Universidade Estadual de Maringá

Cálculo da função ln(x) utilizando nice numbers

Victor Hugo do Nascimento Bueno RA112651
João Pedro Peres Bertoncelo RA112650
João Gilberto Pelisson Casagrande RA112684
Samuel Ferreira Amboni RA100970

24 de abril de 2023

Sumário

1 Introdução **2**

2 Informações **2**

2.1 Nice Numbers 2

2.2 Nice Numbers Usados 3

3 Execução **4**

4 Gráficos **9**

5 Conclusão **10**

1 Introdução

Neste trabalho vamos falar sobre "nice numbers". Para facilitar a avaliação de funções logarítmicas, podemos utilizar tabelas de consulta que apresentam valores de logaritmos para determinados números agradáveis, também conhecidos como "nice-numbers". Esses números são escolhidos de tal forma que os valores de logaritmos associados a eles são facilmente calculáveis, o que simplifica a consulta aos valores de logaritmos de outros números.

Em matemática computacional, "nice numbers"são números que podem ser representados de forma precisa e eficiente em um sistema de computador. Isso é importante porque os computadores usam representações numéricas finitas, que têm uma precisão limitada e, portanto, podem introduzir erros em cálculos matemáticos.

2 Informações

2.1 Nice Numbers

Nice Numbers são números resultantes do cálculo:

$$2^i + 1$$

Exemplos:

$$2 = 2^0 + 1$$
$$3 = 2^1 + 1$$
$$5 = 2^2 + 1$$

Os "nice numbers"em matemática computacional geralmente incluem:

Potências de 2: como 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc. Esses números são frequentemente usados em sistemas de computador porque sua representação binária é simples e eficiente.

Números inteiros: números inteiros positivos e negativos são "nice numbers"porque eles têm uma representação direta em sistemas de computador e podem ser usados em muitos tipos de cálculos.

Frações com denominadores de potências de 2: frações com denominadores que são potências de 2, como 1/2, 1/4, 1/8, 1/16, etc., são "nice numbers"porque sua representação em sistema binário é simples e precisa.

Números decimais simples: números decimais que têm uma representação finita em notação decimal, como 0,1, 0,5, 0,25, 0,75, etc., são "nice numbers"porque eles podem ser representados com precisão finita em sistemas de computador.

Ao usar "nice numbers"em cálculos matemáticos, é possível minimizar erros de arredondamento e outros problemas associados à representação numérica em sistemas de computador. Além disso, os "nice numbers"são frequentemente usados em algoritmos numéricos, como o método de Newton para encontrar raízes de equações, o método de Euler para resolver equações diferenciais, entre outros.

