

Lista de Exercícios 1 - INE5202/INE5232/INE5409
Prof. Sérgio Peters (sergio.peters@ufsc.br)

Resoluções ¹

- 1) (a) $(10.1011)_2 = 1 \cdot 2^1 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = (2.6875)_{10}$
(b) $(10.57)_{10} = (?)_2$

$$(10)_{10} = (?)_2 \longrightarrow \begin{array}{r} 10 \mid 2 \\ 0 \quad 5 \mid 2 \\ 1 \quad 2 \mid 2 \\ 0 \quad 1 \mid 2 \\ 1 \quad 0 \end{array}$$

←

$$(10)_{10} = (1010)_2 = 1 \cdot 2^3 + 1 \cdot 2^1$$

$$(0.57)_{10} = (?)_2 \longrightarrow$$

0.57	0.14	0.28	0.56	0.12	0.24	0.48	0.96	0.92	0.84	0.68	0.36	0.72	0.44	
×2	×2	×2	×2	×2	×2	×2	×2	×2	×2	×2	×2	×2	×2	...
1.14	0.28	0.56	1.12	0.24	0.48	0.96	1.92	1.84	1.68	1.36	0.72	1.44	0.88	

$$(0.57)_{10} = (0.10010001111010\dots)_2$$

Logo, $(10.57)_{10} = (1010.10010001111010\dots)_2$.

- 2) `from functools import reduce`

```
for n in [2, 3, 10, 16]:
    values = reduce(lambda x, _: x + [(n + 1) * x[-1] - 1], range(10), [1/n])
    print("\n".join("{:.30f}".format(i) for i in values))
```

- A função *reduce* é utilizada para criar o padrão requerido pelo enunciado: uma lista de números onde todos os valores estão diretamente relacionados a seus antecessores.
- Aplica-se a fórmula do enunciado no último termo da lista a cada iteração, por 10 vezes, iniciando pela inversa multiplicativa do número fornecido.
- Adiciona-se então mais casas após a vírgula para melhor visualização de como o arredondamento funciona.
- O objetivo do algoritmo é verificar como os valores são influenciados gradativamente por computações anteriores: estes perdem precisão exponencialmente pois um valor decimal não pode ser representado perfeitamente com um número finito de bits, exceto por potências de 2. A saída do algoritmo mostra duas possibilidades para cada caso, e como a quantidade de bits divergentes à divisão inicial, que já não se mostra exata, aumenta ao longo das iterações.
- Exemplo de saída (parcial) do programa:

```
0.333333333333333314829616256247
0.3333333333333333259318465024990
0.3333333333333333037273860099958
0.3333333333333332149095440399833
0.33333333333333328596381761599332
0.33333333333333314385527046397328
0.333333333333333257542108185589314
0.333333333333333030168432742357254
0.3333333333333332120673730969429016
0.33333333333333328482694923877716064
0.33333333333333313930779695510864258
```

- 3) (a) $(3021)_{F!} = 3 \cdot 4! + 0 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! = (77)_{10}$
(b) $(4321)_{F!} = 4 \cdot 4! + 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! = (119)_{10}$
(c) $(10000)_{F!} = 1 \cdot 5! = (120)_{10}$
(d) $(0.02)_{F!} = \frac{0}{2!} + \frac{2}{3!} = (\frac{2}{6})_{10} = (\frac{1}{3})_{10} = (0.333333\dots)_{10}$

¹por Gustavo Zambonin (gustavo.zambonin@grad.ufsc.br)

$$(f) \quad (321.123)_{F!} = 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + \frac{1}{2!} + \frac{2}{3!} + \frac{3}{4!} = \left(\frac{575}{24}\right)_{10} = (\mathbf{23.958\overline{3}})_{10}$$

