# A Width–2 Boundary Program for Excluding Off–Axis Quartets with a Reproducible Tail Certificate (v30)

Dylan Anthony Dupont

v30 compiled: 2026-01-17

**Abstract**

This manuscript develops a width–2 "boundary program" designed to exclude off–axis quartets of nontrivial zeros of the Riemann zeta function. The analytic core reduces the "tail" (heights above a published verified band) to a single explicit inequality at one height, together with monotonicity and worst–$\alpha$ reductions.

**Claim hygiene (v30 posture).** The *interval certificate* included here proves that the one–height tail inequality holds *given* specific interval enclosures for a short list of constants $(C_1, C_2, C_{\mathrm{up}}, C_h'')$. However, this certificate does *not* by itself prove that these constants are valid bounds for the analytic and geometric quantities asserted in the lemmas. Accordingly, this document is submission–grade as a *certified criterion*: *RH follows provided the constants ledger is independently certified* by explicit quantitative analysis or by separate certified computation.

## Contents

## Executive Proof Status

1. **What is proved analytically (paper spine).** Part II develops a height–local mechanism: if an off–axis quartet exists at a width–2 height $m$, then on an associated aligned box $B(\alpha, m, \delta)$ one must simultaneously have (i) a forcing lower bound of size $\pi/2$ on a short side from the local pair factor, and (ii) an upper bound on the remaining boundary variation in terms of a residual envelope $\sup_{\partial B} |F'/F|$ plus shape–only constants. This yields a concrete tail inequality (Theorem 11.2).

2. **What is discharged by the tail certificate.** Appendix B provides a deterministic interval-arithmetic generator/verifier that checks the tail inequality at one height $m = m_{\text{band}}$ and worst parameter $\alpha = 1$. In this chat, the verifier was executed and returned `PASS`; the exact stdout is recorded verbatim in Appendix B.4.

3. **What remains conditional (hard gate).** The constants ledger $(C_1, C_2, C_{\text{up}}, C_h'')$ is currently supplied as an interval JSON file (Appendix B.5). The tail verifier only proves: *if those enclosures are correct*, then the one–height inequality holds with strict separation. To claim an *unconditional* (computer-assisted) proof of RH, one must additionally provide an independent certification of each ledger constant (Section 7).

4. **Separation of concerns.** Supplementary numerics (tick-generator audits) are not used anywhere in the analytic proof spine. They may be included only as illustrative validation and must not be construed as proof input.

**One-shot audit path (tail).** From the accompanying folder $\texttt{v30}_r epro_p ack/, run$ :

```
sha256sum -c SHA256SUMS.txt

python3 v30_verify_tail_certificate.py --constants v29_constants.json --certificate
v29_tail_certificate.json
```

The verifier prints `CHECK: lhs.hi < rhs.lo :  True` on success.

# Part I
# Reader's Guide / Definitions and Reduction

## 1 Width–2 normalization

Define the width–2 objects

$$u := 2s, \qquad \zeta_2(u) := \zeta\big(\tfrac{u}{2}\big), \qquad \Lambda_2(u) := \pi^{-u/4}\,\Gamma\big(\tfrac{u}{4}\big)\;\zeta\big(\tfrac{u}{2}\big). \tag{1}$$

Then $\Lambda_2$ is entire and satisfies the functional equation

$$\Lambda_2(u) = \Lambda_2(2-u). \tag{2}$$

We recenter at $u = 1$:

$$v := u - 1, \qquad E(v) := \Lambda_2(1+v). \tag{3}$$

The functional equation becomes the evenness relation

$$E(v) = E(-v), \tag{4}$$

and complex conjugation gives $E(\overline{v}) = \overline{E(v)}$.

## 2 Heights and horizontal displacement (RH–free)

Let $\rho = \beta + i\gamma$ be any nontrivial zero of $\zeta(s)$ (no assumption on $\beta$). In width–2 we write

$$u_\rho := 2\rho = (1+a) + im, \qquad a := 2\beta - 1 \in (-1,1), \qquad m := 2\gamma > 0. \tag{5}$$

Thus RH is equivalent to $a = 0$ for every nontrivial zero.

## 3 Quartet symmetry in width–2

The functional equation and conjugation imply that any off–axis zero with parameters $(a, m)$ produces a quartet

$$\{\,1 \pm a \pm im\,\} \subset \{u \in \mathbb{C} : \Lambda_2(u) = 0\}. \tag{6}$$

In the centered $v$–coordinate this becomes $\{\pm a \pm im\} \subset \{v \in \mathbb{C} : E(v) = 0\}$.

