Universidade Estadual de Maringá

Avaliação de Precisão e Erros Computacionais - Parte 2
Polinômios de Chebyshev e Método de Newton-Raphson

Gabriel Bitdinger Medeiros - 118542
Peter Mundadi - 116338

Matemática Computacional

28 de março de de 2024

Sumário

1 Introdução

O presente relatório diz respeito aos trabalhos da disciplina de Matemática Computacional que teve como objetivo a análise de duas técnicas de matemática computacional apresentadas em aula: A primeira para o cálculo de $\sin x$ e $\cos x$ utilizando polinômios de Chebyshev. A segunda para o cálculo de $\sqrt{x}$ utilizando o método de Newton-Raphson, comparando dois diferentes chutes iniciais.

1.1 Objetivos

- Fazer uma breve fundamentação teórica
- Implementar algoritmos para exemplificar as diferentes técnicas estudadas
- Comparar a precisão das técnicas implementadas com o valor real

1.2 Componentes do Trabalho

- Código fonte do script python que simula o método de Newton-Raphson para o cálculo de $\sqrt{x}$: RaizquadNR.py
- Código fonte do script python que simula a técnica de cálculo de $\sin x$ e $\cos x$ utilizando polinômios de Chebyshev: Sencos.py
- Este presente relatório explicando o raciocínio por trás da implementação e a análise dos resultados

2 Método de Newton-Raphson

2.1 Fundamentação Teórica

O método de Newton (ou Método de Newton–Raphson), desenvolvido por Isaac Newton e Joseph Raphson, tem o objetivo de estimar as raízes de uma função. Para isso, escolhe-se uma aproximação inicial para esta. Após isso, calcula-se a equação da reta tangente (por meio da derivada) ao gráfico da função nesse ponto e a interseção dela com o eixo das abcissas, a fim de encontrar uma melhor aproximação para a raiz. Repetindo-se o processo, cria-se um método iterativo para encontrarmos a raiz da função. Em notação matemática, o método de Newton é dado pela seguinte sequência recursiva:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1}$$

Onde x_k é a aproximação inicial, $f(x_k)$ é o valor da função no ponto x_k e $f'(x_k)$ é a derivada da função no ponto x_k. O método de Newton é um método iterativo, e a convergência para a raiz é garantida se a derivada da função for contínua e não nula no intervalo de interesse, e se a aproximação inicial for suficientemente próxima da raiz.

No nosso caso, utilizaremos o método de Newton-Raphson para calcular a raiz quadrada de um número x. A função $f(x)$ que queremos encontrar a raiz é dada por $f(x) = x^2 - a$, onde a é o número do qual queremos calcular a raiz quadrada. A derivada de $f(x)$ é dada por $f'(x) = 2x$. Substituindo esses valores na equação (1), obtemos a seguinte equação iterativa para o método de Newton-Raphson:

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} \tag{2}$$

2.2 Implementação

O método de Newton-Raphson é um método iterativo que precisa assumir um x_0 inicial para executar.

A partir desta definição, organizamos scripts (que lembrando, estão completos em anexo nesta pasta), que comparam os resultados do chute inicial criado a partir do algoritmo implementado na parte 1 do trabalho, com um chute realizado pelo método logarítmico da ieee apresentado em sala.

A leitura do código completo faz-se necessária para melhor entendimento. Aqui estão presentes apenas as funções principais.

```
def mySqrt(A,X0,Parada): # usando o trab 1

    Xk = X0
    '''
    print("---------------")
    print("    MySqrt")
    print("    A: ", A)
    print("    X0: ", X0)
    print("    Parada: ", Parada)
    print("---------------")'''
    iter = 1
    #while True:
    for _ in range(100):

        X_prox_k =0.5 * (Xk + A/Xk)
        #print("X_prox_k: ", X_prox_k)

        if abs(X_prox_k - Xk) < Parada:

            return X_prox_k, iter

        Xk = X_prox_k
        iter += 1

    #print(" Iteraes  necessrias: ", iter)


def sqrt_log(A,Parada):

    Xk = math.exp(0.5 * math.log(A))
    #print("Chute log: ", Xk)
    iter = 1
```

```
for _ in range(100):

    X_prox_k =0.5 * (Xk + A/Xk)
    #print("X_prox_k: ", X_prox_k)

    if abs(X_prox_k - Xk) < Parada:
        return X_prox_k, iter

    Xk = X_prox_k
    iter += 1
```

