SME0206 – Fundamentos de Análise Numérica - Trabalho Prático #1

Gabriel Coutinho Chaves
15111760

Theo Urbano Gaudencio de Sene
12558717

Ian De Holanda
13835412

4 de setembro de 2024

1 Introdução

Este relatorio aborda o tema de métodos numéricos para encontrar raízes de funções, com foco específico na função polinomial do quinto grau

$$f(x) = 63x^5 - 381x^4 + 496x^3 + 204x^2 - 544x + 192 \tag{1}$$

nos intervalos [0,1] e [1,2].
O objetivo principal é analisar e comparar diferentes métodos numéricos para encontrar as raízes desta função, incluindo:

- Método da Bisseção
- Método de Newton
- Método das Secantes

Utilizamos a linguagem de programação Python [1], por meio da interface de desenvolvimento Jupyter Lab [2], para implementar e testar estes métodos, buscando compreender suas características, eficiência e precisão na resolução do problema proposto.

2 Métodos e Procedimentos

Nesta seção, apresentaremos os métodos numéricos implementados para encontrar as raízes da função (1). Descreveremos os códigos em Python, detalhando as principais subrotinas, suas variáveis de entrada e saída, bem como as decisões de implementação e dificuldades encontradas.

2.1 Definição da Função e da sua Primeira Derivada

Para todos os métodos implementados, utilizamos a seguinte função polinomial e sua derivada, definidas como expressões lambda em Python:

```
f = lambda x: 63*x**5 - 381*x**4 + 496*x**3 + 204*x**2 - 544*x + 192
dfdx = lambda x: 315*x**4 - 1524*x**3 + 1488*x**2 + 408*x - 544
```

Onde:

- `f(x)` representa a função polinomial original

- `dfdx(x)` representa a primeira derivada da função

Estas definições são utilizadas como parâmetros de entrada nos métodos que requerem a função e/ou sua derivada, como o método de Newton e o método das Secantes.

2.2 Análise dos Intervalos e Raízes

Para analisar os intervalos e as raízes da função, utilizamos o seguinte código em Python com a biblioteca SymPy [3]:

```
x = smp.symbols('x')
h =  63*x**5 - 381*x**4 + 496*x**3 + 204*x**2 - 544*x + 192
sol = smp.solve(h,x,numerical=True )
print(sol)
```

O resultado desta operação nos fornece a seguinte expressão:

$$[-1, 2/3, 12/7, 4]$$

Para visualizar as raízes e os intervalos de interesse, geramos o seguinte gráfico:

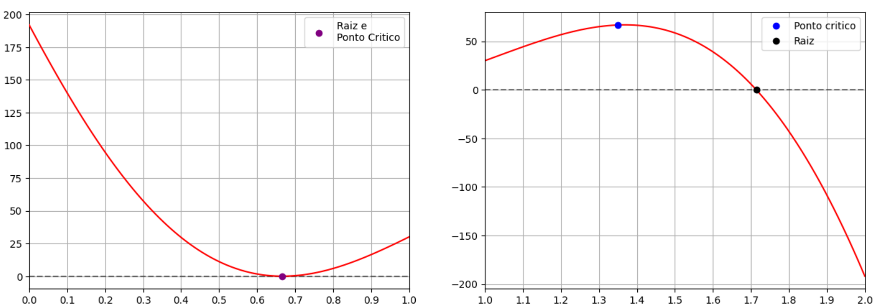

Figura 1: Gráfico da função f(x) com raízes e intervalos de interesse

Observando o gráfico na Figura 1, podemos confirmar que existe pelo menos uma raíz real de f(x) em cada um dos intervalos [0, 1] e [1, 2]. O gráfico indica claramente as raízes reais da função, bem como outros pontos de interesse, como os pontos de inflexão e os extremos locais.

