Trabalho 01 - Erros de arredondamento, representação numérica e precisão

Arthur Vieira Silva

09 de Abril de 2025

Sumário

1 Introdução

Sabe-se que os computadores não são capazes de representar alguns números de forma exata, mas sim aproximada, utilizando a notação de ponto flutuante. Por conseguinte, erros de arredondamento e de truncamento que se acumulam em operações sucessivas podem ocorrer. Além disso, a conversão de valores com precisões distintas pode ocasionar a perda dos algarismos armazenados.

Diante desse contexto, foram propostas, neste trabalho, três tarefas que serão implementadas utilizando a linguagem Python. Sendo assim, o problema inicial é implementar um algoritmo que converta alguns números do sistema binário para o decimal, sem utilizar funções que realizem essa conversão de maneira direita. Em seguida, o objetivo é armazenar alguns valores em uma variável com precisão dupla (64 *bits*) e copiá-los para uma variável com precisão simples (32 *bits*) e, portanto, obter uma precisão estimada do número de algarismos que a variável de 32 *bits* conseguiu armazenar corretamente. Por fim, a última tarefa é estimar o erro de truncamento das operações realizadas entre duas variáveis do tipo *float* de 32 *bits*.

2 Solução numérica dos problemas propostos

2.1 Problema 1

No primeiro problema apresentado, o objetivo é converter alguns números binários para decimal (sem o uso de funções de conversão direta). Para isso, no algoritmo implementado, os cinco números estão armazenados em uma lista no Python e não estão representados utilizando notação científica. Por exemplo, o número $1,100001010001001 \times 2^{-1}$ será armezando como $0,1100001010001001$. Além disso, o tipo dos dados são strings, para que seja possível iterar sobre os seus dígitos e também dividir a parte inteira da fracionária.

Sendo assim, o funcionamento do algoritmo é dividir as partes inteiras e fracionárias de cada número binário e convertê-las para decimal da seguinte forma: um laço *for* será utilizada para iterar sobre cada posição de acordo o número de algarismos da parte inteira ou fracionária. Para a parte inteira,

o número decimal convertido será obtido a partir da multiplicação entre um dado algarismo (0 ou 1) e a potência $2^{numDigitosInteiros-i-1}$, assim é possível obter qual o expoente da potência da mesma forma clássica utilizada na conversão de um número binário para decimal e, além disso, a conversão de cada dígito será somada e armazenada em uma determinada variável utilizada para guardar o valor em decimal do número da sua respectiva parte (nesse caso, a parte inteira). Já para a parte fracionária, o mecanismo é o mesmo e a única alteração é a potência que será $2^{-(i+1)}$. Dessa maneira, ao final dos dois laços, as strings da parte inteira e da parte fracionária serão concatenadas para formar o número decimal desejado, que será adicionado na lista de números decimais.

Assim sendo, a partir dos números decimais fornecidos no exercício, as seguintes conversões foram obtidas utilizando o algoritmo descrito anteriormente:

Números binários	**Números decimais**
$1,100001010001001 \times 2^{-1}$	0.07599029541015625
$1,100001010001001 \times 2^{5}$	48.06337890625
$1,101101100101001 \times 2^{8}$	438.03203125
$1,100100111111011 \times 2^{17}$	51707.00
$1,000111110101010 \times 2^{-14}$	0.00006850436329841614

2.2 Problema 2

Para este problema, a ideia é armazenar valores em uma variável "X"de 64 *bits* e copiá-los para uma variável "Y"de 32 *bits* e, com isso, realizar uma estimativa da precisão decimal equivalente da variável "Y"em cada caso. Dessa forma, para calcular a precisão, é necessário obter o erro relativo, ou seja, entre o valor exato e o valor truncado por meio da fórmula $ER = \left|\frac{(Y-X)}{Y}\right|$. A partir do erro relativo obtido, a precisão estimada em cada caso será calculada utilizando a fórmula $int(|\log_{10}(ER)|)$.

Desse modo, o programa apenas armazena os valores dos números fornecidos no exercício em uma variável de precisão dupla (64 *bits*) e copia o seu valor para uma outra variável de precisão simples (32 *bits*). Assim, calcula-se o erro relativo e a precisão decimal equivalente em cada caso, a partir das fórmulas descritas anteriormente. Também foi necessário adicionar tratamentos de exceções para lidar com o *Underflow* do número da letra b) e com o *Overflow* do número da letra c).

