Universidade Estadual de Maringá

Análise da Inversa da Raiz Quadrada

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

Este documento tem como objetivo apresentar uma análise de erro e tempo para o cálculo da inversa da raiz quadrada utilizando diferentes métodos de cálculo para a mesma. No caso, utilizamos as linguagens de programação C e Python para a implementação dos códigos do trabalho.

2 Newton-Raphson

O método de Newton-Raphson é um algoritmo iterativo utilizado para encontrar raízes de equações não lineares, utilizando um x_0 como um valor de chute inicial. Dessa forma, aprimoramos a precisão de nosso resultado a cada iteração utilizando o valor calculado na iteração anterior.

É do nosso interesse utilizarmos o método de Newton-Raphson para calcular a raiz quadrada de um número não negativo. Dessa forma, a raiz quadrada de x é dada por:

$$Z = \frac{1}{2}(x_0 + \frac{x}{x_0})$$

Utilizando a fórmula dada anteriormente, conseguimos calcular a raiz quadrada. Entretanto, queremos calcular a inversa da raiz quadrada. Para isso, conseguimos calculá-la das seguintes formas:

$$Z = \frac{1}{\frac{1}{2}(x_0 + \frac{x}{x_0})} \tag{1}$$

$$Z = x_0(\frac{3}{2} - \frac{1}{2}(x.x_0^2)) \tag{2}$$

Nosso x_0 é dado por uma função de aproximação da raiz quadrada. Segue o código da mesma na linguagem C:

```
float aprox_sqrt(float num){
    union{
        float f;
        uint32_t k;
    } val = {num};

    val.k -= 1 << 23;
    val.k >>= 1;
    val.k += 1 << 29;

    return val.f;
}
```

Note que usamos apenas operações de Bit Shifting para o cálculo desse valor aproximado. Dessa forma, esse cálculo não adiciona muito custo computacional. Já para os dois métodos propostos, temos os seguintes códigos:

```
float newton_raphson(float num, float x0){
    return 1/(0.5 * (x0 + (num/x0))));
 }

float inv_newton_raphson(float num, float x0){
    return x0 * (1.5f - 0.5f * num * x0 * x0);
}
```

Onde o argumento x_0 é calculado pela função de aproximação, vale ressaltar que para a segunda função apresentada acima o argumento calculado para aproximação é $1/x$.

3 Carmack/Tarolli

O método de cálculo da raiz quadrada proposto por John Carmack/-Tarolli é conhecido como "Fast Inverse Square Root"(ou "Rápida Inversa da Raiz Quadrada", em português) e foi uma grande contribuição para jogos e outras aplicações que utilizam computação gráfica.

A ideia principal por trás desse método é minimizar o custo das operações para o cálculo da Inversa da Raiz Quadrada, onde trocamos operações complexas por aritméticas de bit a bit. Dessa forma, esse método é útil para sistemas onde o desempenho é crítico.

A seguir temos o código implementado na linguagem C que representa Carmack:

```
float inv_sqrt(float num){
    const float num2 = 0.5f * num;

    union{
        float x;
        uint32_t k;
    } u = {num};

    u.k = 0x5f3759df - (u.k >> 1);
    u.x = u.x * (1.5f - num2 * u.x * u.x);

    return u.x;
}
```

Note que diferente de Newton-Raphson essa função não necessita de um valor inicial como parâmetro.

4 Análise de Erro

Nesta seção iremos analisar e comparar a magnitude dos erros de cada algoritmo para o cálculo da inversa da raiz quadrada.

Segue uma tabela com alguns valores de exemplo, onde **x** representa o número o qual calculamos a inversa da raiz quadrada, **1/sqrt** a inversa da raiz quadrada fornecida pela linguagem C, **NR1** para o método 1 de Newton-Raphson, **NR2** para o método 2 de Newton-Raphson e **C/T** para Carmack/Tarolli.

x	1/sqrt	NR1	NR2	C/T
1	1	1	1	0.998307
0.5	1.414214	1.411765	1.406250	1.413860
0.02	7.071068	7.070864	7.070449	7.068994
0.001	31.622776	31.622778	31.622776	31.585064

Figura 1: Tabela Valores Calculados

A seguir temos os gráficos de erro de cada método utilizado, onde o erro é dado pela diferença absoluta entre o valor calculado pelo método e a função sqrt que já é padrão na linguagem C.

Nos gráficos temos os seguintes eixos: eixo x onde ele representa os valores que calculamos a inversa da raiz quadrada e o eixo y representando o erro.

