Inverso da Raiz Quadrada
Matemática Computacional

Felipe Gabriel Comin Scheffel[1]

[1]Universidade Estadual de Maringá (UEM)

{ra117306}@uem.br

Abstract. *This article aims to present and compare more than one implementation of the inverse square root problem, solved using the Newton-Raphson method in three different contexts, analyzing aspects such as accuracy and performance.*

Resumo. *Este trabalho tem como objetivo principal apresentar e comparar mais de uma implementação do problema do inverso da raiz, resolvido utilizando o método de Newton-Raphson em três contextos diferentes, analisando aspectos como precisão e performance.*

1. Introdução

O método de Newton-Raphson é uma técnica iterativa muito utilizada na área de cálculo numérico para encontrar aproximações para as raízes de uma função. Ele parte de uma estimativa inicial e, a partir daí, utiliza a tangente da curva da função para iterativamente refinar essa estimativa até chegar a uma solução satisfatória. O método é bastante eficiente para funções suaves e bem comportadas, e sua implementação pode ser relativamente simples, com poucas linhas de código.

2. Metodologia

O artigo aborda a aplicação do método de Newton-Raphson para estimar o inverso da raiz utilizando três diferentes abordagens: a inversão de NR-sqrt, NR-invsqrt e Fast Square Root. A inversão de NR-sqrt utiliza a fórmula original do método de Newton-Raphson para encontrar a raiz quadrada, seguida pela inversão do resultado obtido. O método NR-invsqrt é uma variante direta que utiliza o método de Newton-Raphson para encontrar diretamente o inverso da raiz quadrada. O método Fast Square Root de Tarolli é uma técnica de aproximação para o cálculo do inverso da raiz quadrada que utiliza um truque matemático usando uma constante ”mágica”para a estimativa inicial.

Todas as implementações utilizam aspectos característicos da representação de ponto flutante no padrão IEEE-754, a fim de reduzir o valor do argumento, simplificando o cálculo. O padrão permite separar um valor ”mantissa”, que representa a parte significativa, do ”expoente”, que representa o grau. É possível aplicar o método apenas na mantissa e manipular o expoente com operações simples para atingir o valor esperado sem afetar a precisão.

2.1. Fundamentação Teórica

O método de Newton-Raphson é um algoritmo numérico utilizado para encontrar raízes de funções. Desenvolvido a partir de uma série de Taylor, permite aproximar a raiz de

uma função ao aproximá-la por uma reta tangente em um ponto inicial e, em seguida, encontrar a interseção dessa reta com o eixo x, o que nos dá uma nova estimativa da raiz.[Akram and Ann 2015] Esse processo é repetido várias vezes até que a estimativa da raiz converja para um valor aceitável.

Assim, tendo uma estimativa x_n, partindo de uma estimativa inicial x_0, é possível obter uma estimativa melhor a partir da fórmula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

onde $f(x)$ é a função para a qual se deseja obter as raízes e $f'(x)$ é a primeira derivada de tal função.

Também é relevante o formato de representação de ponto flutuante com precisão simples IEEE-754, onde um número na base decimal passa a ser composto de 1 bit de sinal, 8 bits para o expoente enviesado e 23 bits representando a mantissa. Ou seja,

$$x_{10} = (s)(E + b)(1 + f_R)$$

, onde s é o bit de sinal, E é o expoente, b é o viés do expoente ($2^8 - 1 = 127$, no caso de precisão simples), e f_R é a mantissa normalizada para $R = 23$ bits.

2.1.1.

Raiz quadrada de um número

Parte-se do conceito base de que encontrar uma raiz quadrada é obter $x = \sqrt{A}$. Ou seja, rearranjando o expoente, se trata de achar uma raiz para da função de forma:

$$f(x) = x^2 - A$$

Que tem a primeira derivada:

$$f'(x) = 2x$$

E assim chega-se na fórmula de estimativa:

$$x_{n+1} = x_n - \frac{x_n^2 - A}{2x_n}$$
$$x_{n+1} = \frac{1}{2}(x_n + \frac{A}{x_n})$$

Esse método pode ser repetido por um número pré-definido de iterações ou até atingir uma tolerância $\epsilon = x_{n+1} - x_0$ desejada.

