# Units in the Last Place, and Floating Point Errors

One “unit in the last place” is the minimum value required to reach the next floating point number. This gap is often labeled “epsilon” (). A notable feature of floating point numbers is the unit in the last place, and therefore the size of errors in representation is relative to the number being considered.

See <https://canvas.wisc.edu/courses/121385/pages/units-in-the-last-place> .

**Definition:**A ULP for a floating point number in base , precision p, and exponent e is an error scale measured relative to  .

Example: consider the number , represented as a single precision floating point binary (base2) number. Here, , , and . (We can approximate e by calculating . That is, representing in base 2 requires a leading digit in the place.) The ULP is . If we try to add in this representation, we do not have enough precision to reach the next floating point number beyond .

We can see this using Julia\_v1 a couple of ways. We can code

Float32(10^8+4)==Float32(10^8)

Which returns a value of “true”. We can also code

nextfloat(Float32(10^8))-Float32(10^8)

Which returns a value of 8.0f0.

Our absolute error is 4.0. Our relative error is . Measured as ULP, our relative error is .

Next consider the number . In single precision, our . To produce an example similar to the first example we might add . Our absolute error would be , our relative error would be , and our ULP error .

Definition: define a function that maps real numbers to floating point representations as

Where S is some suitably defined representation. Then we define *absolute error* as

And relative error as

**Lemma**: If z in base has exponent e, i.e. , then the maximum absolute error on fl(z) is .

**Proof**: If we think of z’s representation in base , this can be expressed as

(if then , i.e. z is exactly representable in base with precision p, and our absolute error will be zero). Meanwhile

And for , i.e. in all but the last digit of . In the last digit, we have either or , whichever is closer (we haven’t defined in detail, yet). The difference between the upper and the lower is therefore

We chose a rounding rule for such that it rounds to whichever floating point number is closer, so the absolute error is at most .

**Lemma**: If z in base has exponent e, the relative error is bounded by .

**Proof**: From the previous lemma, divide by z. Note as above that .

In computation, the upper bound on relative error is called “machine epsilon”. In Julia use eps().

>eps(Float16)=0.000977

>eps(Float32)=1.19e-7

**Homework**: what happens if we consider negative numbers?