UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

Matemática Computacional(6900)

TRABALHO No.4

Cálculo de ln usando nice numbers

Data: 19/04/2023

Professor: Airton Marco Polidorio

Discentes

R.A.	**Nome**
106769	Vítor Rodrigues Gôngora
119188	Pedro Lucas Keizo Honda
120116	Vitor Fernandes Gonçalves da Cruz

Introdução

O cálculo do logaritmo natural é uma operação matemática comum em diversas áreas, desde a matemática pura até a engenharia e as ciências naturais. No entanto, o cálculo do ln(x) para valores arbitrários pode ser computacionalmente custoso, especialmente quando se deseja alta precisão.

Uma estratégia para reduzir o tempo de cálculo é utilizar a técnica de nice numbers, que consiste em pré calcular os valores de ln(k) para um conjunto selecionado de números chamados nice numbers e, em seguida utilizar esses valores para chegar ao resultado aproximado de ln(x).

Objetivo

O presente trabalho tem como objetivo explorar cálculo do logaritmo natural por meio de uma tabela de nice numbers e analisar valores, erros e tempo de execução em relação ao cálculo do ln(x) embutido na linguagem de programação Python.

Explicação matemática

Os nice numbers definido como os números do tipo $\pm\ 2^{\pm i} \pm 1$ são utilizados em um primeiro momento na geração de uma tabela já pré-calculada com alguns valores de ln(x), exemplo:

k (nice number)	ln(k)
1.5	0.4054651081081644
3	1.0986122886681098
5	1.6094379124341003

Faz-se também uma redução do argumento x:

1. Procura-se na tabela o valor k que seja imediatamente superior ao valor x.
2. Calcula-se, então, o argumento reduzido:

$$x^* = \frac{x}{2^k}$$

De posse da tabela e do argumento reduzido x* pode-se iniciar o algoritmo:

1. Passo inicial:

repita
• Encontrar na tabela o maior valor de k que ao multiplicar o valor x_j produza $x_{j+1} = (k * x_j)$ de tal forma que $x_{j+1} < 1$ • $y_{j+1} = y_j + ln(k)$

- $x_j = x_{j+1}$
- $y_j = y_{j+1}$

2. Recuperação de resíduo
   $ln(1 - x) \simeq - x\ (para\ x\ pequeno)$
   $resíduo = - x_j\ (último\ valor\ de\ x\ encontrado)$

