Métodos Numéricos Computacionais

Igor dos Reis Gomes

RELATÓRIO TRABALHO 1 – ZERO DE FUNÇÕES

Métodos Numéricos Computacionais

Bauru

2025

Bauru

2025

Igor dos Reis Gomes

RELATÓRIO TRABALHO 1 – ZERO DE FUNÇÕES

Métodos Numéricos Computacionais

Trabalho teórico-prático apresentado como parte da composição da nota da disciplina de Métodos Numéricos Computacionais – Prof. Douglas Rodrigues.

1. OBJETIVO

O objetivo deste trabalho é desenvolver e aplicar os métodos de Newton, Falsa Posição e de Newton-Raphson, utilizando a linguagem Python, para resolver problemas que provém de contextos reais. Além disso, busca-se verificar a eficiência e precisão de cada método utilizado.

2. EXPLICAÇÃO DO CÓDIGO

Neste trabalho, foi realizado um programa utilizando a linguagem de programação Python, visando resolver os problemas dados. Assim, o programa foi divido em 2 arquivos, sendo esses chamados de funcoes.py e main.py.

- No arquivo funcoes.py,foram implementadas as funções de suporte, incluindo os métodos numéricos, as funções matemáticas dos problemas e suas derivadas.
- No arquivo main.py, foram organizadas as resoluções dos cinco exercícios propostos, além de duas funções para a criação de gráficos comparativos.

2.1. Método da Bisseção

```
def metBissecao(f, a, b, erro):
    inicio = time.perf_counter()
    if f(a) * f(b) >= 0:
          print("Método da Bisseção: Os valores de a e b não satisfazem
f(a) * f(b) < 0\n")
        return None, 0, 0.0, []
    i = 0
    valores_x = []
    while True:
        x = (a + b) / 2
        fx = f(x)


        valores_x.append(x)
        if abs(fx) <= erro or abs(b - a) <= erro:
```

```
            tempo = (time.perf_counter() - inicio)
            return x, i, tempo, valores_x
        if f(a) * fx < 0:
            b = x
        else:
            a = x
        i += 1
```

Nessa função, temos a aplicação do Método da Bisseção, a função tem como parâmetros uma função matemática, o valor de a, de b e o erro. Em cada iteração, o ponto médio é calculado e testado. Os valores de x a cada passo são armazenados em uma lista. Quando o critério de parada é atingido, retorna-se o valor final de x, o número de iterações, o tempo de execução e os valores de x em cada iteração.

2.2. Método da Falsa Posição

```
def metFalsaPosicao(f, a, b, erro):
    inicio = time.perf_counter()
    if f(a) * f(b) >= 0:
            print("Método da Falsa Posição: Os valores de a e b não satisfazem f(a) * f(b) < 0\n")
        return None, 0, 0.0, []
    i = 0
    valores_x = []
    while True:
        x = (a * f(b) - b * f(a)) / (f(b) - f(a))
        fx = f(x)
        valores_x.append(x)
        if abs(fx) <= erro or abs(b - a) <= erro:
            tempo = (time.perf_counter() - inicio)
            return x, i, tempo, valores_x
        if f(a) * fx < 0:
```

```
            b = x
        else:
            a = x
        i += 1
```

A função acima é a aplicação do Método da Falsa Posição, possui os mesmos parâmetros da função do Método da Bisseção, sendo estes: uma função matemática, o valor de a, de b e o erro. Os valores de x para cada iteração são armazenados e é realizado o cálculo do tempo de execução, além da execução do método em si. A função retorna o valor final de x após ser atingido o critério de parada, a quantidade de iterações, o tempo de execução e a lista com todos os valores de x para cada iteração.

2.3. Método de Newton

```
def metNewton(f, df, x0, erro):
    inicio = time.perf_counter()
    maxIteracoes = 100
    i = 0
    valores_x = [x0]
    while True:
        if df(x0) == 0:
            print("Erro matemático: derivada nula\n")
            return None, i, (time.perf_counter() - inicio), valores_x
        x1 = x0 - f(x0) / df(x0)
        i += 1
        valores_x.append(x1)
        if i >= maxIteracoes:
                print("Não convergente: quantidade máxima de iterações
atingida\n")
            return None, i, (time.perf_counter() - inicio), valores_x
        if abs(f(x1)) <= erro:
```

```
            tempo = (time.perf_counter() - inicio)
            return x1, i, tempo, valores_x
        else:
            x0 = x1
```

Agora, por fim, a função apresenta a implementação do Método de Newton, que tem como parâmetros de entrada a função matemática a ser utilizada no método, sua derivada, o valor inicial de x e o erro. Enquanto o critério de parada não é atingido, são realizadas iterações buscando o valor de x. Diferente dos outros métodos, um número máximo de iterações é imposto para evitar loops infinitos em casos de não convergência. Assim como os outros métodos, a função retorna o valor final de x, a quantidade de iterações realizadas, o tempo de execução e a lista com valores de x.

