Universidade Federal de Juiz de Fora

Pós-Graduação em Modelagem Computacional

Métodos Numéricos

**Eduardo Santos de Oliveira Marques**

**Atividade 1**
**Representação e Aritmética de Ponto Flutuante**

Juiz de Fora

2023

**Questão 1.** Desenvolva um programa que conte o número de bits considerados na mantissa de números em ponto flutuante. Leve em consideração apenas os tipos de representação de ponto flutuante que a linguagem de programação que for usada permite manipular (simples, dupla, etc).

**Resolução:**

```
import struct

def count_mantissa_bits(float_number):
    # Converte o número em ponto flutuante para sua representacao binária em bytes
    binary = struct.pack('!d', float_number)

    # Extrai o expoente e a mantissa da representacao binária
    bits = struct.unpack('!Q', binary)[0]
    # Máscara para extrair a mantissa
    mantissa_bits = bits & ((1 << 52) - 1)
    # Conta o número de bits na mantissa
    count = sum(1 for _ in bin(mantissa_bits)[2:] if _ == '1')
    return count

print("Contagem de bits da mantissa em números de ponto flutuante")

while True:
    input_number = input("Insira um número em ponto flutuante (ou 's' para sair): ")

    if input_number.lower() == 's':
        break

    try:
        float_number = float(input_number)
        mantissa_bits = count_mantissa_bits(float_number)
        print(f"O número de bits da mantissa para {float_number} é: {mantissa_bits} bits")
    except ValueError:
        print("Entrada inválida. Tente novamente.")
```

O código foi executado em Python através da plataforma OnlineGDB, abaixo é mostrado o passo a passo:

1. O módulo *struct* foi importado, fornecendo funções para converter entre valores Python e objetos binários de C;

2. A função *count_mantissa_bits(float_number)* recebe um número em ponto flutuante como entrada;

3. Dentro da função, o comando *struct.pack('!d', float_number)* é usado para converter o número em ponto flutuante para sua representação binária em bytes. O argumento *'!d'* indica que se trata de um número de ponto flutuante de precisão dupla (*double*) no formato big-endian (os bytes mais significativos aparecem primeiro);

4. Em seguida, usa-se *struct.unpack('!Q', binary)* para extrair os 64 bits da representação binária. O argumento *'!Q'* indica que os bytes são interpretados como um número inteiro de 64 bits sem sinal (unsigned long long) no formato big-endian;

5. Uma máscara é usada *((1 « 52) - 1)* para extrair somente os bits da mantissa. A expressão *(1 « 52)* cria um número binário com um '1' seguido de 52 '0's (representa o bit implícito '1' na mantissa), subtraindo 1 obtêm uma sequência de 52 '1's;

6. A operação binária & (AND) é aplicada entre o valor dos bits e a máscara para obter apenas a mantissa;

7. A mantissa é convertida em uma representação binária com *bin(mantissa_bits)[2:]*. A chamada de *bin()* retorna uma string que representa o número binário, incluindo o prefixo '0b'. Usando *[2:]*, o prefixo '0b' é removido para obter apenas a sequência binária;

8. O número de bits definidos (1's) é contado na mantissa usando *sum(1 for _ in bin(mantissa_bits)[2:] if _ == '1')*. Cada bit é percorrido na representação binária da mantissa, sendo incrementado sempre que é encontrado um bit igual a '1';

9. O valor do contador é retornado, representando o número de bits na mantissa;

10. No bloco principal do código, um loop *while True* é utilizado para permitir que o usuário insira vários números em ponto flutuante;

11. Dentro do loop, é solicitado ao usuário inserir um número em ponto flutuante. Se o usuário digitar a letra 's', o loop é interrompido e o programa é encerrado.

