Uso de nice numbers para cálculo do logaritmo natural

Bruno Fusieger[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

ra112646@uem.br

Resumo. *Este artigo descreve o método de cálculo do logaritmo natural por tabela de consulta construída com nice numbers, também é apresentada uma análise dos resultados obtidos.*

1. Nice numbers

Nice number é todo número que permite trocar a multiplicação de um ponto flutuante por somas, desde que os números sejam representados no padrão IEEE 754 e, no geral, um formato pode ser $n = \pm 2^{\pm i} \pm 1$. O processo pode ser feito da seguinte forma, seja x um número representado no padrão IEEE 754, soma-se $\pm i$ ao expoente de x e depois soma-se o x original à x. Um exemplo disso pode ser visto abaixo, onde o nice number é 3.

$$9 = (1 + 0{,}125) \cdot 2^3$$
$$9 \cdot 3 = (1 + 0{,}125) \cdot 2^{3+1} + (1 + 0{,}125) \cdot 2^3$$
$$9 \cdot 3 = 27$$

Com base nisso, foi elaborada a função *multiplica,* que realiza a multiplicação de número qualquer a por um nice number com expoente e.

```
float multiplica(float a, int e) {
  union {
      float f;
      unsigned int k;
  } val = {.f = a};

  unsigned char expoente = val.k >> 23;
  float aux = a;
  expoente += e;
  val.k = (val.k & ~(0xFF << 23)) | expoente << 23;
  val.f += aux;
  return val.f;
}
```

2. Tabela de consulta

Para o auxílio no algoritmo foi construída uma tabela de consulta que tem como colunas, o nice number n, o expoente exp e o logaritmo natural $ln(n)$. O primeiro registro da tabela tem expoente oito e as entradas subsequentes se referem ao expoente subtraído de 1, até que o expoente chegue à -23. Essa tabela foi construída em C usando um agregado heterogêneo para representar as colunas, a tabela completa está disponível no anexo I.

```
typedef struct {
    int exp;
    float n;
    float ln;
} NiceNumber;
```

3. Algoritmo

O algoritmo tem como base a invariância do logaritmo natural $ln(1) = 0$, ou ainda $ln(\frac{x}{x}) = 0$. Assumindo k uma constante natural positiva qualquer tem-se:

$$ln(\frac{x}{kx}) = ln(\frac{x}{x}) - ln(k) = 0$$

Note que isso é verdade apenas para $k = 1$, no entanto, como o algoritmo é aproximativo, podemos usar essa equação mantendo x próximo de um e acumulando o logaritmo natural dos nice numbers tabelados que quando multiplicados por x o mantém abaixo de um.

Sendo assim, o primeiro passo do algoritmo é reduzir o valor do argumento para um valor menor do que um. Isso apresenta a primeira consideração ao construir a tabela: o maior nice number tem de ser maior do que o maior valor de entrada do algoritmo, caso contrário a entrada nunca será reduzida para menos do que um, tornando o erro proporcionalmente grande à diferença do maior valor da tabela e da entrada.

Para reduzir o valor da entrada, procura-se o primeiro nice number imediatamente maior do que x na tabela e então, divide-se a entrada por esse número. Em seguida, registra-se o logaritmo do nice number encontrado como y, esses são os valores inicias do algoritmo.

Depois, procura-se por um nice number que quando multiplicado por x atual seja menor do que 1. Quando esse número for encontrado, atribui-se o resultado da multiplicação a x e y como $y - ln(n)$, onde n é o nice number encontrado. Esse processo é repetido até o fim da tabela.

Quando o fim da tabela for atingido, calcula-se o resíduo como a diferença absoluta entre 1 e o último x computado, depois subtrai-se o resíduo de y. O valor obtido ao final é o valor aproximado de $ln(x)$.

4. Resultados

Para a análise de precisão do algoritmo foi utilizado um espaço linear de 1 até 257 com 50 amostra. Observe que a tabela utilizada tem como valor máximo 257 não faria sentido utilizar valores de entrada maiores do que isso.