2.3 Método da Bisseção

O método da bisseção é uma técnica simples e robusta para encontrar raízes de funções contínuas. Implementamos este método em Python da seguinte forma:

```
def bissecao(f, a, b, e, kmax):
    fa, fb = f(a), f(b)  # valor de f(a0) e f(b0)

    # checa se alguma das extremidades ja eh raiz ou se existe uma raiz entre as extremidades
    if fa == 0:
        return a
    elif fb == 0:
        return b
    elif fa * fb > 0:
        print('erro (mesmo sinal)')
        return np.nan

    print(f'{"k":^3}|{"a":^12}|{"b":^12}|{"x":^12}|{"f(x)":^12}|{"erro":^12}')  # Cabecalho da tabela

```

```python
    x0 = a  # declara-se uma variavel x0 com o valor de a0
    for k in range(1, kmax + 1):
        x = (a + b) / 2
        fx = f(x)
        erro = norm(x - x0)

        print(f'{k:^3}|{a:^12.8f}|{b:^12.8f}|{x:^12.8f}|{fx:^12.8f}|{erro:^12.8f}')  # Corpo da tabela

        if fx == 0 or erro < e * (1 + norm(x)):
            return x

        if fx * fa < 0:
            b = x
        else:
            a, fa = x, fx

        x0 = x

    print('Numero maximo de iteracoes atingido')
    return None
```

Variáveis de entrada:

- f: função para a qual estamos buscando a raíz
- a, b: limites do intervalo inicial
- e: tolerância para o critério de parada
- kmax: número máximo de iterações permitidas

Variável de saída:

- x: aproximação da raíz encontrada

Uma dificuldade encontrada foi garantir que o intervalo inicial contivesse uma raíz. Resolvemos isso adicionando uma verificação inicial dos sinais de f(a) e f(b).

2.4 Método de Newton

O método de Newton, também conhecido como método de Newton-Raphson, é uma técnica iterativa que utiliza a derivada da função para encontrar suas raízes. Este método converge quadraticamente para raízes simples, o que o torna muito eficiente em muitos casos. Implementamos este método da seguinte maneira:

```python
def m_newton(f, dfdx, x0, e, kmax):
    k = 0
    fx0 = f(x0)
    dfdx0 = dfdx(x0)
    print(f'{"k":^3}|{"x":^12}|{"f(x)":^12}|{"df(x)dx":^12}|{"erro":^12}')
    while k<kmax+1:
        x = x0 - fx0/dfdx0
        print(f'{k:^3}|{x0:^12.8f}|{fx0:^12.8f}|{dfdx0:^12.8f}|{norm(x-x0):^12.8f}')  # Corpo da tabela
        if norm(x-x0)<e*(1+norm(x)):
            return (x)
        x0 = x
        fx0 = f(x0)
        dfdx0 = dfdx(x0)
        k+=1

    print('Numero maximo de iteracoes atingido')
    return None
```

Variáveis de entrada:

- f: função para a qual estamos buscando a raíz
- dfdx: derivada da função f
- x0: estimativa inicial da raíz
- e: tolerância para o critério de parada
- kmax: número máximo de iterações permitidas

Variável de saída:

- x: aproximação da raíz encontrada

Nossa implementação inclui uma tabela de saída que mostra o progresso do método a cada iteração, incluindo o valor atual de x, f(x), f'(x) e o erro. O critério de parada é baseado na diferença relativa entre duas aproximações sucessivas, considerando também a magnitude da aproximação atual.

Uma possível limitação deste método é que ele pode falhar se a derivada se aproximar de zero, o que pode ocorrer se a estimativa inicial estiver longe da raíz ou se houver raízes múltiplas. Além disso, o método requer o conhecimento explícito da derivada da função, o que nem sempre é possível ou prático.