12. Um bloco 'try-except' é utilizado para lidar com possíveis erros de conversão do número inserido pelo usuário para um número em ponto flutuante;

13. Se a conversão for bem-sucedida, a função *count_mantissa_bits()* é chamada para obter o número de bits da mantissa, sendo exibido o resultado na saída;

14. Se ocorrer um erro de conversão, é exibida uma mensagem de erro.

**Questão 2.** Implemente um programa para calcular o valor da aproximação que segue:

$$e^x \approx P_n(x) = \sum_{k=0}^{n} 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{k=0}^{n} \frac{x^k}{k!}$$

Analise os erros absolutos e relativos que podem surgir durante a computação dos seguintes valores: $x = \{-20, -10, -2, -1, 1, 2, 10, 20\}$. Os erros devem ser calculados em relação à função de exponencial da linguagem de programação que estiverem usando. Para o caso dos expoentes negativos, avalie também:

$$e^{-x} = \frac{1}{P_n(x)}$$

**Resolução:**

```python
import math

def aproximacao_exp(x, n):
    aproximacao = 0.0
    exato = math.exp(x)
    for k in range(n+1):
        aproximacao += x**k / math.factorial(k)
    return aproximacao, exato

def calcular_erros(x_values, n):
    for x in x_values:
        aproximacao, exato = aproximacao_exp(x, n)
        erro_absoluto = abs(exato - aproximacao)
        erro_relativo = abs(exato - aproximacao) / exato
        print(f'x = {x}:')
        print(f'Aproximacao: {aproximacao}')
        print(f'Exato: {exato}')
        print(f'Erro absoluto: {erro_absoluto}')
        print(f'Erro relativo: {erro_relativo}\n')

# Valores de x a serem avaliados
x_values = [-20, -10, -2, -1, 1, 2, 10, 20]

# Grau do polinomio da aproximacao
n = 10

calcular_erros(x_values, n)
```

Outputs:

```
x = -20:                                  x = 1:
Aproximação: 1859623.6807760142           Aproximação: 2.7182818011463845
Exato: 2.061153622438558e-09              Exato: 2.718281828459045
Erro absoluto: 1859623.6807760121         Erro absoluto: 2.7312660577649694e-08
Erro relativo: 902224686472367.4          Erro relativo: 1.0047766310211053e-08

x = -10:                                  x = 2:
Aproximação: 1342.5873015873017           Aproximação: 7.388994708994708
Exato: 4.5399929762484854e-05             Exato: 7.38905609893065
Erro absoluto: 1342.5872561873718         Erro absoluto: 6.138993594273501e-05
Erro relativo: 29572452.274954546         Erro relativo: 8.308224368687552e-06

x = -2:                                   x = 10:
Aproximação: 0.13537918871252214          Aproximação: 12842.305114638448
Exato: 0.1353352832366127                 Exato: 22026.465794806718
Erro absoluto: 4.390547590943372e-05      Erro absoluto: 9184.16068016827
Erro relativo: 0.00032442002454505394     Erro relativo: 0.4169602498070145

x = -1:                                   x = 20:
Aproximação: 0.3678794642857144           Aproximação: 5245469.677248677
Exato: 0.36787944117144233                Exato: 485165195.4097903
Erro absoluto: 2.3114272051927287e-08     Erro absoluto: 479919725.7325416
Erro relativo: 6.283110569681271e-08      Erro relativo: 0.9891882811733473
```

Neste código, a função *aproximacao_exp* calcula a aproximação de $e^x$ utilizando a série de Taylor até o $n$-ésimo termo. Em seguida, a função *calcular_erros* itera atrvavés dos valores de $x$ fornecidos no enunciado, calcula a aproximação, os erros absolutos e relativos correspondentes; e imprime os resultados.

Os resultados fornecem a aproximação para cada valor de $x$, o valor exato calculado pela função *math.exp(x)*, o erro absoluto (diferença entre o valor exato e a aproximação) e o erro relativo (razão entre o erro absoluto e o valor exato).

Nota-se que para valores negativos de $x$, a função *math.exp(x)* retorna $e^{-x}$ diretamente. Portanto, não é preciso calcular o inverso da aproximação para obter $e^{-x}$.

**Questão 3.** Mostre que

$$ER_{\text{truncamento}}(x, \bar{x}) \leq \beta^{1-p}$$
$$ER_{\text{arredondamento}}(x, \bar{x}) \leq \frac{\beta^{1-p}}{2}$$

onde $ER$ representa o erro relativo ao tentar representar o valor $x$ em ponto flutuante num sistema computacional de base $\beta$. O valor de $x$ é aproximado por $x$ quando arredondado ou truncado, e $p$ é o número de dígitos da mantissa.

