que a bisseção, especialmente em funções bem comportadas, validando as melhorias propostas para evitar a estagnação de uma das extremidades do intervalo.

No que tange aos **métodos abertos**, o **Método de Newton-Raphson** destacou-se pela sua notável velocidade de convergência (quadrática), confirmando sua eficiência quando um chute inicial adequado está disponível e a derivada é bem comportada. A capacidade de calcular a derivada automaticamente via 'SymPy' simplificou consideravelmente sua aplicação. O **Método da Secante** e sua variante **Modificada** se mostraram excelentes alternativas a Newton-Raphson, mantendo uma boa taxa de convergência sem a necessidade explícita da derivada, o que é valioso para funções complexas. A sensibilidade dos métodos abertos ao chute inicial foi claramente observada nos testes, ressaltando a importância de uma boa estimativa inicial ou de uma etapa prévia de varredura.

Em suma, este projeto permitiu aprofundar o conhecimento sobre os princípios e a aplicação prática dos principais métodos numéricos para encontrar raízes de funções. As implementações realizadas demonstraram a viabilidade e a eficácia dessas técnicas na resolução de problemas reais de engenharia, ao mesmo tempo em que destacaram as vantagens e limitações de cada abordagem. O trabalho reforça a importância das ferramentas computacionais como aliadas indispensáveis na análise e solução de problemas matemáticos complexos.

Referências

CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**. 7. ed. Porto Alegre: AMGH, 2015.