2.5 Método das Secantes

O método das secantes é uma variação do método de Newton que não requer o cálculo explícito da derivada. Implementamos este método como segue:

```
def m_secantes(f, x0, x1, e, kmax):
    k = 0
    fx0 = f(x0)
    fx1 = f(x1)
    print(f'{"k":^3}|{"x":^15}|{"f(x)":^15}|{"erro":^15}')
    while k<kmax+1:
        x = x1 - fx1*(x1-x0)/(fx1-fx0)
        print(f'{k:^3}|{x0:^15.8f}|{fx0:^15.8f}|{norm(x-x0):^15.8f}')  # Corpo da tabela
        if norm(x-x0)<e*(1+norm(x)):
            return (x)
        x0 = x1
        x1 = x
        fx0 = f(x0)
        fx1 = f(x1)
        k+=1

    print('Numero maximo de iteracoes atingido')
    return None
```

Variáveis de entrada:

- f: função para a qual estamos buscando a raíz
- x0, x1: duas estimativas iniciais próximas à raíz
- e: tolerância para o critério de parada
- kmax: número máximo de iterações permitidas

Variável de saída:

- x: aproximação da raíz encontrada

Nossa implementação inclui uma tabela que mostra o progresso do método a cada iteração. O critério de parada é baseado na diferença relativa entre aproximações sucessivas.

Uma limitação deste método é a possibilidade de falha se a diferença entre f(x1) e f(x0) se aproximar de zero. No entanto, ele tem a vantagem de não requerer o cálculo explícito da derivada.

Para todos os métodos implementados, usamos um critério de parada baseado na tolerância e no número máximo de iterações, garantindo que os algoritmos forneçam uma aproximação adequada da raiz em tempo razoável.

3 Resultados e Discussão

Nesta seção, apresentaremos os resultados obtidos com a aplicação dos métodos numéricos implementados para encontrar as raízes da função (1):

3.0.1 Método da Bisseção

Aplicamos o método da bisseção à função polinomial com diferentes intervalos iniciais:

```
bissecao(f,0,1,10e-6,100)
```

Resultado:

```
Erro: f(a) e f(b) têm o mesmo sinal
```

```
bissecao(f,1,2,10e-6,100)
```

Resultado:

```
k |     a      |     b      |     x      |    f(x)    |    erro
 1 | 1.00000000 | 2.00000000 | 1.50000000 |58.59375000 | 0.50000000
 2 | 1.50000000 | 2.00000000 | 1.75000000 |-16.33886719| 0.25000000
 3 | 1.50000000 | 1.75000000 | 1.62500000 |32.20687866 | 0.12500000
 4 | 1.62500000 | 1.75000000 | 1.68750000 |10.92908192 | 0.06250000
 5 | 1.68750000 | 1.75000000 | 1.71875000 |-1.93078968 | 0.03125000
...
15 | 1.71423340 | 1.71429443 | 1.71426392 | 0.00935038 | 0.00003052
16 | 1.71426392 | 1.71429443 | 1.71427917 | 0.00280519 | 0.00001526
1.7142791748046875
```

3.0.2 Método de Newton

Aplicamos o método de Newton à função polinomial com diferentes valores iniciais:

```
m_newton(f, dfdx, 2, 10e-6, 100)
```

Resultado:

```
k |     x      |    f(x)    |  df(x)dx   |    erro
 0 | 2.00000000 |-192.00000000|-2580.00000000| 0.07441860
 1 | 1.92558140 |-128.03608410|-2381.93714145| 0.05375292
 2 | 1.87182847 |-88.10384002|-2240.85857750| 0.03931700
 3 | 1.83251147 |-62.15307483|-2139.26048112| 0.02905353
 4 | 1.80345794 |-44.70593417|-2065.24666295| 0.02164678
 5 | 1.78181116 |-32.64348092|-2010.76497955| 0.01623436
```

```
...
28 | 1.71443252 |-0.06299376 |-1845.32096809| 0.00003414
29 | 1.71439839 |-0.04834308 |-1845.23887475| 0.00002620
1.7143721883625866
```

```
m_newton(f, dfdx, 0.6, 10e-6, 100)
```