**Resolução:**

Para mostrar as desigualdades dadas, os termos são definidos. Dado um número real $x$ e sua representação em ponto flutuante $\bar{x}$, o erro relativo do truncamento e do arrendondamento são dados respectivamente por:

$$ER_{\text{truncamento}}(x, \bar{x}) = \left|\frac{x - \bar{x}}{x}\right| \tag{1}$$

$$ER_{\text{arredondamento}}(x, \bar{x}) = \left|\frac{x - \bar{x}}{x}\right| \tag{2}$$

onde $\bar{x}$ é o número representado em ponto flutuante mais próximo de $x$.

**1.** Erro de truncamento:

Quando um número real $x$ é truncado para ser representado em ponto flutuante com uma mantissa de $p$ dígitos, o erro de truncamento ocorre devido à perda dos dígitos removidos. Considerando $\bar{x}_t$ como a representação truncada de $x$. Nesse caso, têm-se:

$$\bar{x}_t = \text{trunc}(x, p)$$

onde $\text{trunc}(x, p)$ denota o truncamento de $x$ para $p$ dígitos.

Como o truncamento remove os dígitos além da mantissa de $p$ dígitos, têm-se:

$$|x - \bar{x}_t| \leq \beta^{1-p}$$

Isso ocorre porque, no pior caso, os dígitos removidos têm o valor máximo possível, que é $\beta^{-p}$. Portanto, o erro absoluto máximo de truncamento é limitado por $\beta^{1-p}$.

Dividindo ambos os lados por $|x|$ e substituindo $\bar{x}_t$ por $\bar{x}$, têm-se:

$$\frac{|x - \bar{x}_t|}{|x|} \leq \frac{\beta^{1-p}}{|x|}$$

Como o erro relativo é sempre não negativo, é possível remover os valores absolutos e obter:

$$ER_{\text{truncamento}}(x, \bar{x}) \leq \frac{\beta^{1-p}}{|x|}$$

Porém, como $|x| \geq 1$ (porque $x$ é um número real), pode-se substituir $|x|$ por 1:

$$ER_{\text{truncamento}}(x, \bar{x}) \leq \beta^{1-p}$$

**Portanto, a desigualdade $\mathbf{ER_{truncamento}(x, \bar{x}) \leq \beta^{1-p}}$ é válida.**

**2.** Erro de arredondamento:

Quando um número real $x$ é arredondado para ser representado em ponto flutuante com uma mantissa de $p$ dígitos, o erro de arredondamento ocorre devido à imprecisão na representação. Considerando $\bar{x}_r$ como a representação arredondada de $x$. Nesse caso, têm-se:

$$\bar{x}_r = \text{round}(x, p)$$

onde $\text{round}(x, p)$ denota o arredondamento de $x$ para $p$ dígitos.

Para o arredondamento, o erro absoluto máximo ocorre quando $x$ está exatamente no meio entre dois números representáveis em ponto flutuante com mantissa de $p$ dígitos. Nesse caso, o erro absoluto máximo é metade da distância entre esses dois números, que é $\frac{\beta^{1-p}}{2}$. Portanto, têm-se:

$$|x - \bar{x}_r| \leq \frac{\beta^{1-p}}{2}$$

Dividindo ambos os lados por $|x|$ e substituindo $\bar{x}_r$ por $\bar{x}$, temos:

$$\frac{|x - \bar{x}_r|}{|x|} \leq \frac{\beta^{1-p}}{2|x|}$$

Assim como no caso anterior, pode-se remover os valores absolutos e substituir $|x|$ por 1:

$$ER_{\text{arredondamento}}(x, \bar{x}) \leq \frac{\beta^{1-p}}{2}$$

**Portanto, a desigualdade $\mathbf{ER_{arredondamento}(x, \bar{x}) \leq \frac{\beta^{1-p}}{2}}$ é válida.**

# APÊNDICE A – Códigos da Atividade

Abaixo são apresentados os códigos realizados, todos desenvolvidos e testados na plataforma `https://www.onlinegdb.com/`. Abaixo seguem os links das questões:

- **Questão 1:** `https://onlinegdb.com/hk_d8vuPP`
- **Questão 2:** `https://onlinegdb.com/eVUBJ7zns`