2.4. Funções matemáticas e suas derivadas

```
# Função do Problema 1
def funcaoEx1(p):
    return 5 * p - 100


# Derivada da função do Problema 1
def dfEx1(p):
    return 5


# Função do Problema 2
def funcaoEx2(v):
    Is = 1e-12
    n = 1
    vt = 0.025
    i = 0.010
```

```
    return Is * (math.exp(v / (n * vt)) - 1) - i

# Derivada da função do Problema 2
def dfEx2(v):
    Is = 1e-12
    n = 1
    vt = 0.025
    return (Is / (n * vt)) * math.exp(v / (n * vt))


# Função do Problema 3
def funcaoEx3(x):
    ka = 1 * 10e-5
    c = 0.1

    return x**2 / (c - x) - ka


# Derivada da função do Problema 3
def dfEx3(x):
    c = 0.1
    return (2 * x * (c - x) + x ** 2) / ((c - x) ** 2)


# Função do Problema 4
def funcaoEx4(epsilon, sigma, r):
    termo1 = (sigma / r) ** 13
    termo2 = (sigma / r) ** 7
    return 24 * epsilon * (2 * termo1 - termo2)


# Derivada da função do Problema 4
def dfEx4(epsilon, sigma, r):
    return 24 * epsilon * (-26 * (sigma ** 13) / (r ** 14) + 7 * (sigma
** 7) / (r ** 8))
```

```python
# Função do Problema 5
def funcaoEx5(k):
    return k - 1 - math.exp(-k)

# Derivada da função do Problema 5
def dfEx5(k):
    return math.exp(-k) + 1
```

No trecho acima, são todas as funções matemáticas de cada Problema proposto no trabalho e suas derivadas já calculadas, para que posteriormente possam ser chamadas no arquivo principal para executar o problema e buscar as suas soluções.

2.5. Resolução dos exercícios

Todos os exercícios seguem uma estrutura semelhante. O usuário pode escolher entre utilizar valores padrão para os parâmetros a, b, x0 e erro, ou informar seus próprios valores. A seguir, os três métodos são aplicados ao problema, e seus resultados são exibidos ao usuário.

```python
def ex05():
    print("\nProblema 5 - Engenharia")
    usar_padrao = input("Usar valores padrão? (s/n): ").strip().lower()

    if usar_padrao == 'n':
        a = float(input("Informe o valor de a: "))
        b = float(input("Informe o valor de b: "))
        x0 = float(input("Informe o valor de x0: "))
        erro = float(input("Informe o valor do erro: "))
    else:
        a, b = 0.1, 2.0
```

```
        x0 = 1.0

        erro = 1e-5

          print(f"\nUsando valores padrão: a = {a}, b = {b}, x0 = {x0},
erro = {erro}\n")

               raizBissecao,    itB,    tempoB,    valores_bissecao    =
funcoes.metBissecao(funcoes.funcaoEx5, a, b, erro)

           raizFalsaPosicao,   itF,   tempoF,   valores_falsaposicao   =
funcoes.metFalsaPosicao(funcoes.funcaoEx5, a, b, erro)

                raizNewton,    itN,    tempoN,    valores_newton    =
funcoes.metNewton(funcoes.funcaoEx5, funcoes.dfEx5, x0, erro)


    if raizBissecao is not None:

        print(f"\nValor de K (Bisseção): {raizBissecao:.5f}")

    if raizFalsaPosicao is not None:

        print(f"Valor de K (Falsa Posição): {raizFalsaPosicao:.5f}")

    if raizNewton is not None:

        print(f"Valor de K (Newton-Raphson): {raizNewton:.5f}\n")


    print(f"Iterações (Bisseção): {itB}, Tempo: {tempoB:.8f} s")

    print(f"Iterações (Falsa Posição): {itF}, Tempo: {tempoF:.8f} s")

   print(f"Iterações (Newton-Raphson): {itN}, Tempo: {tempoN:.8f} s\n")


            plot_convergencia(valores_bissecao,   valores_falsaposicao,
valores_newton)


    plot_comparativo(itB, itF, itN, tempoB, tempoF, tempoN)
```

As funções para a execução dos Métodos de Bisseção, Falsa Posição e Newton são chamadas com os devidos parâmetros para solucionar o problema, os valores retornados são armazenados em variáveis. Caso o método tenha convergido, é mostrado os valores da resolução das equações. Além disso, também é mostrado o número de iterações realizadas e o tempo de execução em segundos

de cada método, e por fim, são chamadas as funções de geração de gráficos de comparação de cada método.