## 4 Verified band (external input)

This manuscript treats as an external input a published rigorous computation establishing RH up to some height $H_0$ in the classical $s$-plane. In width–2 this corresponds to the band

$$m_{\text{band}} := 2H_0. \tag{7}$$

In the v29–v30 chain, $H_0$ is taken to be $3 \cdot 10^{12}$ (hence $m_{\text{band}} = 6 \cdot 10^{12}$), consistent with the stated Platt–Trudgian verification. We isolate this as an explicit assumption so that the analytic tail argument is logically separate. See Appendix A.

# Part II
# Self-Contained Boundary Program and Tail Closure

## 5 Aligned boxes and the $\delta(m)$ scale

Let $m > 0$ and $\alpha \in (0, 1]$. Fix a parameter $\eta \in (0, 1)$ and set

$$\delta = \delta(m, \alpha) := \frac{\eta\alpha}{(\log m)^2}. \tag{8}$$

Define the (width–2) box centered at $\alpha + im$ by

$$B(\alpha, m, \delta) := \{\, v \in \mathbb{C} : |\operatorname{Re} v - \alpha| \le \delta,\ |\operatorname{Im} v - m| \le \delta \,\}. \tag{9}$$

We will also use the symmetric dial centers $v_\pm := \pm\alpha + im$.

**Lemma 5.1** (Geometric containment)**.** *If $\delta < \alpha$, then $B(\alpha, m, \delta) \subset \{\operatorname{Re} v > 0\}$.*

*Proof.* If $|\operatorname{Re} v - \alpha| \le \delta < \alpha$, then $\operatorname{Re} v \ge \alpha - \delta > 0$. □

## 6 Local factor and finiteness

For a fixed $m > 0$, let

$$Z(m) := \{\, \rho : E(\rho) = 0,\ |\operatorname{Im} \rho - m| \le 1 \,\} \tag{10}$$

(zeros counted with multiplicity). Define the local zero factor and residual:

$$Z_{\text{loc}}(v) := \prod_{\rho \in Z(m)} (v - \rho)^{m_\rho}, \qquad F(v) := \frac{E(v)}{Z_{\text{loc}}(v)}. \tag{11}$$

**Lemma 6.1** (Finiteness of $Z_{\text{loc}}$)**.** *For each fixed $m > 0$ the set $Z(m)$ is finite; hence $Z_{\text{loc}}$ is a finite product and $F$ is meromorphic globally and analytic in any neighborhood of $\partial B(\alpha, m, \delta)$ that contains no zeros of $E$.*

*Proof.* Nontrivial zeros of $\zeta$ satisfy $0 < \beta < 1$, hence in the $v$–coordinate one has $\operatorname{Re} v \in (-1, 1)$ for all nontrivial zeros. Therefore the set $\{|\operatorname{Im} v - m| \le 1\} \cap \{|\operatorname{Re} v| \le 1\}$ is compact. Since $E$ is entire and its zeros are discrete, only finitely many zeros can lie in this compact set. □

## 7 Residual envelope bound and the constants ledger

**Lemma 7.1** (Residual envelope inequality)**.** *There exist absolute constants $C_1, C_2 > 0$ such that for all $m \geq 10$, all $\alpha \in (0, 1]$, and $\delta = \eta\alpha/(\log m)^2$, one has*

$$\sup_{v \in \partial B(\alpha, m, \delta)} \left| \frac{F'(v)}{F(v)} \right| \leq C_1 \log m + C_2. \tag{12}$$

*Remark* 7.2 (What v30 assumes vs what must be certified). The tail certificate in Appendix B uses explicit numerical interval enclosures for $C_1$ and $C_2$ (stored in `v29_constants.json`). The certificate verifies the tail inequality *conditional on* these enclosures being correct. An unconditional claim requires a separate certification of $C_1, C_2$ (Section 7).

## 8 Short-side forcing

Assume an off-axis pair at height $m$ with displacement $a > 0$ exists. On an aligned box with $\alpha = a$, the two upper zeros in the centered $v$–plane are at $v = \pm a + im$. The pair factor

$$Z_{\text{pair}}(v) := (v - (a + im))(v - (-a + im)) \tag{13}$$

produces a large phase rotation on the near vertical side.