Nota-se agora a decompisação de um número decimal para a sua representação IEEE-754 ao relizar uma operação de raiz quadrado:

$$\sqrt{x_{10}} = \sqrt{(1+f_R)2^E} = \begin{cases} 2^{\frac{E}{2}}\sqrt{1+f_R} & \text{, se E é par} \\ 2^{\frac{E+1}{2}}\frac{1}{\sqrt{2}}\sqrt{1+f_R} & \text{, senão} \end{cases}$$

Assim, é possível obter a raiz quadrada do número calculando a raiz quadrada de sua mantissa e dividindo seu expoente por 2 (realizando o ajuste necessário se E for ímpar).

E, como estimativa inicial, será utilizada metade da mantissa mais o bit de normalização:

$$x_0 = 1 + \frac{f_R}{2}$$

.

2.1.2. Inverso da Raiz quadrada de um número

Seguindo a mesma lógica, partimos da função:

$$f(x) = \frac{1}{x^2} - A = x^{-2} - A$$

Que tem a primeira derivada:

$$f'(x) = -2x^{-3}$$

E assim chega-se na fórmula de estimativa:

$$x_{n+1} = x_n - \frac{x_n^{-2} - A}{-2x_n^{-3}}$$
$$x_{n+1} = x_n(\frac{3}{2} - \frac{A}{2}x_n^2)$$

A relação com o padrão IEEE-754 continua a mesma, com a adição de que o sinal do expoente deve ser invertido e, caso o resultado da aproximação da mantissa seja menor que 1, esse resultado deve ser normalizado multiplicando por 2 e subtraindo 1 do expoente[Hertz et al. 2016].

A escolha inicial será o inverso da escolha inicial do método anterior, ou seja:

$$x_0 = \frac{2}{2 + f_R}$$

2.1.3. Fast Square Root (Tarolli)

O algoritmo de inversão rápida da raiz quadrada foi visto pela primeira vez no código fonte do jogo de tiro em primeira pessoa Quake III Arena. Ele estima rapidamente o inverso da raiz quadrada de uma variável de ponto flutuante, sendo usada na normalização de vetores durante os cálculos de iluminação e reflexão do jogo.

Resumidamente, se trata de uma iteração do método de Newton-Raphson realizado na representação inteira do número; o seu diferencial é o método da escolha inicial, que utiliza uma constante mágica, da qual é subtraído metade do valor da mantissa do número. A escolha dessa constante decorre de diversas propriedades matemáticas ligadas a representação de ponto flutuante, e são explicadas com mais detalhes por Lomont em [Lomont 2003].

2.2. Implementação

Nessa seção, será comentado sobre o processo de desenvolvimento do script, além de serem apresentados trechos relevantes de código e instruções para sua utilização. Nos trechos de código, serão omitidos comentários e validações a fim de apresentar a lógica principal de forma mais limpa.

Foi desenvolvida a implementação utilizando a linguagem C, compilado no Windows com Clang. A linguagem foi escolhida visto a necessidade de realizar operações a nível de bits na memória, o que é facilitado pelas funções nativas de bishift, union e struct bitfields.

Também foi implementado um programa básico em python com pandas e matplotlib para renderizar os gráficos dos resultados.

O programa completo está disponível no github, em *schelip/newton-raphson*.

2.2.1. NR-sqrt

Para a utilização dos componentes da representação IEEE-754, foi necessária a definição de um tipo union chamado `f_ieee_754`, que é usado para manipular os bits de um float em IEEE 754.

```
typedef union {
    float f;
    struct
    {
        unsigned int mantissa : 23;
        unsigned int exponent : 8;
        unsigned int sign : 1;

    } bits;
} f_ieee_754;
```

Em seguida, foram definidas duas funções para a conversão entre o `float` da linguagem e o tipo definido:

```
void float_to_ieee_754(float x, char *sign, int* exponent,
   float *mantissa) {
    f_ieee_754 x_ieee;
    x_ieee.f = x;
    *exponent = x_ieee.bits.exponent - 127;
    *sign = x_ieee.bits.sign;
    *mantissa = (float)x_ieee.bits.mantissa / (1 << 23);
}

float ieee_754_to_float(char sign, float mantissa, int
   exponent) {
    int exp = exponent + 127;
    unsigned int mantissa_bits = (unsigned int)(mantissa *
       (1 << 23));
    mantissa_bits |= 1 << 23;
    f_ieee_754 float_bits;
    float_bits.bits.sign = sign;
    float_bits.bits.exponent = exp;
    float_bits.bits.mantissa = mantissa_bits;
    return float_bits.f;
}
```