3. Cálculo final do logaritmo
   $ln(x\ *) = y_j + resíduo$

Código

```python
from ctypes import Structure, Union, c_float, c_uint32,
c_uint8
from matplotlib import pyplot as plt
import math
import time

class struct (Structure):
     _fields_ = [("f", c_uint32,23),
                ("e", c_uint32, 8),
                ("s",c_uint32,1)
                ]

class IEE754(Union):
     _fields_ = [("x",c_float),
                 ("bits",struct)]

def novo_numero_IEEE(num):
    y = IEE754()
    y.x = num
    return y

def IEEE_NEG(x):
    num = novo_numero_IEEE(x)
    num.bits.s = 0 if num.bits.s == 1 else 1
    return num.x
```

```
def IEEE_POW_2(exp):
    exp = int(exp)
    x = novo_numero_IEEE(2)

    a = c_uint8(x.bits.e)

    if(exp > 0):
        b = c_uint8(exp - 1)
        while b.value != 0:
            # O valor do carry é calculado
            carry = c_uint8(a.value & b.value)
            # O valor da soma é calculado
            a = c_uint8(a.value ^ b.value)
            # O valor do carry é shiftado para esquerda
            b = c_uint8(carry.value << 1)
    elif(exp < 0):
        b = c_uint8(-exp + 1)
        while b.value != 0:
            # O valor do borrow é calculado
            borrow = c_uint8((~a.value) & b.value)
            # XOR
            a = c_uint8(a.value ^ b.value)
            # O valor do borrow é shiftado para esquerda
            b = c_uint8(borrow.value << 1)

    x.bits.e = a.value

    return x # Retorna o valor final

def gerar_nice_numbers(inicio, fim):
    nice_numbers = []

    for i in range(inicio, fim+1):
         # Encontra os nice numbers no formato
         # +- (2 ** +-i) +- 1

         nice_number_1 = IEEE_POW_2(i).x + 1
         nice_number_2 = IEEE_POW_2(i).x - 1
```

```
        nice_number_3 = IEEE_POW_2(IEEE_NEG(i)).x + 1
        nice_number_4 = IEEE_POW_2(IEEE_NEG(i)).x - 1

        nice_number_5 = IEEE_NEG(IEEE_POW_2(IEEE_NEG(i)).x +
1)
        nice_number_6 = IEEE_NEG(IEEE_POW_2(IEEE_NEG(i)).x -
1)

        nice_number_7 = IEEE_NEG(IEEE_POW_2(i).x + 1)
        nice_number_8 = IEEE_NEG(IEEE_POW_2(i).x - 1)

        nice_numbers.append(nice_number_1)
        nice_numbers.append(nice_number_2)
        nice_numbers.append(nice_number_3)
        nice_numbers.append(nice_number_4)
        nice_numbers.append(nice_number_5)
        nice_numbers.append(nice_number_6)
        nice_numbers.append(nice_number_7)
        nice_numbers.append(nice_number_8)

    nice_numbers.sort()
    # Remove números duplicados
    nice_numbers = list(dict.fromkeys(nice_numbers))
    return nice_numbers

def gerar_tabela_ln_da_lista(lista_numeros):
    # Modifica a lista para apenas número positivos
    lista_numeros = [i for i in lista_numeros if i > 0]

    dict_ln = {}

    for i in lista_numeros:
        dict_ln[i] = math.log(i, math.e)

    return dict_ln

def reduzir_argumento(x, lista):
    for i in lista:
```

```
        if i > x:
            imediatamente_superior = i
            break

    x_red = x / imediatamente_superior

    return x_red, imediatamente_superior

def recuperacao_residuo(xn):
    return abs(1 - xn)

def ln(x, lista_ln, nice_numbers):
    argumento_reduzido, numero_imediatamente_superior =
reduzir_argumento(x, nice_numbers)

    xj = argumento_reduzido
    yj = lista_ln[numero_imediatamente_superior]
    iter = len(lista_ln.keys())

    while iter > 0:
        maior_k = 0

        for i in lista_ln.keys():
            # Se k for 1, não há mudança no resultado
            if(i != 1 and i * xj < 1):
                maior_k = i

        # Condição de parada
        # Percorreu toda a lista de logaritmo e não encontrou
        # valor tal que i * xj < 1
        if(maior_k == 0):
            break

        xj = maior_k * xj
        yj = yj - lista_ln[maior_k]
        iter = iter -1

    resultado_ln = yj - recuperacao_residuo(xj)
```

```
    return resultado_ln

def main():
    nice_numbers = gerar_nice_numbers(-8, 8)
    lista_ln = gerar_tabela_ln_da_lista(nice_numbers)

    erro, x_list, tempo_nice_numbers,tempo_calculadora,
resultado_calculadora, resultado_nice_numbers =
[],[],[],[],[],[]
    for x in range(1,100):
        star_calculadora = time.time()
        ln_calculadora = math.log(x)
        end_calculadora = time.time()

        start = time.time()
        ln_nice_numbers = ln(x, lista_ln, nice_numbers)
        end = time.time()

        erro.append(abs(ln_nice_numbers - ln_calculadora))

        tempo_nice_numbers.append(end -start)

tempo_calculadora.append(end_calculadora-star_calculadora)

        x_list.append(x)
        resultado_calculadora.append(ln_calculadora)
        resultado_nice_numbers.append(ln_nice_numbers)

    # Erro
    plt.plot(x_list,erro,label = 'ln(X)-Calculadora X
Nice-Numbers')
    plt.ylabel('Erro')
    plt.xlabel('Argumento')
    plt.legend()
    plt.show()

    # Tempo
    plt.plot(x_list,tempo_nice_numbers,label =
'Tempo-Nice-Numbers')
```

```
    plt.plot(x_list,tempo_calculadora,label =
'Tempo-Calculadora')

    plt.ylabel('Tempo')
    plt.xlabel('Argumento')
    plt.legend()
    plt.show()

    # Resultado
    plt.plot(x_list,resultado_calculadora,label =
'ln(X)-Calculadora',color ='blue')
    plt.plot(x_list,resultado_nice_numbers,label = 'ln(X)-
Nice-Numbers',color =  'red')
    plt.yscale('log')
    plt.ylabel('ln(x)')
    plt.xlabel('Argumento')
    plt.legend()
    plt.show()

main()
```

