UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

Matemática Computacional(6900)

TRABALHO No.3

Cálculo da raiz quadrada da inversa

Data: 30/03/2023

Professor: Airton Marco Polidorio

Discentes

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

Introdução

Ainda no âmbito da otimização do tempo e custo de processamento procurou-se uma maneira de realizar o cálculo da raiz quadrada da inversa de um número. Um dos algoritmos aqui exemplificados foi utilizado no desenvolvimento do jogo *Quake III Arena* de 1999 para o cálculo do ângulo de incidência e reflexão para a iluminação e sombreamento.

Figura 1. Captura de imagem do jogo *Quake III Arena*

O cálculo da raiz quadrada da inversa de um número geralmente dependia de muitas operações de divisão tornando-se computacionalmente muito custoso e demorado. O algoritmo de Tarolli consegue produzir um resultado com um número de dígitos de acertos alto o suficiente com apenas um passo de divisão, permitindo assim a execução dos gráficos em tempo real existentes em jogos como o *Quake III Arena*.

Objetivo

O presente trabalho tem como objetivo explorar três formas distintas de calcular a raiz quadrada da inversa de um número e analisar suas diferenças quanto ao número de casas decimais de precisão e custo de processamento. Os algoritmos a serem analisados são: cálculo da raiz com Newton-Raphson e posterior inversão, cálculo da inversa da raiz diretamente com Newton-Raphson e o cálculo da inversa da raiz com Tarolli.

Cálculo da raiz com Newton-Raphson e posterior inversão

Esse método de cálculo da inversa da raiz de A propõe que primeiro seja calculado $\sqrt{A}$ e detendo o valor da raiz, a sua inversa.

Para calcularmos $\sqrt{A}$ com o método de Newton-Raphson prosseguimos da seguinte maneira:

Pelo método de Newton-Raphson temos que:

$$(1)\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Dado que:

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

Então:

$$(3)\ f'(x) = 2x$$

Substituindo (2) e (3) em (1) tem-se:

$$(4)\ x_{k+1} = x_k - \frac{x_k^2 - A}{2x_k}$$

Simplificando (4), temos:

$$0.5 * \left(x_k + \frac{A}{x_k}\right)$$

Quando k = 0 um bom chute para X_0 é $aproximacao\_da\_raiz(A)$

```python
class union(Union):
    _fields_ = [("x", c_float),
                ("k", c_int32)]

def aproximacao_da_raiz(A):
        val= union()
        val.x = A
        val.k -= 1<<23
        val.k >>=1
        val.k += 1<<29
        return val.x
```

A operação em (4) é realizada repetidas vezes até chegar-se ao número de dígitos de precisão desejados. Após determinar-se o valor de $\sqrt{A}$ faz-se a operação de divisão para encontrar-se $\frac{1}{\sqrt{A}}$.

Cálculo da inversa da raiz diretamente com Newton-Raphson

O cálculo de $\frac{1}{\sqrt{A}}$ é dado pela seguinte forma:

Pelo método de Newton-Raphson temos que:

$$(1)\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$Fazendo\ \frac{1}{\sqrt{A}} = x \Rightarrow x^2 = \left(\frac{1}{\sqrt{A}}\right)^2 \Rightarrow \frac{1}{A} = x^2 \Rightarrow A = \frac{1}{x^2} \Rightarrow$$

$$(2)\ f(x) = \frac{1}{x^2} - A = 0$$

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

$$Aplicando\ em\ (2)\ e\ (3)\ em\ (1), temos:$$

$$x_{k+1} = x_k - \left(\frac{\frac{1}{x_k^2} - A}{\frac{-2}{x_k^3}}\right) = x_k - \left[\left(\frac{1}{x_k^2} - A\right).\frac{x_k^3}{-2}\right] = x_k - \frac{x_k^3}{-2x_k^2} - \frac{Ax_k^3}{2}$$

$$x_k - \frac{x_k}{-2} - \frac{Ax_k^3}{2} = x_k + \frac{x_k}{2} - \frac{Ax_k^3}{2} = \frac{3x_k}{2} - \frac{Ax_k^3}{2} =$$

$$x_k.\left(\frac{3}{2} - \frac{Ax_k^2}{2}\right)$$

Assim, pode-se afirmar que para se calcular a inversa da raiz quadrada diretamente com Newton-Raphson basta calcularmos pela seguinte fórmula:

$(4)\ x_{k+1} = x_k.\left(\frac{3}{2} - \frac{Ax_k^2}{2}\right)$,onde k é o número de iterações e XO é o chute inicial.

