fn conservative_continuous_gaussian_tail_to_alpha

Michael Shoemate

September 16, 2025

> This proof resides in **"contrib"** because it has not completed the vetting process.

Proof for `conservative_continuous_gaussian_tail_to_alpha`.

Definition 0.1. Define $X \sim \mathcal{N}(0, s)$, a random variable following the continuous gaussian distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \tag{1}$$

Definition 0.2. The error function is defined as:

$$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{2}$$

Definition 0.3. The complementary error function is defined as:

$$\text{erfc}(z) = 1 - \text{erf}(z) \tag{3}$$

Lemma 0.4. The implementation of `erfc` differs from a conservatively rounded implementation by no greater than one 32-bit float ulp.

Proof. The following code conducts an exhaustive search.

```python
# The result from this check is that the erfc function in statrs errs by at most 1 f32 ulp.

# First disagreement is at 0.5.
# Execution is slowest at inputs around 15, before switching to the next approximating curve.

# To use all CPUs, floats are sharded modulo the number of CPUs (less two).
# It may be necessary to restart this program at a later float to free memory.

import struct
import multiprocessing

# pip install gmpy2
import gmpy2
from opendp._data import erfc

# specifically check max ulp distance from a conservative upper bound
gmpy2.get_context().round = gmpy2.RoundUp


def floatToBits(f):
    s = struct.pack(">f", f)
    return struct.unpack(">l", s)[0]


```

```python
def bitsToFloat(b):
    s = struct.pack(">l", b)
    return struct.unpack(">f", s)[0]


def worker(offset, step):
    max_err = 0
    print(f"running {offset}")
    # iterate through all 32-bit floats
    for bits in range(floatToBits(0.0), floatToBits(float("inf")), step):
        bits += offset
        if offset == 0 and bits > 0 and (bits // step) % 10_000 == 0:
            prop_done = bits / floatToBits(float("inf"))
            print(
                f"{prop_done:.2%} done, with max discovered f32 ulp error of: {max_err}. Currently at: {bitsToFloat(bits)}"
            )

        f32 = bitsToFloat(bits)
        f32_ulp_err = abs(floatToBits(erfc(f32)) - floatToBits(gmpy2.erfc(f32)))
        max_err = max(max_err, f32_ulp_err)
    print(max_err)


if __name__ == "__main__":
    n_cpus = multiprocessing.cpu_count() - 2
    processes = []
    for cpu in range(n_cpus):
        p = multiprocessing.Process(target=worker, args=(cpu, n_cpus))
        p.start()
        processes.append(p)

    [p.join() for p in processes]
```

Upon completion, the greatest discovered error is at most 1 ulp. □

Theorem 0.5. Assume $X \sim \mathcal{N}(0, s)$, and $t > 0$.

$$\alpha = P[X \geq t] = \frac{1}{2}\text{erfc}\left(\frac{t}{\sigma\sqrt{2}}\right) \tag{4}$$

Proof.

$$\begin{aligned} \alpha &= P[X \geq t] \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_t^\infty e^{-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2} dt && \text{by } 0.1 \\ &= \frac{1}{2}\left(1 + \text{erf}\frac{t}{\sigma\sqrt{2}}\right) && \text{by } 0.2 \\ &= \frac{1}{2}\text{erfc}\left(\frac{t}{\sigma\sqrt{2}}\right) && \text{by } 0.3 \end{aligned}$$

The implementation of this bound uses conservative rounding down within erfc, as erfc is monotonically decreasing. The outcome of erfc is increased by one 32-bit float ulp, which guarantees a conservatively larger value, by 0.4. Therefore the entire computation results in a conservatively larger bound on the mass of the tail of the continuous gaussian distribution. □