4.1 Newton-Raphson Método 1

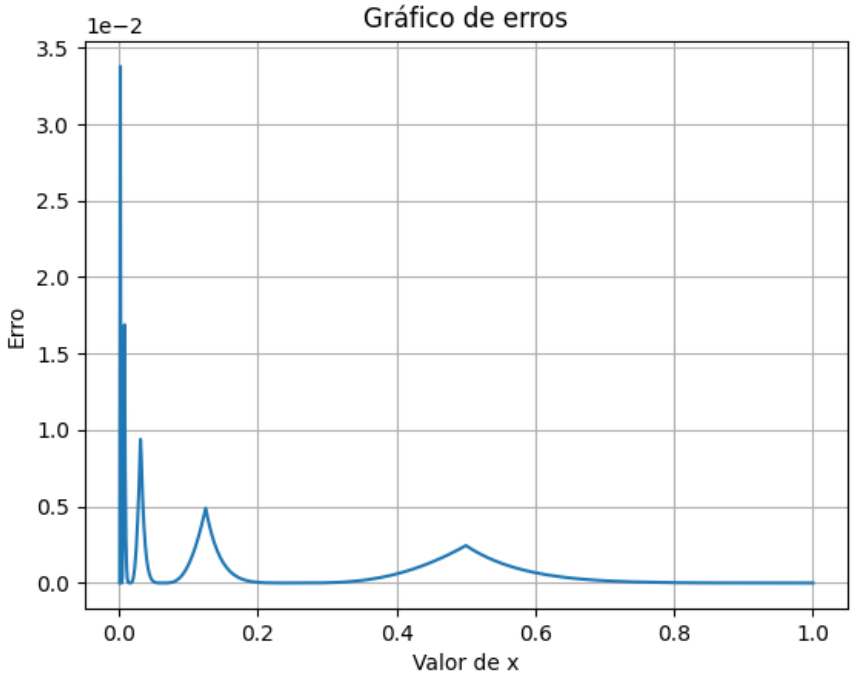

Figura 2: Newton-Raphson Método 1

Por esse gráfico percebemos que esse método nos garante até 1 casa decimal de precisão na maioria dos casos e em alguns intervalos como $[0.2; 0.4]$ e $[0.8; 1.0]$ ele garante duas casas decimais de precisão.

4.2 Newton-Raphson Método 2

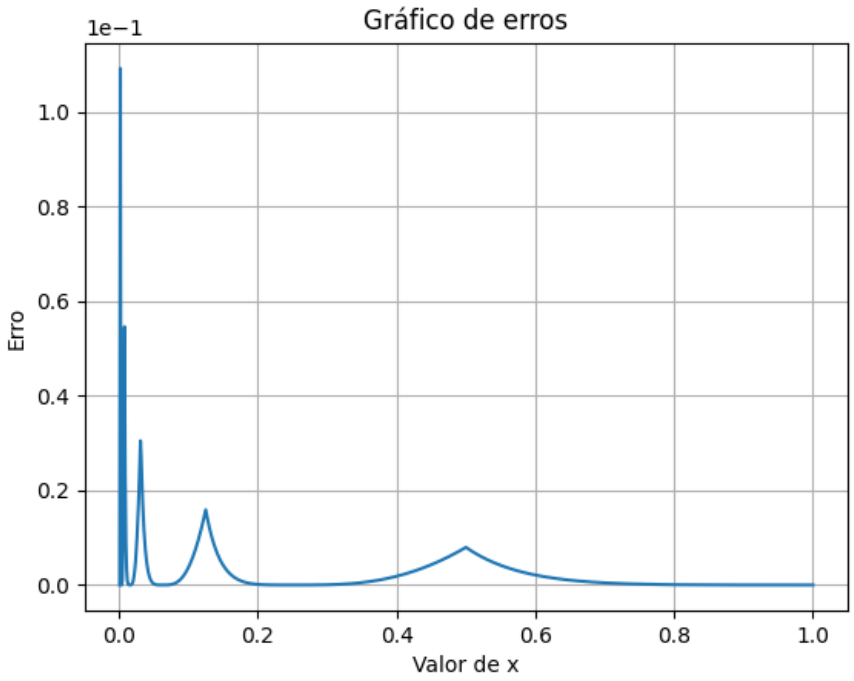

Figura 3: Newton-Raphson Método 2

Esse método apresenta a maior disparidade no erro, indo de valores extremamente altos para uma entrada de valor próximo a zero, porém se estabiliza para números mais altos.

