Когда арифметика лжёт: числа с плавающей точкой

Откройте Python и наберите:

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
```

Это не баг Python. То же самое выдаст любой современный язык — C, C++, Java, JavaScript. И этому есть фундаментальная причина: **двоичная система счисления** не умеет точно представить десятичную дробь $0,1$.

Двоичные дроби и почему $0,1$ *— это «*$0,333\ldots$*»*

В десятичной системе $1/3 = 0,3333\ldots$ — бесконечная дробь. То же самое в двоичной системе случается с $0,1$:

$$0,1_{10} = 0,0001\,1001\,1001\,1001\ldots_2 \tag{1}$$

Дробь периодическая. Любой реальный компьютер хранит её, оборвав после конечного числа разрядов (обычно 52 двоичных знака после запятой — стандарт *IEEE 754, double precision*). Поэтому число, которое программист записывает как 0.1, на самом деле равно

$$0,1 \approx 0,100\,000\,000\,000\,000\,005\,551\,115\ldots \tag{2}$$

Когда мы складываем такое «искажённое $0,1$» с искажённым $0,2$, ошибки усиливаются — и получается тот самый «лишний хвостик» $4 \cdot 10^{-17}$, который мы видели в листинге.

> **i** Машинная точность $\varepsilon_{\text{маш}}$
>
> **Машинной точностью** называют наименьшее положительное число ε, такое что в компьютерной арифметике $1 + \varepsilon \neq 1$. Для стандарта double оно равно $\varepsilon_{\text{маш}} = 2^{-52} \approx 2,22 \cdot 10^{-16}$.
>
> Это означает: какое бы хорошее число x вы ни записали, оно известно компьютеру лишь с относительной точностью $\sim \varepsilon_{\text{маш}}$.

Когда это становится опасным: катастрофическое сокращение

Маленькая ошибка в 17-м знаке — мелочь? Не всегда. Рассмотрим безобидное на вид вычисление:

$$g(x) = \frac{1 - \cos x}{x^2} \qquad \text{при} \quad x = 10^{-8}. \tag{3}$$

По формуле Тейлора $1 - \cos x \approx x^2/2$, так что $g(10^{-8}) \approx 0{,}5$. Что выдаст компьютер?

```
>>> from math import cos
>>> x = 1e-8
>>> (1 - cos(x)) / x**2
0.0          # вместо 0.5!
```

Что произошло? Число $\cos(10^{-8}) \approx 1 - 5 \cdot 10^{-17}$ так близко к единице, что компьютер просто хранит его как *ровно единицу* (поправка $5 \cdot 10^{-17}$ меньше $\varepsilon_{\text{маш}}$). Разность $1 - \cos x$ обнулилась, и весь ответ обнулился вместе с ней.

Это явление называется **катастрофическим сокращением** (*catastrophic cancellation*): когда из двух близких чисел вычитают и теряют все значащие цифры разом.

Пример 0.1. Спасение через переписывание формулы

Воспользуемся тождеством $1 - \cos x = 2\sin^2(x/2)$:

$$g(x) = \frac{2\sin^2(x/2)}{x^2} = \left(\frac{\sin(x/2)}{x/2}\right)^2 \cdot \frac{1}{2}. \tag{4}$$

Внутри теперь нет вычитания близких чисел, и Python выдаёт 0.4999999999999983 — погрешность лишь в последнем знаке.

Важно

Главный практический вывод. Если ваш численный алгоритм выдаёт странный ответ, первая гипотеза, которую следует проверить, — это не «бага в библиотеке», а *катастрофическое сокращение* в вашей формуле. Часто его можно убрать, переписав формулу алгебраически эквивалентным, но численно устойчивым способом.

Историческая справка

Стандарт *IEEE 754* был принят в 1985 г., в значительной мере благодаря Уильяму Кэхэну (*William Kahan*, премия Тьюринга 1989 г.). До этого каждый производитель процессоров делал плавающую арифметику по-своему, и одна и та же программа на разных машинах могла давать заметно разные ответы. Кэхэн также является автором знаменитого **алгоритма компенсированного суммирования**, который позволяет складывать миллионы чисел с почти двойной точностью.