Universidade Federal de Santa Catarina - UFSC
Centro Tecnológico - CTC
Departamento de Informática e Estatística - INE

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

Resoluções [1]

1) (a) $(10,1011)_2 = 1 \cdot 2^1 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \mathbf{(2,6875)_{10}}$

(b) $(10,57)_{10} = (?)_2$

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

$$\longleftarrow$$

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

$$(0,57)_{10} = (?)_2 \longrightarrow$$

$$\begin{array}{cccccccccccccccc} 0,57 & 0,14 & 0,28 & 0,56 & 0,12 & 0,24 & 0,48 & 0,96 & 0,92 & 0,84 & 0,68 & 0,36 & 0,72 & 0,44 & \\ \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \ldots \\ 1,14 & 0,28 & 0,56 & 1,12 & 0,24 & 0,48 & 0,96 & 1,92 & 1,84 & 1,68 & 1,36 & 0,72 & 1,44 & 0,88 & \end{array}$$

$$(0,57)_{10} = (0,10010001111010\ldots)_2$$

Logo, $(10,57)_{10} = \mathbf{(1010,10010001111010\ldots)_2}$.

2)

```python
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 à 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
0.333333333333333259318465024990
0.333333333333333037273860099958
0.333333333333332149095440399833
0.333333333333328596381761599332
0.333333333333314385527046397328
0.333333333333257542108185589314
0.333333333333030168432742357254
0.333333333332120673730969429016
0.333333333328482694923877716064
0.333333333313930779695510864258
```

3) (a) $(3021)_{F!} = 3 \cdot 4! + 0 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! = \mathbf{(77)_{10}}$

(b) $(4321)_{F!} = 4 \cdot 4! + 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! = \mathbf{(119)_{10}}$

(c) $(10000)_{F!} = 1 \cdot 5! = \mathbf{(120)_{10}}$

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

(d) $(0,02)_{F!} = \frac{0}{2!} + \frac{2}{3!} = (\frac{2}{6})_{10} = (\frac{1}{3})_{10} = \mathbf{(0,333333\ldots)_{10}}$

(e) $(0,113)_{F!} = \frac{1}{2!} + \frac{1}{3!} + \frac{3}{4!} = (\frac{19}{24})_{10} = \mathbf{(0.791\overline{6}\ldots)_{10}}$

(f) $(321,123)_{F!} = 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + \frac{1}{2!} + \frac{2}{3!} + \frac{3}{4!} = (\frac{575}{24})_{10} = \mathbf{(23.958\overline{3})_{10}}$

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) $(41,1)_{10} = (?)_2 = (?)_{16}$

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

$$\longleftarrow$$

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

$$(0,1)_{10} = (?)_2 \longrightarrow$$

$$\begin{array}{cccccccccccc} 0,1 & 0,2 & 0,4 & 0,8 & 0,6 & 0,2 & 0,4 & 0,8 & 0,6 & 0,2 & \\ \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \times 2 & \ldots \\ 0,2 & 0,4 & 0,8 & 1,6 & 1,2 & 0,4 & 0,8 & 1,6 & 1,2 & 0,4 & \end{array}$$

$$(0,1)_{10} = (0,00011001100110011\ldots)_2$$

Logo, $(41,1)_{10} = (101001,0\overline{0011})_2 = (0010\ 1001,0001\ \overline{1001})_2 = \mathbf{(29,1\overline{9})_{16}}$. 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) $NM = (\beta - 1) \cdot \beta^{t-1} = (2-1) \cdot 2^{3-1} = \mathbf{4}$

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

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

(d)

0	1	0	0	1	1	0	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 $\{\boldsymbol{x \in \mathbb{R} \mid -(0,0625)_{10} < x < (0,0625)_{10}}\}$.

0	1	1	1	0	1	0	0

$\longrightarrow M.P. = (0,111)_2 \cdot (2^3)_{10} = (0,875)_{10} \cdot (2^3)_{10} = (7)_{10}$

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

(e) ???

(f) ???

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

s_1	d_1	d_2	d_3	0	e_1	e_2	e_3

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

(b) $NE = S - I + 1 = 7 - 0 + 1 = \mathbf{8}$

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

(d)

0	1	0	0	0	0	0	0

$\longrightarrow m.p. = (0,1)_2 \cdot (2^0)_{10} = (2^{-1})_{10} \cdot (1)_{10} = \mathbf{(0,5)_{10}}$

Logo, a região de *underflow* é $\{\boldsymbol{x \in \mathbb{R} \mid -(0,5)_{10} < x < (0,5)_{10}}\}$.

0	1	1	1	0	1	1	1

$\longrightarrow m.p. = (0,111)_2 \cdot (2^7)_{10} = (0,875)_{10} \cdot (2^7)_{10} = \mathbf{(112)_{10}}$

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

(e) ???

9)

```
from numpy import float32 as single

h = 1/2
x = 2/3 - h
y = 3/5 - h
e = 3*x - h
f = 5*y - h
g = e/f

for i in [h, x, y, e, f, g]:
    print("{:.55f}\n{:.55f}\n".format(single(i), i))
```

- O processo descrito em código pode ser reescrito aritmeticamente da seguinte maneira:

$$g = \frac{e}{f} = \frac{3 \cdot x - h}{5 \cdot y - h} = \frac{3 \cdot (\frac{2}{3} - h) - h}{5 \cdot (\frac{3}{5} - h) - h} = \frac{3 \cdot (\frac{2}{3} - \frac{1}{2}) - \frac{1}{2}}{5 \cdot (\frac{3}{5} - \frac{1}{2}) - \frac{1}{2}} = \frac{3 \cdot \frac{1}{6} - \frac{1}{2}}{5 \cdot \frac{1}{10} - \frac{1}{2}} = \frac{\frac{1}{2} - \frac{1}{2}}{\frac{1}{2} - \frac{1}{2}} = \frac{0}{0}$$

  Porém, nota-se que na saída do programa, as variáveis `x` e `y` são responsáveis pelo resultado válido (`g` $= 1$), em virtude da imprecisão de seu armazenamento quando $\beta = 2$ (as multiplicações realizadas nunca serão iguais a um valor exato binariamente como $\frac{1}{2}$). Assim, como `e` $=$ `f` e $e, f \neq 0$, então $\frac{e}{f} = g = 1$.

- Saída do programa:

```
0.5000000000000000000000000000000000000000000000000000000
0.5000000000000000000000000000000000000000000000000000000

0.1666666716337203979492187500000000000000000000000000000
0.1666666666666666296592325124947819858789443969726562500

0.1000000014901161193847656250000000000000000000000000000
0.0999999999999999777955395074968691915273666381835937500

-0.0000000000000001110223024625156540423631668090820312500
-0.0000000000000001110223024625156540423631668090820312500

-0.0000000000000001110223024625156540423631668090820312500
-0.0000000000000001110223024625156540423631668090820312500

1.0000000000000000000000000000000000000000000000000000000
1.0000000000000000000000000000000000000000000000000000000
```

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("{:.70f}%".format((desired - obtained) / desired))
```

- 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:

```
0.0000069152325516942810432695874778286082573686144314706325531005859375%
0.0000069138016857893659460184906939694826633058255538344383239746093750%
```