Assignment: Inverse Standard Normal (Probit) — Numerical Engineering Exercise

OP Kiitorata Trainee Program

Dear candidate,

You will receive a *single header-only* baseline implementation `InverseCumulativeNormal.hpp` that computes $\Phi^{-1}(x)$ via bisection (slow but accurate). Your task is to turn it into a production-grade implementation while *preserving the public API.*

This brief refines the earlier instructions by outlining one practical way to *derive* a fast rational approximation. Treat it as a template—you may use this or an equivalent approach, provided you meet the targets and explain your reasoning.

What you receive (API — keep unchanged)

- `quant::InverseCumulativeNormal(double average=0.0, double sigma=1.0);`
- `double operator()(double x) const;`
- `void operator()(const double* in, double* out, std::size_t n) const;`
- `static double standard_value(double x);`

Your tasks

1. **Replace bisection with a piecewise fast core** (central + tails) using rational functions evaluated by Horner's method.
2. **Polish with one or two refinement steps** (Halley preferred) to reach full double-precision accuracy.
3. **Keep properties**: monotonicity, symmetry $\Phi^{-1}(1-x) = -\Phi^{-1}(x)$, finite outputs on $(0, 1)$.
4. **Implement and speed up the vector overload** (SIMD optional; auto-vectorization is fine).
5. **Document** the derivation, error measurements, and performance.

Centering and symmetry. Let $g(x) = \Phi^{-1}(x)$. The normal symmetry implies $g(0.5 + u) = -g(0.5 - u)$, so around $x = \frac{1}{2}$ it is odd. Set

$$u = x - \tfrac{1}{2}, \qquad r = u^2.$$

Approximate.

$$g(x) \approx u\,\frac{P(r)}{Q(r)} = u\,\frac{a_0 + a_1 r + \cdots + a_m r^m}{\underbrace{1}_{Q(0)=1} + b_1 r + \cdots + b_n r^n}.$$

Scale invariance: $(\alpha P, \alpha Q)$ gives the same ratio, so we fix $b_0 = 1$. Rearranged *linear* system:

$$u \sum_{i=0}^{m} a_i r^i \;-\; z^\star \sum_{j=1}^{n} b_j r^j \;\approx\; z^\star.$$

Sampling $\{x_k\}$ yields $A\theta \approx y$ with $\theta = [a_0, \ldots, a_m, b_1, \ldots, b_n]^\top$. Solve the (optionally weighted) ridge least-squares:

$$\min_\theta \; \|W(A\theta - y)\|_2^2 + \lambda \|\theta\|_2^2, \quad W = \mathrm{diag}(w_1, \ldots, w_K).$$

Tips:

- **Do the fit in Python:** NumPy `lstsq` (and tiny λ if needed), use SciPy `norm.ppf` or mpmath for $z^\star$.
- Degrees: start with $m/n \in \{5/5,\, 6/6\}$.
- Nodes: Chebyshev-like grid mapped to $[10^{-6}, 1 - 10^{-6}]$ (include endpoints).
- Weights: slightly upweight samples near the central/tail join to ease matching.
- Export coefficients to C++ as `constexpr`; evaluate P, Q via Horner.

2. Tail approximation

For x near 0 or 1, let $m = \min(x, 1-x)$, $t = \sqrt{-2 \log m}$, $s = \mathrm{sign}(x - \frac{1}{2})$ and approximate

$$g(x) \approx s\,\frac{C(t)}{D(t)} = s\,\frac{c_0 + c_1 t + \cdots + c_p t^p}{\underbrace{1}_{D(0)=1} + d_1 t + \cdots + d_q t^q}.$$

Same scale invariance: fix $D(0) = 1$ to keep the system linear.
Linear system (left tail, $s = -1$):

$$s \sum_{i=0}^{p} c_i t^i \;-\; z^\star \sum_{j=1}^{q} d_j t^j \;\approx\; z^\star.$$

Tips:

- Nodes: log-spaced in x on $\left[10^{-16},\, x_{\mathrm{low}}/10\right]$ plus a short linear grid to $x_{\mathrm{low}} \approx 0.02425$; mirror by symmetry.
- Degrees: start with $p/q \in \{7/7,\, 9/9\}$.
- Weights: upweight both the extreme tail and near-join points.
- Refinement: after joining, apply 1–2 Halley steps; for very small/large x use a stable tail residual.

3. Joining the pieces

Choose a boundary $x_{\text{low}} \in [0.02, 0.05]$ and set $x_{\text{high}} = 1 - x_{\text{low}}$. Ensure the piecewise function is *continuous* at the join; derivative continuity is nice but not required (the refinement step will correct a small kink).

