Relatório: Método do Ponto Fixo

Marcelo Roner
Universidade Federal de Goiás

25 de fevereiro de 2025

Sumário

1 Introdução

A busca por soluções de equações não lineares $f(x) = 0$ é parte fundamental de inúmeros problemas em engenharia, física, economia e outras áreas. Dentre as diversas técnicas numéricas existentes, o *Método do Ponto Fixo* oferece uma abordagem relativamente simples, mas ilustrativa, para entender o conceito de iteração. A ideia central consiste em reformular a equação $f(x) = 0$ na forma $x = g(x)$, de modo que a raiz do problema seja um ponto fixo da função g.

Este relatório descreve o Método do Ponto Fixo, aborda a teoria de convergência, apresenta sua implementação computacional em Python e discute os resultados obtidos a partir de duas variações do método.

2 Fundamentação Teórica

Seja o problema de determinar x^* tal que $f(x^*) = 0$. Se escrevemos a equação como

$$x = g(x), \tag{1}$$

então x^* será um *ponto fixo* de g, ou seja, $g(x^*) = x^*$. Para construir uma sequência $(x_n)_{n\geq 0}$, definimos

$$x_{n+1} = g(x_n). \tag{2}$$

Condição de Convergência Local: Se existe um intervalo $[a, b]$ contendo x^* tal que $g(x)$ mapeia $[a, b]$ em si mesmo e

$$\max_{x\in[a,b]} |g'(x)| < 1, \tag{3}$$

então a sequência $x_{n+1} = g(x_n)$ converge para o ponto fixo x^*. Intuitivamente, a inclinação de g deve ser pequena o suficiente (em módulo) para que o processo iterativo seja contrativo.

2.1 Análise de Erro

Seja $e_n = x_n - x^*$ o erro na n-ésima iteração. Supondo g derivável em x^*, podemos fazer uma expansão de Taylor:

$$e_{n+1} = x_{n+1} - x^* = g(x_n) - g(x^*) \approx g'(x^*)\,(x_n - x^*) = g'(x^*)\,e_n.$$

Assim, $|e_{n+1}| \approx |g'(x^*)|\,|e_n|$. Portanto, se $|g'(x^*)| < 1$, o erro tende a diminuir geometricamente a cada iteração.

3 Métodos Aplicados

Neste trabalho, comparamos duas formas de definir $g(x)$, partindo da mesma função original $f(x)$:

- **Método 2**: Definimos

  $$g(x) = x - \lambda\, f(x), \tag{4}$$

  onde λ é um *parâmetro de relaxação* (ou *step size*), escolhido para tentar satisfazer $|g'(x)| < 1$ em um dado intervalo.

- **Método 3**: Baseia-se em
$$g(x) = x - \frac{f(x)}{f'(x)}, \tag{5}$$
o que é essencialmente o Método de Newton reformulado como um ponto fixo.

O **Método 2** é mais simples e não requer derivadas de f, mas pode ser sensível à escolha de λ. Já o **Método 3** (semelhante ao Newton) tende a apresentar convergência mais rápida, embora dependa do cálculo de $f'(x)$ e, em alguns casos, possa exigir mais cautela perto de pontos onde $f'(x) = 0$.

4 Implementação Computacional

A seguir, apresentamos um trecho de código em Python empregando as bibliotecas *SymPy* (para manipulação simbólica) e *Matplotlib* (para geração de gráficos). O algoritmo faz:

- Verificação da condição de convergência aproximada (3).
- Execução da iteração de ponto fixo conforme a escolha do método.
- Geração de gráficos comparando o número de iterações e a evolução dos erros.

```python
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

# Definição da função e intervalo
x = sp.symbols('x')
f_expr = sp.exp(-x) - x
intervalo = (0, 2)
tol = 1e-6
max_iter = 100

# Derivada da função para o Método 3
f_prime = sp.diff(f_expr, x)

# Lista de valores de lambda para o Método 2
lambdas = np.linspace(-2, 2, 40)
lambdas = lambdas[lambdas != 0]

# Função para verificar convergência
# Retorna True se |g'(x)| < 1 no intervalo, e g(x) permanece no intervalo

def verificar_convergencia(g, x, intervalo):
    a, b = intervalo
    g_prime = sp.diff(g, x)

    try:
        g_prime_func = sp.lambdify(x, sp.Abs(g_prime), modules=['numpy'])
    except:
        return False

    x_vals = np.linspace(a + 1e-6, b - 1e-6, 1000)
    try:
        g_prime_vals = g_prime_func(x_vals)
        if np.all(np.isfinite(g_prime_vals)) and np.max(g_prime_vals) < 1:
```

```
            g_func = sp.lambdify(x, g, modules=['numpy'])
            g_vals = g_func(x_vals)
            if np.all((g_vals >= a) & (g_vals <= b)) and np.all(np.isfinite(g_vals)):
                return True
        else:
            return False
    except:
        return False
    return False

# Método do ponto fixo

def ponto_fixo(g_func, x0, tol=1e-6, max_iter=100):
    x_n = x0
    iteracoes = [x_n]
    for _ in range(max_iter):
        x_next = g_func(x_n)
        iteracoes.append(x_next)
        if abs(x_next - x_n) < tol:
            return x_next, iteracoes, True
        x_n = x_next
    return x_n, iteracoes, False

# Aplicação do Método 2

def aplicar_ponto_fixo_metodo2(f_expr, intervalo, x0=None, tol=1e-6, max_iter=100):
    x = sp.symbols('x')
    resultados = []
    for lam in lambdas:
        g_expr = x - lam * f_expr
        if verificar_convergencia(g_expr, x, intervalo):
            g_func = sp.lambdify(x, g_expr, modules=['numpy'])
            if x0 is None:
                x0 = (intervalo[0] + intervalo[1]) / 2
            raiz, iteracoes, convergiu = ponto_fixo(g_func, x0, tol=tol,
            ↪  max_iter=max_iter)
            num_iter = len(iteracoes) - 1
            resultados.append({'lambda': lam, 'raiz': raiz, 'iteracoes': iteracoes,
                               'num_iter': num_iter, 'convergiu': convergiu})
        else:
            resultados.append({'lambda': lam, 'raiz': None, 'iteracoes': None,
                               'num_iter': None, 'convergiu': False})
    return resultados

# Aplicação do Método 3

def aplicar_ponto_fixo_metodo3(f_expr, intervalo, x0=None, tol=1e-6, max_iter=100):
    x = sp.symbols('x')
    g_expr = x - f_expr / f_prime
    if verificar_convergencia(g_expr, x, intervalo):