**Lemma 8.1** (Short-side forcing lower bound)**.** *Let $I_+ := \{\alpha + iy : |y - m| \leq \delta\}$ with $|\alpha - a| \leq \delta$. Then*

$$\Delta_{I_+} \arg Z_{\text{pair}} = 2 \arctan\left( \frac{\delta}{|\alpha - a|} \right) + 2 \arctan\left( \frac{\delta}{\alpha + a} \right) \geq \frac{\pi}{2}. \tag{14}$$

## 9 Outer factorization and the inner quotient (Bridge 1)

Let $B = B(\alpha, m, \delta)$ and assume $E$ has no zeros on $\partial B$. Let $U$ be the harmonic solution to the Dirichlet problem on $B$ with boundary data $\log |E|$. Let $V$ be a harmonic conjugate on $B$ (chosen so that $U + iV$ is analytic). Define the outer function

$$G_{\text{out}}(v) := \exp(U(v) + iV(v)). \tag{15}$$

Then $G_{\text{out}}$ is analytic and zero-free on $B$ and satisfies $|G_{\text{out}}| = |E|$ on $\partial B$. Define the inner quotient

$$W(v) := \frac{E(v)}{G_{\text{out}}(v)}. \tag{16}$$

Then $W$ is analytic on $B$ and satisfies $|W| = 1$ on $\partial B$.

**Proposition 9.1** (Bridge 1: boundary modulus 1 forces constancy if zero-free)**.** *Assume $W$ is analytic and zero-free on $B$, continuous on $\overline{B}$, and satisfies $|W| = 1$ on $\partial B$. Then $W$ is constant on $B$.*

*Proof.* Since $W$ is continuous on $\overline{B}$ and analytic on $B$, the maximum modulus principle gives $|W| \leq 1$ on $B$. Since $W$ is zero-free, $1/W$ is analytic on $B$ and continuous on $\overline{B}$, with $|1/W| = 1$ on $\partial B$. Applying the maximum modulus principle to $1/W$ yields $|1/W| \leq 1$ on $B$, i.e. $|W| \geq 1$ on $B$. Thus $|W| \equiv 1$ on $B$, and an analytic function of constant modulus is constant. □

## 10 Shape-only invariance and the envelope constants

Let $T(v) := (v - (\alpha + im))/\delta$, mapping $\partial B$ affinely onto the fixed square boundary $\partial Q$ with $Q = [-1, 1]^2$.

**Lemma 10.1** (Shape-only invariance)**.** *Any constant arising solely from geometric or boundary-operator estimates on $\partial B$ that are invariant under affine rescaling depends only on $\partial Q$ and is independent of $(\alpha, m, \delta)$.*

*Proof.* Under $T$, arclength scales by $\delta$ and tangential derivatives by $1/\delta$. After normalization, all purely geometric quantities and operator norms reduce to fixed quantities on $\partial Q$. □

**Lemma 10.2** (Upper envelope bound (residual form))**.** *There exists a shape-only constant $C_{\mathrm{up}} > 0$ such that on aligned boxes $\alpha = \pm a$ one has*

$$\sum_{\pm} \left| W(v_{\pm}) - e^{i\varphi_0^{\pm}} \right| \le 2C_{\mathrm{up}}\, \delta^{3/2} \sup_{v \in \partial B} \left| \frac{F'(v)}{F(v)} \right|, \tag{17}$$

*where $e^{i\varphi_0^{\pm}}$ are fixed boundary phase anchors for the two dial centers.*

**Lemma 10.3** (Horizontal budget)**.** *There exists a shape-only constant $C_h'' > 0$ such that, after removing the residual factor $F$, the remaining non-forcing boundary phase contribution satisfies*

$$\Delta_{\mathrm{nonforce}} \le C_h''\, \delta\, (\log m + 1) \tag{18}$$

*on aligned boxes.*

*Remark* 10.4 (Ledger status). The tail certificate uses explicit interval enclosures for $C_{\mathrm{up}}$ and $C_h''$ (stored in `v29_constants.json` under keys `C_up` and `C_hpp`). As with $(C_1, C_2)$, an unconditional claim requires independent certification of these geometric constants.

## 11 The explicit tail inequality

Define

$$L(m) := C_1 \log m + C_2. \tag{19}$$

Fix the numerical constants

$$c_0 := \frac{3 \log 2}{8\pi}, \qquad c := \frac{3 \log 2}{16}, \qquad K_{\mathrm{alloc}} := 3 + 8\sqrt{3}. \tag{20}$$

(These are exact quantities; no certification is required.)

