Cálculo da Raiz Quadrada pelo Método de Newton-Raphson através do Padrão IEEE-754 para Ponto-Flutuante

Ricardo Henrique Brunetto[1]

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

ra94182@uem.br

1. Introdução

Deseja-se obter uma maneira de estimar a raiz quadrada com base no método iterativo de Newton-Raphson, que realiza a estimativa de raízes de polinômios e equações transcedentais através da expansão da Série de McLaurin-Taylor.

Tal método permite obter uma precisão específica através de convergência quadrática (se houver).

Este artigo fará uma breve explicação do método de Newton-Raphson e do padrão IEEE 754 de armazenamento de ponto-flutuante que servirão de base para o desenvolvimento da estratégia utilizada. Não serão feitas explanações a respeito da Série de Taylor-McLaurin, visto que foge ao foco do artigo.

2. Fundamentação Teórica

Aqui serão abordados dois tópicos principais em que se baseará o cálculo da raiz quadrada posteriormente na Seção 3: o método de Newton-Raphson e o padrão IEEE 754.

2.1. Método de Newton-Raphson para determinação de raízes

O Método de Newton-Raphson se aplica aos polinômios e às equações transcedentais para determinar soluções de sistemas de equações simultâneas, com várias variáveis. No escopo deste artigo, seu uso será restringido para determinação de raízes em equações polinomiais com uma única variável e, portanto, será feito um breve resumo a respeito de seu funcionamento.

O método de Newton-Raphson se principia com uma raiz aproximada, x_k, de uma equação $f(x) = 0$, e se vale da expansão em série de Taylor para formular um algoritmo computacional que define uma fórmula de recursão para uso.

O desenvolvimento em série de Taylor de uma função nas vizinhanças do ponto x_k tem a forma

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)$$

onde $f'(X_i)$ indica a derivada primeira de $f(x_k)$ calculada no ponto $(x_k, f(x_k))$.

Dessa forma, dispõe-se da reta tangente ao gráfico da função $f(x)$, traçada no ponto x_k, para deteminação da próxima estimativa. Nota-se, que, ao admitir que a função $f(x)$ aproxima-se de sua(s) raiz(es), ocorre $f(x) = 0$ e, portanto, da equação acima:

$$f(x_k) + f'(x_k)(x - x_k) = 0$$

Alternativamente, pode-se expressar tal equação como sendo:

$$f'(x_k) = \frac{f(x_k)}{x_k - x}$$

Admitindo x como a próxima estimativa, sendo esta mais satisfatória que x_k, tem-se a seguinte equação de recorrência, que caracteriza o Método de Newton-Raphson:

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

2.2. Padrão IEEE 754

Aqui será realizada uma curta abordagem ao padrão IEEE 754 para armazenamento e aritmética binária. Não serão realizadas considerações a respeito da quantidade de bits nos diferentes tipos de precisão que o padrão oferece. Nesse caso, o foco torna-se especificamente a maneira como os números são acomodados na máquina para entendimento de como se pode manipular esta estrutura a fim de obter cálculos mais rápidos e precisos.

Qualquer número x pode ser representado por $x = mb^t$, onde m é a mantissa, b é a base e t é o expoente.

Um número decimal x está representado no padrão IEEE 754 da seguinte forma:

$$x_{10} = (-1)^{S_2}(1 + M_{10})2^{E_{10}-1023}$$

onde:

- R_b implica que R está na base b;
- S é o valor do bit do sinal;
- M é o valor da soma dos bits da mantissa;
- E é o valor do expoente (onde $E \in \mathbb{N}$, pois é enviasado);
- 1023 refere-se ao **BIAS** supondo 11 bits de precisão do expoente.