Análise dos resultados

• Tabela 01 - Logaritmos

x	ln(x) da calculadora	ln(x) usando a tabela
1	0.0	8.632194907631438e-06
2	0.6931471805599453	0.6931556870794752
3	1.0986122886681098	1.0986153658373967
4	1.3862943611198906	1.3862974382891777
5	1.6094379124341003	1.6094416298912544
6	1.791759469228055	1.7917612939682495
7	1.9459101490553132	1.9459231226911209
8	2.0794415416798357	2.079467490727081 4

9	2.1972245773362196	2.197227654505509
10	2.302585092994046	2.3025935995135782

• Tabela 02 - Cálculo do erro

x	Erro ln(x) calculadora - ln(x) tabela
1	8.632194907631438e-06
2	8.506519529882794e-06
3	3.077169286935799e-06
4	3.0771692871578438e-06
5	3.7174571541065404e-06
6	1.8247401945004071e-06
7	1.2973635807655981e-05
8	2.5949047245621415e-05
9	3.0771692896003344e-06
10	8.506519532325285e-06

Podemos observar através da tabela de erros que ao se utilizar a técnica dos nice numbers, conseguimos obter pelo menos 4 casas decimais após a vírgula de precisão.

• Tabela 03 - Tempo de execução (ns)

x	ln(x) da calculadora	ln(x) usando a tabela
1	7.152557373046875e-07	0.00018477439880371094
2	1.430511474609375e-06	0.00015544891357421875
3	7.152557373046875e-07	0.0001690387725830078
4	7.152557373046875e-07	0.00013566017150878906
5	4.76837158203125e-07	0.0001392364501953125
6	4.76837158203125e-07	0.00013971328735351562

7	7.152557373046875e-07	0.0001668930053710937 5
8	9.5367431640625e-07	0.0001609325408935547
9	7.152557373046875e-07	0.0001437664031982422
10	2.384185791015625e-07	0.0001542568206787109 4

Podemos observar que em relação ao tempo de execução para os valores avaliados, a técnica de nice numbers foi uma técnica bem pior que a função ln(x) nativa na linguagem de programação python.