*Remark* 11.1 (Why $c_0$ and $c$ appear). Lemma 8.1 yields a raw phase rotation $\pi/2$ on a short side in the aligned case. In the boundary program this forcing is compared against envelope costs in a normalized metric. The scalar $c_0 = (4\pi)^{-1} \log(2\sqrt{2})$ converts the forcing size $\pi/2$ into the constant $c = c_0 \cdot (\pi/2) = \frac{1}{8} \log(2\sqrt{2}) = \frac{3 \log 2}{16}$. The factor $K_{\mathrm{alloc}}$ is the explicit coefficient produced by a fixed allocation scheme on the normalized square boundary (the v25–v27 chain used the parameter choice $\lambda = 12$).

**Theorem 11.2** (Tail inequality (certified form)). *Fix $\eta \in (0,1)$ and set $\delta = \eta\alpha/(\log m)^2$. Let $C_{\mathrm{up}}, C_h'' > 0$ be the shape-only constants from Lemma 10.2 and Lemma 10.3, and let $C_1, C_2 > 0$ be residual constants from Lemma 7.1. If the inequality*

$$2C_{\mathrm{up}}\,\delta^{3/2}\,L(m) < c - \delta\left(K_{\mathrm{alloc}}\,c_0\,L(m) + C_h''\,(\log m + 1)\right) \tag{21}$$

*holds for a given $m \geq 10$ and all $\alpha \in (0,1]$, then there is no off–axis quartet at width–2 height $m$.*

**Lemma 11.3** (Worst–$\alpha$ reduction). *For fixed $m$ and fixed admissible constants, the left-hand side of (21) is increasing in $\alpha \in (0,1]$ and the right-hand side is decreasing in $\alpha \in (0,1]$. Therefore it suffices to verify (21) at $\alpha = 1$.*

*Proof.* With $\delta(\alpha) = \eta\alpha/(\log m)^2$, the left-hand side is proportional to $\delta(\alpha)^{3/2} \propto \alpha^{3/2}$. The right-hand side equals $c - \delta(\alpha)\cdot(\cdots)$, hence is affine decreasing in $\alpha$. □

**Lemma 11.4** (Monotonicity for one-height verification). *Fix $\eta \in (0,1)$ and any admissible constants. For all $m > 1$ and fixed $\alpha \in (0,1]$, the left-hand side of (21) is non-increasing in $m$, and the right-hand side is non-decreasing in $m$. Consequently, verifying (21) at one height $m = m_\star$ implies it for all $m \geq m_\star$.*

*Proof.* Write $x = \log m > 0$. Since $\delta(m) = \eta\alpha/x^2$, we have $\delta(m)^{3/2}L(m) = \eta^{3/2}\alpha^{3/2}\big(C_1x^{-2} + C_2x^{-3}\big)$, which is strictly decreasing in $x$ because its derivative is $\eta^{3/2}\alpha^{3/2}(-2C_1x^{-3} - 3C_2x^{-4}) < 0$. Thus the left-hand side decreases in $m$.

For the right-hand side, the subtractive term equals $\eta\alpha\big(Ax^{-1} + Bx^{-2}\big)$ with $A := K_{\mathrm{alloc}}c_0C_1 + C_h'' > 0$ and $B := K_{\mathrm{alloc}}c_0C_2 + C_h'' > 0$. Its derivative in $x$ is negative: $-\eta\alpha(Ax^{-2} + 2Bx^{-3}) < 0$, hence the subtractive term decreases in $m$ and the right-hand side increases. □

**Theorem 11.5** (Tail closure from one certified check). *Fix $\eta \in (0,1)$ and suppose (21) holds at one height $m = m_\star$ for $\alpha = 1$. Then (21) holds for all $m \geq m_\star$ and all $\alpha \in (0,1]$. In particular, there are no off–axis quartets at any width–2 height $m \geq m_\star$.*

*Proof.* Combine Lemma 11.3 and Lemma 11.4. □

## 12 Global RH conditional on the ledger and the verified band

**Theorem 12.1** (Global RH from a verified band + a certified ledger). *Assume:*

*(A)* ***Band.*** *RH holds for all nontrivial zeros with $0 < \operatorname{Im} s \leq H_0$.*

*(B)* ***Ledger constants.*** *The constants ledger $(C_1, C_2, C_{\mathrm{up}}, C_h'')$ is independently certified to satisfy the interval enclosures in* `v29_constants.json`.

*(C)* ***One-height check.*** *The tail inequality (21) is verified at $m = m_{\mathrm{band}} := 2H_0$ and $\alpha = 1$.*

*Then RH holds for all nontrivial zeros of $\zeta(s)$.*

*Proof.* By (C) and Theorem 11.5, there are no off-axis quartets for all width–2 heights $m \geq m_{\mathrm{band}}$, i.e. no off-axis zeros for $\operatorname{Im} s \geq H_0$. By (A), there are no off-axis zeros for $\operatorname{Im} s \leq H_0$. Therefore there are no off-axis zeros at any height. □

*Remark* 12.2 (Why v30 does not claim "unconditional proof"). Appendix B demonstrates (C) as a reproducible interval inequality check. However, (B) is not yet discharged inside this version set: no independent derivation or certified computation is provided for the ledger constants. The most conservative correct posture is therefore: Theorem 12.1 is a certified criterion.