A função nr-sqrt começa verificando se o número de entrada é zero, e retorna `INFINITY` se for o caso. Em seguida, ela chama a função `float_to_ieee_754`. O código ajusta o expoente, e a seguir calcula a aproximação da mantissa do resultado usando o método de Newton-Raphson. O valor inicial para a iteração é `xk_1 = 1.0 + mantissa * 0.5`. A cada iteração, é calculado um novo valor `xk_1` usando a equação `xk_1 = 0.5 * (xk + mantissa / xk)`. O loop continua até que o valor absoluto da diferença entre `xk_1` e `xk` seja menor que uma tolerância `EPSILON` ou até que o número máximo de iterações `MAX_NR_ITER` seja alcançado.

```
#define EPSILON 5e-17
#define N_EXEC 1000000
#define SQRT_2 1.41421356237309504880168872420969807
float nr_sqrt (float x) {
    if (x == 0) return INFINITY;

    float mantissa;
    int exponent;
    char sign;

    float_to_ieee_754(x, &sign, &exponent, &mantissa);

    int is_odd = 0;
    if (exponent & 1) {
        is_odd = 1;
        exponent--;
    }
```

```
    exponent >>= 1;

    double xk;
    double xk_1 = 1.0 + mantissa * 0.5; // 1 + (f/2)
    mantissa += 1.0;
    int k = 0;
    do {
        xk = xk_1;
        // 1/2(xk + A/xk)
        xk_1 = 0.5 * (xk + mantissa / xk);
        k++;
    } while(fabs(xk_1 - xk) > EPSILON && k < MAX_NR_ITER);

    float result;
    if (is_odd) result = (xk_1 * SQRT_2);
    else result = xk_1;

    return 1 / ieee_754_to_float(sign, result, exponent);
}
```

Por fim, o código calcula o resultado final, ajustando caso seja necesário devido a um expoente impar, realiza a conversão para `float`, e retorna seu inverso.

2.2.2. NR-invsqrt

A implementação segue o mesmo padrão de NR-sqrt, apenas invertendo o expoente, alterando a fórmula para obter a escolha inicial e da iteração de Newton-Raphson, e realizando um ajuste a mais no resultado caso necessário.

```
float nr_invsqrt (float x) {
    if (x == 0) return INFINITY;

    float mantissa;
    int exponent;
    char sign;

    float_to_ieee_754(x, &sign, &exponent, &mantissa);

    int is_odd = 0;
    if (exponent & 1) {
        is_odd = 1;
        exponent++;
    }
    exponent >>= 1;
    exponent = -exponent; // inverter expoente

    double xk;
```

```
    double xk_1 = 2.0 / (2 + mantissa); // 2/(2 + f)
    mantissa += 1.0;
    double mantissa_half = mantissa * 0.5;
    int k = 0;
    do {
        xk = xk_1;
        // xk(3/2 - Axk/2)
        xk_1 = xk * (1.5 - mantissa_half * xk * xk);
        k++;
    } while(fabs(xk_1 - xk) > EPSILON && k < MAX_NR_ITER);

    float result = xk_1;
    if (is_odd) result = (xk_1 * SQRT_2);
    else result = xk_1;

    if (result < 1) {
        result *= 2;
        exponent -= 1;
    }

    return ieee_754_to_float(sign, result, exponent);
}
```

2.2.3. Fast Square Root (Tarolli)

Ao contrário dos outros dois métodos, essa implementação não utiliza os componentes da representação IEEE-754 de forma separada e explícita. Ao invés disso, é utilizada uma union para poder representar o número como inteiro, e essa representação é utilizada com a constante mágica para obter uma escolha inicial. Então, o valor representado como ponto flutuante é utilizado para realizar uma iteração de Newton-Raphson, e o resultado é retornado.

```
float fast_invsqrt_tarolli(float x) {
    float xhalf = 0.5f * x;
    union {
        float x;
        int k;
    } u;
    u.x = x;
    u.k = 0x5f375286 - (u.k >> 1); // what the fuck?
    u.x = u.x * (1.5f - xhalf * u.x * u.x);
    return u.x;
}
```

