analise_de_erros

May 21, 2023

#

Análise de Erros

1 1. Erros de Arredondamento e Arimtética no Computador

A aritmética executada por um calculadora ou computador é diferente daquela dos nossos cursos de álgebra ou cálculo. Consideremos alguns exemplos…

2 Exemplos

[ ]:
```
# Exemplo 1
a1 = 2
b1 = 1 + 1
print(f'a = {a1}')
print(f'b = {b1}')
print(a1 == b1)
```

[ ]:
```
# Exemplo 2
a2 = 1-1/7
b2 = 6/7
print(f'a = {a2}')
print(f'b = {b2}')
print(a2 == b2)
```

[ ]:
```python
# Exemplo 3
import numpy as np
a3 = np.sqrt(3)**2            # sqrt(a) é a raiz quadada de a
b3 = 3
print(f'a = {a3}')
print(f'b = {b3}')
print(a3 == b3)
```

Observação: O operador relacional 'a == b' retorna o valor True caso 'a' seja igual a 'b' e False caso contrário.

Espere erros devido ao arredondamento sempre que forem feitos cálculos usando números que não sejam potências de 2. Manter esse erro sob controle é extremamente importante quando o número de cálculos for grande.

2.1 1.1 Dentro do PC

Cálculos internos à máquina são, em geral, efetuados em outras bases que não a base 10 (**decimal**).

A base decimal é a base numérica padrão utilizada em nosso sistema cotidiano. Os números decimais são compostos por dígitos de 0 a 9, e cada posição em um número decimal representa uma potência de 10.

$$324_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$$

Existem outras bases numéricas importantes na programação: a base **binária**, a **base octal** e a **base hexadecimal**.

A base **binária** é a base numérica utilizada pelos computadores para armazenar e processar informações. Os números binários são compostos apenas pelos dígitos 0 e 1, e cada posição em um número binário representa uma potência de 2.

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13_{10}$$

A base binária é essencial em áreas como a engenharia de computação e a ciência da computação, pois permite que as informações sejam armazenadas e processadas de forma eficiente pelos computadores.

A base **octal** é uma base numérica que utiliza os dígitos de 0 a 7. Cada posição em um número octal representa uma potência de 8.

$$314_8 = 3 \cdot 8^2 + 1 \cdot 8^1 + 4 \cdot 8^0 = 192 + 8 + 4 = 204_{10}$$

A base octal é utilizada principalmente em sistemas de permissões de arquivos em sistemas operacionais Unix e Linux, bem como em linguagens de programação como o C e o Perl.

A base **hexadecimal** é uma base numérica que utiliza os dígitos de 0 a 9 e as letras de A a F (representando os valores de 10 a 15). Cada posição em um número hexadecimal representa uma potência de 16.

$$E3_{16} = 14 \cdot 16^1 + 3 \cdot 16^0 = 227_{10}$$

A base hexadecimal é amplamente utilizada em programação, especialmente em linguagens de programação como o C, o C++, o Java e o Python, para representar cores, endereços de memória, números de porta e outros valores numéricos.

O coprocessador numérico para PCs implementa uma representação de 64 bits para números reais, conhecida como real longo. O primeiro bit é o indicador de sinal (s), seguido por um expoente de 11 bits (c), chamado de característica, e uma mantissa binária de 52 bits (f). A base do expoente é 2. A célula a seguir mostra que um número binárico com 52 dígitos corresponde a um número decimal contendo entre 15 e 16 dígitos decimais.

[15]:
```
# Sobre a afirmação anterior...
print(10**15<2**52<10**16)
```

```
True
```

Podemos assumir que um número representado neste sistema tenha pelo menos 15 dígitos decimais de precisão. O expoente de 11 dígitos binários dá um intervalo de 0 a $2^{11}-1=2047$. No entanto, usando apenas inteiros positivos para o expoente não teríamos uma representação adequada de números de pequena magnitude. Para garantir que os números com magnitude pequena sejam igualmente representáveis, 1023 é subtraído da característica (expoente). Então o intervalo do expoente é na verdade, de -1023 a 1024.

