PADRÃO IEEE 754: REPRESENTAÇÃO EM PONTO FLUTUANTE E EFEITOS DO ARREDONDAMENTO

Diogo Lepri Moreira[1]
Raphael Garcia Palma [2]

RESUMO

Este artigo apresenta um estudo sobre o padrão IEEE 754 de precisão simples, amplamente utilizado na representação de números em ponto flutuante pelos computadores. O objetivo principal é compreender como valores decimais são convertidos para um formato binário normalizado, limitado por uma quantidade fixa de bits, e como essa limitação afeta os resultados de cálculos numéricos. O trabalho foi dividido em três etapas: conversões manuais de números decimais para o formato IEEE 754, implementação de uma calculadora em Python para visualizar campos como sinal, expoente e mantissa, e a análise de operações envolvendo números que não possuem representação binária exata, como 0,1. Os experimentos mostraram pequenas diferenças entre o valor decimal “ideal” e o valor aproximado armazenado pelo computador. Constatou-se que o arredondamento e o limite de bits podem introduzir erros que se acumulam ao longo das operações. A partir disso, conclui-se que compreender o IEEE 754 é essencial para interpretar resultados numéricos com mais precisão e evitar conclusões equivocadas em cálculos aparentemente simples.

Palavras-chave: IEEE 754. Ponto flutuante. Arredondamento. Erro numérico..

1 INTRODUÇÃO

A área de cálculo numérico trabalha constantemente com aproximações, tanto em algoritmos quanto em representações internas dos números. Um aspecto que passa despercebido por muitos estudantes é que, dentro do computador, os valores reais não são armazenados como nós os vemos em decimal; eles são convertidos para binário e representados com uma quantidade limitada de bits. Isso significa que nem todos os números podem ser representados

[1] Graduando do curso de Ciência da Computação do Centro Universitário Filadélfia – UniFil. E-mail: diogo.lepri@edu.unifil.br
[2] Graduando do curso de Ciência da Computação do Centro Universitário Filadélfia – UniFil. E-mail: raphaelgpalma@edu.unifil.br

exatamente.

O padrão IEEE 754 surgiu para padronizar essa representação e garantir que sistemas diferentes trabalhassem da mesma forma com números em ponto flutuante. Ele define como separar um número em sinal, expoente e mantissa, bem como as regras de arredondamento, casos especiais e formatos de precisão.

O objetivo deste trabalho é entender, de forma prática, como funciona o padrão IEEE 754 de precisão simples e como a limitação de bits afeta operações básicas. Para isso, foram realizados três tipos de atividade: conversão manual de números decimais para o formato IEEE 754, implementação de uma calculadora em Python para visualização dos campos internos e experimentos com operações comuns, como somar 0,1 e 0,2. Além disso, incluíram-se comentários pessoais sobre o impacto dessas representações no uso real do computador.

2 DESENVOLVIMENTO

No ponto flutuante, um número real é representado aproximadamente da seguinte forma:

x=(−1)s×(1.f)×2(e−bias)x=(−1)s×(1.f)×2(e−bias)

No formato de precisão simples (32 bits), o IEEE 754 utiliza:

- 1 bit de sinal(s)
- 8 bits de expoente(e), com viés de 127
- 23 bits de mantissa(f), representando a parte fracionária do número normalizado

O processo começa convertendo o número decimal para binário (parte inteira e parte fracionária). Em seguida, normaliza-se o número para a forma 1,mantissa×2e1,mantissa×2e, ajusta-se o expoente com o viés 127 e armazena-se apenas a parte fracionária na mantissa. Como o espaço da mantissa é limitado, alguns números precisam ser aproximados, gerando erros que podem ser

observados em cálculos posteriores.

Esse tipo de erro se relaciona com temas discutidos em métodos numéricos, como arredondamento, truncamento e cancelamento, que afetam a precisão final das operações.