4. Refinement to full precision

Use one or two steps of **Halley**:

$$z \leftarrow z - \frac{r}{1 - \frac{1}{2} z\, r}, \qquad r = \frac{\Phi(z) - x}{\phi(z)}.$$

Note. The baseline implementation uses the direct residual $r = (\Phi(z) - x)/\phi(z)$. In the tails this suffers *catastrophic cancellation* because $\Phi(z)$ and x are nearly equal. Use the following algebra to obtain a numerically stable residual.

Right tail $(x > 0.5)$**.** Let $y = 1 - x \ll 1$ and $q = Q(z) = 1 - \Phi(z) \ll 1$:

$$r = \frac{\Phi(z) - x}{\phi(z)} = \frac{y - q}{\phi(z)} = -\frac{q - y}{\phi(z)} = -\frac{y\big(e^{\log q - \log y} - 1\big)}{\phi(z)} = -\frac{y\,\mathrm{expm1}\big(\log Q(z) - \log y\big)}{\phi(z)}.$$

Left tail $(x < 0.5)$**.** Let $y = x \ll 1$ and $a = \Phi(z) = 1 - Q(-z)$:

$$r = \frac{a - y}{\phi(z)} = \frac{y\big(e^{\log a - \log y} - 1\big)}{\phi(z)} = \frac{y\,\mathrm{expm1}\big(\log Q(-z) - \log y\big)}{\phi(z)}.$$

In a central band (e.g. $x \in [10^{-8},\, 1 - 10^{-8}]$) the direct residual is fine; switch to the log/expm1 forms above in the tails.

Why this works. The rational pieces give $\sim 10^{-5} \ldots 10^{-8}$ raw error quickly; *one* stable Halley step typically lands $\lesssim 10^{-12}$; a *second* step often pushes below 10^{-13}.

5. Alternative approach (optional)

The least–squares rational fit above is just one workable path. You may use *any* approach as long as you preserve the API and meet the accuracy/performance targets; briefly justify your choices and keep symmetry/monotonicity in mind.

One example only: fit in a transformed variable (e.g., the logit $y = \log \frac{x}{1-x}$ or the tail variable $t = \sqrt{-2 \log m}$ with $m = \min(x, 1 - x)$), then map back to x. If you do this, consider nonuniform sampling that emphasizes the central/tail join and extreme tails.

Targets (aim to meet or exceed)

- **Accuracy on** $[10^{-12}, 1 - 10^{-12}]$**:**
  - Without refinement: max abs error $\leq 10^{-4}$ (indicative).
  - With 1–2 Halley steps (stable residual): max abs error $\leq \mathbf{1 \times 10^{-10}}$ (goal $\leq 10^{-12}$).

  - Symmetry and strict monotonicity on $(0, 1)$.

- **Performance (report on your machine):**
  - Scalar throughput: $\geq$ **10**$\times$ faster than baseline bisection on 10^6–10^7 calls.
  - Vector overload: **parallelize or vectorize (method of your choice)** and achieve $\geq$ **1.5**$\times$ speedup over a naive single-thread loop on $n = 10^6$.

Testing & measurement

- **Correctness:** round-trip check $\Phi(\hat{z}) \approx x$ over a dense grid (and random points), including tails; symmetry test $g(1 - x) = -g(x)$.
- **Derivative sanity:** numerically verify $\dfrac{d}{dx}g(x) = \dfrac{1}{\phi(g(x))}$ at sample points.
- **Performance:** report ns/eval for 10^6 calls.

What to submit

- Modified `InverseCumulativeNormal.hpp` (header-only solution).
- `DESIGN.md` (max 1 page) with:
  - Your piecewise forms, degrees (m/n) and (p/q), join choice, and *how* you fitted coefficients (nodes, weights, any ridge).
  - Error evaluation (max/mean/$p99$), and how you computed a *stable* Halley residual.
  - Performance measurements and any vectorization notes.
  - Non-idealities or limitations.
- (Optional) tiny driver you used for timing.

Rules

- Keep the public API and names unchanged.
- Do not call any library inverse CDF / `erfinv` in the *final* core.
- Using `exp`, `log`, `erfc`, `expm1`, `log1p` is fine.
- If you use literature coefficients, say so and still show *your* validation and performance; deriving your own is a plus.

Evaluation rubric

Correctness & robustness	40%
Performance & vector path	20%
Engineering quality	20%
Design note (clarity, evidence)	20%