Para economizar armazenamento e fornecer uma representação exclusiva para cada número de ponto flutuante, impomos a seguinte normalização. O uso desse sistema fornece um número de ponto flutuante da forma:

$$(-1)^s 2^{c-1023}(1+f).$$

Considere por exemplo o número de máquina:

$$n = 0\,10000000011\,1011100100010000000000000000000000000000000000000000.$$

Considerando a forma $n=(-1)^s 2^{c-1023}(1+f)$, temos que bit mais a esquerda sendo 0 indica que o número n é positivo $(-1)^0=1$ (se fosse $s=1$, teríamos n negativo). Os próximos 11 bits 10000000011 que fornecem a característica(expoente) são equivalentes ao decimal

$$c = 1\cdot 2^{10}+0\cdot 2^9+0\cdot 2^8+0\cdot 2^7+0\cdot 2^6+0\cdot 2^5+0\cdot 2^4+0\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0 = 1024+2+1 = 1027.$$

Portanto, a parte exponencial do número é dada por $2^{1027-1023}=2^4=16$. Os 52 bits finais especificam que a mantissa seja:

$$f = 1\cdot\left(\frac{1}{2}\right)^1 + 1\cdot\left(\frac{1}{2}\right)^3 + 1\cdot\left(\frac{1}{2}\right)^4 + 1\cdot\left(\frac{1}{2}\right)^5 + 1\cdot\left(\frac{1}{2}\right)^8 + 1\cdot\left(\frac{1}{2}\right)^{12}.$$

Por fim,

$$n = (-1)^s 2^{c-1023}(1+f) = (-1)^0 2^4\left(1+\left(\frac{1}{2}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\frac{1}{256}+\frac{1}{4096}\right)\right) = 27.56640625.$$

Além disso,

$$x < 0\ 10000000011\ 1011100100010000000000000000000000000000000000000000 < y$$

temos que este está entre os números de máquina

$$x = 0\ 10000000011\ 1011100100001111111111111111111111111111111111111111$$

e

$$y = 0\ 10000000011\ 1011100100010000000000000000000000000000000000000001$$

sendo x e y os números de máquina mais próximos de n.

Neste caso, o número de máquina 27.56640625 representa não somente ele próxprio como metade dos números reais que estão entre 27.56640625 e seus dois números de máquina mais próximos. Para ser mais preciso,

$$27.56640625 \equiv [27.566406249999998223643160599749535322189331054687 5, 27.5664062500000017763568394002 50$$

O símbolo $\equiv$ usado nos diz que quais quer números no intervalo à direita terá a mesma representação, isto é, 27.56640625.

2.1.1 Exemplo:

Obtenha o número de máquina correspondente ao número binário:

$$n = 1\ 10000000101\ 0010101000000000000000000000000000000000000000000000.$$

Solução: Temos que

$$s = -1\ e\ c = 1 \cdot 2^{10} + 1 \cdot 2^2 + 1 \cdot 2^0 = 1024 + 4 + 1 = 1029$$

donde segue que a parte exponencial de n é $2^{1029-1023} = 2^6$. Continuando,

$$f = 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^7.$$

Portanto,

$$n = (-1)^1 2^{1029-1023} \left(1 + \left(\frac{1}{8} + \frac{1}{32} + \frac{1}{128}\right)\right) = -74.5$$

Observação: Para facilitar, indique as posições conforme a seguir: $ $

$$n = 1\ 10000000\underline{1}0\underline{1}\ 00\underline{1}0\underline{1}0\underline{1}000000000000000000000000000000000000000000000$$

isto é, a parte exponecial tem suas posições começando de 0 indo da direita para a esquerda e a mantissa tem suas posições começando de 1 indo da esquerda para a direita. Note que o número de máquina -74.5 representará uma classe de números e não somente o real -74.5.

2.2 Limites

O **menor** número positivo normalizado que pode ser representado tem $s = 0$, $c = 1$ e $f = 0$ e é equivalente a $ $

$$(-1)^0 2^{1-1023}(1+0) = 2^{-1022}(1+0) \approx 0.2225 \times 10^{-307}.$$

e o **maior** tem $s = 0$, $c = 2046$ e $f = 1 - 2^{-52}$ e é equivalente a $ $

$$(-1)^0 2^{2046-1023}(1+(1-2^{-52})) = 2^{1023}(2-2^{-52}) \approx 0.17977 \times 10^{309}$$

Números maiores que 0.2225×10^{-307} resultam em **underflow** e igualados a zero. Por outro lado, números maiores que 0.17977×10^{309} resultam em **overflow**. Para mais informações consulte [1].