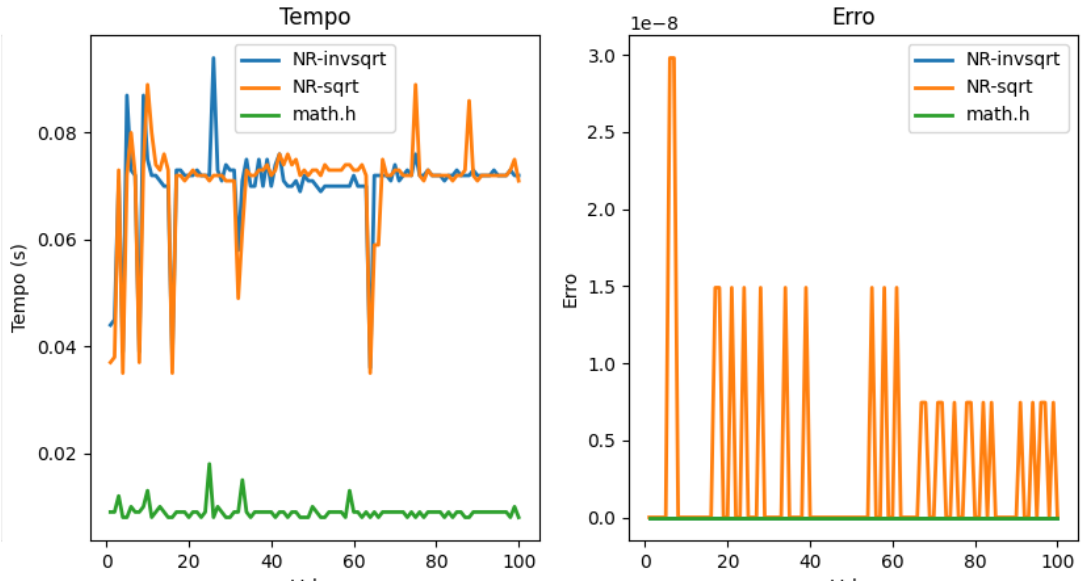
3. Discussão

Nesse tópico, serão apresentados os resultados obtidos pelos diferentes métodos, assim como será apresentada uma análise do erro e da performance de cada método e como esse resultado se compara com o dos outros métodos.

3.0.1. NR-sqrt vs NR-invsqrt

A primeira parte da análise será sobre a precisão e tempo de execução de NR-sqrt e NR-invsqrt. Ambos conseguiram atingir precisão satisfatória, com erro máximo de ordem 10^-8. Percebeu-se que NR-invsqrt tem uma precisão melhor que NR-sqrt, tendo resultado igual ao da biblioteca nativa da linguagem em todos os conjuntos de dados analisados, enquanto NR-sqrt apresentou erro em alguns valores. Quanto a performance, NR-sqrt pareceu ser levemente mais rápida, mas não chegou a demonstrar uma diferença realmente relevante.

A Figura demostra uma comparação entre os métodos para os valores de 0 a 100, incluindo a medição de tempo para 1000000 execuções e o erro comparando com `1 / sqrt(x)`, da biblioteca padrão.



Seguem os resultados e cada iteração para `x = 36`

NR-SQRT

```
Iter: 1 | xk: 1.06250000000000000000 | xk_1: 1.06066176470588224845
Iter: 2 | xk: 1.06066176470588224845 | xk_1: 1.06066017178101734686
Iter: 3 | xk: 1.06066017178101734686 | xk_1: 1.06066017177982141462
Iter: 4 | xk: 1.06066017177982141462 | xk_1: 1.06066017177982141462
Expoente original impar, multiplicando por sqrt(2)
Resultado Aprox. Mantissa: 1.12500000000000000000
Expoente: 2
```