PROCEDIMENTOS

1)Três valores foram escolhidos:

- 5,75
- –13,25
- 0,1

O objetivo foi aplicar passo a passo:

1. Conversão da parte inteira para binário
2. Conversão da parte fracionária por multiplicações sucessivas
3. Normalização
4. Cálculo do expoente ajustado pelo viés
5. Definição do bit de sinal
6. Construção final dos 32 bits

As tabelas completas com todos os passos estão no Anexo.

2)Para visualizar e entender melhor o formato, foi implementado um programa em Python que:

- recebe dois números reais
- converte cada um para o formato IEEE 754 de 32 bits
- exibe sinal, expoente e mantissa
- realiza a operação (soma ou subtração)
- compara o valor resultante com a representação em IEEE 754 calculada manualmente

O código foi comentado para facilitar a leitura e está incluído integralmente no Apêndice.

3)Os seguintes testes foram realizados:

- 0,1 + 0,2, para observar a diferença entre o valor esperado (0,3) e o resultado interno
- operações entre números com módulos muito diferentes
- comparação da mantissa antes e depois das operações

O propósito foi analisar como pequenos erros se manifestam nas operações e como eles se acumulam.

RESULTADOS

As conversões manuais mostraram claramente como os campos do IEEE 754 são obtidos. No caso do 5,75, a normalização leva ao formato 1,01112×221,01112×22, resultando em expoente 129 e uma mantissa que começa com 0111. Isso demonstra como o padrão organiza internamente um número aparentemente simples.

Para o **0,1**, o processo revelou que a parte fracionária origina uma dízima binária, que não cabe inteira nos 23 bits. Assim, é necessário arredondar, explicando por que 0,1 nunca é representado exatamente no computador.

Nos experimentos com o Python, observou-se que operações como 0,1 + 0,2 retornam valores próximos de 0,3, mas não exatamente iguais. Quando o resultado é analisado em IEEE 754, nota-se que a mantissa já chega à operação com aproximações prévias, e a soma envolve dois números já arredondados.

Em operações envolvendo subtração de valores muito próximos, ocorreu o chamado cancelamento, eliminando parte significativa dos dígitos úteis e ampliando o erro relativo. Isso reforçou a importância de conhecer o padrão de representação para evitar armadilhas numéricas.

Do ponto de vista prático, esses experimentos mostram que o computador não “erra”; ele apenas segue regras de representação que são finitas.

3 CONCLUSÃO

O estudo permitiu compreender de forma prática como números reais são representados internamente no padrão IEEE 754 e como a limitação de bits influencia cálculos simples. A realização das conversões manuais auxiliou na visualização do funcionamento do formato, enquanto os testes com a calculadora em Python evidenciaram erros de arredondamento e cancelamento.

Percebeu-se que operações aparentemente simples, como 0,1 + 0,2, podem produzir resultados ligeiramente diferentes do esperado por conta das aproximações necessárias na mantissa. Além disso, observou-se que certos tipos de operações podem amplificar esses erros.

Conclui-se que compreender o padrão IEEE 754 contribui para o domínio de cálculo numérico e para a interpretação crítica dos resultados gerados por computadores, especialmente em aplicações que exigem alta precisão.

REFERÊNCIAS

AQUINO, F. J. A. ***Tópicos de métodos numéricos com Scilab: computação científica para engenheiros*.** Rio de Janeiro: PoD, 2020.

VIANA, G. V. R. ***Padrão IEEE 754 para Aritmética Binária de Ponto Flutuante*.** Universidade Estadual do Ceará, Departamento de Estatística e Computação.

IBM. ***IEEE Floating-Point Overview*.** Disponível em: https://www.ibm.com/docs/pt-br/xl-fortran-aix/16.1.0.