Quando k = 0 um bom chute para X_0 é $\frac{1}{aproximacao\_da\_raiz(A)}$.

A operação em (4) é realizada repetidas vezes até chegar-se ao número de dígitos de precisão desejados. Após k iterações a operação devolve $\frac{1}{\sqrt{A}}$.

Cálculo com o algoritmo de Tarolli

Para se realizar o cálculo da inversa de $\sqrt{A}$ podemos utilizar o algoritmo de Tarolli.

```
def raiz_inversa_tarolli(x):
    x2 = 0.5*x
    u = union()
    u.x  = x
    u.k = 0x5f3759df - (u.k >> 1)
    u.x = u.x *(1.5 - x2*u.x*u.x)
    return u.x
```

O algoritmo de Tarolli é uma versão otimizada do algoritmo “fast inverse square root” utilizado em Quake III Arena. Ele funciona ao tratar o ponto flutuante de 32-bits como um inteiro de 32-bits e realizando manipulações que se aproximam da inversa da raiz quadrada. O algoritmo foi otimizado para utilizar apenas uma iteração de Newton-Raphson para aumentar sua velocidade.

Explicando de uma maneira detalhada e passo a passo:

1. A função recebe um único parâmetro ‘x’, que é o número em que queremos calcular a inversa da raiz quadrada.
2. A variável x2 é iniciada como a metade de x.
3. Um objeto do tipo union é criado. Ela é um tipo de data que permite ver a mesma parte da memória como dois tipos de dados diferentes (um inteiro e ponto flutuante).
4. O valor de x é armazenado na propriedade x do objeto union.
5. A propriedade k do objeto union inicia com o valor constante ‘0x5f3759df’, que é um número mágico utilizado no algoritmo de Tarolli.
6. A propriedade k do objeto union é shiftado para direita por um bit.
7. A propriedade k do objeto union é subtraído pelo valor original de (‘0x5f3759df’) e o resultado é armazenado em k.
8. O valor de x é multiplicado pela expressão (1.5 - x2*u.x*u.x).
9. O resultado final é armazenado na propriedade x do objeto union e é retornado pela função.

O algoritmo de Tarolli funciona utilizando uma combinação de manipulação de bit e aritmética de ponto flutuante para se aproximar à inversa da raiz quadrada de um número.