4) (a) $(10111.1101)_2 = (0001\ 0111.1101)_2 = (17.D)_{16} = 1 \cdot 16^1 + 7 \cdot 16^0 + 13 \cdot 16^{-1} = (\frac{381}{16})_{10} = \mathbf{(23.8125)}_{10}$
(b) $(BD.0E)_{16} = (1011\ 1101.0000\ 1110)_2 = 1 \cdot 2^7 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 + 1 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} = (\frac{24199}{128})_{10} = \mathbf{(189.0546875)}_{10}$
(c) $(41.1)_{10} = (?)_2 = (?)_{16}$

$$(41)_{10} = (?)_2 \longrightarrow \begin{array}{cccccccc} 41 & \begin{array}{|c} 2 \\ 1 \end{array} & \begin{array}{|c} 2 \\ 20 \end{array} & \begin{array}{|c} 2 \\ 10 \end{array} & \begin{array}{|c} 2 \\ 5 \end{array} & \begin{array}{|c} 2 \\ 2 \end{array} & \begin{array}{|c} 2 \\ 1 \end{array} & \begin{array}{|c} 2 \\ 0 \end{array} \\ & & 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

$$(41)_{10} = (101001)_2 = 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^0$$

$$(0.1)_{10} = (?)_2 \longrightarrow$$

0.1	0.2	0.4	0.8	0.6	0.2	0.4	0.8	0.6	0.2	
$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	$\times 2$	\dots
0.2	0.4	0.8	1.6	1.2	0.4	0.8	1.6	1.2	0.4	

$$(0.1)_{10} = (0.00011001100110011\dots)_2$$

Logo, $(41.1)_{10} = (101001.000\overline{11})_2 = (0010\ 1001.0001\ \overline{1001})_2 = (\mathbf{29.19})_{16}$. Haverá perda de dígitos significativos.

5) $F(\beta, t, I, S) \longrightarrow F(2, 3, -3, +3) \longrightarrow$

s_1	d_1	d_2	d_3	s_2	e_1	e_2	e_3
-------	-------	-------	-------	-------	-------	-------	-------

$$(a) \quad NM = (\beta - 1) \cdot \beta^{t-1} = (2 - 1) \cdot 2^{3-1} = 4$$

(b) $NE = S - I + 1 = 3 - (-3) + 1 = \mathbf{7}$

(c) $NR = 2 \cdot NM \cdot NE + 1 = \mathbf{57}$

(d) $\boxed{0 \mid 1 \mid 0 \mid 0 \mid 1 \mid 1 \mid 0 \mid 0} \longrightarrow m.p. = (0.1)_2 \cdot (2^{-3})_{10} = (2^{-1})_{10} \cdot (2^{-3})_{10} = (0.0625)_{10}$

Logo, a região de *underflow* é $\{x \in \mathbb{R} \mid -(\mathbf{0.0625})_{10} < x < (\mathbf{0.0625})_{10}\}$.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \longrightarrow M.P. = (0.111)_2 \cdot (2^3)_{10} = (0.875)_{10} \cdot (2^3)_{10} = (7)_{10}$$

Logo, a região de *overflow* é $\{x \in \mathbb{R} \mid x < -(7)_{10} \cup x > (7)_{10}\}$.

(e) A precisão decimal equivalente é aproximadamente $\log_{10}(2^3) = 3 \cdot \log_{10}(2) \approx \mathbf{0.90308998699}$.

(f) $F(2, 3, 0, 6)$ (polarização $p = +3$)

6) $F(\beta, t, I, S) \rightarrow F(2, 3, 0, +7) \rightarrow$

s_1	d_1	d_2	d_3	0	e_1	e_2	e_3
-------	-------	-------	-------	---	-------	-------	-------

$$(a) \quad NM = (\beta - 1) \cdot \beta^{t-1} = (2 - 1) \cdot 2^{3-1} = 4$$

(b) $NE = S - I + 1 = 7 - 0 + 1 = 8$

(c) $NR = 2 \cdot NM \cdot NE + 1 = \mathbf{65}$

(d) $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \longrightarrow m.p. = (0.1)_2 \cdot (2^0)_{10} = (2^{-1})_{10} \cdot (1)_{10} = \mathbf{(0.5)_{10}}$

Logo, a região de *underflow* é $\{x \in \mathbb{R} \mid -(0.5)_{10} < x < (0.5)_{10}\}$.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow m.p. = (0.111)_2 \cdot (2^7)_{10} = (0.875)_{10} \cdot (2^7)_{10} = (\mathbf{112})_{10}$$

Logo, a região de *overflow* é $\{x \in \mathbb{R} \mid x < -(112)_{10} \cup x > (112)_{10}\}$.

