The Bernstein constant: ten rigorously certified digits

Hanyu Yang · ORCID 0009-0005-0419-4070

Version 1.0.0, published 2026-08-26. DOI 10.5281/zenodo.22106774 (concept), 10.5281/zenodo.22106775 (version 1.0.0).

This PDF is a typeset rendering of `NOTE.md` from branch `main` of https://github.com/SeverinVisionary/bernstein-constant-certificate at commit `b14c5e818f917b6c63d93bfdf50a282408e13eae` (file SHA-256 `b5e88e94896235654727aff03e0cb658ce0ac3be70be34bb32d22e35893a8460`, as listed in that commit's `CHECKSUMS.sha256`). The text below is that file unchanged; only this title block was added. Landing page: https://severinvisionary.github.io/bernstein-constant-certificate/

The release's canonical claim, machine-parsable and checked by `make headline` against `RELEASE_CLAIM.json`:

```
BEGIN CERTIFIED ENCLOSURE
lower_endpoint: 0.280169499016595460711186
upper_endpoint: 0.28016949904799999998
rounded_value: 0.2801694990
correctly_rounded_places: 10
END CERTIFIED ENCLOSURE
```

Computational note and data release.

Summary

Let `E_k(f; K)` be the error of best uniform polynomial approximation of degree at most `k`. Bernstein (1913) proved that

```
beta = lim_{n -> inf}  2n * E_{2n}( |x| ; [-1, 1] )
```

exists. This release contains a machine-checkable certificate for

```
0.280169499016595460711186  <=  beta  <=  0.28016949904799999998
```

Both endpoints are proved; the printed decimals are rounded **outward**, so the printed endpoints are themselves valid bounds. The enclosure has width `3.140454e-11` and determines

```
beta = 0.2801694990
```

to **ten correctly-rounded decimal places**, in the sense that `floor(L*10^k + 1/2) == floor(U*10^k + 1/2)` for all `k <= 10`. It fails at `k = 11`, and Section 6 explains why eleven places are unreachable **with this witness** for a provable reason rather than a budgetary one — a statement about the released coefficient vector, not about every vector at `m = 64000`.

This is a **computational** contribution. The underlying inequality is classical (see `LITERATURE.md`); this release contributes a large-scale rigorous evaluation of it, and a certificate whose aggregation and digit claim can be re-checked from the raw data. It is not a new theorem in approximation theory, and it is not presented as one. No claim is made about whether comparable or better rigorous enclosures exist elsewhere; see section 7.

Normalization

Exactly which constant the endpoints refer to. This block is compared character-for-character (whitespace collapsed) against the `normalization` field of `RELEASE_CLAIM.json` by `make headline-audit`, so the note and the executable claim cannot drift apart.

```
BEGIN NORMALIZATION
beta = lim_{n->inf} 2n * E_{2n}(|x|; [-1,1]), where E_k(f; [-1,1]) is the error of best uniform
approximation to f on [-1,1] by real polynomials of degree at most k, in the sup norm. The upper
endpoint is 2*mu_m with mu_m = inf_a sup_{t>=0} |R_m(t; a)|, R_m(t) = cos(pi t) H(t), H(t) = F(t)
- a_0 - sum_{k=1}^{m} a_k/(t^2 - c_k^2), c_k = (2k-1)/2, F(t) = (t/2)[psi(t/2+3/4) -
psi(t/2+1/4)]; Varga-Carpenter (1985) eqs. (3.5), (3.8), (3.9), (3.10).
END NORMALIZATION
```

1. Method

Varga and Carpenter (*Constr. Approx.* **1** (1985) 333-348, eqs. 3.5-3.10), attributing the limiting relation to Bernstein, define

```
R_m(t; a) = cos(pi t) * [ F(t) - ( a_0 + sum_{k=1}^{m} a_k / (t^2 - c_k^2) ) ],
c_k       = (2k - 1) / 2,
F(t)      = (t/2) * [ psi(t/2 + 3/4) - psi(t/2 + 1/4) ],
mu_m      = inf over real a of  || R_m(.; a) ||_{L_inf[0, inf)},
```

and establish `beta = 2 * lim_m mu_m` with `mu_m` nonincreasing, hence

```
beta <= 2 * mu_m       for every nonnegative integer m.
```

Because `mu_m` is an **infimum**, no optimality proof is required. For any concrete coefficient vector `a`,

```
beta <= 2 * mu_m <= 2 * || R_m(.; a) ||_inf ,
```

so search and proof separate cleanly: a near-optimal `a` is found numerically (and its optimality is never claimed or needed), and then a single rigorous sup-norm bound is certified.