Código

Python

```python
from ctypes import c_float, c_int32,Union
import ctypes
import math
import time
import matplotlib.pyplot as plt

class union(Union):
    _fields_ = [("x", c_float),
                ("k", c_int32)]


def desenhar_ponto(ponto,color,text,i,j):
```

```
    plt.plot(ponto[0], ponto[1], marker="o", markersize=5,
markeredgecolor=color,
markerfacecolor=color,label=text,)#Posicao real
    # plt.set_title(text)
    plt.legend()

def raiz_quadrada_newton_rapson(A):
    x0 =aproximacao_da_raiz(A)
    xk = x0
    x_k_1 = 0.5*(xk+A/xk)
    xk  = x_k_1
    return xk

def raiz_inversa_newton_rapson(A):
    D = 0.5*A #constante
    C = 1/aproximacao_da_raiz(A)#constante
    x0 =C
    xk = x0
    x_k_1 = xk*(1.5 - D*(xk**2))
    xk  = x_k_1

    return xk

def raiz_inversa_tarolli(x):
    x2 = 0.5*x
    u = union()
    u.x  = x
    u.k = 0x5f3759df - (u.k >> 1)
    u.x = u.x *(1.5 - x2*u.x*u.x)
    return u.x

def raiz_calculadora(x):
    return math.sqrt(x)

def aproximacao_da_raiz(A):
        val= union()
        val.x = A
        val.k -= 1<<23
        val.k >>=1
```

```
        val.k += (1<<29)
        return val.x
def main():
    N = 1000

erro_list_calculadora_raiz_newton_rapson,erro_list_calculadora
_inversa_taroli,erro_list_calculadora_inversa_direto_newton_ra
pson = [],[],[]
    x_list = []
    inversa_raiz_newton_rapson_list = []
    inversa_direto_newton_rapson_list = []
    inversa_taroli_list = []
    inversa_calculadora_list = []

    tempo_inversa_raiz_newton_rapson = []
    tempo_inversa_direto_newton_rapson = []
    tempo_inversa__tarolin = []
    tempo_inversa_calculadora = []
    for x in range(1,N,1):

        start_raiz_quadrada_newton_rapson = time.time()
        raiz = raiz_quadrada_newton_rapson(x)

        inversa_raiz_newton_rapson = 1/raiz
        fim_raiz_quadrada_newton_rapson = time.time()

tempo_inversa_raiz_newton_rapson.append(fim_raiz_quadrada_newt
on_rapson-start_raiz_quadrada_newton_rapson)


inversa_raiz_newton_rapson_list.append(inversa_raiz_newton_rap
son)

        start_inversa_direto_newton_rapson =  time.time()
        inversa_direto_newton_rapson  =
raiz_inversa_newton_rapson(x)
        fim_inversa_direto_newton_rapson = time.time()
```

```
tempo_inversa_direto_newton_rapson.append(fim_inversa_direto_n
ewton_rapson-start_inversa_direto_newton_rapson)

inversa_direto_newton_rapson_list.append(inversa_direto_newton
_rapson)


        start_inversa_taroli=  time.time()
        inversa_taroli = raiz_inversa_tarolli(x)
        fim_inversa_taroli =  time.time()


tempo_inversa__tarolin.append(fim_inversa_taroli-start_inversa
_taroli)
        inversa_taroli_list.append(inversa_taroli)

        start_inversa_calculadora = time.time()
        inversa_calculadora = 1/raiz_calculadora(x)
        fim_inversa_calculadora = time.time()


tempo_inversa_calculadora.append(fim_inversa_calculadora-start
_inversa_calculadora)
        inversa_calculadora_list.append(inversa_calculadora)

        #Calculo dos erros
        erro_calculadora_raiz_newton_rapson =
abs(inversa_calculadora - inversa_raiz_newton_rapson)
        erro_calculadora_inversa_taroli =
abs(inversa_calculadora - inversa_taroli  )
        erro_calculadora_inversa_direto_newton_rapson =
abs(inversa_calculadora - inversa_direto_newton_rapson )


erro_list_calculadora_raiz_newton_rapson.append(erro_calculado
ra_raiz_newton_rapson)
```

```
erro_list_calculadora_inversa_taroli.append(erro_calculadora_i
nversa_taroli)

erro_list_calculadora_inversa_direto_newton_rapson.append(erro
_calculadora_inversa_direto_newton_rapson)
        x_list.append(x)

        #
print(inversa_calculadora,inversa_raiz_newton_rapson,inversa_d
ireto_newton_rapson,inversa_taroli)
    #Gráficos


plt.plot(x_list,erro_list_calculadora_inversa_direto_newton_ra
pson,label='Calculadora X inversa_direto_newton_rapson ')

plt.plot(x_list,erro_list_calculadora_inversa_taroli,label='Ca
lculadora X Tarolli')

plt.plot(x_list,erro_list_calculadora_raiz_newton_rapson,label
='Calculadora X raiz_newton_rapson')

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

    plt.plot(x_list,inversa_raiz_newton_rapson_list,label =
"1/newton_rapson")
    plt.plot(x_list,inversa_direto_newton_rapson_list,label =
"direta newton_rapson")
    plt.plot(x_list,inversa_taroli_list,label = "Tarolli")
    plt.plot(x_list,inversa_calculadora_list,label =
"Calculadora")

    plt.yscale('log')
    plt.xlabel('Argumento')
    plt.ylabel('Valores')
```

```
    plt.legend()
    plt.show()

    plt.plot(x_list,tempo_inversa_raiz_newton_rapson,label =
"1/newton_rapson")
    plt.plot(x_list,tempo_inversa_direto_newton_rapson,label =
"direta newton_rapson")
    plt.plot(x_list,tempo_inversa__tarolin,label = "Tarolli")
    plt.plot(x_list,tempo_inversa_calculadora,label =
"Calculadora")

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


main()
```

OBS:
Para se calcular X_0, o chute inicial, foi utilizado o seguinte algoritmo:

```
def aproximacao_da_raiz(A):
        val= union()
        val.x = A
        val.k -= 1<<23
        val.k >>=1
        val.k += (1<<29)
        return val.x
```

