張老師，各位同仁：

因為張老師提出的一個深刻的問題，我翻出自己寫的書《計算機概論 16 講》第 D 講（第 13 講的十六進制數字），複習了「浮點數」：資訊工程界有三種浮點數的規格，我這裡一律指「雙精度浮點數」（double precision floating numbers）。上週日（11 月 11 日光棍節）早上，我用 C 語言草草地寫了一支程式，把記憶體內的浮點數老實列印出來：印出 64 個 0 或 1。這篇短文向張老師和大家做個主題為〈**浮點數運算與顯示的有效位數**〉的簡報，內容都用前述程式確認過。

我把第 D 講（共 8 頁，其實那本書的每一講都是 8 頁，頁碼是二進制的）的原稿複印出來，附在本文之後，敬請參酌。附件是該書付梓前的原稿，未經外審的修訂與編輯的校訂，但是本質上應該跟出版品沒有差別。我自認為這一講，把「浮點數」解說得足夠清楚了。據說有些同仁，在教「數值分析」這門課的前兩週，正式進入數值分析內容之前，就用這一講當作教材。

我們輸入給電腦的任何數 x，必須先換成最靠近的浮點數 $\mathrm{fl}(x)$，才能進入電腦。我們用電腦做 $x+y$ 的時候，實際做的是 $\mathrm{fl}(\mathrm{fl}(x)+\mathrm{fl}(y))$，電腦硬體保證每一次的 $\mathrm{fl}(\cdot)$ 都會是「最佳結果」，也就是誤差最小的結果（詳於後）：「誤差」表示「絕對誤差」，即 $|x-\mathrm{fl}(x)|$。雖然每一步的誤差都很小，但是很多步的計算之後，顯然誤差會「傳播」開來；「數值分析」的研究課題之一，就是控制誤差的「傳播」不至於「氾濫成災」。我們不進入這個細節。

就拿 0.1 這個超級無害的小數來說，0.1 的二進制數字卻變成了無窮循環小數：

$$0.0001100110011\cdots_b = 0.\overline{00011}_b$$

我們不妨驗算一下：

$$0.\overline{00011}_b = p + \frac{p}{2^4} + \frac{p}{2^8} + \cdots + \frac{p}{2^{4n}} + \cdots \text{，其中 } p = \frac{1}{2^4} + \frac{1}{2^5} = \frac{3}{32}$$

所以

$$0.\overline{00011}_b = \frac{3}{32} \times \frac{16}{15} = \frac{1}{10} = 0.1 \text{。}$$

要把 $0.\overline{00011}_b$ 寫成浮點數，首先要把它計成二進制的科學記號：

$$0.1 = 1.10011001100110011\cdots_b \mathrm{B}\ \ -4$$

其中 B 的前面是二進制小數，B 的後面是 2 的指數，我用十進制整數表現，比較容易讀。電腦的「**有限性**」呈現在以下兩個方面：

1. 數值範圍有限：B 只能是介於 -1022 和 1023 之間（含）的整數。
2. 解析度的有限：在每一個 $[2^k, 2^{k+1})$ 區間內，均勻分布 2^{52} 個浮點數。（負浮點數為正數的鏡射，不另討論。）

在每一個 $[2^k, 2^{k+1})$ 區間內，相鄰兩個浮點數的距離是 $\epsilon_k = 2^{-52+k}$，所以如果 $x \in [2^k, 2^{k+1})$，則 $|x-\mathrm{fl}(x)| \le \frac{1}{2}\epsilon_k$；可是其相對誤差則為定數 $\frac{1}{2}\epsilon_0 = \frac{1}{2^{53}}$。當 x 落在相鄰兩個浮點數 f_1、f_2 之間，如果 x 小於其中點，則 $\mathrm{fl}(x) = f_1$；如果 x 大於其中點，則 $\mathrm{fl}(x) = f_2$；但是如果 x 恰為中點，則硬體遵循一套「零捨壹入」的特殊規則，使得它選擇 f_1 或 f_2 的機率各半，我們就不要再深入細節了吧。

前述性質 2 的意思是，浮點數僅保留科學記號之 52 位小數。為了方便閱讀，以下我給每四個 bits 寫一個逗點，把 52 位小數分成 13 節：

$$0.1 = 1.1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001_b \text{ B } -4$$

紅色數字代表超過 52 位的部份，因為它超過 ϵ_0 的一半，所以選擇了較大的浮點數，得到

$$\text{fl}(0.1) = 1.1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1010_b \text{ B } -4$$

