

# fn conservative\_continuous\_gaussian\_tail\_to\_alpha

Michael Shoemate

November 29, 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:

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

**Definition 0.2.** The error function is defined as:

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

**Definition 0.3.** The complementary error function is defined as:

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

**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.

```
1 # The result from this check is that the erfc function in statsr errs by at most 1 f32 ulp.
2
3 # First disagreement is at 0.5.
4 # Execution is slowest at inputs around 15, before switching to the next approximating curve
5
6 # To use all CPUs, floats are sharded modulo the number of CPUs (less two).
7 # It may be necessary to restart this program at a later float to free memory.
8
9 import struct
10 import multiprocessing
11
12 # pip install gmpy2
13 import gmpy2
14 from opendp._data import erfc
15
16 # specifically check max ulp distance from a conservative upper bound
17 gmpy2.get_context().round = gmpy2.RoundUp
18
19
20 def floatToBits(f):
21     s = struct.pack(">f", f)
22     return struct.unpack(">L", s)[0]
23
24
```

```

25 def bitsToFloat(b):
26     s = struct.pack(">1", b)
27     return struct.unpack(">f", s)[0]
28
29
30 def worker(offset, step):
31     max_err = 0
32     print(f"running {offset}")
33     # iterate through all 32-bit floats
34     for bits in range(floatToBits(0.0), floatToBits(float("inf")), step):
35         bits += offset
36         if offset == 0 and bits > 0 and (bits // step) % 10_000 == 0:
37             prop_done = bits / floatToBits(float("inf"))
38             print(
39                 f"\{prop_done:.2%\} done, with max discovered f32 ulp error of: {max_err}."
40             )
41
42         f32 = bitsToFloat(bits)
43         f32_ulp_err = abs(floatToBits(erfc(f32)) - floatToBits(gmpy2.erfc(f32)))
44         max_err = max(max_err, f32_ulp_err)
45     print(max_err)
46
47
48 if __name__ == "__main__":
49     n_cpus = multiprocessing.cpu_count() - 2
50     processes = []
51     for cpu in range(n_cpus):
52         p = multiprocessing.Process(target=worker, args=(cpu, n_cpus))
53         p.start()
54         processes.append(p)
55
56     [p.join() for p in processes]

```

Upon completion, the greatest discovered error is at most 1 ulp.  $\square$

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

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

*Proof.*

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

The implementation of this bound uses conservative rounding down within  $\operatorname{erfc}$ , as  $\operatorname{erfc}$  is monotonically decreasing. The outcome of  $\operatorname{erfc}$  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.  $\square$