CSC 165

floating-point

week 11, lecture 2

Danny Heap

heap@cs.toronto.edu

www.cdf.toronto.edu/~heap/165/F09

resources: chapter 7 of course notes

http://docs.python.org/tutorial/floatingpoint.html

http://en.wikipedia.org/wiki/IEEE_754-2008

Due to peer pressure, we will celebrate Nov. 30th next Monday. — E4 due next Monday.

A3

recall: number representation

If you fix the cost of arithmetic operations, you fix the size of numbers
Each number is given the same space (usually bits)

Result: floating numbers are represented in scientific notation using some base β, a fixed number of digits, t, a certain range of exponents $e \in [e_{\min}, e_{\max}]$, and some way to store the sign.

Suppose your base $\beta = 2$, you allow a bit for the sign, you have room for $t = 3$ digits, and your exponents are from $[-2, 3]$. Normalize the representation of non-zero numbers so there is one non-zero digit to the left of the radix point

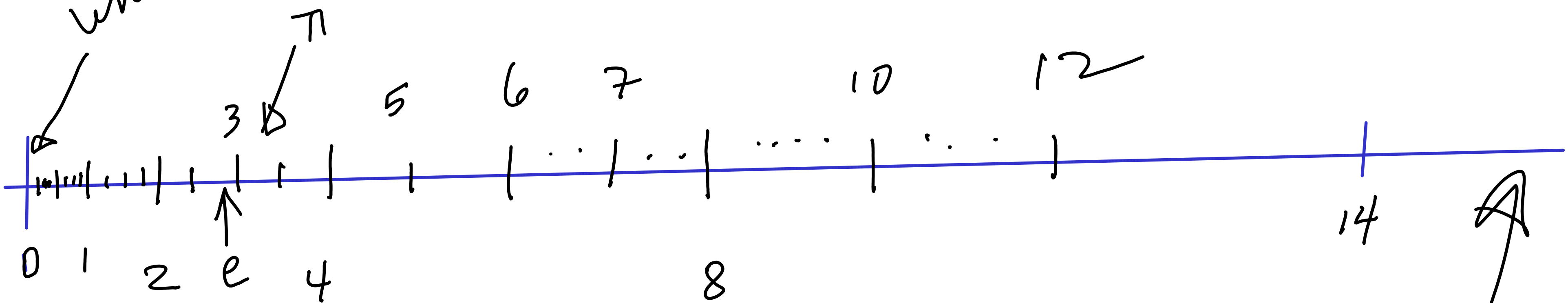
The smallest positive number you can represent in this system? $1.00 \times 2^{-2} = \frac{1}{4}$
The largest positive number you can represent in this system? $1.11 \times 2^{3} = 14$

1.00×2^{-2} ... 1.00×2^{-1}

a number list

1.00×2^{3}

underflow



1.

A number-line of the entire list of positive numbers isn't evenly-spaced

However the ratio of the gaps to the magnitude is roughly constant

overflow

How many positive numbers are there in total?

rounding

Since we're on a budget for digits and exponents, we can't represent infinitely many numbers.

Suppose our base $\beta = 10$, we have $t = 3$ digits, and exponents $[-3, -2, -1, 0, 1, 2]$.

How should we represent $e \approx 2.718281828459045$ or $\pi \approx 3.141592653589793$?

2.72×10^0

truncate or round-to-nearest

This leads to three varieties of error (all with a sewage analogy):

- *overflow:* There is no way to represent numbers larger than 9.99×10^2, so 999.5 is a problem.
- *underflow:* There is no way to represent positive numbers smaller than 1.00×10^{-3}, so $\frac{1}{1001}$ is a problem.
- *rounding error:* (throughflow?) There is no way to represent numbers strictly between adjacent numbers we can represent, so 1.001 is a problem.

absolute, relative

In our base-ten number system of the previous page, using round-to-nearest, the absolute error representing e is $|2.72 - 2.7182818284590451\ldots|$

We care about relative error. A millimeter error is a disaster in eye surgery but pretty acceptable in transcontinental air travel.

x – target quantity
x' – approximation

Compare the error to the quantity being sought.
So, if $x \neq 0$, the relative error is $\frac{|x-x'|}{|x|}$

$re = \frac{0.1}{1.0}$

Compare the relative error when $x = 1.0$ and $x' = 1.1$? How about about when $x = 100.0$ and $x' = 100.1$?

$\frac{0.1}{100}$

bounding round-to-nearest

If you had infinitely many digits and base β, you could *exactly* represent a number $d_0.d_1d_2\ldots d_{t-1}\ldots \times \beta^e$

t-1 steps to right of radix

But, if you're limited to t digits, you have to round up or down:

$d_0.d_1d_2\ldots d_{t-1} \times \beta^e$ or $d_0.d_1d_2\ldots(d_{t-1}+1) \times \beta^e$

diff $0.0\ldots .1 \times \beta^e$

$= \dfrac{1 \times \beta^{e-(t-1)}}{2}$

What's the maximum difference between x and x'?
(take into account round-to-nearest)

re $\dfrac{\beta^{\cancel{e}-(t-1)}}{\cancel{\beta^e} \cdot 2} = \left[\dfrac{\beta^{1-t}}{2}\right]$

What's the maximum relative error?
Use the fact that $1.0\ldots 0 \times \beta^e$ is the smallest possible denominator

$1.00 - \times \beta^e = \beta^e$

big relative error!

What relative error does the python shell produce for

1.0000000000000004 - 1.0000000000000003 = 0.6 − 01

By tweaking the numbers, you can make the error as bad as you like. . .

billions of percent error

The internal (binary) representation of floats is obscured by

the decimal display. Try `shipwright.binFloat()`