把上述二進位數字「全部」轉化為十進制數字，結果如下：

$$\text{fl}(0.1) = 0.1000,0000,0000,0000,0555,1115,1231,2578,2702,1181,5834,0454,1015,6250$$

因為 $0.1 \in [2^{-4}, 2^{-3})$，所以 0.1 和 fl(0.1) 之間的誤差不超過 $2^{-57} \approx 7 \times 10^{-18}$，所以前面從小數點下第 18 位開始，就是不值得參考的，也就是上面那個數的紅色部分（它大約是 5.6×10^{-18}）。那一串紅色數字其實是沒有參考價值的「殘渣」，它們是因為把浮點數尾巴的二進制數字轉換成十進制數字而造成的。例如，當 B 是 0 的時候，浮點數最末的 1 的意思是

$$2^{-52} = 2.2204,4604,9250,3130,8084,7263,3361,8164,0625 \times 10^{-16}$$

一般而言，因為 x 和 fl(x) 之間的相對誤差總是 $2^{-53} \approx 1.11 \times 10^{-16}$，所以當 fl($x$) 寫成十進制的科學記號數字時，從小數點下第 16 位開始，就都可能是「殘渣」而不值得參考了。以下根據張老師的建議，舉出一系列的例子，我將小數點下第 15 位標示出來，在它之後都可能是殘渣。其中「=」右邊是指雙精度浮點數的計算結果，所以「=」並非數學的「等於」意涵。

$\frac{2}{3} = 6.666666666666666296592\cdots \times 10^{-1}$ → 0.666666666666667

$1\frac{2}{3} = 1.666666666666666740681\cdots \times 10^{0}$ → 1.66666666666667

$2\frac{2}{3} = 2.666666666666666518636\cdots \times 10^{0}$ → 2.66666666666667

$4\frac{2}{3} = 4.666666666666666962726\cdots \times 10^{0}$ → 4.66666666666667

$8\frac{2}{3} = 8.666666666666666074547\cdots \times 10^{0}$ → 8.66666666666667

$16\frac{2}{3} = 1.666666666666666785090\cdots \times 10^{1}$ → 16.6666666666667

$32\frac{2}{3} = 3.266666666666666429819\cdots \times 10^{1}$ → 32.6666666666667

$64\frac{2}{3} = 6.466666666666667140361\cdots \times 10^{1}$ → 64.6666666666667

$128\frac{2}{3} = 1.286666666666666571927\cdots \times 10^{2}$ → 128.666666666667

$256\frac{2}{3} = 2.566666666666666856144\cdots \times 10^{2}$ → 256.666666666667

$512\frac{2}{3} = 5.12666666666666\mathbf{6}287710\cdots\times10^{2}$ → 512.666666666667

$1024\frac{2}{3} = 1.02466666666666\mathbf{6}742457\cdots\times10^{3}$ → 1024.66666666667

觀察以上系列，得知科學記號數字才能穩定地決定「可靠」的數值和「殘渣」的區隔。如果採用固定小數點下 15 位的顯示格式，則尾數就會越錯越多，所以我們不該採用那種顯示方法。如果一律設定顯示的有效位數為 15（按照課程說明手冊約定的語言），例如以上箭頭右側的數字，則都會是正確的。

張老師挑選這一組例子的原因是 $2/3 = 0.\overline{10}_b$。所以，把它加上 2、4、8、16…的效果，使得浮點數的小數點後，先插入 1 個、2 個、3 個、4 個…零，然後才開始 10_b 的循環。而它們的 52 個 bits 的最後一節將會以 1011_b 和 0101_b 的次序循環。看起來，計算結果的（十進制）尾數似乎也會出現循環現象。的確如此，它們的「尾數」會在 875、625 之間循環，可是它們出現在小數點下第 40 位以後；它們都是「殘渣」。對於有意義的數值部分，實際的影響是「10_b 循環」越晚發生，絕對誤差就越大，正確的小數位數逐漸變少，但是相對誤差保持不變，也就是正確的有效位數保持不變。

其實，只要你叫它做，C 語言可以輸出 32 位，甚至 64 位數字，就像我在前面表現的那樣。可是我們知道，更多的數字只是「殘渣」，沒有參考價值。輸出 16 位以內的科學記號數字，或者輸出 15 位以內的有效數字，才能保持數值計算結果的一致可信度。在高中數學課程裡使用計算機（calculator）的時候，我們建議一律採用 2 位、3 位或 4 位的有效數字；不論計算機多麼便宜簡樸，相信它總是辦得到 4 位正確有效數字的計算。