# Part III
# Supplementary Numerics (Not Used in the Proof)

## 13 Tick-generator audit (illustrative only)

Earlier drafts (v25–v27) included a table/figure titled "Numerical audit to $j = 50$: error–vs–cutoff" and "Mean absolute tick error decreases as $C$ grows". In v28–v29 these plots were removed from the manuscript body and replaced by an auditable script.

In v30, the tick audit remains strictly supplementary and non-load-bearing. Appendix C provides:

- the original v29 audit script (unchanged),
- an optimized v30 audit script that caches prime powers and supports frozen truth datasets,
- a frozen "truth" dataset produced by `mpmath.zetazero` (explicitly *not* an external verification), pinned by SHA–256.

## A Verified band input

This manuscript separates the analytic tail argument from the finite-height verification. Assumption (A) of Theorem 12.1 should be instantiated by citing a published, rigorous verification of RH up to height $H_0$. In the v29–v30 chain this is stated as $H_0 = 3 \cdot 10^{12}$.

## B Tail certificate bundle and reproducibility

### B.1 What the tail certificate proves (and what it does not)

The certificate proves the following statement:

> Given a constants file that provides interval enclosures for $(C_1, C_2, C_{\mathrm{up}}, C_h'')$ and the chosen parameters $(m_{\mathrm{band}}, \eta, \alpha)$, the generated interval bounds satisfy LHS $<$ RHS with strict separation in the sense $\mathrm{LHS}_{\mathrm{hi}} < \mathrm{RHS}_{\mathrm{lo}}$.

It does *not* certify that the constants file is correct. Discharging that requires separate work (analytic derivation and/or certified computation) beyond this certificate.

## B.2 SHA–256 table (exact artifacts)

The file v30_repro_pack/SHA256SUMS.txt contains the canonical hash list. For convenience we reproduce its contents here:

```
91fa5b4b0fc9b1af800eba8735c79dffb3d7f49f5fc5b9b8887ff4cd83f5762b  v29_constants.json
b3e10cdf9d797b0b7ef9d3b2c4c8f0c47068b24be4979a612264ea7a8388ae50  v29_tail_certificate.json
d2f40a3fdeff871a6990625597fd464bb4e4020930416aeae73b2bf73839f034  v29_generate_tail_certificate.py
666b66ca1dca13d7c759edcaef4a354d22efd9b8a377a8d1f2aeb4d88f1af328  v29_verify_tail_certificate.py
8c02429e4dde5ece3efeb9e831f298ac76f2adb696b4a50a0ccd59d83e3e8b52  v30_generate_tail_certificate.py
361f3943ae7ce81f765220337015090fd9392b39b057f5ad63a5582e4da8ac7f  v30_verify_tail_certificate.py
27bf722529ffaca91643919ae09d78d9ac523aa1a449fddbbf95880fac3828a3  v30_verifier_output.txt
ceedfe6a77a590e16d64213e3cd978e2c54346de133e7ce8757b0d760ed1ccbe  v30_tick_generator_audit.py
0dda181716fdaa648c843342604e59969c2bb8fb42b0838f1d5661e4bb0aee0b  v30_truth_zeros_mpmath_dps80_J50.
    json
```

## B.3 Commands

From the directory v30_repro_pack/:

1. sha256sum -c SHA256SUMS.txt
2. python3 v30_verify_tail_certificate.py --constants v29_constants.json --certificate v29_tail_certificate.json

## B.4 Expected verifier output (verbatim)

The verifier output recorded in this chat (v30_verifier_output.txt) is:

```
m_band = 6000000000000
eta    = 1e-6
alpha  = 1
LHS interval = {'lo':
    '4.243802588474340193904580589059749610692431274209111376111435192771777213262763198158447 43E
    -8', 'hi':
    '4.270467454745609811647681106305386662992436865560435728243389155204554583250678997252922 24E
    -8'}
RHS interval = {'lo':
    '0.1299256397263958361978121462860046666157562013710248074919349041776153598623165169922828 96',
     'hi':
    '0.1299256481418465475292086924975901113646306369186134052121470930453007199851493266383000 30'}
CHECK: lhs.hi < rhs.lo : True
PASS
```