4.3 Carmack/Tarolli

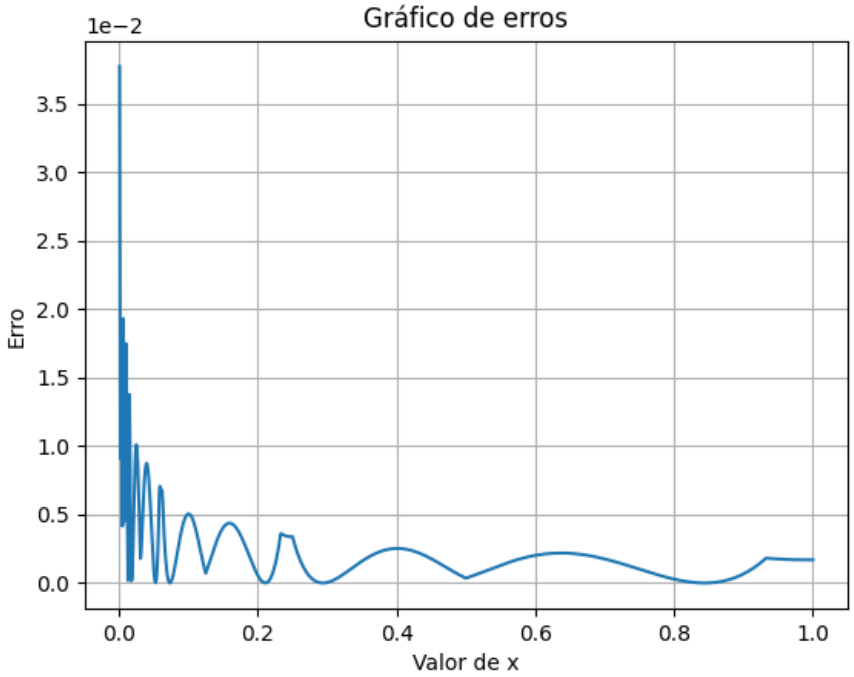

Figura 4: Carmack/Tarolli

Pode-se perceber que esse método possuí a maior instabilidade nos erros dentre os métodos testados.

5 Análise de Tempo

Nesta seção serão apresentados os resultados obtidos fazendo uma análise de tempo simples com os algoritmos implementados.

Para a realização dessa análise utilizamos o seguinte código na linguagem C, considerando os códigos para cada algoritmo apresentados anteriormente, segue o código do mesmo:

```
int main(){
    clock_t start, end;
    double cpu_time_used;

    //NR1
    float num = 0.001;
    start = clock();
    while(num <= 10000){
        float result1 = newton_raphson(num, aprox_sqrt(num));
        num += 0.001;
    }
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("nr1, tempo de execucao: %f segundos\n", cpu_time_used);

    //NR2
    num = 0.001;
    start = clock();
    while(num <= 10000){
        float result1 = inv_newton_raphson(num, aprox_sqrt(1/num));
        num += 0.001;
    }
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("nr2, tempo de execucao: %f segundos\n", cpu_time_used);

    // carmack/tarolli
    num = 0.001;
    start = clock();
    while(num <= 10000){
        float result1 = inv_sqrt(num);
        num += 0.001;
    }
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("tarolli, tempo de execucao: %f segundos\n", cpu_time_used);

    return 0;
}
```

A seguir temos uma tabela apresentando os resultados obtidos executando o código anterior, onde a coluna **Algoritmo** representa o algoritmo em que está sendo utilizado para a execução, **Tempo de Execução** representa o tempo em segundos e **Teste** determina em qual teste esse algoritmo foi executado. Além disso, as siglas **NR1**, **NR2** e **C/T** são respectivamente os algoritmos Newton-Raphson método 1, Newton-Raphson método 2 e Carmack/Tarolli.

Algoritmo	Tempo de Execução	Teste
NR1	0.124539s	1
NR2	0.090738s	1
C/T	0.059834s	1
NR1	0.137658s	2
NR2	0.107164s	2
C/T	0.065614s	2
NR1	0.131743s	3
NR2	0.088176s	3
C/T	0.058444s	3

Figura 5: Tabela Tempo de Execução

Através dos resultados, percebemos na média das execuções o algoritmo de Carmack/Tarolli apresenta um tempo de execução menor comparado aos outros dois.

6 Conclusão

Ao avaliarmos os resultados obtidos em estudo constatamos que o método 1 de Newton-Raphson apresentou a melhor precisão computacional para o problema.

Dessa forma, podemos concluir que a utilização de cada um dos métodos depende do contexto. Caso seja desejado mais precisão podemos utilizar o método 1 de Newton-Raphson, entretanto caso o desempenho seja crítico podemos optar por um método mais rápido como o método de Carmack/Tarolli.