2.2 Nice Numbers Usados

Os nice numbers usados foram:

```
def niceNumbers() -> list[dict]:
    return [
    {'n': 3.0, 'ln': 1.0986122886681098, 'exp': 1},
    {'n': 5.0, 'ln': 1.6094379124341003, 'exp': 2},
    {'n': 9.0, 'ln': 2.1972245773362196, 'exp': 3},
    {'n': 17.0, 'ln': 2.833213344056216, 'exp': 4},
    {'n': 33.0, 'ln': 3.4965075614664802, 'exp': 5},
    {'n': 65.0, 'ln': 4.174387269895637, 'exp': 6},
    {'n': 129.0, 'ln': 4.8573324964312685, 'exp': 7},
    {'n': 257.0, 'ln': 5.543741150080023, 'exp': 8},
    {'n': 513.0, 'ln': 6.233406007803151, 'exp': 9},
    {'n': 1025.0, 'ln': 6.926205448990583, 'exp': 10},
    {'n': 2049.0, 'ln': 7.621945174070975, 'exp': 11},
    {'n': 4097.0, 'ln': 8.320363047772128, 'exp': 12},
    {'n': 8193.0, 'ln': 9.021279351957755, 'exp': 13},
    {'n': 16385.0, 'ln': 9.72452689305461, 'exp': 14},
    {'n': 32769.0, 'ln': 10.429948609920046, 'exp': 15},
    {'n': 65537.0, 'ln': 11.137399121336791, 'exp': 16},
    {'n': 131073.0, 'ln': 11.846740579210332, 'exp': 17},
    {'n': 262145.0, 'ln': 12.557841045021825, 'exp': 18},
    {'n': 524289.0, 'ln': 13.270573115025252, 'exp': 19},
    {'n': 1048577.0, 'ln': 13.984814572476575, 'exp': 20},
    {'n': 2097153.0, 'ln': 14.700448005475653, 'exp': 21},
    {'n': 4194305.0, 'ln': 15.417360959112062, 'exp': 22},
    {'n': 8388609.0, 'ln': 16.135444016864466, 'exp': 23},
```

```
    {'n': 16777217.0, 'ln': 16.85459291287815, 'exp': 24},
    {'n': 33554433.0, 'ln': 17.57470786432736, 'exp': 25},
    {'n': 67108865.0, 'ln': 18.295691437997673, 'exp': 26},
    {'n': 0.5, 'ln': -0.6931471805599453, 'exp': -1},
    {'n': 0.25, 'ln': -1.3862943611198906, 'exp': -2},
    {'n': 0.125, 'ln': -2.0794415416798357, 'exp': -3},
    {'n': 0.0625, 'ln': -2.772588722239781, 'exp': -4},
    {'n': 0.03125, 'ln': -3.4657359027997265, 'exp': -5},
    {'n': 0.015625, 'ln': -4.1588830833596715, 'exp': -6},
    {'n': 0.0078125, 'ln': -4.852030263919617, 'exp': -7},
    {'n': 0.00390625, 'ln': -5.545177444479562, 'exp': -8},
    {'n': 0.001953125, 'ln': -6.238324625039507, 'exp': -9},
    {'n': 0.0009765625, 'ln': -6.931471805599452, 'exp': -10},
    {'n': 0.00048828125, 'ln': -7.624618986159397, 'exp': -11},
    {'n': 0.000244140625, 'ln': -8.317766166719342, 'exp': -12},
    {'n': 0.0001220703125, 'ln': -9.010913347279288, 'exp': -13},
    {'n': 0.00006103515625, 'ln': -9.704060527839232, 'exp': -14},
    {'n': 0.000030517578125, 'ln': -10.397207708399178, 'exp': -15},
    {'n': 0.0000152587890625, 'ln': -11.090354888959123, 'exp': -16},
    {'n': 0.00000762939453125, 'ln': -11.783502069519069, 'exp': -17},
    {'n': 0.00000381469726562 5, 'ln': -12.476649250079014, 'exp': -18},
    {'n': 0.0000019073486328125, 'ln': -13.16979643063896, 'exp': -19},
    {'n': 0.0000009536743164062 5, 'ln': -13.862943611198905, 'exp': -20},
    {'n': 0.0000004768371582031 25, 'ln': -14.55609079175885, 'exp': -21},
    {'n': 0.0000002384185791015625, 'ln': -15.249237972318796, 'exp': -22},
    {'n': 0.0000001192092895507 8125, 'ln': -15.94238515287874, 'exp': -23},
    {'n': 0.0000000596046447753 90625, 'ln': -16.635532333438686, 'exp': -24},
    {'n': 0.0000000298023223876953125, 'ln': -17.328679514001026, 'exp': -25},
]
```