2.6. Criação dos gráficos

Para a criação dos gráficos, foram feitas duas funções, segue abaixo:

```
def plot_comparativo(itB, itF, itN, tempoB, tempoF, tempoN):

    metodos = ['Bisseção', 'Falsa Posição', 'Newton-Raphson']

    iteracoes = [itB, itF, itN]

    tempos = [tempoB, tempoF, tempoN]



    plt.figure(figsize=(10, 6))

        plt.bar(metodos, iteracoes, color=['dodgerblue', 'darkorange',
'forestgreen'])

    plt.title("Número de Iterações por Método")

    plt.ylabel("Iterações")

    plt.tight_layout()

    plt.show()



    plt.figure(figsize=(10, 6))

          plt.plot(metodos, tempos, marker='s', linestyle='dashed',
color='darkgreen', label='Tempo')

    plt.title("Tempo de Execução por Método")

    plt.ylabel("Tempo (s)")

    plt.grid(True)

    plt.tight_layout()

    plt.show()



def plot_convergencia(valores_bissecao, valores_falsaposicao,
valores_newton):

    plt.figure(figsize=(12, 7))

   cores = ['blue', 'orange', 'green']
```

```
    if valores_bissecao:
        plt.plot(range(len(valores_bissecao)), valores_bissecao, 'o-', color=cores[0], label='Bisseção')

    if valores_falsaposicao:
        plt.plot(range(len(valores_falsaposicao)), valores_falsaposicao, 's-', color=cores[1], label='Falsa Posição')

    if valores_newton:
        plt.plot(range(len(valores_newton)), valores_newton, '^-', color=cores[2], label='Newton-Raphson')

    plt.title('Convergência dos Métodos Iterativos')
    plt.xlabel('Iterações')
    plt.ylabel('Valor da Raiz')
    plt.grid(True)
    plt.legend()
    plt.show()
```

A primeira função apresenta, em um gráfico de barras, o número de iterações necessárias para cada método encontrar a solução e também apresenta outro gráfico, este agora sendo de linhas, o tempo de execução correspondente a cada método. Já a segunda função mostra a convergência dos métodos ao longo das iterações, ou seja, os valores da raiz a cada iteração de cada método feito.

3. ANÁLISE E DISCUSSÃO

Neste tópico, será realizada a comparação e a análise de eficiência e precisão dos três métodos implementados para cada Problema proposto, para cada exercício, serão utilizados com valores iniciais os padrões propostos no código. Essa comparação leva em conta os resultados numéricos obtidos, o número de iterações necessárias para convergência e o tempo de execução medido. Nos 5 exercícios

realizados, a execução dos três métodos tiveram como resultados valores bastante próximos, notando-se então que, quando os métodos convergem, todos podem ter uma boa precisão. Segue abaixo o resultado de cada exercício.