APÊNDICE – CÓDIGO-FONTE DA CALCULADORA IEEE 754

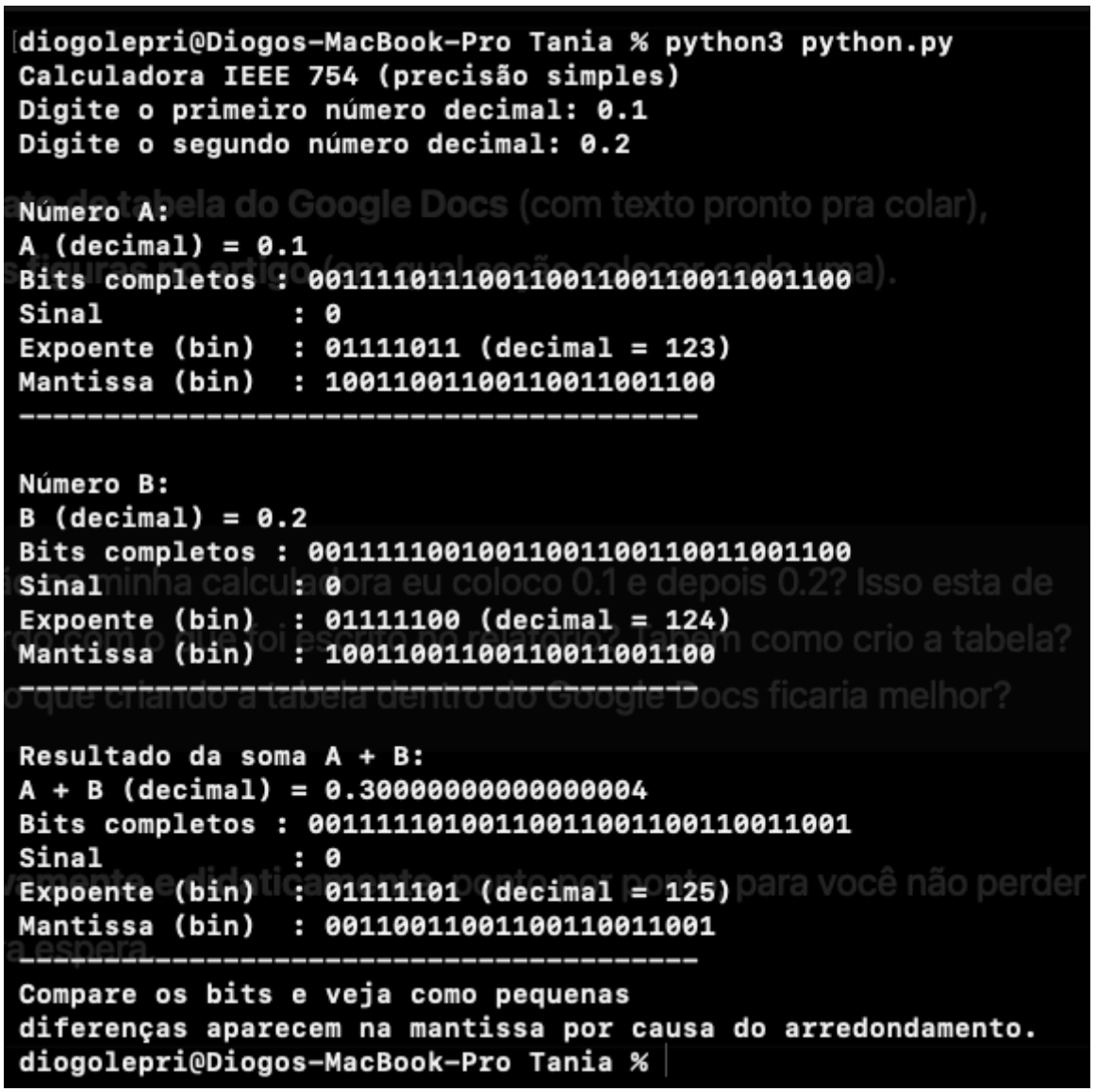
Figura 1 – Trecho do código da calculadora IEEE 754 em Python

```python
import math

def float_to_ieee754(x: float) -> str:
    if x < 0:
        sign_bit = '1'
    else:
        sign_bit = '0'
    v = abs(x)
    if v == 0.0:
        return sign_bit + '0' * 31

    e = math.floor(math.log2(v))
    m = v / (2 ** e)
    frac = m - 1.0

    mantissa_bits = ""
    for _ in range(23):
        frac *= 2
        if frac >= 1.0:
            mantissa_bits += '1'
            frac -= 1.0
        else:
            mantissa_bits += '0'
    exponent = e + 127
    exponent_bits = format(exponent, '08b')
    return sign_bit + exponent_bits + mantissa_bits

def mostrar_campos(bits: str) -> None:
    s = bits[0]
    e = bits[1:9]
    f = bits[9:]

    print(f"Bits completos : {bits}")
    print(f"Sinal          : {s}")
    print(f"Expoente (bin) : {e} (decimal = {int(e, 2)})")
    print(f"Mantissa (bin) : {f}")
    print("-" * 40)
def main():
```

```
    print("Calculadora IEEE 754 (precisão simples)")
    a = float(input("Digite o primeiro número decimal: "))
    b = float(input("Digite o segundo número decimal: "))

    print("\nNúmero A:")
    a_bits = float_to_ieee754(a)
    print(f"A (decimal) = {a}")
    mostrar_campos(a_bits)

    print("\nNúmero B:")
    b_bits = float_to_ieee754(b)
    print(f"B (decimal) = {b}")
    mostrar_campos(b_bits)

    c = a + b
    c_bits = float_to_ieee754(c)

    print("\nResultado da soma A + B:")
    print(f"A + B (decimal) = {c}")
    mostrar_campos(c_bits)

    print("Compare os bits e veja como pequenas")
    print("diferenças aparecem na mantissa por causa do arredondamento.")

if __name__ == "__main__":
    main()
```