[16]:
```
# No Jupyter
import sys    #fornece acesso a algumas variáveis e funções do interpretador␣
↪Python
print(f'menor = {sys.float_info.min}')
print(f'maior = {sys.float_info.max}')
```

```
menor = 2.2250738585072014e-308
maior = 1.7976931348623157e+308
```

2.3 Trucamento, Arredondamento e Algarismos Significativos

Qualquer número real positivo dentro do intervalo numérico da máquina pode ser normalizado na forma $ $

$$y = 0.d_1 d_2 ... d_k d_{k+1} ... \times 10^n.$$

2.3.1 Truncamento

A forma em ponto flutuante de y, denotada por $fl_k(y)$, é obtida terminando a mantissa de y em k algarismos decimais. $ $

$$fl_k(y) = 0.d_1 d_2 ... d_k \times 10^n.$$

2.3.2 Arredondamento

A forma em ponto flutuante de y, denotada por $fl_k(y)$, é obtida terminando a mantissa de y em k algarismos decimais. Contudo o algarismo d_k é aumentado de 1 caso $d_{k+1} \geq 5$.

$$fl_k(y) = 0.\delta_1\delta_2...\delta_k \times 10^n.$$

Usamos δ para ilustrar que o arredondamento de d_k pode acarretar mudanças em $d_{k-1}, ..., d_2, d_1$.

2.3.3 Exemplo:

O número π tem uma expansão decimal infinita da forma $\pi = 3.14159265...$

a) Escreva π na forma decimal normalizada;

b) Apresente a forma em ponto flutuante de π usando truncamento para 5 casas decimais;

c) Apresente a forma em ponto flutuante de π usando arredondamento também para 5 casas decimais.

Solução: a) Temos que a forma normalizada de π é \$ \$

$$\pi = 3.14159265 \times 10^1.$$

b) A forma em ponto flutuante de π usando truncamento com 5 casas decimais é: \$ \$

$$fl_5(\pi) = 0.31415 \times 10^1 = 3.1415.$$

c) A forma em ponto flutuante de π usando arredondamento com 5 casas decimais é: \$ \$

$$fl_5(\pi) = (0.31415 + 1) \times 10^1 = 0.31416 = 3.1416.$$

Definição: Se p^* é uma aproximação de p, o **erro absoluto** é $|p - p^*|$ e o **erro relativo** é $\frac{|p-p^*|}{|p|}$, contanto que $p \neq 0$.

2.3.4 Exemplo:

Calcule o erro relativo e o erro absoluto ao se considerar a aproximação

a) $p^* = 0.3100 \times 10^1$ de $p = 0.3000 \times 10^1$.

b) $p^* = 0.3100 \times 10^{-3}$ de $p = 0.3000 \times 10^{-3}$.

Solução: a) Temos que $ $

$$|p - p^*| = |0.3000 \times 10^1 - 0.3100 \times 10^1| = 0.1 \times 10^{-1} \ e$$

$$\frac{|p - p^*|}{|p|} = \frac{|0.3000 \times 10^1 - 0.3100 \times 10^1|}{|0.3000 \times 10^1|} = 0.333\overline{3} \times 10^{-1}$$

b) Temos que

$$|p - p^*| = |0.3000 \times 10^4 - 0.3100 \times 10^4| = 0.1 \times 10^3 \ e$$

$$\frac{|p - p^*|}{|p|} = \frac{|0.3000 \times 10^4 - 0.3100 \times 10^4|}{|0.3000 \times 10^4|} = 0.333\overline{3} \times 10^{-1}$$

[17]:
```
# a)
p = round(0.3000*10**1,4)
p_aprox = round(0.3100*10**1,4)
e_abs = round(abs(p-p_aprox),4)
e_rel = round(abs((p-p_aprox)/p),10)
print(f'Erro absoluto = {e_abs}')
print(f'Erro relativo = {e_rel}')
```

```
Erro absoluto = 0.1
Erro relativo = 0.0333333333
```

[18]:
```
# b)
p = round(0.3000*10**4,4)
p_aprox = round(0.3100*10**4,4)
e_abs = round(abs(p-p_aprox),4)
e_rel = round(abs((p-p_aprox)/p),10)
print(f'Erro absoluto = {e_abs}')
print(f'Erro relativo = {e_rel}')
```

```
Erro absoluto = 100.0
Erro relativo = 0.0333333333
```

Note que o erro relativo é, em geral, uma medida melhor que o erro absoluto pois leva em conta o tamanho do número que está sendo aredondado.