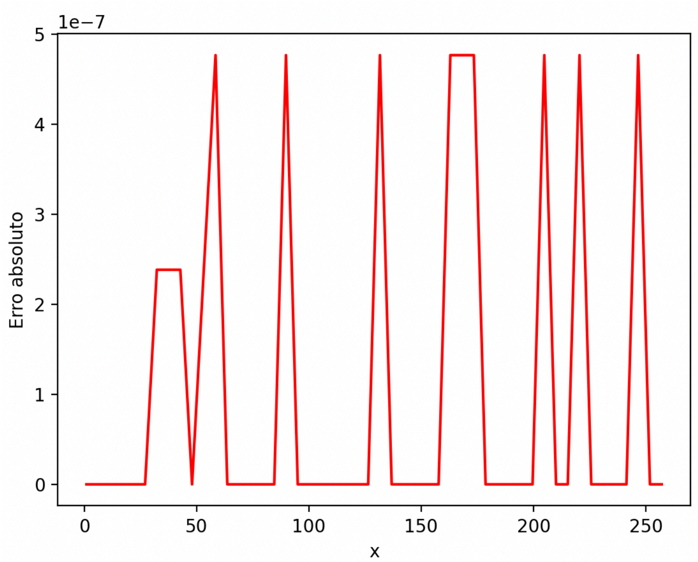

Figura 1.Erro absoluto

É observado que esse algoritmo tem erro a partir de 7 casas decimais, o que demonstra que ele tem um boa precisão. E como todo o código conta com apenas uma multiplicação, que é a utilizada na redução do valor de entrada, ele apresenta uma boa eficiência computacional.

Observe ainda que esse algoritmo é sensível a tabela de nice numbers, quanto maior a tabela, mais preciso será o algoritmo, e como dito anteriormente, o valor do maior nice number também restringe o maior valor de entrada.

Anexo I

```
NiceNumber NICE[] = {
{.exp = 8, .n = 257.0f, .ln = 5.549076080322265625},
{.exp = 7, .n = 129.0f, .ln = 4.8598122596740722265625},
{.exp = 6, .n = 65.0f, .ln = 4.174387454986572265625},
{.exp = 5, .n = 33.0f, .ln = 3.4965076446533203125},
{.exp = 4, .n = 17.0f, .ln = 2.833213329315185546875},
{.exp = 3, .n = 9.0f, .ln = 2.1972246170043945 3125},
{.exp = 2, .n = 5.0f, .ln = 1.6094379425048828125},
{.exp = 1, .n = 3.0f, .ln = 1.098612308502197265625},
{.exp = 0, .n = 2.0f, .ln = 0.693147182464599609375},
{.exp = −1, .n = 1.5, .ln = 0.405465096235275268554 6875},
{.exp = −2, .n = 1.25, .ln = 0.22314354777336120605 46875},
{.exp = −3, .n = 1.125, .ln = 0.117783032357692718505859375},
{.exp = −4, .n = 1.0625, .ln = 0.060624621808528900146484375},
{.exp = −5, .n = 1.03125, .ln = 0.030771657824516296386718 75},
{.exp = −6, .n = 1.015625, .ln = 0.015504186972975730895996 09375},
{.exp = −7, .n = 1.0078125, .ln = 0.0077821402810513973236083984375},
{.exp = −8, .n = 1.00390625, .ln = 0.00389864039607346057891845703125},
{.exp = −9, .n = 1.001953125, .ln = 0.001951220096088945865631103515625},
{.exp = −10, .n = 1.0009765625, .ln = 0.0009760859538801014423370361328125},
{.exp = −11, .n = 1.00048828125, .ln = 0.0004881620698142796754837036132812 5},
{.exp = −12, .n = 1.000244140625, .ln = 0.000244110822677612304 6875},
{.exp = −13, .n = 1.0001220703125, .ln = 0.00012206286191940307617 1875},
{.exp = −14, .n = 1.00006103515625, .ln = 0.00006103329360485076904296875},
{.exp = −15, .n = 1.000030517578125, .ln = 0.000030517112463712692260 7421875},
{.exp = −16, .n = 1.0000152587890625, .ln = 0.0000152586726471781730651 85546},
{.exp = −17, .n = 1.00000762939453125, .ln = 0.0000076293654274195432662 9638},
{.exp = −18, .n = 1.000003814697265625, .ln = 0.0000038146899896673858165740},
{.exp = −19, .n = 1.0000019073486328125, .ln = 0.0000019073468138230964541 43},
{.exp = −20, .n = 1.00000095367431640625, .ln = 0.0000009536738616588991135 3},
{.exp = −21, .n = 1.000000476837158203125, .ln = 0.000000476837044516287278 3},
{.exp = −22, .n = 1.0000002384185791015625, .ln = 0.00000023841855067985306 9},
{.exp = −23, .n = 1.00000011920928955078125, .ln = 0.00000011920928244535389}
};
```

Anexo II

```c
#include <math.h>
#include <stdio.h>
#define LINHAS 32

float ln_x(float a) {
  unsigned int cursor = 0;
  NiceNumber atual = NICE[cursor];

  while (cursor < LINHAS - 1 && NICE[cursor + 1].n > a) {
    cursor += 1;
    atual = NICE[cursor];
  }

  float x = a / atual.n;
  float y = atual.ln;

  while (cursor < LINHAS - 1) {
    float mult = multiplica(x, atual.exp);

    while (cursor < LINHAS - 1 && mult >= 1) {
      cursor += 1;
      atual = NICE[cursor];
      mult = multiplica(x, atual.exp);
    }

    if (cursor < LINHAS - 1) {
      x = mult;
      y = y - atual.ln;
    }
  }
  return y - fabs(1 - x);
}
```