        g_func = sp.lambdify(x, g_expr, modules=['numpy'])
        if x0 is None:
            x0 = (intervalo[0] + intervalo[1]) / 2
        raiz, iteracoes, convergiu = ponto_fixo(g_func, x0, tol=tol,
        ↪  max_iter=max_iter)
        num_iter = len(iteracoes) - 1
        return {'raiz': raiz, 'iteracoes': iteracoes, 'num_iter': num_iter,
        ↪  'convergiu': convergiu}
```

```python
    else:
        return None

# Comparação entre os métodos
resultados_metodo2 = aplicar_ponto_fixo_metodo2(f_expr, intervalo)
resultado_metodo3 = aplicar_ponto_fixo_metodo3(f_expr, intervalo)

lambdas_validos = []
num_iters_metodo2 = []
for res in resultados_metodo2:
    if res['convergiu']:
        lambdas_validos.append(res['lambda'])
        num_iters_metodo2.append(res['num_iter'])

# Gráfico de comparação (número de iterações vs lambda)
plt.figure(figsize=(10, 5))
plt.plot(lambdas_validos, num_iters_metodo2, 'o-', label='Método 2')
if resultado_metodo3:
    plt.axhline(y=resultado_metodo3['num_iter'], color='r', linestyle='--', label='Método 3')
plt.xlabel('Lambda')
plt.ylabel('Número de Iterações')
plt.title('Número de Iterações até Convergência')
plt.legend()
plt.grid(True)
plt.show()

# Análise detalhada para o melhor lambda do Método 2
if resultado_metodo3:
    best_result_metodo2 = min(resultados_metodo2, key=lambda r: r['num_iter'] if r['convergiu'] else max_iter)
    iteracoes_metodo2 = best_result_metodo2['iteracoes']
    erros_metodo2 = [abs(iteracoes_metodo2[i+1] - iteracoes_metodo2[i]) for i in range(len(iteracoes_metodo2)-1)]

    iteracoes_metodo3 = resultado_metodo3['iteracoes']
    erros_metodo3 = [abs(iteracoes_metodo3[i+1] - iteracoes_metodo3[i]) for i in range(len(iteracoes_metodo3)-1)]

    # Plotagem comparativa
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    # (a) Convergência dos métodos
    axes[0].plot(range(len(iteracoes_metodo2)), iteracoes_metodo2, marker='o', linestyle='-',
                 label=f"Método 2 ( = {best_result_metodo2['lambda']:.2f})")
    axes[0].plot(range(len(iteracoes_metodo3)), iteracoes_metodo3, marker='s', linestyle='-', label='Método 3')
    axes[0].set_title('Evolução das Aproximações')
    axes[0].set_xlabel('Iteração')
    axes[0].set_ylabel('Aproximação de x')
    axes[0].grid(True)
    axes[0].legend()

    # (b) Erro absoluto em escala log
    axes[1].semilogy(range(1, len(erros_metodo2) + 1), erros_metodo2, marker='o', linestyle='-',
                     label=f"Método 2 ( = {best_result_metodo2['lambda']:.2f})")
```

```
    axes[1].semilogy(range(1, len(erros_metodo3) + 1), erros_metodo3, marker='s',
    ↪  linestyle='-', label='Método 3')
    axes[1].set_title('Erro Absoluto')
    axes[1].set_xlabel('Iteração')
    axes[1].set_ylabel('Erro')
    axes[1].grid(True, which='both', ls='--')
    axes[1].legend()

    plt.tight_layout()
    plt.show()
```