A estrutura permite trabalhar com mantissa M_2 (supondo m bits de precisão) de duas formas: inteira ($v_z(M) \in \mathbb{N}$) ou fracionada ($v_f(M) \in \mathbb{R}$).

$$v_z(M) = \sum_{k=0}^{m-1} M_b 2^b$$

$$v_f(M) = \sum_{k=1}^{m} M_k 2^{-k}$$

onde b e k indexam o "vetor" de bits da mantissa conforme ilustra a figura abaixo.

Salienta-se que, de qualquer forma, $v_z(M) = v_f(M)$.

Assim, passam a existir duas maneiras de se trabalhar com a recuperação do valor armazenado:

$$x = (-1)^S(1 + v_f(M))2^{E-1023}$$

ou

$$x = (-1)^S(1 + \frac{v_z(M)}{2^m})2^{E-1023}$$

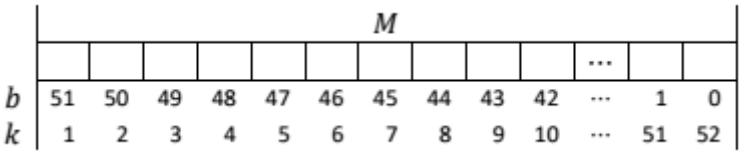

Figure 1. Indexação da mantissa

3. Desenvolvimento

Deseja-se, nesta etapa, desenvolver um método para aplicar Newton-Raphson aproveitando-se do padrão IEEE 754 para calcular a raiz quadrada de um número A em base 10. Sabe-se que

$$x = \sqrt{A} \implies x^2 = A \implies f(x) = x^2 - A = 0$$

Nota-se que encontrar a raiz quadrada de A significa encontrar o zero da função $f(x)$. Para tal utilizar-se-á Newton-Raphson.

Da Seção 2.2, sabe-se que $A = m_a 2^{e_a}$. Da mesma forma, $x = m_x 2^{e_x}$.

Ao aplicar Newton-Raphson em $f(x)$, tendo x_k como estimativa inicial, tem-se:

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

Percebe-se que o expoente se torna desnecessário para a inclusão no métoodo de Newton-Raphson. Isso porque, matematicamente, tem-se para bases 2:

$$\sqrt{x2^k} = \begin{cases} \sqrt{x}2^{\frac{k}{2}}, & k \, mod \, 2 = 0 \\ \sqrt{2x}2^{\frac{k-1}{2}}, & k \, mod \, 2 = 1 \end{cases}$$

Dessa forma, é necessário calcular, inevitavelmente, a raiz da **mantissa** ($\sqrt{x}$). Portanto, as estimativas devem ter como base o valor da própria mantissa de A. Nesse ínterim, o expoente serve meramente para verificar o valor a ser retornado e ajustar os cálculos da estimativa inicial e valor final de retorno. Assim, as iterações do método de Newton-Raphson percorrem na mantissa da estimativa inicial.

Tal estimativa inicial é calculada com base na própria mantissa de A, sendo:

$$x_0 = m_A * 0.5 + 1.0$$

Ou seja, a estimativa inicial (x_0) consiste na metade do valor da mantissa de A (m_A) acrescido do bit escondido.

Essa estimativa, em geral, corresponde a uma conversão mais apurada (em menos iterações) para o valor da raiz.

4. Algoritmo

Antes de aplicar Newton-Raphson, definem-se algumas operações que podem ser feitas através do padrão IEEE-754. É importante especificar uma devida precisão. Existem duas

formas básicas de fazer isso no método: especificar o número de iterações; especificar o erro relativo a ser tolerado (diferença nos resultados entre as iterações). Por motivos de precisão, escolher-se-á a segunda opção. Assim, define-se:

```
#define EPSILON 1e-20
```

Em seguida, iniciar-se-á pela definição da função `doubleToIeee`, converte um valor real de ponto-flutuante na estrutura IEEE-754 definida previamente.

```
void doubleToIeee(double valor, int *s, int *exp, double *mant){
  *exp = floor(log2(valor));
  if(valor < 0){
    valor = valor*(-1);
    *s = 1;
  }else{
    *s = 0;
  }
  *mant = valor/(pow(2, *exp)) - 1.0;
}
```

Com objetivo oposto, defini-se a função `ieeeToDouble`:

```
double ieeeToDouble(double mantissa, int expoente, int sinal){
  return (mantissa * pow(-1, sinal) * pow(2, expoente));
}
```

Deseja-se, neste momento, deseja-se construir uma função que faça o cálculo da raiz quadrada. Para tanto, fazer-se-á uso da técnica de Newton-Raphson, discorrida na Seção 2.