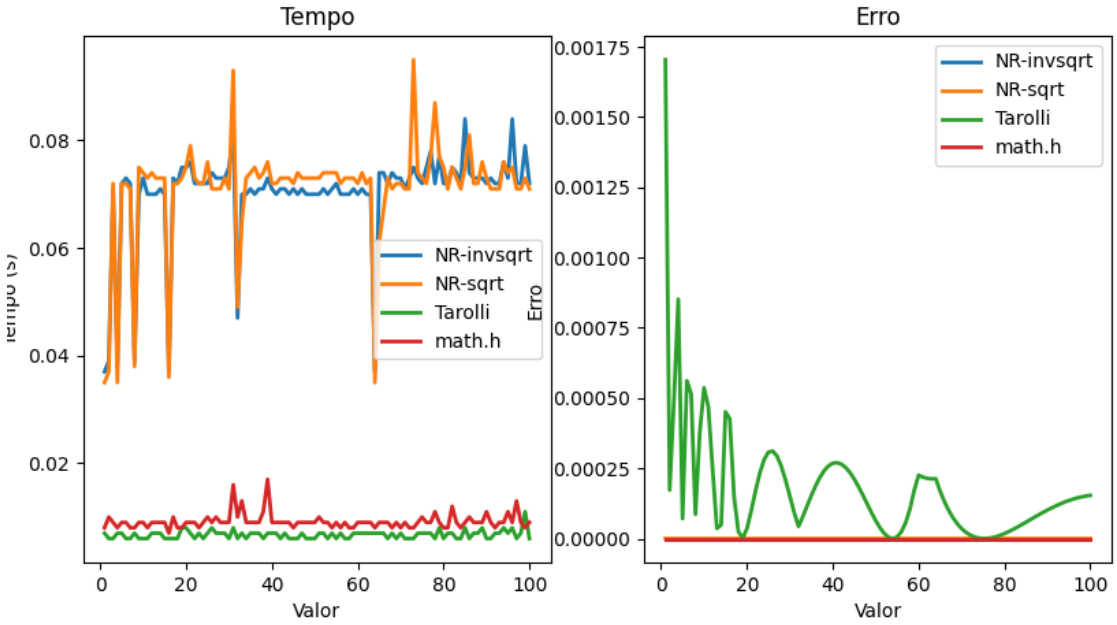
```
Resultado: 1 / 6 = 0.16666667163372039794921875
Erro comparado com a funcao sqrt(): 0.00000000000000000000000000
```

NR-INVSQRT

```
Iter: 1 | xk: 0.94117647058823528106 | xk_1: 0.94280480358233254012
Iter: 2 | xk: 0.94280480358233254012 | xk_1: 0.94280904155348821405
Iter: 3 | xk: 0.94280904155348821405 | xk_1: 0.94280904158206346732
Iter: 4 | xk: 0.94280904158206346732 | xk_1: 0.94280904158206335630
Expoente original impar, multiplicando por sqrt(2)
Resultado Aprox. Mantissa: 1.12500000000000000000
Expoente: -3
NR-invsqrt: 0.16666667163372039794921875
```

3.0.2. NR simples vs Fast Square Root

Agora, analisando como os métodos demostrados se comparam com o terceiro método, de Tarolli. O resultado foi exatamente o esperado: a precisão de Tarolli é bem menor, apesar de ainda aceitável para aplicações como a de sua origem, garantindo apenas 3 casas de precisão contra as 7 de NR-sqrt e 20 de NR-invsqrt. Porém, seu desempenho é bem melhor, sendo aproximadamente 4 vezes mais rápido que os outros dois métodos, sendo inclusive mais rápido que o da biblioteca padrão.



4. Conclusão

No decorrer das atividade, foi possível desenvolver a implementação de um script que realize o cálculo aproximado do inverso da raiz quadrada com três técnicas diferentes e realizar uma comparação tanto entre os resultados quanto entre os próprios processos. Com isso, também foi possível realizar uma análise sobre o erro e seus impactos em métodos de estimação e no contexto geral da computação.

Referências

[Akram and Ann 2015] Akram, S. and Ann, Q. U. (2015). Newton raphson method. *International Journal of Scientific & Engineering Research*, 6(7):1748–1752.

[Hertz et al. 2016] Hertz, E., Thuning, N., Bärring, L., Svensson, B., and Nilsson, P. (2016). Algorithms for implementing roots, inverse and inverse roots in hardware.

[Lomont 2003] Lomont, C. (2003). Fast inverse square root. *Tech-315 nical Report*, 32.