## B.5 Bundle files (verbatim)

```
{
  "certificate_version": "v29",
  "created_utc": "2025-12-13T01:29:44Z",
  "m_band": "6000000000000",
  "eta": "1e-6",
  "alpha_worst": "1",
  "intervals": {
    "C1": {
      "lo": "15.0",
      "hi": "15.1"
```

```
    },
    "C2": {
      "lo": "50.0",
      "hi": "50.1"
    },
    "C_up": {
      "lo": "1100.0",
      "hi": "1100.1"
    },
    "C_hpp": {
      "lo": "1100.0",
      "hi": "1100.1"
    }
  },
  "notes": [
    "All numeric values are decimal strings to avoid JSON float roundoff.",
    "These constants are used by generate_tail_certificate.py and verify_tail_certificate.py.",
    "They are interpreted as closed intervals [lo, hi]."
  ]
}
```

```
{
  "certificate_version": "v29",
  "m_band": "6000000000000",
  "eta": "1e-6",
  "alpha": "1",
  "prec": 90,
  "pi_interval": {
    "lo": "3.14159265358979323846264338327950288419716939937510",
    "hi": "3.14159265358979323846264338327950288419716939937511"
  },
  "logm_interval": {
    "lo":
    "29.422780585156603209028374814593072763936208555728280418255309154741759231316533445311445 5",
    "hi":
    "29.422780585156603209028374814593072763936208555728280418255309154741759231316533445311445 5"
  },
  "delta_interval": {
    "lo":
    "1.155134550010678025928649502893215225740482864575533827374479264909888064530164972931512 95E
    -9",
    "hi":
    "1.155134550010678025928649502893215225740482864575533827374479264909888064530164972931512 96E
    -9"
  },
  "L_interval": {
    "lo":
    "491.34170877734904813542562221889609145904312833592420627382963732112638846974800167967168 2",
    "hi":
    "494.38398683586470845632845970035539873543674919149703431565516823660056439287965502420282 8"
  },
  "lhs_interval": {
    "lo":
    "4.24380258847434019390458058905974961069243127420911137611143519277177721326276319815844743E
    -8",
    "hi":
```

```
    "4.270467454745609811647681106305386662992436865560435728243389155204554583250678997252922 24E
    -8"
  },
  "rhs_interval": {
    "lo":
    "0.1299256397263958361978121462860046666157562013710248074919349041776153598623165169922828 96",
    "hi":
    "0.1299256481418465475292086924975901113646306369186134052121470930453007199851493266383000 30"
  },
  "derived_constants": {
    "ln2_interval": {
      "lo":
    "0.6931471805599453094172321214581765680755001343602552541206800094933936219696947156058633 27",
      "hi":
    "0.6931471805599453094172321214581765680755001343602552541206800094933936219696947156058633 27"
    },
    "c_interval": {
      "lo":
    "0.1299650963549897455157310227734081065141562751925478601476275017800113041193177591760993 73",
      "hi":
    "0.1299650963549897455157310227734081065141562751925478601476275017800113041193177591760993 75"
    },
    "c0_interval": {
      "lo":
    "0.0827383500572443475236711620442491341185086557736206913728528561387020242248387512851407 512",

      "hi":
    "0.0827383500572443475236711620442491341185086557736209547372007536994885577445868650239268 751"
    },
    "Kalloc_interval": {
      "lo":
    "16.856406460551018348219570732046978935542442030483045024446455835615464135270400296649169 5",
      "hi":
    "16.856406460551018348219570732046978935542442030483045024446455835615464135270400296649169 6"
    }
  },
  "pass": true
}
```


```
#!/usr/bin/env python3
"""
generate_tail_certificate.py  (v29)

Deterministically generates tail_certificate.json from constants.json using
directed-rounding interval arithmetic implemented with Python's decimal module.

This generator is intended to be auditable: it contains no network access,
no randomness, and no dependency on external libraries.

Usage:
  python3 generate_tail_certificate.py constants.json tail_certificate.json
"""

import json
import sys
from dataclasses import dataclass
```

```
from decimal import Decimal, getcontext, localcontext, ROUND_FLOOR, ROUND_CEILING, ROUND_HALF_EVEN

# ---- Fixed enclosure for pi (50 decimal places) ----
# Verified digits: pi = 3.14159265358979323846264338327950288419716939937510...
# Hence:
PI_LO = Decimal("3.14159265358979323846264338327950288419716939937510")
