Universidade de Brasília
Departamento de Ciências Mecânica
Programa de Pós-Graduação

Programa 2 – Cálculo de Raízes: Métodos de Bissecção e Falsa Posição

Disciplina: Métodos Numéricos
Professor: Rafael Gabler Gontijo
Data: 3 de abril de 2025

Aluno: Eng. Lucas Wanick — Mestrando em Engenharia Mecânica

Introdução

O presente relatório apresenta o desenvolvimento do Programa 2 da disciplina de Métodos Numéricos, cujo objetivo é implementar os métodos de Bissecção e Falsa Posição para o cálculo de raízes de funções não lineares e transcendentais.

Formulação do Problema

Foram analisadas cinco funções fornecidas no enunciado, representando expressões algébricas com termos polinomiais, logarítmicos e trigonométricos. As raízes foram calculadas dentro de intervalos definidos para cada função.

Metodologia

A equação geral para o número de iterações foi utilizada:

$$n = \left\lfloor \log_2 \left(\frac{x_u - x_l}{\text{tol}} \right) \right\rfloor + 1$$

Como o uso de bibliotecas externas foi vetado, funções como $\ln(x)$ e $\sin(x)$ foram implementadas por meio de séries de Taylor, com técnicas de redução de argumento e mudança de base para garantir estabilidade numérica.

Implementação

A seguir, destacam-se trechos do código implementado.

Função ln(x)

```
def ln_estavel(x, n=100):
    ln2=0.6931471805599453
    if x <= 0:
        raise ValueError("Log␣indefinido␣para␣x␣<=␣0")
    k = 0
    while x > 2:
        x /= 2
        k += 1
    while x < 0.5:
        x *= 2
        k -= 1
    return k * ln2 + ln_taylor(x, n)
```

Função sin(x)

```
def sin_taylor(x, n=20):
    pi = 3.141592653589793
    x = x % (2 * pi)
    if x > pi:
        x -= 2 * pi
    termo = x
    soma = x
    sinal = -1
    for i in range(1, n):
        termo *= x * x / ((2 * i) * (2 * i + 1))
        soma += sinal * termo
        sinal *= -1
    return soma
```

Cálculo das Iterações

```
def calcular_max_iter(xu, xl, tol):
    razao = abs(xu - xl) / tol
    if razao <= 1:
        return 1
    ln2 = 0.6931471805599453
    ln_aprox = ln_estavel(razao, n=100)
    return int(ln_aprox / ln2) + 1
```

Método da Bissecção

```
def bisseccao(f, xl, xu, tol):
    iteracoes = []
    max_iter = calcular_max_iter(xu, xl, tol)
    xr_ant = None
    for i in range(1, max_iter + 1):
        xr = (xl + xu) / 2
        fr = f(xr)
        fl = f(xl)
        ea = abs(xr - xr_ant) if xr_ant is not None else abs(xu - xl
            )
        iteracoes.append((i, xr, fr, ea))
        if fr == 0 or ea < tol:
            break
        elif fl * fr < 0:
            xu = xr
        else:
            xl = xr
        xr_ant = xr
    return xr, iteracoes
```

Resultados

Abaixo, tabela comparativa para a função $f_1(x) = -0.5x^2 + 2.5x + 4.5$ com tolerância 10^{-5}:

Método	Raiz	Iterações	f(Raiz)	Erro final
Bissecção	6.4051342010	19	≈ 0	$\approx 9.5 \times 10^{-6}$
Falsa Posição	6.4051231903	12	$\approx 6 \times 10^{-6}$	$\approx 3.6 \times 10^{-6}$

Tabela 1: Comparação entre métodos para $f_1(x)$

Discussão

Ambos os métodos apresentaram convergência adequada à tolerância estabelecida. O método da falsa posição atingiu a precisão desejada com menos iterações, o que reforça sua eficiência para funções com comportamento suave. A reescrita de funções matemáticas sem bibliotecas não comprometeu a robustez das soluções.

Conclusão

A implementação dos métodos da bissecção e da falsa posição permitiu a obtenção de raízes reais com precisão controlada, conforme os critérios de tolerância estabelecidos. A reescrita das funções matemáticas usuais, como o seno e o logaritmo natural, viabilizou a execução dos algoritmos mesmo sem dependências externas. Os resultados demonstraram convergência consistente, e o número de iterações se manteve

compatível com as estimativas teóricas baseadas na análise logarítmica do intervalo. As diferenças no comportamento entre os métodos ficaram evidentes na comparação tabular, em especial na taxa de convergência. O código foi estruturado de modo a favorecer a análise dos dados e possibilitar reuso em futuras aplicações.