Portanto, após a execução do programa, a saída encontrada foi a seguinte:

X	**Y**	**Precisão**
$5,21$	5.210000038146973	8 dígitos
$3.5e-94$	0.0	0 dígitos (*Underflow*)
$4.7499999999999994e+113$	inf	0 dígitos (*Overflow*)

2.3 Problema 3

Por fim, o último problema proposto pede para estimar o truncamento de algumas operações realizadas com registros de 32 *bits* (variáveis tipo *float*). Nesse contexto, como os valores fornecidos serão armazendos em variávies de precisão simples, após a realização de operações entre essas variávies, pode ser que não seja possível representar o resultado da operação em um registro de 32 *bits* então, os algarismos que não são possíveis de serem representados, são truncados/aproximados para ser viável a representação do resultado dessa operação em uma variável do tipo *float*.

Sendo assim, o algoritmo implementado armazena os dois valores fornecidos em variáveis de 32 *bits* e também em variáveis de 64 *bits* (a fim de estimar o truncamento). E, assim, realiza-se as operações (soma, subtração, multiplicação e divisão) tanto entre as duas variáveis *float* quanto entre as duas variáveis *double*. Após o cálculo de cada operação, calcula-se o erro absoluto $|x - \overline{x}|$ e o erro relativo $\frac{|x-\overline{x}|}{|x|}, x \neq 0$.

Com isso, após a execução do algoritmo, a seguinte saída foi obtida:

x (32 *bits*)	**y (32 *bits*)**	**x (64 *bits*)**	**y (64 *bits*)**
0.20000000298023224	0.19999000430107117	0.2	0.19999

- $x + y$:
    1. 32 $bits = 0.3999900221824646$
    2. 64 $bits = 0.39999$
    3. Erro absoluto $= 2.2182464587405804e - 08$
    4. Erro relativo $= 5.545754790721219e - 08$
- $x - y$:
    1. 32 $bits = 9.998679161071777e - 06$
    2. 64 $bits = 1.0000000000010001e - 05$
    3. Erro absoluto $= 1.3208389382235453e - 09$
    4. Erro relativo $= 0.00013208389382222244$
- $x \times y$:
    1. 32 $bits = 0.03999800235033035$
    2. 64 $bits = 0.039998000000000006$
    3. Erro absoluto $= 2.3503303470118198e - 09$
    4. Erro relativo $= 5.876119673513224e - 08$
- x/y:
    1. 32 $bits = 1.0000499486923218$
    2. 64 $bits = 1.000050002500125$
    3. Erro absoluto $= 5.380780332409074e - 08$
    4. Erro relativo $= 5.380511293392453e - 08$

3 Análise dos resultados

Sobre o primeiro problema, a sua análise é bastante simples, note que os números binários fornecidos foram corretamente convertidos para o decimal correspondente, a conversão realizada respeitou a parte inteira e fracionária dos números, posicionando o ponto na posição correta ao converter o número.

Agora no segundo problema, é possível visualizar que, na letra **a)**, a precisão decimal equivalente da variável "Y"foi de 8 dígitos, assim sendo, observa-se um pequeno erro nos últimos dígitos, devido à limitação no armazenamento dos *bits* da variável "Y". Em relação à letra **b)**, o número armazenado na variável de precisão dupla foi tão pequeno que não foi possível armazená-lo em uma variável de precisão simples, uma vez que está fora da sua faixa de valores e, nesse caso, ocorreu um *underflow* e o valor armazenado foi o 0. Por fim, na letra **c)**, ocorreu o inverso, ou seja, o valor armazenado na variável de 64 *bits* é tão grande que ultrapassou o valor máximo que pode ser armazenado em uma variável de 32 *bits*, resultado em um *overflow* e, portanto, a variável recebeu inf (infinito).

Para finalizar, no terceiro problema, já é possível visualizar que, apesar dos valores nos dois tipos de variáveis serem os mesmos, ocorre um erro de truncamento antes mesmo das operações serem realizadas. Após a realização das operações, é possível afirmar que a soma, a multiplicação e a divisão apresentaram erros absolutos e relativos muito pequenos, não sendo signaficativos o suficiente para prejudicarem o resultado da operação. Por outro lado, a operação de subtração apresentou um erro relativo muito elevado, apesar do erro absoluto ainda ser baixo, isso se deve ao fato da operação estar sendo realizada entre dois números muito próximos entre si, ocasionando um evento chamado "cancelamento catastrófico", no qual pode haver perda de inúmeros dígitos de precisão após a operação ter sido efetuada.