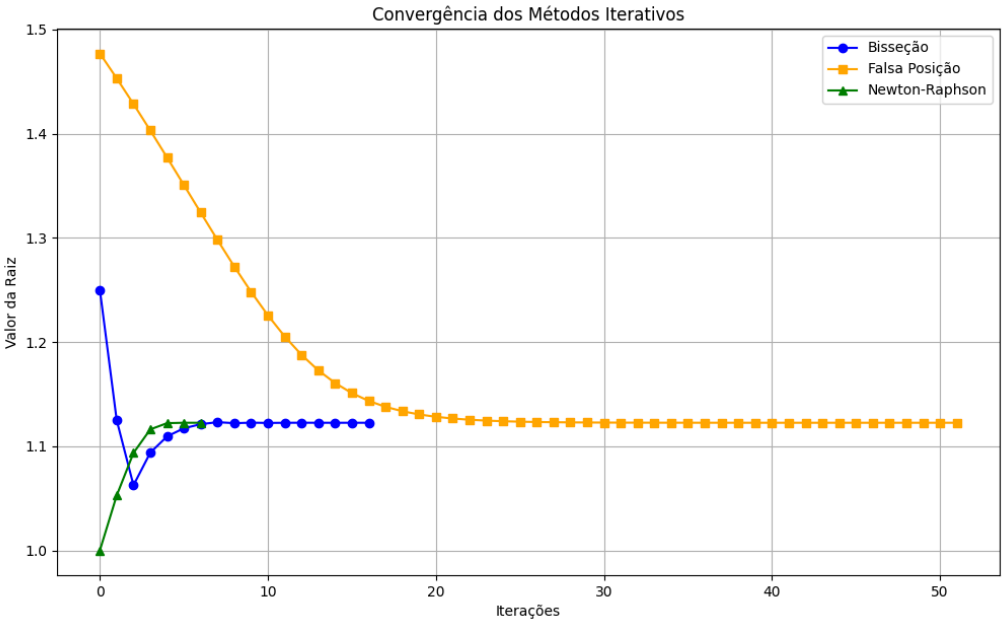
Tabela 1 - Resultados dos exercícios

Exercício	Método	Resultado	Iterações	Tempo (s)
Problema 1	Bisseção	20.00000	22	0.00002440
	Falsa Posição	20.00000	0	0.00000220
	Newton-Raphson	20.00000	1	0.00000230
Problema 2	Bisseção	0.57563	10	0.00002750
	Falsa Posição	0.57563	9	0.00002050
	Newton-Raphson	0.57565	20	0.00002980
Problema 3	Bisseção	0.00301	6	0.00001900
	Falsa Posição	0.00296	30	0.00004030
	Newton-Raphson	0.00317	3	0.00000800
Problema 4	Bisseção	1.12246	16	0.00003740
	Falsa Posição	1.12246	51	0.00010500
	Newton-Raphson	1.12246	6	0.00001180
Problema 5	Bisseção	1.27847	16	0.00003770
	Falsa Posição	1.27847	6	0.00001800
	Newton-Raphson	1.27846	3	0.00000980

Fonte: Elaboração própria

Além da tabela, foram gerados gráficos para auxiliar na análise dos resultados obtidos e a eficiência de cada método. O tipo de gráfico para cada exercício é igual, o que se altera é somente os valores, segue abaixo os gráficos com os valores do Problema 4:

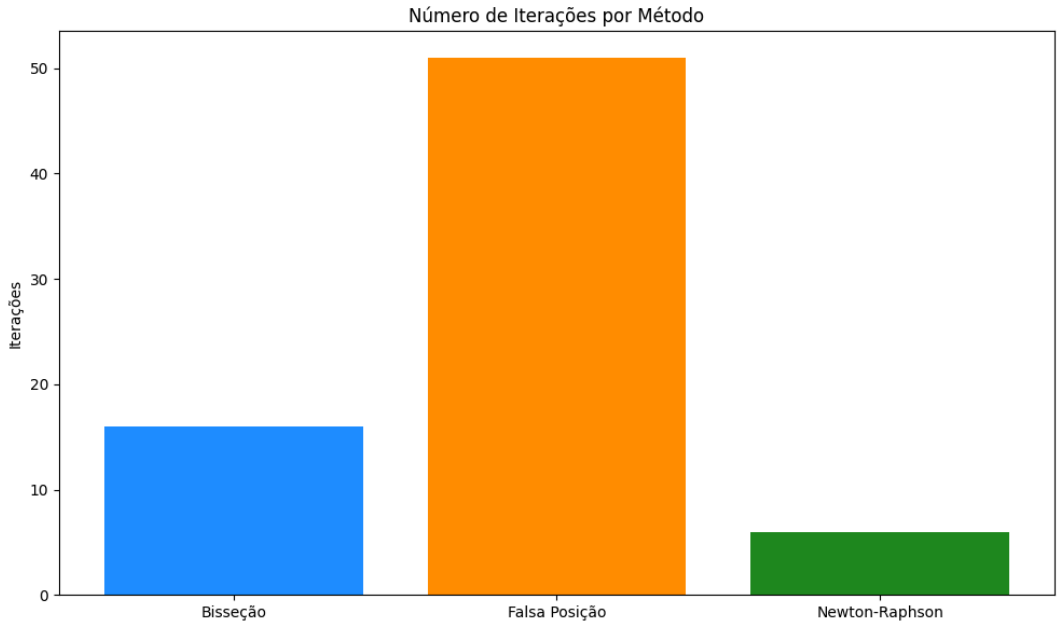
Figura 1 - Gráfico de convergência dos métodos



Fonte: Elaboração Própria usando matplotlib

A figura 1 apresenta um gráfico com a convergência de cada um dos métodos aplicados, mostrando como os valores se aproximam da raiz conforme o número de iterações. No Problema 4, é evidente que o método de Newton-Raphson foi o mais eficiente, seguindo logo depois pelo da Bisseção e por último o da Falsa Posição.