Comparação entre os métodos

• Tabela 01 - Cálculo da inversa da raiz

X	1/sqrt(X)	Raiz depois inversão - NR	Inversa da raiz Direta - NR	Tarolli
1	1	1	1	0.9983071684837341
2	0.7071067811865475	0.7058823529411765	0.7037037037037037	0.7069300413131714
3	0.5773502691896258	0.5773195876288659	0.5772594752186588	0.5768468379974365
4	0.5	0.5	0.5	0.4991535842418670 7
5	0.4472135954999579	0.4472049689440994	0.4471879286694 1015	0.4471410214900 9705
6	0.4082482904638631	0.4081632653061224	0.4080000000000 0003	0.4076813757419586
7	0.3779644730092272	0.3776824034334764	0.3771600300525 92	0.3774441480636 5967
8	0.3535533905932 7373	0.3529411764705 826	0.3518518518518 5186	0.3534650206565 857
9	0.3333333333333 333	0.3330557868442 9643	0.3325439999999 9995	0.3329532146453 8574
10	0.3162277660168 3794	0.3161094224924 012	0.3158852981338 188	0.3156857788562 7747

• Tabela 02 - Cálculo do Erro

Utilizando o 1/sqrt(x) como base de comparação para o cálculo do erro.

X	Raiz depois inversão - NR	Inversa da raiz Direta - NR	Tarolli
1	0	0	0.0016928315162658691
2	0.0012244282453709454	0.0034030774828437416	0.000176739873376075
3	3.068156075991535e-05	9.079397096700692e-05	0.0005034311921893186
4	0	0	0.0008464157581329346
5	8.626555858537444e-06	2.5666830547776964e-05	7.25740098608818e-05
6	8.502515774067021e-05	0.00024829046386304343	0.0005669147219044546
7	0.0002820695757507763	0.00080444295663517 63	0.0005203249455675296
8	0.0006122141226854727	0.00170153874142187 08	8.83699366880375e-05
9	0.0002775464890368395	0.00078933333333336 41	0.0003801186879475726
10	0.0001183435244367370 1	0.00034246788301911524	0.0005419871605604754

Podemos observar através do cálculo do erro:

- Raiz depois inversão (Newton-Raphson): utilizando o chute inicial com o algoritmo de aproximação da raiz, conseguimos atingir uma precisão de pelo menos 2 casas decimais após a vírgula, às vezes até mais, chegando a 5 casas decimais de precisão.
- Inversa da raiz direta (Newton-Raphson): igualmente como o algoritmo anterior, utilizando o chute inicial com o algoritmo, obtivemos um resultado de pelo menos 2 casas decimais de precisão, podendo alcançar precisões maiores a depender do valor do argumento.
- Algoritmo de Tarolli: o algoritmo de Tarolli garante pelo menos uma precisão de duas casas decimais após a vírgula, sendo bem consistente em sua precisão como mostrado na tabela do cálculo do erro.

• Tabela 03 - Tempo de execução (ns)

X	1/sqrt(X)	Raiz depois inversão - NR	Inversa da raiz Direta - NR	Tarolli
1	1.430511474609375e-06	8.58306884765625e-06	3.0994415283203125e-06	6.914138793945312 5e-06
2	1.192092895507812 5e-06	5.960464477539062 5e-06	7.152557373046875e-06	2.861022949218 75e-06
3	1.430511474609375e-06	6.437301635742187 5e-06	4.768371582031 25e-06	3.337860107421 875e-06
4	1.192092895507812 5e-06	7.62939453125e-06	4.291534423828125e-06	2.861022949218 75e-06
5	7.152557373046875e-07	4.768371582031 25e-06	2.861022949218 75e-06	1.907348632812 5e-06
6	7.152557373046875e-07	4.053115844726562 5e-06	3.0994415283203125e-06	2.384185791015 625e-06
7	9.536743164062 5e-07	4.529530029296875e-06	2.861022949218 75e-06	2.145767211914 0625e-06
8	9.536743164062 5e-07	4.768371582031 25e-06	2.622604370117187 5e-06	2.145767211914 0625e-06
9	7.152557373046875e-07	.4.291534423828125e-06	3.337860107421 875e-06	2.861022949218 75e-06
10	4.768371582031 25e-07	4.291534423828125e-06	2.861022949218 75e-06	2.145767211914 0625e-06

Podemos analisar em relação ao tempo de execução que, os 3 algoritmos, os 2 por Newton-Raphson e o algoritmo de Tarolli obtiveram resultados bem parecidos, não tendo uma diferença de tempo significativa, podemos observar também que ao utilizar a função padrão da linguagem para o cálculo da raiz quadrada (sqrt(x)), obtivemos o melhor resultado em relação à tempo de execução.