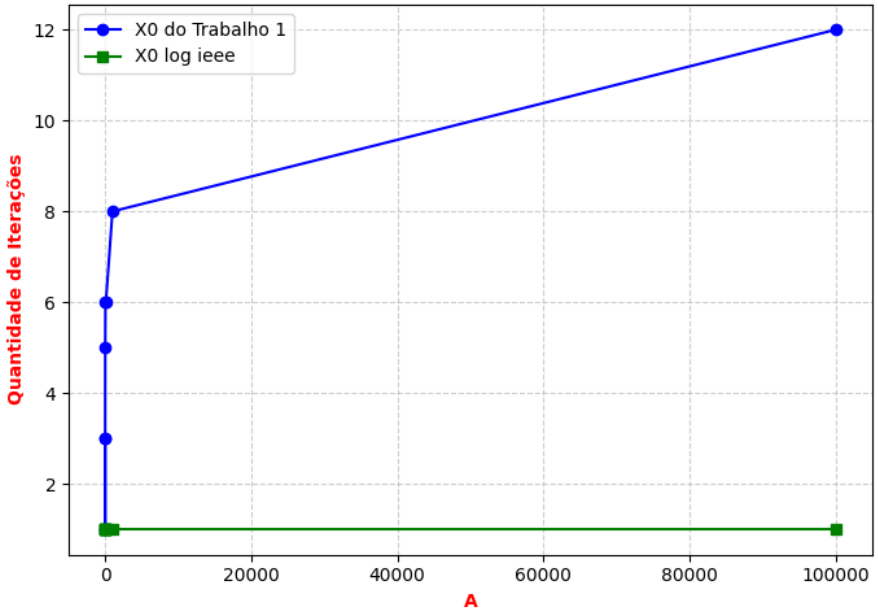

Figura 1: Gráfico comparando a quantidade K de iterações necessárias para o x_0 chutado pelo trabalho 1, e chutado pelo logaritmo da ieee.

2.3 Resultados

Podemos observar a partir destes resultados que a quantidade de iterações necessárias para atingir a condição de parada do erro pré-especificado aumenta conforme o argumento A aumenta ao utilizarmos nosso chute inicial do trabalho 1. Porém, ao utilizarmos o chute do método log ieee, conseguimos sempre o resultado em uma iteração.

3 Polinômios de Chebyshev

3.1 Fundamentação Teórica

Nomeados em homenagem ao matemático russo **Pafnuty Chebyshev**,os polinômios de Chebyshev se tornaram uma ferramenta poderosa no campo da análise numérica e teoria da aproximação devido a suas propriedades únicas como por exemplo a distribuição dos seus nós, que são os pontos onde o polinômio atinge valores extremos.

Ao utilizar os nós de Chebyshev como pontos de interpolação, podemos obter um polinômio interpolador que minimiza o erro máximo de aproximação, o que é uma característica extremamente desejável em muitas aplicações. Esta propriedade dos polinômios de Chebyshev é conhecida como "fenômeno de Runge", onde a escolha dos nós uniformemente espaçados leva a uma piora significativa no erro de interpolação nos extremos do intervalo.

Além disso, os polinômios de Chebyshev são especialmente úteis na aproximação de funções periódicas, como as funções trigonométricas, devido à sua natureza oscilatória e ao fato de que seus zeros estão concentrados nas extremidades do intervalo. Isso os torna mais eficazes do que os polinômios de Lagrange, por exemplo, na representação de funções trigonométricas em termos de séries de Chebyshev.

Essa capacidade dos polinômios de Chebyshev de proporcionar aproximações precisas e eficientes é extremamente valiosa em uma ampla gama de aplicações, desde a computação científica até a engenharia e a física aplicada. Em essência, os polinômios de Chebyshev são uma ferramenta poderosa que os cientistas e engenheiros têm à disposição para lidar com problemas numéricos de forma precisa e eficiente.

Os polinômios de Chebychev de grau n, denotados por $T_n(x)$, estão restritos ao intervalo $x \in [-1, +1]$. Essa restrição pode ser generalizada para o intervalo $x \in [a, b]$.

Os polinômios de primeira ordem de Chebyshev são definidos recursivamente da seguinte forma:

$$T_0(x) = 1 \tag{3}$$

$$T_1(x) = x \tag{4}$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{5}$$

A partir dela, podemos assumir os seguintes polinômios:

n	$T_n(x)$
0	1
1	x
2	$2x^2 - 1$
3	$4x^3 - 3x$
4	$8x^4 - 8x^2 + 1$
5	$16x^5 - 20x^3 + 5x$
6	$32x^6 - 48x^4 + 18x^2 - 1$
7	$64x^7 - 112x^5 + 56x^3 - 7x$
8	$128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$
9	$256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$
10	$512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1$
11	$1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x$
12	$2048x^{12} - 6144x^{10} + 6912x^8 - 3584x^6 + 840x^4 - 72x^2 + 1$
13	$4096x^{13} - 13312x^{11} + 16640x^9 - 9984x^7 + 2912x^5 - 364x^3 + 13x$
14	$8192x^{14} - 28672x^{12} + 39424x^{10} - 26880x^8 + 9408x^6 - 1568x^4 + 98x^2 - 1$
15	$16384x^{15} - 61440x^{13} + 92160x^{11} - 70400x^9 + 28800x^7 - 6048x^5 + 560x^3 - 15x$
16	$32768x^{16} - 131072x^{14} + 212992x^{12} - 180224x^{10} + 84480x^8 - 21504x^6 + 2688x^4 - 128x^2 + 1$
17	$65536x^{17} - 278528x^{15} + 487424x^{13} - 452608x^{11} + 239360x^9 - 71808x^7 + 11424x^5 - 816x^3 + 17x$
18	$131072x^{18} - 589824x^{16} + 1105920x^{14} - 1118208x^{12} + 658944x^{10} - 228096x^8 + 44352x^6 - 4320x^4 + 162x^2 - 1$
19	$262144x^{19} - 1245184x^{17} + 2490368x^{15} - 2723840x^{13} + 1770496x^{11} - 695552x^9 + 160512x^7 - 20064x^5 + 1140x^3 - 19x$
20	$5242888x^{20} - 2621440x^{18} + 5570560x^{16} - 6553600x^{14} + 4659200x^{12} - 2050048x^{10} + 549120x^8 - 84480x^6 + 6600x^4 - 200x^2 + 1$