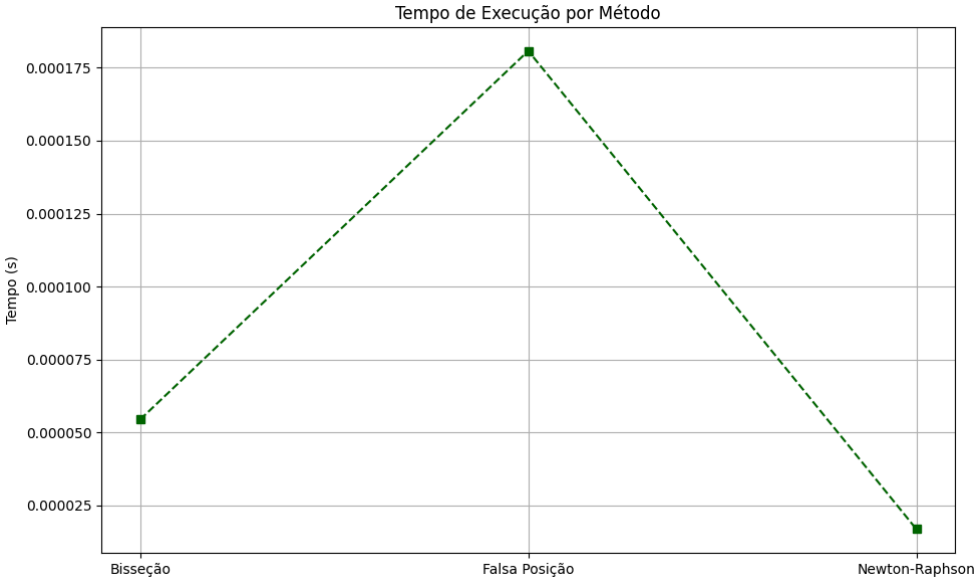
Figura 2 - Gráfico com número de iterações por método



Fonte: Elaboração Própria usando matplotlib

Já a figura 2 mostra um gráfico de barras exibindo a quantidade de iterações necessárias para cada método atingir a sua precisão. Neste problema, o método de Newton-Raphson foi o que menos precisou de iterações para obter o resultado final.

Figura 3 - Gráfico de tempo de execução por método



Fonte: Elaboração Própria usando matplotlib

Por fim, a figura 3 exibe um gráfico que compara o tempo de execução em segundos de cada método. No problema 4, que é o mostrado, verifica-se que o método de Newton-Raphson foi o que executou mais rapidamente.

Analisando os gráficos e a tabela, podemos verificar que no geral, a execução dos três métodos, utilizando os valores padrões de cada exercício, foram capazes de encontrar as raízes corretas com bastante precisão, porém em termos de eficiência, os métodos apresentaram diferenças entre si. O método de Newton mostrou-se como o mais rápido dentre os três sem perder sua precisão, especialmente quando o valor inicial era próximo da raiz, porém um de seus problemas é que é necessário saber qual é a derivada da função executada, que muitas vezes não é possível saber.

Por outro lado, o método da Bisseção se mostrou sendo o mais estável, visto que dificilmente o método não convergia, porém teve um número de iterações significativamente maior. Já o método da Falsa Posição, teve um comportamento

variado, em alguns casos foi mais eficiente que o Bisseção e outras vezes não, necessitando de muitas iterações a mais para conseguir o resultado. O grande problema desses dois métodos é que é necessário saber inicialmente qual é o intervalo que a raiz se encontra.

4. CONCLUSÃO

Com base na implementação e aplicação dos métodos da Bisseção, Falsa Posição e Newton-Raphson e seus resultados obtidos, é possível observar que todos os três métodos são capazes de se obter soluções corretas, mas cada um possui um desempenho diferente. Os resultados reforçam que a escolha do método correto é dependente da análise da função utilizada e o quão bom são os valores iniciais. Assim, percebe-se que a aplicação desses métodos não depende apenas da realização do algoritmo, mas também de uma análise de suas funções e quais serão os chutes iniciais para que se possa garantir a convergência e a execução eficiente do método.