Gráfico 1 - Cálculo da inversa da raiz

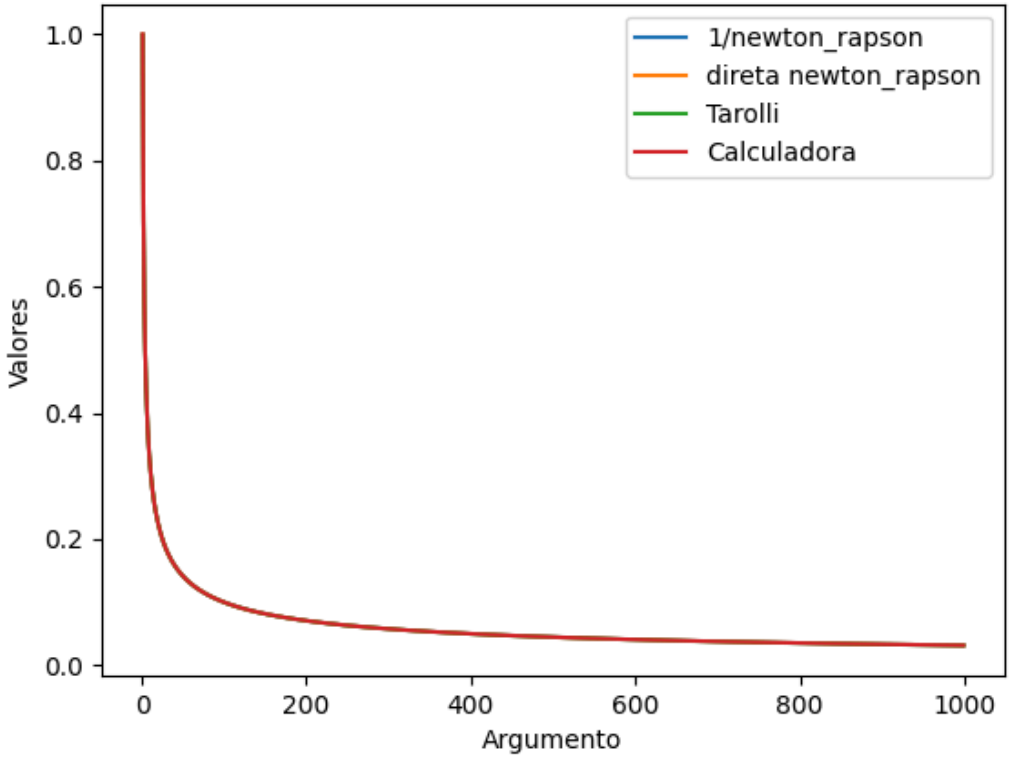


Gráfico 2 - Cálculo do Erro

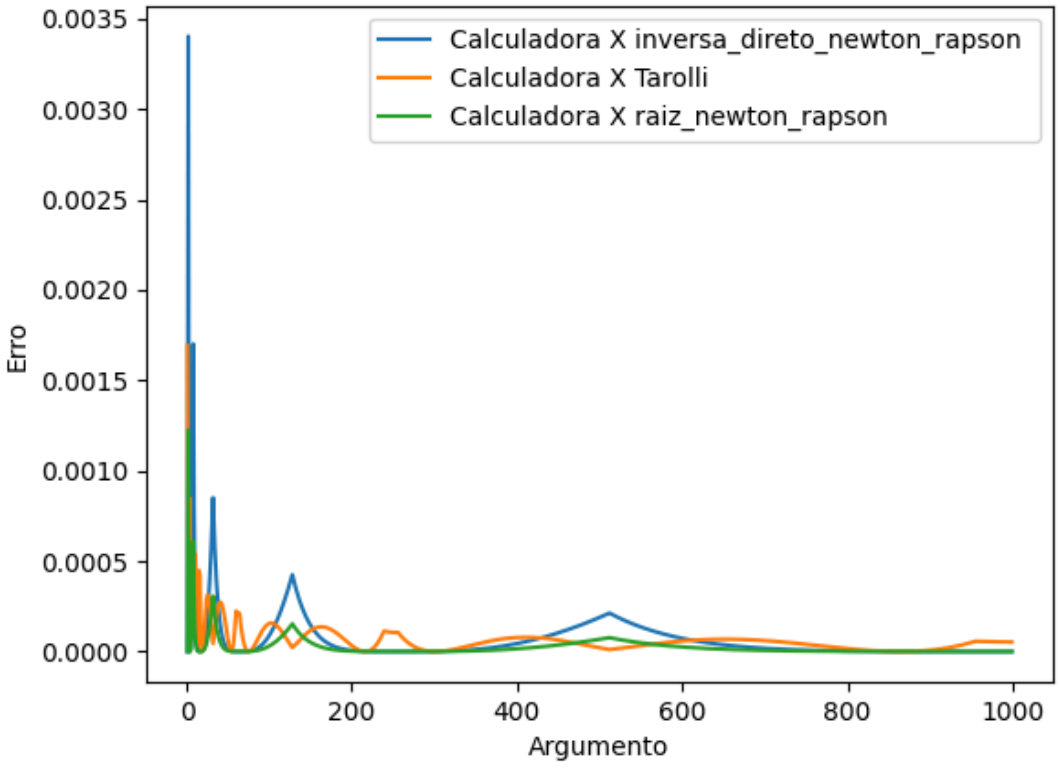
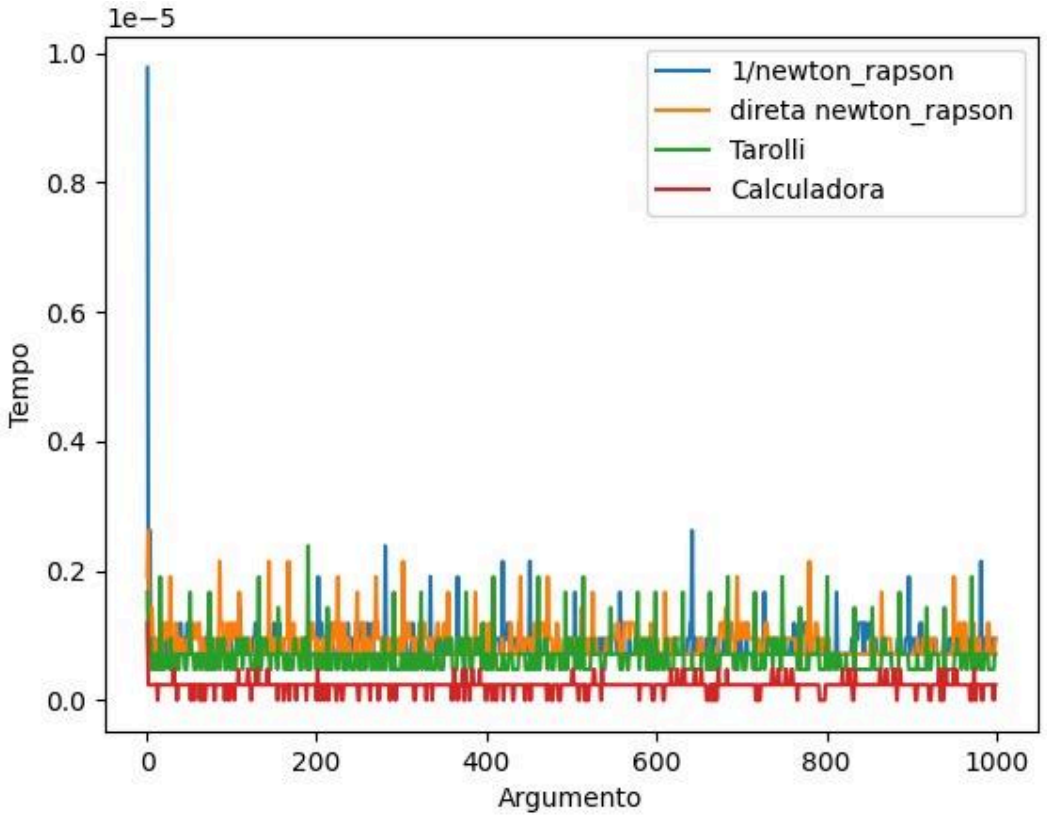


Gráfico 3 - Tempo de execução



Conclusão

Em conclusão, foram exploradas três formas distintas de calcular a raiz quadrada da inversa de um número: cálculo da raiz com Newton-Raphson e posteriormente a inversão, cálculo da inversa da raiz diretamente com Newton-Raphson e o cálculo da inversa da raiz com Tarolli. Foi analisado o número de casas decimais de precisão e o custo de tempo para processamento de cada algoritmo. Concluiu-se que o algoritmo de Tarolli possui uma precisão alta o suficiente para ser utilizado em tempo real em jogos, como o Quake III Arena, e possui um tempo de processamento eficiente. A escolha do método adequado a ser utilizado para se calcular a raiz quadrada da inversa de um número deve ser considerado as necessidades específicas de cada projeto, em relação a precisão e custo de processamento.