Fonte: elaborado pelo autor (2025)

Figura 2 – Execução da calculadora IEEE 754 para os valores 0.1 e 0.2



Fonte: elaborado pelo autor (2025)

ANEXO – TABELAS DE CONVERSÃO MANUAL

Tabela 1 – Conversão de 5,75 para IEEE 754 (precisão simples)

Número decimal	**5,75**
Parte inteira (decimal → binário)	**$5 \rightarrow 101_2$**
Parte fracionária (decimal → binário)	**$0{,}75 \rightarrow 0{,}11_2$**
Binário completo	**$101{,}11_2$**
Forma normalizada	**$1{,}0111_2 \times 2^2$**
Expoente real (*e*)	**2**
Expoente com viés (e + 127)	**2 + 127 = 129**
Expoente em binário (8 bits)	**$129 \rightarrow 10000001_2$**
Bit de sinal (*s*)	**0 (número positivo)**
Mantissa (23 bits)	**01110000000000000000000**
Palavra final de 32 bits	**0 10000001 01110000000000000000000**

Tabela 2 – Conversão de –13,25 para IEEE 754 (precisão simples)

Número decimal	**–13,25**
Parte inteira (decimal → binário)	**$13 \rightarrow 1101_2$**
Parte fracionária (decimal → binário)	**$0{,}25 \rightarrow 0{,}01_2$**
Binário completo (módulo)	**$1101{,}01_2$**
Forma normalizada	**$1{,}10101_2 \times 2^3$**
Expoente real (*e*)	**3**
Expoente com viés (e + 127)	**3 + 127 = 130**
Expoente em binário (8 bits)	**$130 \rightarrow 10000010_2$**
Bit de sinal (*s*)	**1 (número negativo)**
Mantissa (23 bits)	**10101000000000000000000**
Palavra final de 32 bits	**1 10000010 10101000000000000000000**

Tabela 3 – Conversão de 0,1 para IEEE 754 (precisão simples)

Número decimal	**0,1**
Parte inteira (decimal → binário)	**$0 \rightarrow 0_2$**
Parte fracionária (decimal → binário)	**$0{,}1 \approx 0{,}00011001100110011\ldots_2$ (dízima periódica)**
Forma normalizada	**$1{,}10011001100110011\ldots_2 \times 2^{-4}$**
Expoente real (*e*)	**–4**
Expoente com viés (e + 127)	**-4 + 127 = 123**
Expoente em binário (8 bits)	**$123 \rightarrow 01111011_2$**
Bit de sinal (*s*)	**0 (número positivo)**
Mantissa (23 bits)	**10011001100110011001101**
Palavra final de 32 bits	**0 01111011 10011001100110011001101**