Em geral não se pode determinar o erro exato em uma aproximação. O que se faz é encontrar um limitante para o erro, o que fornece o **pior caso de erro**, ou um valor *não muito exagerado* que certamente seja superior ao erro exato.

Observação: Uma expressão mais adequada para **pior caso de erro** seria o **melhor entre os piores erros**, uma vez que, em teoria, o erro pode ser praticamente infinito. Podemos, por exemplo, dizer que 1000 é uma estimativa para 1200 ou para 10000…

Definição: Diz-se que o número p^* aproxima p até t **algarismos significativos** se t for o maior inteiro não negativo para o qual $ $

$$\frac{|p-p^*|}{|p|} \leq 5 \times 10^{-t}.$$

2.3.5 Cota Superior Para o Erro Devido a Utilização de Truncamento

A representação de números de máquina em Ponto Flutuante $fl_k(y)$ é tal que o erro relativo é: $ $

$$\left|\frac{y - fl_k(y)}{y}\right| = \left|\frac{0.d_1d_2...d_kd_{k+1}... \times 10^n - 0.d_1d_2...d_k \times 10^n}{0.d_1d_2...d_kd_{k+1}... \times 10^n}\right| =$$

$$= \left|\frac{0.d_{k+1}d_{k+2}... \times 10^{n-k}}{0.d_1d_2... \times 10^n}\right| = \left|\frac{0.d_{k+1}d_{k+2}...}{0.d_1d_2...}\right| \times 10^{-k}$$

Uma vez que $d_1 \neq 0$ temos que o denominador é limitado inferiormente por 0.1. Além disso, note que o numerador é limitado superiormente por 1. Portanto, $ $

$$\left|\frac{y - fl_k(y)}{y}\right| \leq \frac{1}{0.1} \times 10^{-k} = 10 \times 10^{-k} = 10^{-k+1}.$$

[19]:
```python
# Erro relativo máximo com k = 4 (4 casas decimais de truncamento)
k = 4
y = 0.100599664
fl_y =  0.1005
e_rel_y = abs((y-fl_y)/y)
if e_rel_y < 10**(-k+1):
    print(f'e_rel = {e_rel_y} < {10**(-k+1)} ')
```

```
e_rel = 0.0009906991339454106 < 0.001
```

Esperava-se um erro da ordem de no máximo $10^{-4+1} = 0.1 \times 10^{-3} = 0.001$ como de fato ocorreu (neste caso). Ao final, no tópico **Extras** apresentamos dois códigos que tentam estourar essa cota (sem sucesso, é claro)…

2.3.6 Cota Superior Para o Erro Devido a Utilização de Arredondamento

A representação de números de máquina em Ponto Flutuante $fl_k(y)$ é tal que o erro relativo é: $ $

$$\left|\frac{y - fl_k(y)}{y}\right| = \left|\frac{0.d_1d_2...d_kd_{k+1}... \times 10^n - 0.\delta_1\delta_2...\delta_k \times 10^n}{0.d_1d_2...d_kd_{k+1}... \times 10^n}\right| =$$

$$= \left|\frac{0.\delta'_{k+1}\delta'_{k+2}... \times 10^{n-k}}{0.d_1d_2... \times 10^n}\right| = \left|\frac{0.\delta'_{k+1}\delta'_{k+2}...}{0.d_1d_2...}\right| \times 10^{-k}$$

Uma vez que $d_1 \neq 0$ temos que o denominador é limitado inferiormente por 0.1. Além disso, note que o numerador é limitado superiormente por 0.5. Portanto, $ $

$$\left|\frac{y - fl_k(y)}{y}\right| \leq \frac{0.5}{0.1} \times 10^{-k} = 5 \times 10^{-k} = 0.5 \times 10^{-k+1}.$$

[20]:
```
# Erro relativo máximo (Veja em extras como foi gerado o valor de y para␣
↪encostar em 0.5*10**(-k+1), k = 4)
k = 4
y = 0.10005081956958761
fl_y =  0.1001
e_rel_y = abs((y-fl_y)/y)
if e_rel_y < 0.5*10**(-k+1):
    print(f'e_rel = {e_rel_y} < {0.5*10**(-k+1)} ')
```

```
e_rel = 0.0004915544982435259 < 0.0005
```

Esperava-se um erro da ordem de no máximo $0.5 \times 10^{-4+1} = 0.5 \times 10^{-3} = 0.0005$ como de fato ocorreu!