(e) A precisão decimal equivalente é aproximadamente $\log_{10}(2^3) = 3 \cdot \log_{10}(2) \approx \mathbf{0.90308998699}$.

7) O formato de ponto flutuante de precisão dupla tem 1 bit de sinal, 11 bits para expoente e 52 bits explícitos para decimais (o 1º bit é implícito por conta do armazenamento do número de acordo com o padrão IEEE 754).

Também foi decidido neste padrão **facilitar** o armazenamento do expoente. Assim, um conceito chamado *exponent bias*, calculado por $2^{k-1} - 1$ onde k é o número de bits do expoente, foi criado com esta função. Este *bias* é essencial para a codificação do expoente como um valor **unsigned**. Para obter o expoente verdadeiro, subtrai-se o *bias* do expoente representado.

Assim sendo, $bias = 2^{11-1} - 1 = 1023$. $e_{min} = 1 - 1023 = -1022$ e $e_{max} = 2046 - 1023 = 1023$. (Os expoentes 0 e 2047 são reservados.)

Portanto, o menor número passível de ser representado com precisão dupla é $2^{-1022} \approx \mathbf{2.225 \cdot 10^{-308}}$, e o maior número é 2^{1023} com todos os bits da mantissa ativados, ou seja, $2^{1023} \cdot (2 - 2^{-52}) \approx \mathbf{1.797 \cdot 10^{308}}$. Às custas da perda de precisão, existem números que vão abaixo da fronteira de *underflow*, chamados de **subnormais**.

- ```
x1 = -0.0161072374089670233843208 x2 = -62.0838927625910343977011507
x1 = -0.0161072374089685811660022 x2 = -62.0838927625970313783909660
x1 = -0.0161 x2 = -62.08
x1 = -0.0161072 x2 = -62.0839
```

- $$\begin{aligned} \text{(c)} \quad \left| \frac{VA-VE}{VE} \right| &= \left| \frac{-0.0161 - (-0.0161072)}{-0.016172} \right| = \left| \frac{7.2 \cdot 10^{-6}}{-0.016172} \right| \approx \mathbf{0.000447} \\ \left| \frac{VA-VE}{VE} \right| &= \left| \frac{-62.08 - (-62.0839)}{-62.0839} \right| = \left| \frac{3.9 \cdot 10^{-3}}{-62.0839} \right| \approx \mathbf{0.00006282} \end{aligned}$$

- 12)** (a) Sabe-se que  $(10)_{10} = (1010)_2$ . O cálculo da parte decimal segue abaixo:  $-(0.05)_{10} = (?)_2$

$$\begin{array}{cccccccccc} 0.05 & 0.1 & 0.2 & 0.4 & 0.8 & 0.6 & 0.2 & 0.4 & 0.8 & 0.6 \\ \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \dots \\ \color{red}{0.1} & \color{red}{0.2} & \color{red}{0.4} & \color{red}{0.8} & \color{red}{1.6} & \color{red}{1.2} & \color{red}{0.4} & \color{red}{0.8} & \color{red}{1.6} & \color{red}{1.2} \end{array}$$

$$-(0.05)_{10} = -(0.000011)_2 = 1 \cdot 2^{-5} \cdot -(0.1001)_2$$

$$(-10.05)_{10} = (1010.0000\overline{1100})_2 = (1.0100000\overline{1100})_2 \cdot 2^3$$

|                  |   |          |                         |
|------------------|---|----------|-------------------------|
| Resultado final: | 1 | 10000010 | 01000001100110011001101 |
|------------------|---|----------|-------------------------|