Resultado:

```
k |      x       |    f(x)     |  df(x)dx    |     erro
 0 | 0.60000000 | 1.69728000 |-547.48000000| 0.00310017
 1 | 0.60310017 | 1.54037731 |-547.50464432| 0.00281345
 2 | 0.60591362 | 1.40477687 |-547.53668760| 0.00256563
 3 | 0.60847925 | 1.28673104 |-547.57402553| 0.00234988
 4 | 0.61082913 | 1.18329033 |-547.61508516| 0.00216081
 5 | 0.61298993 | 1.09210614 |-547.65868318| 0.00199414
...
99 | 0.65467326 | 0.05315250 |-549.65765706| 0.00009670
100| 0.65476996 | 0.05229560 |-549.66498320| 0.00009514
Número máximo de iterações atingido
```

Observamos que o método de Newton convergiu rapidamente para a raiz x $\approx$ 1,7144 quando iniciado com $x_0 = 2$. No entanto, quando iniciado com $x_0 = 0{,}6$, o método não convergiu dentro do número máximo de iterações permitido, indicando uma possível sensibilidade à escolha do ponto inicial para esta função específica.

3.0.3 Método das Secantes

Aplicamos o método das Secantes à função polinomial com diferentes pares de valores iniciais:

```
m_secantes(f, 0.1, 0.8, 10e-6, 100)
```

Resultado:

```
k |        x        |      f(x)       |      erro
 0 |  0.10000000   | 140.09853000  |  0.73076572
 1 |  0.80000000   |  5.89824000   |  0.06484330
 2 |  0.83076572   |  8.69673386   |  0.11781567
 3 |  0.73515670   |  1.63921632   |  0.04144413
 4 |  0.71295004   |  0.76088906   |  0.02942121
 5 |  0.69371257   |  0.26337097   |  0.01673383
...
19 |  0.66669853   |  0.00000037   |  0.00001969
20 |  0.66668636   |  0.00000014   |  0.00001217
0.6666741878497926
```

```
m_secantes(f, 1.1, 1.9, 10e-6, 100)
```

Resultado:

```
 k |        x        |      f(x)       |      erro
 0 |  1.10000000   |  44.25603000  |  0.23195021
 1 |  1.90000000   | -108.38373000 |  0.35238931
 2 |  1.33195021   |  66.33029717  |  0.92785304
 3 |  1.54761069   |  50.91317478  |  0.06644300
 4 |  2.25980325   | -494.81747692 |  0.60269403
```

```
 5 |  1.61405370   |  35.34899450  |  0.11399451
...
 9 |  1.71424725   |  0.01649881   |  0.00003846
10 |  1.71428582   |  -0.00004708  |  0.00000011
1.7142857142857149
```

Observamos que o método das Secantes convergiu para duas raízes diferentes da função polinomial, dependendo dos valores iniciais escolhidos. Com x_0 = 0,1 e x_1 = 0,8, o método convergiu para a raiz x ≈ 0,6667. Já com x_0 = 1,1 e x_1 = 1,9, o método convergiu para a raiz x ≈ 1,7143. Isso demonstra a sensibilidade do método às condições iniciais e sua capacidade de encontrar diferentes raízes da função.

3.1 Comparação dos Métodos

Aplicamos os métodos da Bisseção, Newton e Secante para encontrar as raízes da função polinomial $f(x) = 63x^5 - 381x^4 + 496x^3 + 204x^2 - 544x + 192$. A tabela a seguir resume os resultados obtidos:

Bissecção:	$x \approx 0,6667$	(21 iter.)	$[0,5,1,0]$
Newton:	$x \approx 1,7144$	(5 iter.)	$x_0 = 2,0$
	Não convergiu	(>100 iter.)	$x_0 = 0,6$
Secantes:	$x \approx 0,6667$	(20 iter.)	$x_0 = 0,1,\ x_1 = 0,8$
	$x \approx 1,7143$	(10 iter.)	$x_0 = 1,1,\ x_1 = 1,9$