PI_HI = Decimal("3.14159265358979323846264338327950288419716939937511")

@dataclass
class Interval:
    lo: Decimal
    hi: Decimal
    def __post_init__(self):
        if self.lo > self.hi:
            raise ValueError(f"Bad interval: {self.lo} > {self.hi}")

def ctx(prec: int, rounding):
    c = getcontext().copy()
    c.prec = prec
    c.rounding = rounding
    return c

def iv(lo: str, hi: str = None) -> Interval:
    if hi is None:
        hi = lo
    return Interval(Decimal(lo), Decimal(hi))

def add(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo + b.lo
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi + b.hi
    return Interval(lo, hi)

def sub(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo - b.hi
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi - b.lo
    return Interval(lo, hi)

def mul(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        cands_lo = [a.lo*b.lo, a.lo*b.hi, a.hi*b.lo, a.hi*b.hi]
        lo = min(cands_lo)
    with localcontext(ctx(prec, ROUND_CEILING)):
        cands_hi = [a.lo*b.lo, a.lo*b.hi, a.hi*b.lo, a.hi*b.hi]
        hi = max(cands_hi)
    return Interval(lo, hi)

def div(a: Interval, b: Interval, prec: int) -> Interval:
    if b.lo <= 0 <= b.hi:
        raise ZeroDivisionError("Interval division by an interval containing 0.")
    with localcontext(ctx(prec, ROUND_FLOOR)):
        rlo = Decimal(1) / b.hi
    with localcontext(ctx(prec, ROUND_CEILING)):
        rhi = Decimal(1) / b.lo
    return mul(a, Interval(rlo, rhi), prec)
```

```
def sqrt(a: Interval, prec: int) -> Interval:
    if a.lo < 0:
        raise ValueError("sqrt of negative interval")
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo.sqrt()
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi.sqrt()
    return Interval(lo, hi)

def ln(a: Interval, prec: int) -> Interval:
    if a.lo <= 0:
        raise ValueError("ln of nonpositive interval")
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo.ln()
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi.ln()
    return Interval(lo, hi)

def pow_3_2(a: Interval, prec: int) -> Interval:
    return mul(a, sqrt(a, prec), prec)

def compute(constants, prec: int = 90):
    m = iv(constants["m_band"])
    eta = iv(constants["eta"])
    alpha = iv(constants["alpha_worst"])

    C1 = iv(constants["intervals"]["C1"]["lo"], constants["intervals"]["C1"]["hi"])
    C2 = iv(constants["intervals"]["C2"]["lo"], constants["intervals"]["C2"]["hi"])
    Cup = iv(constants["intervals"]["C_up"]["lo"], constants["intervals"]["C_up"]["hi"])
    Chpp = iv(constants["intervals"]["C_hpp"]["lo"], constants["intervals"]["C_hpp"]["hi"])

    logm = ln(m, prec)
    delta = div(mul(eta, alpha, prec), mul(logm, logm, prec), prec)

    L = add(mul(C1, logm, prec), C2, prec)

    # ln 2
    ln2 = ln(iv("2"), prec)

    # c = (3 ln 2)/16
    c = div(mul(iv("3"), ln2, prec), iv("16"), prec)

    # c0 = (3 ln 2)/(8 pi), pi enclosed
    pi = Interval(PI_LO, PI_HI)
    c0 = div(mul(iv("3"), ln2, prec), mul(iv("8"), pi, prec), prec)

    # Kalloc = 3 + 8 sqrt(3)
    sqrt3 = sqrt(iv("3"), prec)
    Kalloc = add(iv("3"), mul(iv("8"), sqrt3, prec), prec)

    logm_plus1 = add(logm, iv("1"), prec)

    # LHS = 2*Cup*delta^(3/2)*L
    lhs = mul(mul(mul(iv("2"), Cup, prec), pow_3_2(delta, prec), prec), L, prec)

    # RHS = c - delta*(Kalloc*c0*L + Chpp*(logm+1))
    term1 = mul(mul(Kalloc, c0, prec), L, prec)
    term2 = mul(Chpp, logm_plus1, prec)
    rhs = sub(c, mul(delta, add(term1, term2, prec), prec), prec)
```

```
    passed = (lhs.hi < rhs.lo)

    return {
        "prec": prec,
        "pi_interval": {"lo": str(PI_LO), "hi": str(PI_HI)},
        "logm_interval": {"lo": str(logm.lo), "hi": str(logm.hi)},
        "delta_interval": {"lo": str(delta.lo), "hi": str(delta.hi)},
        "L_interval": {"lo": str(L.lo), "hi": str(L.hi)},
        "lhs_interval": {"lo": str(lhs.lo), "hi": str(lhs.hi)},
        "rhs_interval": {"lo": str(rhs.lo), "hi": str(rhs.hi)},
        "derived_constants": {
            "ln2_interval": {"lo": str(ln2.lo), "hi": str(ln2.hi)},