This release certifies `m = 64000`.

Normalisation. Factor `2n`, even degree `2n`, interval `[-1, 1]`. Every other convention in the literature (`n` vs `2n`, `[0,1]` vs `[-1,1]`) rescales `beta`. The reduction `E_{2n}(|x|; [-1,1]) = E_n(sqrt(t); [0,1])` is an identity, not an approximation.

2. The upper endpoint

`|R_m|` is bounded rigorously on `[0, inf)` by splitting the domain at the poles:

- **Finite part `[0, T_0]`, `T_0 = 64000 > c_m = 63999.5`.** The `m` poles at `t = c_k` cut `[0, T_0]` into exactly `m + 1 = 64001` intervals. Each is certified by adaptive branch-and-bound in Arb (python-flint) ball arithmetic at 160 bits, with every emitted bound an **exact rational**. The zeros of `cos(pi t)` cancel the rational poles, so the apparent singularities are removable and each pole value is finite.
- **Tail `[T_0, inf)`.** Discharged by a monotonicity lemma ("Lemma T"): for `t > c_m` every term of `H'` is positive, so `H` increases to `lambda := 1/2 - a_0`, and a single point check `H(T_0) >= -lambda` gives `|H| <= lambda` on the whole tail.

Both lemmas require `a_k > 0` (checkable exactly from the witness) **and** `F` strictly increasing. The latter is not self-evident — every individual series term eventually decreases and the leading order of `F'` cancels (`F' = Theta(1/t^3)`) — so it is proved here via the integral representation

```
F(t) = integral_0^inf  e^{-v} / ( 2 cosh( v / (2t) ) )  dv ,
```

whose integrand is manifestly strictly increasing in `t` for each fixed `v > 0`, with `F(0+) = 0` and `F(inf) = 1/2`. (Checked against the `psi` form at 40 digits; agreement `<= 8e-40`.)

The run is sharded: 1069 shards, 436,201,931 cells, no interval exhausting its cell cap and no bound hitting the width floor. Aggregation is a strict recombination that proves the shards partition the 64001 intervals **exactly once**, then takes the exact rational maximum.

```
U_finite = 70042374761999999931084805387138536290 / 50000000000000000000000000000000000000
2 * U    = 70042374761999999931084805387138536290 / 25000000000000000000000000000000000000
         = 0.28016949904799999997243392215485541 6516
printed  = 0.280169499047999999 8        (rounded OUTWARD, hence itself proved)
```

3. The lower endpoint

Unchanged from the previous milestone: `beta >= B_m(lambda)` (VC eq. 4.6) at `m = 1000`, giving `0.280169499016595460711186` (decimal **floored**, hence itself proved). Witness from L-BFGS-B on `B_m` with the closed-form `O(m^2)` gradient.

The lower endpoint is cross-checked by two independent implementations — Arb and `mpmath.iv` — which agree to 19 decimals on the same witness. The upper endpoint is **not** so cross-checked; see Section 5.

The lower endpoint did not need to move: it already cleared its side of the tenth-place rounding cell by `6.660e-11`.

4. What the ten places rest on

The digit criterion is **two-sided**. Place `k` is determined iff `floor(L*10^k + 1/2) == floor(U*10^k + 1/2)`; both endpoints must lie in the same rounding cell. For the tenth place the cell is `[0.28016949895, 0.28016949905)`:

endpoint	boundary	headroom
`U < 0.28016949905`	binding side	`2.000e-12`
`L >= 0.28016949895`	satisfied	`6.660e-11`

Both endpoints are load-bearing; neither is redundant. At `k = 11` they diverge (`...902` vs `...905`).

5. What is proved, and what is trusted

Stated plainly, because the distinction matters more than the digit count.

Proved. The inequality chain; the partition of `[0, T_0]`; the exactness of every per-interval rational bound *given a correct kernel*; the outward rounding of both printed endpoints; the ten-place count from the two endpoints.

Trusted.