以上所言，相信都能符合電腦上執行的浮點數計算。但是，計算機的計算元件（ALU）比電腦的硬體簡單，通常不符合以上標準規格。但是，張老師已經發現了一種「逆向工程」的探索方法。以下，我用國教院採購（我長期借用）的 Fuh Bao FX-180 計算機為例。

1. 輸入 $5\div3$ 得到固定 10 位的輸出 1.666666667。
2. 接著輸入 -1.666666666 （9 位小數），按「=」獲得 6.7×10^{-10}。因此猜想它大概在 10^{-10} 之後發生殘渣。
3. $-10\div\log 2 \approx 33.2$，所以它的機器精度也許是 2^{-32} 或 2^{-33} 或附近的數。
4. 利用 y^x 按鍵，實驗 $1+2^{-32}-1$ 得到 2.3×10^{-10}，而 $1+2^{-33}-1$ 得到 1.2×10^{-10}，然後 $1+2^{-34}-1$ 就得到 0 了。可見它的精度是 2^{-33}，亦即其浮點數有 33 位小數。
5. 所以，這台計算機的相對誤差是 $2^{-33}\approx1.16\times10^{-10}$。它只有 9 位「可靠的」有效數字。而它選擇以 10 位數字輸出計算結果，也是合理的設計。

最後還有一個問題：為何 $5\div3$ 得到 1.666666667 但是 $2\div3$ 卻是 0.666666666？經過一些實驗，我獲得以下「反向工程」的猜想。我認為這是計算機的輸出演算法所造成的結果，而這個結果是「合理」的。前面說過，它可以輸出 9 位「可靠的」有效數字，所以當輸出 9 位以內的有效數字時，就不在尾數做四捨五入；反之，如果輸出了 10 位數字（它的螢幕僅能顯示 10 位數字），則最後一位是不可靠的，它做了四捨五入。例如 $2\div300$ 得到 0.006666666，只有 7 位有效數字，顯示的是未經四捨五入的正確數值。此時如果再輸入 -0.006666666 則看到 6.667×10^{-10}，可見它還有更多正確的有效數字沒有顯示出來。

當 FX-180 切換成科學記號數字的時候，它僅顯示 8 位數字，因為不滿 9 位，所以都是可靠的。

我試了 6^{30} 得到 2.2107391×10^{23}，以及 6^{-23} 得到 1.2662551×10^{-18}，都確定尾數是正確的，並沒有做四捨五入，因為它們其實是：

$$6^{30}=2.21073919720\cdots\times10^{23}\ ,\ 6^{-23}=1.26625519805\cdots\times10^{-18}$$

用同樣的手法，我測試了 Google 模擬計算機（以下簡稱 GG）。只要在 Chrome 裡面搜尋「1+1」就會呼叫出那個模擬計算機。（我不知道有沒有其他呼叫辦法？也不知道在其他瀏覽器裡面是否有一樣的結果？）。GG 明明可以用電腦的硬體做計算，卻要「降級」模擬一個僅能顯示 12 位數字的計算機。當 GG 算完一個數，我們按 [AC] 把顯示器歸 0 之後，會看到上方出現灰色的小字 Ans = …，顯示電腦硬體的計算結果。我猜想 GG 是用電腦的 CPU 做計算，只有在顯示答案的時候，模擬計算機的表現。當顯示的有效數字不足 12 個時，全部都可靠，所以尾數沒有四捨五入。觀察 $2\div3$ 和 $2\div300$ 都是如此。當 GG 採用科學記號數字時，它跟 FX-180 一樣顯示 8 位數字，可是 GG 卻在尾數做了四捨五入，6^{30} 和 6^{-23} 皆如此；所以 GG 的科學記號結果跟 FX-180 在尾數可能差 1。

像這樣的「不一致」實在也沒什麼道理，只能怪計算機的浮點數運算和輸出格式，並不像電腦有一個「業界」共同約定的標準。在高中數學課堂裡，如果要求每位老師了解以上的原理說明，會不會要求太多了？不論如何，在操作上，只要了解：(1) 電腦的有限性必定產生誤差，(2) 輸出格式也會造成少許困擾，然後 (3) 不要用太多「有效位數」就不會錯，這樣是不是就夠了？

請各位老師給建議，然後我也許投稿到《電子報》吧。謝謝大家。

單維彰

2018 年 11 月 14 日