Encontrar a raiz quadrada de um número A significa encontrar a raiz da função $f(x) = x^2 - A$. Nota-se que $f'(x) = 2x$. Deseja-se, também, obter grande precisão. Para tanto, deve-se utilizar um valor reduzido de argumento. Nesse ínterim, uma boa estimativa inicial para o método de Newton-Raphson é a mantissa do valor no padrão IEEE-754, uma vez que a mantissa é sempre maior que zero e menor que 2.

Além do chute inicial, o padrão IEEE-754 pode ser utilizado no cálculo sumário do método Newton-Raphson. Isso significa que, em vez de calcular a raiz da função (valor decimal), calcula-se a mantissa da raiz da função. A manipulação da mantissa fica a cargo da seguinte lógica: basta dividí-lo por 2 (raiz quadrada), se for par; caso contrário, decrementa-se o expoente e corrige-se a mantissa na devolução do resultado (multiplicando-a por $\sqrt{2}$).

Dessa forma, tem-se, seguindo o método de Newton-Raphson, o cálculo da Raiz Quadrada:

```
double raiz_quadrada (double mantissa, int expoente){
  int isImpar = 0, k = 1;
  if(*expoente & 1){
    isImpar = 1;
    expoente--;
  }
  expoente *= 0.5;
  double xk_1 = mantissa * 0.5 + 1.0;
  mantissa += 1.0;
  double xk = 0, aux;
  do{
```

```
    xk = xk_1;
    xk_1 = xk - (((xk*xk - mantissa)/xk)*0.5);
  }while((fabs(xk_1 - xk) > EPSILON));
  if(isImpar) return (xk_1 * 1.41421356237309504880168872420969807);
  return xk_1;
}
```

onde $1.41421356237309504880168872420969807 = \sqrt{2}$.

Dessa forma, calcula-se a raiz quadrada através do método Newton-Raphson com o padrão IEEE-754 com precisão absoluta de 19 casas decimais e dúvida na vigésima casa.

5. Casos Teste

Testar-se-á a convergência para o cálculo das seguintes raízes:

$$A = 36$$

```
Expoente: 2
#1      [xk]: 1.06250000000000000000    [xk+1]: 1.06066176470588220000
#2      [xk]: 1.06066176470588220000    [xk+1]: 1.06066017178101730000
#3      [xk]: 1.06066017178101730000    [xk+1]: 1.06066017177982120000
#4      [xk]: 1.06066017177982120000    [xk+1]: 1.06066017177982120000

RAIZ CONVERGIDA: 1.50000000000000000000 VALOR DECIMAL: 6.00000000000000000000
```

$$A = 2.58$$

```
Expoente: 0
#1      [xk]: 1.14500000000000000000    [xk+1]: 1.13581877729257650000
#2      [xk]: 1.13581877729257650000    [xk+1]: 1.13578166976623130000
#3      [xk]: 1.13578166976623130000    [xk+1]: 1.13578166916005460000
#4      [xk]: 1.13578166916005460000    [xk+1]: 1.13578166916005460000

RAIZ CONVERGIDA: 1.60623784042090100000 VALOR DECIMAL: 1.60623784042090100000
```