1. **A single kernel implementation.** The 1069 shards were produced by eleven parallel workers running **one** kernel (the author's record of the campaign; historical, not established by anything in this archive). That is one implementation executed eleven times — parallelism, not independent evidence. Manifest-hash equality shows only that the shipped kernel bytes match the digest every manifest asserts; it does **not** establish that those bytes were executed, and it says nothing about whether they are correct. A kernel error would be common-mode across all eleven workers. The upper endpoint therefore rests on one trusted interval implementation, in the ordinary sense of rigorous numerics.
2. **The arithmetic stack is only partly pinned.** The manifests record `python_flint == 0.6.0`, which is the **Python wrapper**; the FLINT/Arb **C library** performing every ball operation is not recorded. The shards also span two container images and two `mpmath` versions (the latter is read only for a version string and is not load-bearing).
3. **The transcription of Varga-Carpenter eqs. (3.5)-(3.10).** Read from the author scan and recorded verbatim in the repository, but a paraphrase error there would be invisible to every check in this package.

Partial mitigations, none of which is a proof.

- An **independent implementation** ships in `audit/` (`make level2`). It evaluates `R(t)` from the mathematical definition using `mpmath digamma`, sharing no code with the certifying kernel, and checks `|R(t)|` against the bound of the region **actually containing** `t` — piece-local across all 161 pieces of the split interval 0, on both sides of and NEAR every seam (the seam abscissae themselves are not evaluated), plus the argmax interval, every near-maximal interval, and a stratified spread. **533 points, none beyond the finite domain**, no violation found. Runtime and the screen's own measured discrepancy are printed on every run, so this claim is reproducible rather than reported.

  Read precisely: every point is evaluated by a float64 screen, and the points that are re-evaluated at 40 digits are the calibration subset, the apparent violations, and anything within the tight band. A point understated by the screen beyond that band, and outside the calibration subset, is never re-decided in high precision — so the measured discrepancy is calibration, not a bound over all 533 points. It remains sparse falsification: it can refute a bound, never certify one.
- **The argmax interval was re-derived from scratch, on a different platform.** Interval 9350 carries the maximum that determines the published endpoint. It was recertified with `--shard 9350:9351` on **macOS ARM64**, against producers that ran **Linux x86_64** with a different FLINT build, and reproduced the stored exact rational **bit-for-bit** at the same cell count (6633). Shipped as `validation/argmax_9350_reproduction.json`. This is the interval determining the maximum among the **stored** bounds, so it is the single most informative one to re-run — but every interval matters to validity, and a different interval carrying an understated stored bound could hide a larger true residual. Weigh it as: strong evidence for local reproducibility of interval 9350, moderate evidence against gross FLINT-build sensitivity, and **no** evidence against a common-mode defect in the kernel, which this re-run shares.
- **The kernel is validated end to end at tractable m against an independent published reference** (`make validate-small`). At `m = 5, 10, 20` the complete campaign runs in seconds, and Varga-Carpenter 1985 Table 3.1 publishes `2*mu_m` for exactly those `m` — computed independently, forty years ago, with other software. The test is the *direction* of the inequality: since `mu_m` is an infimum and the witness is not optimal, a certified `2*U_m` must sit just ABOVE the published value; a bound below it is a **regression failure**. It is not by itself proof of a bug: VC's table decimals are reported values without a documented rounding direction or error enclosure, so an arbitrarily tiny shortfall could be theirs rather than ours.

m	2*U_m certified here	2*mu_m (VC 1985)	excess
5	0.28177999318000	0.28177999262427	5.56e-10
10	0.28056827228799	0.28056814808466	1.24e-07
20	0.28026802660998	0.28026791812802	1.08e-07

The excess is the witness's suboptimality plus branch-and-bound slack, and it tracks the known gap between these witnesses and VC's own.

- **600 pole intervals were recomputed twice**, by accident: six shard filenames collide across two worker branches with different file hashes, and the pairs differ **only** in a recorded git commit — their bounds and cell counts are bit-identical.
- The audit levels are **tested against 36 mutation classes, 37 expected outcomes** (the understated-bound mutation is staged once and carries two: Level 1 must accept it, Level 2 must reject it), each rejected by the level responsible for it. The split is itself informative: aggregation (Level 1) catches structural defects — missing, duplicated, malformed, mis-sized, mis-parameterised, hash-inconsistent — but **cannot** catch an understated bound, because it never evaluates `R(t)`. That case is caught by Level 2, and only where Level 2 samples. A checker never shown to fail proves nothing, and a checker's reach must be stated rather than assumed.

6. Why eleven places are out of reach at m = 64000

Not a budget limit. The witness's own tail level `lambda = 1/2 - a_0 = 0.140084749516649675893...` is a rigorous **lower** bound on `|| R_m ||_inf`, because `|R_m| -> lambda` along the integers. Hence:

- the branch-and-bound overshoot on this witness is at most `U_finite - lambda = 7.350e-12`; and
- **no** certification at this `m`, on this witness, at **any** halting target, could report a bound below `2*lambda = 0.2801694990333`.