Figura 2: Polinômios $T_n(x)$ de Chebychev

3.2 Implementação

Aproximação de seno utilizando polinômios de Chebyshev:

```
def aprox_seno(x):
    x=abs(x)

    S_X = x * (1 + calc_exp(x,2) * (-1 / fat(3) + calc_exp(x,2) *
        (1 / fat(5) + calc_exp(x,2) * (-1 / fat(7) + calc_exp(x,2)
        * (1 / fat(9) - calc_exp(x,2) * (11 / (fat(11) * 4) -
        calc_exp(x,2) * (11 / (fat(11) * 4) - calc_exp(x,2) * (77 /
        (fat(11) * 64) - calc_exp(x,2) * (55 / (fat(11) * 256) -
        calc_exp(x,2) * (11 / (fat(11) * 1024)))))))))))

    return S_X
```

Aproximação de cosseno utilizando polinômios de Chebyshev:

```
def aprox_cosseno(x):
    x=abs(x)

    C_X3 = 1 + ((-calc_exp(x,2)/2) *(1-
        (calc_exp(x,2)/12)*((-calc_exp(x,2)/30)*((1-calc_exp(x,2)/56)*((1-calc_exp(x,2)
```

```
    + (calc_exp(x,2)/fat(12) * (9/256 + calc_exp(x,2) *
    (-105/256 + calc_exp(x,2) * (7/4 + calc_exp(x,2) * (-27/8 +
    calc_exp(x,2) * (3)))))) - 1/479001600*2048
return C_X3
```

3.3 Resultados

Ao fazer a aproximação do seno e cosseno dos angulos: $\left(\frac{-\pi}{4}\right), \left(\frac{-\pi}{6}\right)$, 0 , $\left(\frac{\pi}{6}\right)$ e $\left(\frac{\pi}{4}\right)$ e comparando o resultado com o da função pronta do python para raiz, obtemos o seguinte grafico, que representa o erro em nosso método:

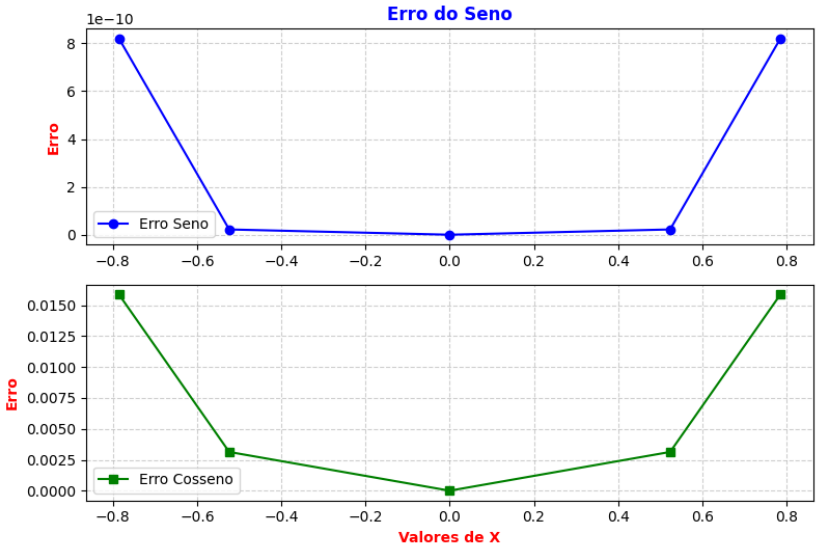

Figura 3: Comparação de erro entre as funções seno e cosseno aproximadas por polinômio de Chebychev e as funções seno e cosseno reais

Podemos notar que nosso método para o Seno possui um erro muito menor que o método implementado para o cosseno, na casa de $1 * 10^{-10}$.

4 Conclusão

O projeto alcançou sucesso ao aplicar e analisar os conceitos de precisão e erros computacionais em métodos de cálculo de raízes e seno e cosseno trigonométricos, além de comparar a precisão entre diferentes técnicas. A implementação e a análise detalhada desses métodos proporcionaram um aprofundamento no entendimento do âmbito da Matemática Computacional. Este estudo não apenas fortaleceu a compreensão das técnicas envolvidas, mas também destacou a importância da precisão numérica e da análise de erros em contextos computacionais.