5 Resultados e Discussão

Para o exemplo $f(x) = e^{-x} - x$, empregamos o intervalo $(0, 2)$ e variamos λ no Método 2 entre -2 e 2. A Figura 1 exibe o número de iterações necessário para a convergência em função de λ. Já as Figuras 2-(a) e 2-(b) comparam a evolução das aproximações e o erro absoluto (em escala logarítmica) entre o melhor caso do Método 2 e o Método 3:

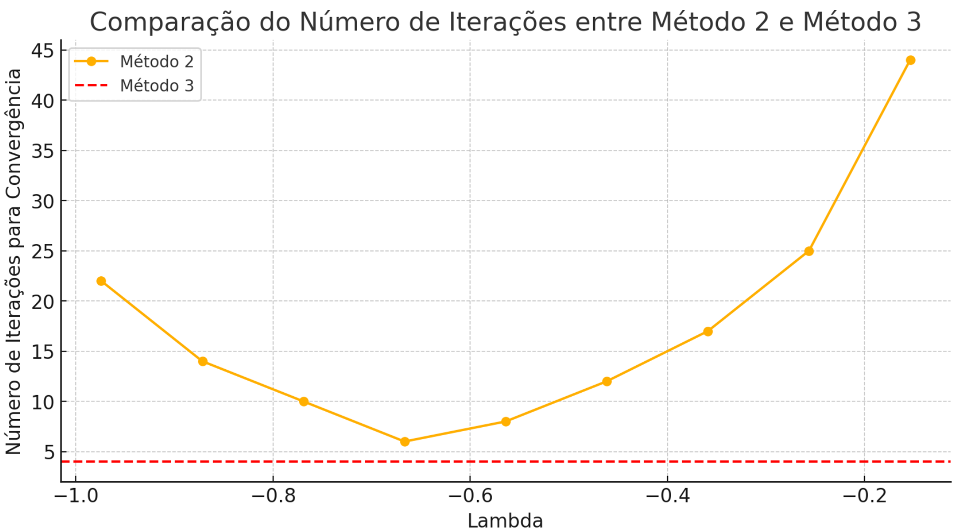


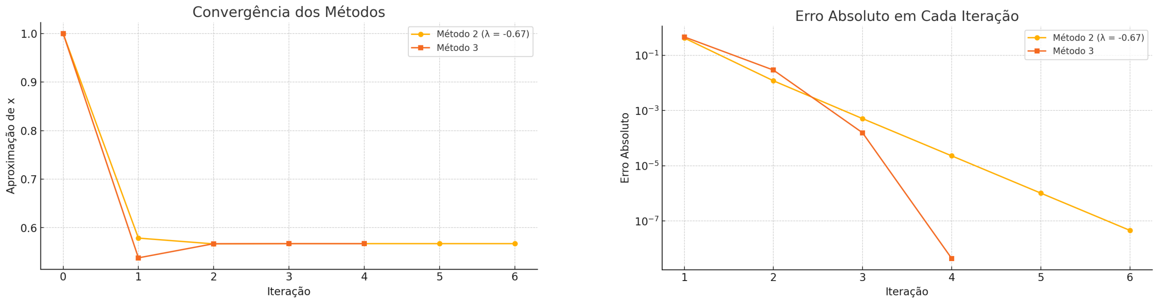
Figura 1: Número de iterações para convergência em função de λ (Método 2). A linha tracejada vermelha representa o número de iterações do Método 3, que não depende de λ.

Observamos que o Método 3 (inspirado no Método de Newton) convergiu mais rápido e, sobretudo, de maneira independente do parâmetro λ. Por outro lado, o Método 2 pode atingir convergência satisfatória se λ for bem escolhido, mas exibe forte sensibilidade a esse valor.

6 Conclusão

O **Método do Ponto Fixo** oferece uma base importante para compreender métodos iterativos na resolução de equações não lineares. A forma da função $g(x)$ determina a robustez e a velocidade de convergência do processo. Em termos de aplicação:

- O **Método 2** $\big($Equação (4)$\big)$ exige apenas a avaliação de $f(x)$, mas a escolha do parâmetro λ é crucial para garantir $|g'(x)| < 1$ e um bom desempenho.



(a) Evolução das aproximações (x_n).

(b) Erro absoluto em cada iteração.

Figura 2: Comparação entre o melhor caso do Método 2 e o Método 3. (a) Evolução de x_n ao longo das iterações. (b) Decaimento do erro absoluto em escala logarítmica.

- O **Método 3** $\big($Equação (5)$\big)$ costuma apresentar convergência mais rápida (semelhante ao Método de Newton), porém depende do cálculo de $f'(x)$ e pode falhar se $f'(x^*) \approx 0$ ou se a escolha inicial x_0 não estiver em uma região de convergência.

Para problemas de grande porte ou funções mais complexas, combinações de estratégias (como *line search* ou *regularizações*) podem ser necessárias. Ainda assim, a análise mostrada aqui ilustra como pequenas variações em $g(x)$ podem produzir comportamentos distintos de convergência.

7 Referências

Referências

[1] Jaan Kiusalaas, *Numerical Methods in Engineering with Python*, Cambridge University Press, 2005.

[2] N. B. Franco, *Cálculo Numérico*, Pearson Prentice Hall, 2007.

[3] S. Lynch, *Dynamical Systems with Applications Using Python*, Springer, 2018.

[4] M. A. G. Ruggiero, V. L. Lopes, *Cálculo Numérico: aspectos teóricos e computacionais*, Makron Books, 1997.