$$A = 3.625$$

```
Expoente: 0
#1      [xk]: 1.40625000000000000000    [xk+1]: 1.34756944444444440000
#2      [xk]: 1.34756944444444440000    [xk+1]: 1.34629180802485050000
#3      [xk]: 1.34629180802485050000    [xk+1]: 1.34629120178376250000
#4      [xk]: 1.34629120178376250000    [xk+1]: 1.34629120178362590000
#5      [xk]: 1.34629120178362590000    [xk+1]: 1.34629120178362590000

RAIZ CONVERGIDA: 1.90394327646597720000 VALOR DECIMAL: 1.90394327646597720000
```

$$A = 27$$

```
Expoente: 2
#1      [xk]: 1.34375000000000000000    [xk+1]: 1.29978197674418610000
#2      [xk]: 1.29978197674418610000    [xk+1]: 1.29903831853703580000
#3      [xk]: 1.29903831853703580000    [xk+1]: 1.29903810567667530000
#4      [xk]: 1.29903810567667530000    [xk+1]: 1.29903810567665800000
#5      [xk]: 1.29903810567665800000    [xk+1]: 1.29903810567665800000

RAIZ CONVERGIDA: 1.29903810567665800000 VALOR DECIMAL: 5.19615242270663200000
```

$$A = 15874$$

```
Expoente: 6
#1      [xk]: 1.46887207031250000000    [xk+1]: 1.39403879423544890000
#2      [xk]: 1.39403879423544890000    [xk+1]: 1.39203023492146820000
#3      [xk]: 1.39203023492146820000    [xk+1]: 1.39202878584715340000
#4      [xk]: 1.39202878584715340000    [xk+1]: 1.39202878584639910000
#5      [xk]: 1.39202878584639910000    [xk+1]: 1.39202878584639910000

RAIZ CONVERGIDA: 1.96862598815773040000 VALOR DECIMAL: 125.99206324209474000000
```

$$A = 6.023e23$$

```
Expoente: 39
#1      [xk]: 1.49642176588137720000    [xk+1]: 1.41408048508089460000
#2      [xk]: 1.41408048508089460000    [xk+1]: 1.41168313691175000000
#3      [xk]: 1.41168313691175000000    [xk+1]: 1.41168110129975810000
#4      [xk]: 1.41168110129975810000    [xk+1]: 1.41168110129829040000
#5      [xk]: 1.41168110129829040000    [xk+1]: 1.41168110129829040000

RAIZ CONVERGIDA: 1.41168110129829040000 VALOR DECIMAL: 776079892794.54980000000000000000
```

A precisão se mantém estável para valores pequenos de A. Isso ocorre porque o cálculo da raiz é executado a partir da mantissa do valor de A no padrão IEEE-754 e encontra a mantissa da raiz. No caso de valores grandes, a mantissa também é definida com precisão, porém o valor decimal final é comprometido visto que boa parte da mantissa representa os números da parte inteira, como é o último caso teste, da Constante de Avogadro.

6. Conclusão

Em relação ao algoritmo desenvolvido, fazer-se-á uma análise a partir de cada iteração do método. Nota-se que a quantidade de iterações depende do tempo de convergência em relação à precisão desejada.

Dessa forma, conclui-se que é possível aplicar Newton-Raphson onde cada iteração necessita de:

- 1 comparação
- 3 subtrações
- 2 multiplicações
- 1 divisão

As operações de pré-processamento e pós-processamento não são contabilizadas nesta análise, pois ocorrem somente uma única vez e não interferem significativamente na precisão. Além disso, as operações que envolvem divisão por 2 (multiplicação por 0.5) ou multiplicação por 2 podem ser feitas pela máquina através de *bit-shift*.

Provavelmente é possível executar melhoras nesta técnica, talvez através do tratamento da mantissa como um número inteiro, para que se possa executar *bit-shift*. Isso substituiria as multiplicações e divisões por 2. Contudo, para aplicação em software, a conversão requer aplicação de uma fórmula que pode não compensar o uso do deslocamento de bits. Para tanto, é necessário um estudo mais profundo.