- 13)**

|   |          |                            |
|---|----------|----------------------------|
| 1 | 00000000 | 10000000000000000000000001 |
|---|----------|----------------------------|

 (Número subnormal)

$$e = 0, f \neq 0 \longrightarrow v = (-1)^s \cdot \left( \sum_{i=1}^{23} b_{23-i} 2^{-i} \right) \cdot 2^{e-126} = (-1)^1 \cdot (2^{-1} + 2^{-23}) \cdot 2^{-126} \approx -\mathbf{5.877473 \cdot 10^{-39}}$$

- 14)** (a)  $(-1.51 \cdot 10^{+37})_{10} = (-1.011010111000010011101111010 \dots)_2 \cdot 2^{123}$   
 $s = 1$  (número negativo)  
 $e = 123 + 127 = (250)_{10} = (11111010)_2$  (*bias* + 123 bits decimais)  
 $f = (01101011100001001110111)_2 + (1)_2 = (01101011100001001111000)_2$  (primeiros 23 bits + arredondamento) Resultado final: 

|          |                 |                                |
|----------|-----------------|--------------------------------|
| <b>1</b> | <b>11111010</b> | <b>01101011100001001111000</b> |
|----------|-----------------|--------------------------------|

- (b) Convertendo novamente o número para decimal, tem-se  $VA = (-1.51000004197986229847484292417978368 \cdot 10^{37})_{10}$ .  
O percentual de erro é igual a  $\left| \frac{VA-VE}{VE} \right| \cdot 100 \approx \mathbf{2.7801 \cdot 10^{-6}\%}$ .

- 15)** (a)  $(-1.1 \cdot 10^{-41})_{10} = (-1.111010101001110111001100000 \dots)_2 \cdot 2^{-137}$   
 $s = 1$  (número negativo)  
 $e = -137 + 127 = -10 < 0 \rightarrow$  número subnormal  
 É necessário mover o decimal para que o número possa ser representado com  $e = 0$ . Então,  
 $(-1.111010101001110111001100000 \dots)_2 \cdot 2^{-137} = (-0.0000000001111010101001 \dots)_2 \cdot 2^{-126}$ .  
 Resultado final: 

|          |                 |                                |
|----------|-----------------|--------------------------------|
| <b>1</b> | <b>00000000</b> | <b>00000000001111010101010</b> |
|----------|-----------------|--------------------------------|

(b) Convertendo novamente o número para decimal, tem-se  $VA = -1.1000192944949814 \cdot 10^{-41}$ .  
O percentual de erro é igual a  $\left| \frac{VA-VE}{VE} \right| \cdot 100 = \mathbf{1.754044998309 \cdot 10^{-3}\%}$ .

16) Caso seja viável calcular o resultado desejado à mão, ou utilizando ferramentas de maior precisão, é possível verificar o erro relativo sem maiores dificuldades. Porém, na maioria dos casos isso não acontece, e é necessário observar como o computador se comporta nestes casos. Esta aritmética de ponto flutuante é uma grande área de estudo do cálculo numérico.

```
17, 18) from functools import reduce
from math import factorial as fact
from numpy import float32 as single

n = 4
x = -.111
desired = 0.894938748929031

for y in [single(x), x]:
 obtained = reduce(lambda w, i: w + (y**i)/fact(i), range(n), 0)
 print("{:.65f}%".format(abs((desired - obtained) / desired) * 100))
```

- O decimal desejado foi retirado do [Wolfram Alpha](#), para comparação com os resultados calculados pelo programa em variáveis de precisão simples e dupla, respectivamente. O erro da variável de precisão simples é maior pela quantidade reduzida de bits disponíveis para armazenar a parte decimal do resultado.
- Novamente, a função *reduce* é utilizada para simular o cálculo da série de Maclaurin com os argumentos fornecidos.
- Saída do programa:

```
0.00069152325516942810771509053680006218201015144586563110351562500%
0.00069138016857893662848316695956896182906348258256912231445312500%
```