Gráfico 1 - Gráficos dos valores de ln(x) em escala logarítmica

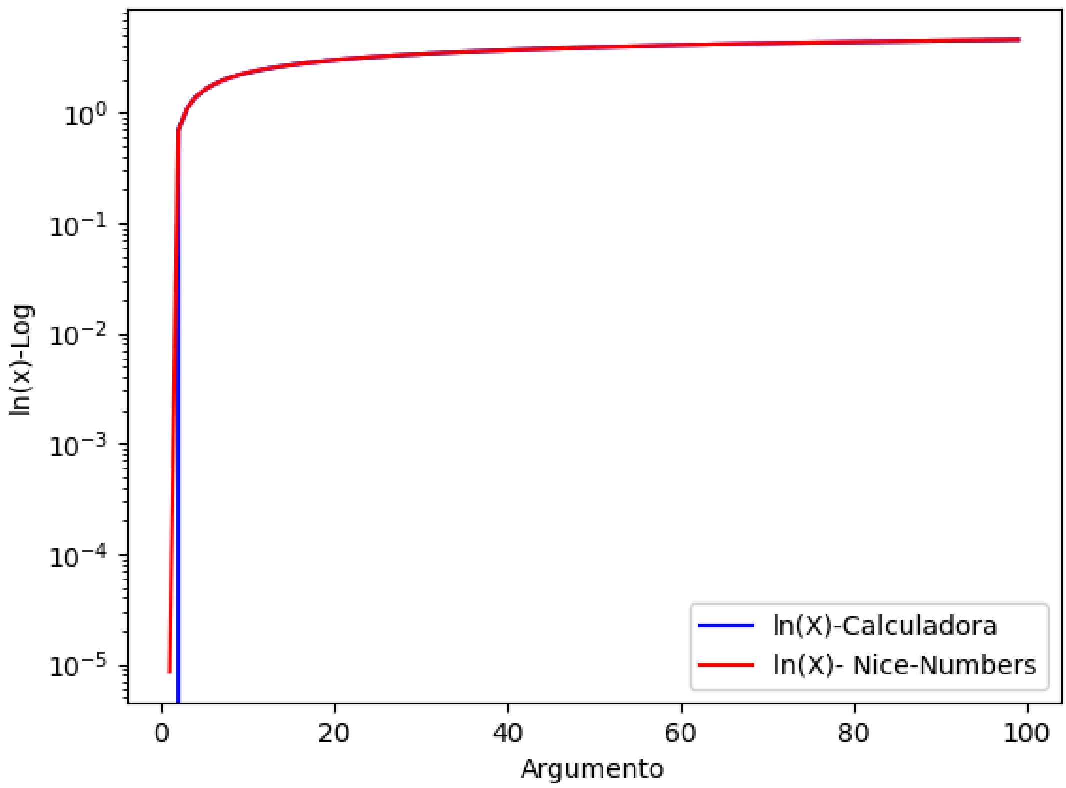


Gráfico 2 - Gráfico do erro

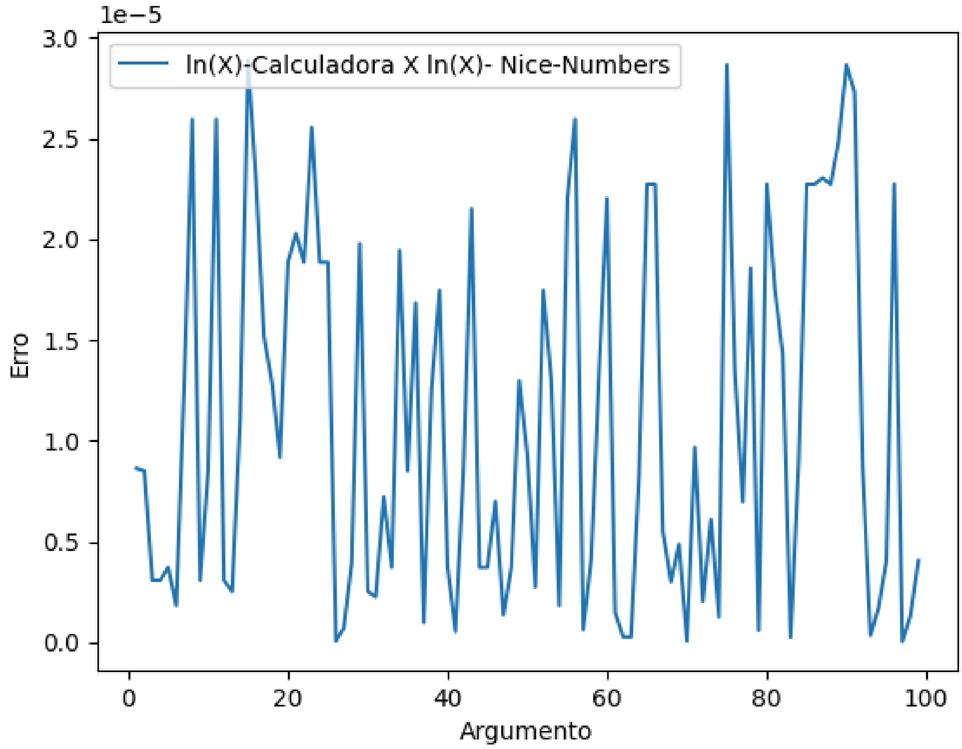
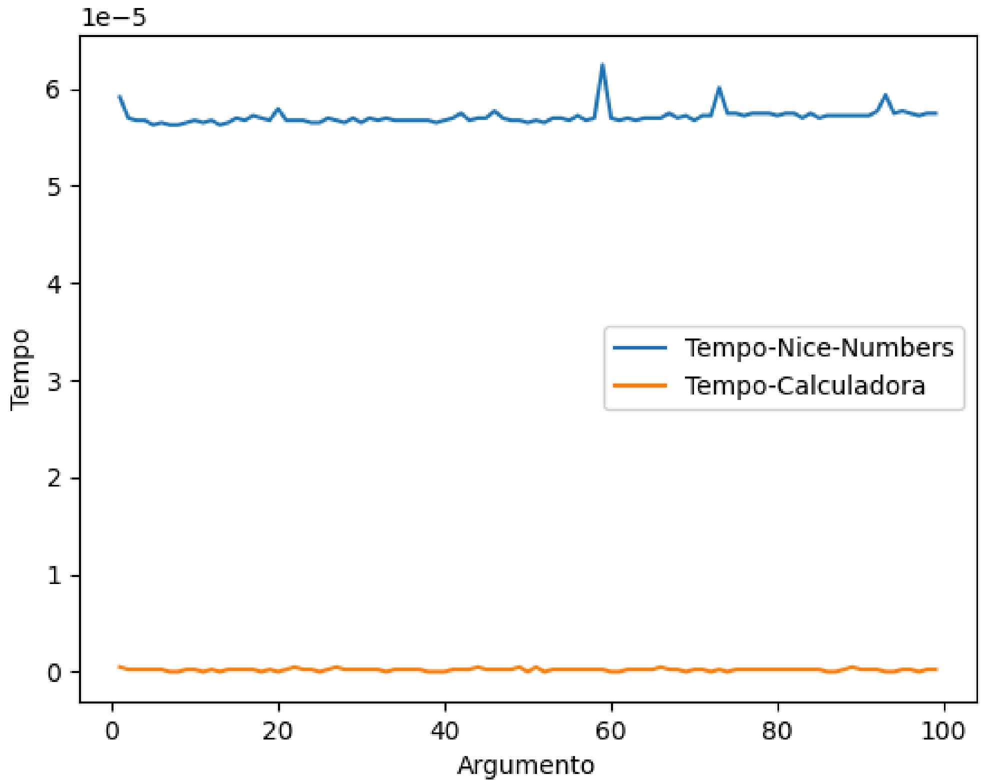


Gráfico 3 - Gráfico do tempo de execução



Conclusão

Este trabalho explorou a técnica de nice numbers para o cálculo do logaritmo natural e comparou seus resultados com o cálculo embutido do ln(x) na linguagem de programação python. Observamos que a técnica de nice numbers pode ser uma estratégia eficaz para aproximar o valor do ln(x), especialmente quando se deseja alta precisão e o valor de x está próximo a um dos nice numbers pré-calculados. No entanto, nossos experimentos mostraram que o cálculo do ln(x) embutido na linguagem de programação python apresentou desempenho superior em termos de tempo de execução, isso se deve em parte às otimizações implementadas na linguagem para o cálculo de funções matemáticas.