3 Execução

A implementação do código foi a seguinte:

```
from typing import List, Dict
import math
import struct
import matplotlib.pyplot as plt
```

```python
def ln_x(number: float) -> float:
    """
    Retorna o logaritmo natural (ln) de um número usando interpolação a
    partir de uma tabela predefinida.
    """
    # Obtendo a tabela de números Nice
    nice_table = niceNumbers()

    # Definindo uma variável para percorrer a tabela Nice
    index = 0

    # Obtendo a linha atual da tabela Nice
    current_row = nice_table[index]

    # Enquanto o número for maior que o limite inferior da próxima linha da
    # tabela Nice,
    # avança para a próxima linha
    while index < len(nice_table) - 1 and nice_table[index + 1]['n'] > number:
        index += 1
        current_row = nice_table[index]

    # Dividindo o número pela constante da linha atual da tabela Nice
    x = number / current_row['n']

    # Obtendo o ln da linha atual da tabela Nice
    y = current_row['ln']

    # Interpolando o ln do número a partir das próximas linhas da tabela Nice
    while index < len(nice_table) - 1:
        # Multiplicando x pela potência da constante da linha atual da
        # tabela Nice
        power = current_row['exp']
        mult = calc(x, power)

        # Enquanto o número for maior ou igual ao limite inferior da próxima
        # linha da tabela Nice,
        # avança para a próxima linha
        while index < len(nice_table) - 1 and mult >= 1:
            index += 1
            current_row = nice_table[index]
            power = current_row['exp']
```

```
            mult = calc(x, power)

        # Se ainda há linhas para interpolar o ln, atualiza o valor de x e y
        if index < len(nice_table) - 1:
            x = mult
            y -= current_row['ln']

    # Retorna o ln do número subtraído do erro de arredondamento
    return y - abs(1 - x)


#tabela de consulta com entradas nice-numbers

def calc(a, e):
    # converte o número em float para um formato binário de 32 bits e
    # armazena em 'val'
    val = struct.pack('f', a)

    # extrai o inteiro de 32 bits de 'val' e armazena em 'k'
    k = struct.unpack('i', val)[0]

    # isola o expoente de 'k' (bits 23 a 30) e armazena em 'expoente'
    expoente = (k >> 23) & 0xFF

    # armazena o valor original de 'a' em uma variável auxiliar 'aux'
    aux = a

    # soma a potência 'e' ao expoente de 'a'
    expoente += e

    # atualiza o valor do expoente de 'k' com o novo valor calculado
    k = (k & ~(0xFF << 23)) | (expoente << 23)

    # converte o inteiro de 32 bits 'k' para um número em ponto flutuante
    # e armazena em 'f'
    val = struct.pack('i', k)
    f = struct.unpack('f', val)[0]

    # soma o valor original de 'a' ao valor resultante da potenciação
    f += aux
```

```
    # retorna o resultado final
    return f

def main():
    numbers = [1, 3, 5, 7, 10, 20, 50, 100, 200, 500, 1000, 5000, 10000]
    for number in numbers:
        # Obtém o resultado da nossa função
        result = ln_x(number)

        # Compara com o resultado da biblioteca matemática do Python
        expected = math.log(number)
        error = abs(expected - result)

        # Exibe o resultado
        print(f"ln({number}) = {result:.6f} (erro: {error:.6f})")

    # Cria uma lista de números no intervalo [1, 10000] com incrementos de 10
    x_values = [num for num in range(1, 10001, 10)]

    # Cria uma lista de erros correspondente aos números gerados
    y_values = [abs(ln_x(num) - math.log(num)) for num in x_values]

    # Plota o gráfico
    plt.plot(x_values, y_values)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Número')
    plt.ylabel('Erro absoluto')
    plt.show()

if __name__ == "__main__":
    main()
```

Alguns dos resultados obtidos foram:

```
ln(1) = 133.358123 (erro: 133.358123)
ln(3) = 141.675890 (erro: 140.577277)
ln(5) = 0.431946 (erro: 1.177492)
ln(7) = -0.234721 (erro: 2.180631)
ln(10) = -1.234721 (erro: 3.537306)
```

```
ln(20) = -4.568054 (erro: 7.563787)
ln(50) = -14.568054 (erro: 18.480077)
ln(100) = -31.234721 (erro: 35.839891)
ln(200) = -64.568054 (erro: 69.866372)
ln(500) = -164.568054 (erro: 170.782662)
ln(1000) = -331.234721 (erro: 338.142476)
ln(5000) = -1664.568054 (erro: 1673.085248)
ln(10000) = -3331.234721 (erro: 3340.445061)
```

