STAT 243 Problem Set 6

Treves Li

2024-10-30

Collaboration Statement

I did not collaborate with anyone.

Question 1

> In class we said that in the exponent of the 8 byte floating point representation, $e \in 0, ..., 2047$, since e is represented with 11 bytes ($2^{11} = 2048$). So that would suggest that the largest and smallest magnitude numbers (ignoring the influence of the 52 bit in the mantissa) are 2^{1024} and 2^{-1023}.

Question 1a

> However, 2^{1024} overflows. Create a numpy `float64` number larger than 10^{308} that overflows and is represented as `inf`. What is its bit-wise representation? Why does it now make sense that we can't work with 2^{1024} as a regular number?
>
> Note that there is still a bit of a mystery here in that it doesn't appear that the sequence 0 followed by 63 1s (or simply 64 1s) is used in any way, even though those would be a natural candidate to represent `inf` (and `-inf`). Extra credit for investigating and figuring out why that is (it might be difficult to determine and I don't know the answer myself).

I created a "big float" using `np.float64()` with a value larger than 10^{308}. To view its 64-bit representation in binary, I used `.view(np.int64)` to interpret it as an integer, then printed it in IEEE 754 standard format.

```
import numpy as np

# Create float64 greater than 10^308
big_float = np.float64(1.0e+309)
print(big_float)

# Get binary repr
big_float_bits = big_float.view(np.int64)
print(f"{big_float_bits:064b}")
```

```
inf
0111111111110000000000000000000000000000000000000000000000000000
```

The output shows that we have maxed out the allowable bits for the exponent field. IEEE 754 `float64` format uses an exponent range up to 2^{1023}, and values exceeding this limit trigger an overflow.

This bit pattern (where the exponent field is all `1`s and the mantissa field is all `0`s) is reserved specifically for representing "infinity". This design choice means we can't use 2^{1024} as a finite number, since the all-`1`s exponent configuration uniquely represents "infinity". In light of this, it makes sense then that we can't use 2^{1024} as a "regular" number (at least for `float64`).

Question 1b

> What is the bitwise repr of 2^{-1022} Given that, what would the bit-wise repr of be 2^{-1023} (work this out conceptually, not by trying to use 2^{-1023} in Python)? What number does that bitwise representation actually represent?
>
> I asked that you not try to use 2^{-1023} in Python. Doing that and exploring what is going on is part (c).

We know that floating points in IEEE 754 double-precision have the following structure:

$$(-1)^s \times 2^{e-1023} \times 1.f$$

This means that we can represent 2^{-1022} by matching the exponent of 2:

$$e - 1023 = -1022 \quad \Longrightarrow \quad e = 1$$

So, to represent 2^{-1022}, we want the exponent portion to simply be 1 in binary, which we can write out as:

0 00000000001 0000000000000000000000000000000000000000000000000000

Given the example above, we expect the bitwise representation of 2^{-1023} to be where the exponent is 0 (while also keeping the sign and the mantissa 0):

0 00000000000 0000000000000000000000000000000000000000000000000000

To see what 2^{-1023} actually represents in decimal, we can convert the binary representation to a 64-bit unsigned integer using `numpy`, and see that is 0.

```
# Prefix with 0b because we're giving numpy a binary literal
bin_repr = 0b0000000000000000000000000000000000000000000000000000000000000000

bin2int = np.uint64(bin_repr)

bin2int.view(np.float64)
```

```
np.float64(0.0)
```

Question 1c

Extra credit: By trial and error, find the base 10 representation of the smallest positive number that can be represented in Python. Hint: it's rather smaller than 1×10^{-308}. Explain how it can be that we can store a number smaller than 1×2^{-1022}, which is the value of the smallest positive number that we saw above. Start by looking at the bit-wise representation of 1×2^{-1023} and see it is not the same as what you worked out in part (b). Given the actual bit-wise representation of 1×2^{-1023}, show the progression of numbers smaller than that that can be represented exactly and show the smallest number that can be represented in Python written in both base 2 and base 10.

Hint: you'll be working with numbers that are not normalized (i.e., denormalized); numbers that do not have 1 as the fixed number before the radix point in the floating point representation we discussed in Unit 8.

We start