That floor still determines the tenth place and still fails at the eleventh. Ten places is therefore a property of **this witness** at `m = 64000`, not an artefact of the halting target that happened to be chosen: no amount of further branch-and-bound refinement on this coefficient vector can reach an eleventh place.

This argument is witness-specific, and deliberately says no more. The floor `2*lambda` is computed from *this* witness's own `a_0`. Nothing here rules out a different admissible coefficient vector at the same `m = 64000` whose `lambda` is smaller and which does reach the eleventh place — and nothing here claims the released witness is optimal, because no optimality is proved or needed (`mu_m` is an infimum, see `TRUSTED_INPUTS.md` T1). So: **an eleventh place requires a different witness and/or a larger `m`.**

7. Relation to the published record

Varga and Carpenter (1985), eq. (1.16), give the rigorous enclosure `l_20 <= beta <= 2*mu_100`, reporting `0.280168546002042...` and `0.280173379101718...`; an outward rendering of those reports is `0.2801685460 <= beta <= 0.2801733792`, width `4.83e-6`, determining five correctly-rounded decimal places. The same paper gives a 50-digit Richardson *estimate* (eq. 1.18), which is not a proved bound. Both endpoints here are inside the corresponding published ones, and the width ratio against that outward rendering is `153901.32` (see `LITERATURE.md` — the ratio against the full reported table decimals is `153898.12`; this release quotes the outward one, since only an outward rendering is a valid enclosure).

No priority claim is made, and none is implied. `LITERATURE.md` records the Varga-Carpenter enclosure used as the comparison baseline, and nothing else. This release makes **no statement** about whether later rigorous

enclosures exist, about novelty, or about being best known. The author's admission rule requires a subject-matter expert to be contacted before any such statement may be made, and that step is **open**. Readers aware of intervening work are asked to open an issue so the comparison can be corrected.

That open step does **not** gate this deposit. It gates any claim of novelty, priority, or best-known status — none of which is made here. What is deposited is a scoped computational certificate and its audit trail, and it stands or falls on the checks in `VERIFY.md`. If a priority claim is ever added, the expert-contact step must close first.

8. Contents and verification

See `README.md` for the file map and `VERIFY.md` for exact commands. The verification is layered so that the headline number can be re-derived with nothing but a Python standard library:

level	needs	re-derives
1	CPython only	the partition, and the exact rational maximum, from the raw shards
2	+ `mpmath`	independent evaluation of `R(t)` against the certified bounds
3	+ `python-flint`	the full strict recombination, including the Lemma T tail

Level 1 alone reproduces `2U` exactly from the 1069 shard files.

Trusted inputs, stated explicitly

Three things are believed rather than checked here, and a reader should weigh them before quoting anything above.

1. The Varga-Carpenter transcription. The reduction `beta <= 2 mu_m` and the finite-`m` formula for `R_m` are taken from Varga and Carpenter (1985), eqs. (3.5)-(3.10). No gate in this release establishes that the implemented formula and its index conventions are exactly those of the source. Every backend here shares that transcription, so agreement between them cannot detect a shared factor, sign or index error. An independent re-derivation from the paper is the check that would close this, and it has not been done.

2. The monotonicity lemmas — now PROVED, no longer assumed. The branch-and-bound relies on `H` being strictly increasing between consecutive poles (so a pole-free cell attains its extremes at the endpoints), on a corresponding tail statement, and on a pole-adjacent correction bound. These are proved in **PROOFS.md**, from Gauss's integral for the digamma difference. The decisive step is that

```
F(t) = integral_0^inf e^{-v} / (2 cosh(v/(2t))) dv,
```

whose integrand is pointwise strictly increasing in `t` — which settles the sign of `F'` that a term-by-term argument cannot, since the leading order of `F'` cancels. `make lemmas` corroborates every stated conclusion numerically on the shipped witness. This item used to read "stated here as an assumption"; it is no longer an assumption.

3. The lower-bound formula. `B_m(lam)` and its admissibility conditions are likewise transcribed. The two backends reduce library and implementation risk; they do not reduce theorem-transcription risk at all.

What the cross-checks DO establish is narrower than it may appear: that two independent arithmetic libraries, given the same formula and the same witness, agree to 19 decimals.