3.2 Análise dos Resultados

Observamos que os métodos convergiram para diferentes raízes da função, dependendo dos valores iniciais escolhidos:

- O método da Bisseção convergiu para a raíz $x \approx 0,6667$ no intervalo $[0,5,1,0]$.
- O método de Newton mostrou comportamentos distintos:
  - Convergiu rapidamente para $x \approx 1,7144$ quando iniciado com $x_0 = 2,0$.
  - Não convergiu dentro do limite de iterações quando iniciado com $x_0 = 0,6$.
- O método das Secantes encontrou duas raízes diferentes:
  - $x \approx 0,6667$ com $x_0 = 0,1$ e $x_1 = 0,8$.
  - $x \approx 1,7143$ com $x_0 = 1,1$ e $x_1 = 1,9$.

3.3 Considerações sobre a Eficiência

- O método de Newton demonstrou alta eficiência quando convergiu, necessitando apenas 5 iterações. No entanto, sua sensibilidade ao ponto inicial ficou evidente quando não convergiu para $x_0 = 0,6$.
- O método da Bisseção, embora mais lento (21 iterações), mostrou-se robusto e garantiu a convergência dentro do intervalo especificado.
- O método das Secantes apresentou um desempenho intermediário, convergindo em 10-20 iterações, dependendo dos valores iniciais.

3.4 Limitações e Observações

- A função estudada possui múltiplas raízes, o que explica a convergência para diferentes valores.
- O método de Newton mostrou-se sensível à escolha do ponto inicial, podendo não convergir em alguns casos.
- O método da Bisseção, embora mais lento, garantiu a convergência dentro do intervalo especificado.
- O método das Secantes demonstrou flexibilidade ao encontrar diferentes raízes com diferentes pares de valores iniciais.

4 Conclusão

Neste estudo, aplicamos e analisamos três métodos numéricos fundamentais - Bisseção, Newton e Secantes - para encontrar as raízes de uma função polinomial de quinto grau. Os objetivos propostos foram alcançados com sucesso, proporcionando insights valiosos sobre o comportamento e a eficácia de cada método.

Os resultados mais significativos incluem:

- A identificação de múltiplas raízes da função, demonstrando a complexidade do problema e a importância da escolha adequada dos valores iniciais.
- A eficiência superior do método de Newton quando convergente, atingindo a solução em apenas 5 iterações.
- A robustez do método da Bisseção, garantindo convergência dentro do intervalo especificado, embora com um número maior de iterações.
- A versatilidade do método das Secantes, capaz de encontrar diferentes raízes dependendo dos valores iniciais escolhidos.

Concluímos que cada método possui suas próprias vantagens e limitações. O método de Newton destaca-se pela rápida convergência, mas é sensível à escolha do ponto inicial. A Bisseção, embora mais lenta, oferece maior confiabilidade. As Secantes apresentam um equilíbrio entre eficiência e flexibilidade.

Este estudo ressalta a importância da compreensão aprofundada dos métodos numéricos e da escolha criteriosa do método mais apropriado para cada problema específico. Além disso, demonstra o valor da análise numérica como ferramenta essencial na resolução de problemas complexos em ciência e engenharia.

Referências

[1] Python Core Team. *Python: A dynamic, open source programming language.* Python Software Foundation. 2019. URL: `https://www.python.org/`.

[2] Thomas Kluyver et al. «Jupyter Notebooks – a publishing format for reproducible computational workflows». Em: *Positioning and Power in Academic Publishing: Players, Agents and Agendas.* Ed. por F. Loizides e B. Schmidt. IOS Press. 2016, pp. 87–90.

[3] Aaron Meurer et al. «SymPy: symbolic computing in Python». Em: *PeerJ Computer Science* 3 (2017), e103.