2.3.7 Adptação a Erros de Máquina Envolvendo as Operações $+, -, \times, \div$

A seguir apresentaremos um exemplo envolvendo erros computacionais associados às operações de $+, -, \times$ e $\div$. A abordagem deste tópico será breve. Para mais informações consulte [1].

2.3.8 Exemplo:

Considerando $x = \frac{5}{7}$ e $y = \frac{1}{3}$ calcule as operações de $+, -, \times$ e $\div$ admitindo um truncamento de 5 algarismos para cálculos. Calcule os erros absoluto e relativo.

O respectivos valores de truncamento de $x = \frac{5}{7}$ e $y = \frac{1}{3}$ com 5 algarismos são, respectivamente, $fl_5(x) = 0.71428$ e $fl_5(y) = 0.33333$. A tabela a seguir apresenta todos os cálculos:

Operao	*Resultado*	*Valor Real*	*Erro Absoluto*	*Erro Relativo*
$x + y$	0.10476×10^1	22/21	0.190×10^{-4}	0.182×10^{-4}
$x - y$	0.38095×10^0	8/21	0.238×10^{-4}	0.625×10^{-5}
$x \times y$	0.23809×10^0	5/21	0.524×10^{-5}	0.220×10^{-4}
$x \div y$	0.21428×10^1	15/7	0.571×10^{-5}	0.267×10^{-4}

2.3.9 Exemplo:

Considerando $x = \frac{5}{7}$ e $y = \frac{1}{3}$ calcule as operações de $+, -, \times$ e $\div$ admitindo um arredondamento de 5 algarismos para cálculos. Calcule os erros absoluto e relativo.

O respectivos valores de truncamento de $x = \frac{5}{7}$ e $y = \frac{1}{3}$ com 5 algarismos são, respectivamente, $fl_5(x) = 0.7142\underline{9}$ e $fl_5(y) = 0.33333$. A tabela a seguir apresenta todos os cálculos:

$Operao$	$Resultado$	$ValorReal$	$ErroAbsoluto$	$ErroRelativo$
$x+y$	0.10476×10^1	$22/21$	0.190×10^{-4}	0.182×10^{-4}
$x-y$	0.38095×10^0	$8/21$	0.762×10^{-5}	0.200×10^{-4}
$x \times y$	0.23809×10^0	$5/21$	0.524×10^{-5}	0.220×10^{-4}
$x \div y$	0.21429×10^1	$15/7$	0.429×10^{-4}	0.200×10^{-4}

2.3.10 Exemplo:

Considerando $x = \frac{3}{7}$, $y = \frac{2}{3}$ e $z = \frac{1}{9}$ calcule o valor de $(x+y) \times z^2$ considerando truncamento e arredondamento para 5 casas decimais. Calcule os erros absoluto e relativo.

2.4 Referências

[1] BURDEN, R.L; FAIRES, J. D. Análise Numérica. 8. ed. São Paulo: CENGAGE Learning, 2008.

2.5 Extras

[ ]:
```
import random
k = 4
for n in range(1,10000000):
    y = numero_aleatorio = random.random()
    f = math.trunc(10**k*y)/10**k
    if y>0.1:
        e = abs((y-f)/y)
        if e>0.99*10**(-k+1):
            print(f'y = {y}')
            print(f'f ={f}')
            print(f'e = {e}')
            print('------')
```

[ ]:
```
import random
k = 4
for n in range(1,10000000):
    y = numero_aleatorio = random.random()
    f =round(y,k)
    if y>0.1:
        e = abs((y-f)/y)
        if e>0.49*10**(-k+1):
            print(f'y = {y}')
            print(f'f ={f}')
            print(f'e = {e}')
            print('------')
```