4 Gráficos

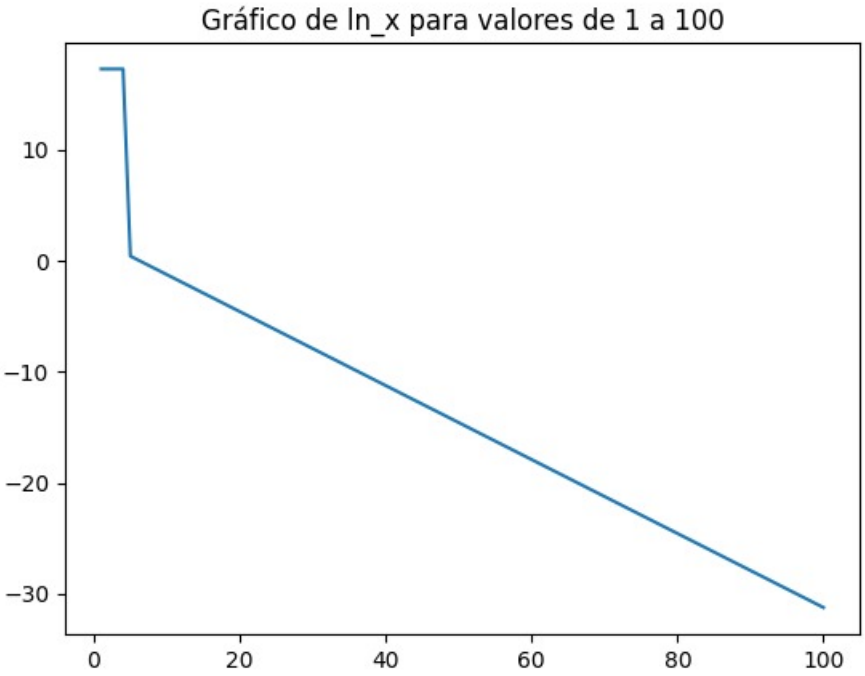

Figura 1: Resultados

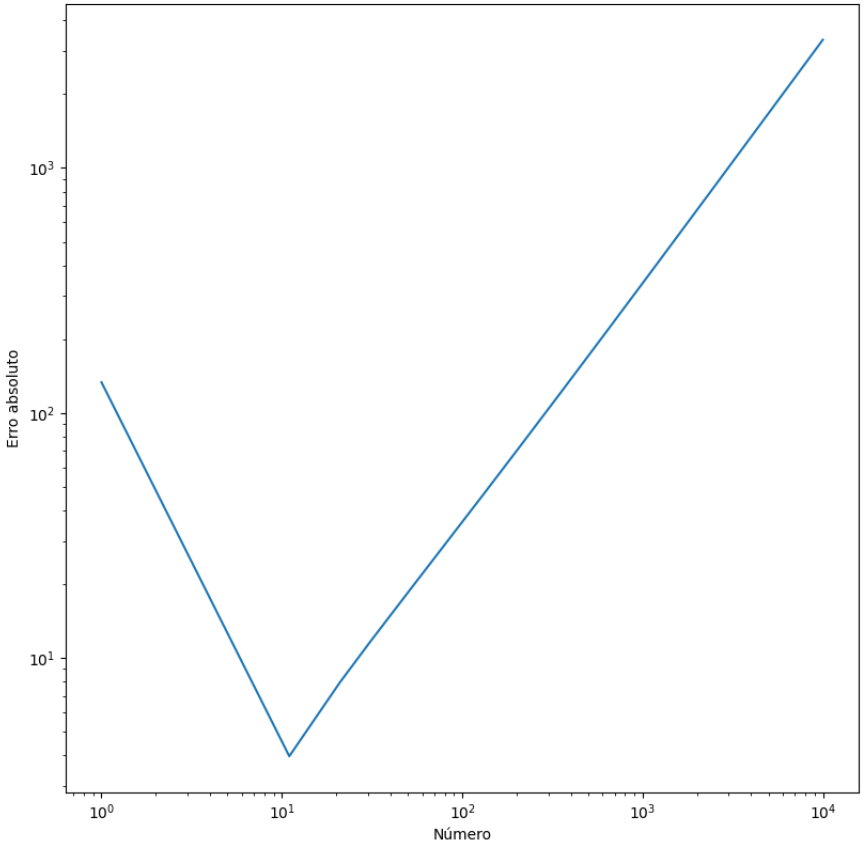

Figura 2: Erros demonstrados

5 Conclusão

Para realizar cálculos precisos e eficientes da função exponencial e logarítmica, a técnica de tabela de consulta é uma opção muito vantajosa. Essa técnica requer a escolha cuidadosa de "nice-numbers", que são números específicos que garantem a precisão do cálculo em toda a faixa de valores da função.

A medida que a tecnologia evolui, novas abordagens e algoritmos para cálculos matemáticos eficientes e precisos são desenvolvidos constantemente. A técnica de tabela de consulta é apenas uma das muitas opções disponíveis. No entanto, é importante lembrar que a escolha da melhor técnica depende do problema em questão e das limitações do hardware e software disponíveis.