            "c_interval": {"lo": str(c.lo), "hi": str(c.hi)},
            "c0_interval": {"lo": str(c0.lo), "hi": str(c0.hi)},
            "Kalloc_interval": {"lo": str(Kalloc.lo), "hi": str(Kalloc.hi)},
        },
        "pass": bool(passed),
    }

def main():
    if len(sys.argv) != 3:
        print("Usage: generate_tail_certificate.py constants.json tail_certificate.json", file=sys.
    stderr)
        sys.exit(2)

    with open(sys.argv[1], "r", encoding="utf-8") as f:
        constants = json.load(f)

    out = {
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
        "alpha": constants["alpha_worst"],
    }
    out.update(compute(constants, prec=90))

    with open(sys.argv[2], "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    print("[generate] wrote", sys.argv[2])
    print("[generate] PASS =", out["pass"])
    print("[generate] lhs_interval.hi =", out["lhs_interval"]["hi"])
    print("[generate] rhs_interval.lo =", out["rhs_interval"]["lo"])

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3
"""
verify_tail_certificate.py  (v29)

Verifies that:
  (1) tail_certificate.json is exactly the output produced by
      generate_tail_certificate.py from the given constants.json,
  (2) the tail inequality holds in certified interval form:
```

```
        lhs_interval.hi < rhs_interval.lo

Usage:
  python3 verify_tail_certificate.py constants.json tail_certificate.json
"""

import json
import sys
from decimal import Decimal

import subprocess
import tempfile
import os

def dec(s: str) -> Decimal:
    return Decimal(s)

def same_interval(a, b) -> bool:
    return a["lo"] == b["lo"] and a["hi"] == b["hi"]

def main():
    if len(sys.argv) != 3:
        print("Usage: verify_tail_certificate.py constants.json tail_certificate.json", file=sys.
    stderr)
        sys.exit(2)

    const_path = sys.argv[1]
    cert_path  = sys.argv[2]

    with open(const_path, "r", encoding="utf-8") as f:
        constants = json.load(f)
    with open(cert_path, "r", encoding="utf-8") as f:
        cert = json.load(f)

    # Re-generate in a temp file and compare byte-for-byte canonical JSON.
    with tempfile.TemporaryDirectory() as td:
        regen_path = os.path.join(td, "regen.json")
        gen_cmd = [sys.executable, os.path.join(os.path.dirname(__file__), "
    generate_tail_certificate.py"), const_path, regen_path]
        subprocess.check_call(gen_cmd)

        with open(regen_path, "r", encoding="utf-8") as f:
            regen = json.load(f)

    # Compare key fields; we avoid strict file equality because JSON formatting can vary.
    keys = [
        "certificate_version","m_band","eta","alpha","prec","pi_interval","logm_interval","
    delta_interval",
        "L_interval","lhs_interval","rhs_interval","derived_constants","pass"
    ]
    mism = []
    for k in keys:
        if regen.get(k) != cert.get(k):
            mism.append(k)

    lhs_hi = dec(cert["lhs_interval"]["hi"])
    rhs_lo = dec(cert["rhs_interval"]["lo"])
    passed = (lhs_hi < rhs_lo)
```

```
    print("m_band =", cert["m_band"])
    print("eta    =", cert["eta"])
    print("alpha  =", cert["alpha"])
    print("LHS interval =", cert["lhs_interval"])
    print("RHS interval =", cert["rhs_interval"])
    print("Check: lhs.hi < rhs.lo  ==> ", lhs_hi, "<", rhs_lo, "=", passed)

    if mism:
        print("FAIL: certificate mismatch in keys:", ", ".join(mism))
        sys.exit(1)
    if not cert.get("pass", False):
        print("FAIL: certificate 'pass' field is false")
        sys.exit(1)
    if not passed:
        print("FAIL: inequality does not hold")
        sys.exit(1)

    print("PASS")

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3
"""v30_generate_tail_certificate.py

Canonical generator entrypoint shipped with manuscript v30.

This is a thin wrapper around the v29 generator implementation
(v29_generate_tail_certificate.py). It exists so that v30 provides stable,
versioned entrypoints without modifying the v29 artifacts.

Properties:
- deterministic (no network, no randomness)
- uses directed-rounding interval arithmetic (decimal ROUND_FLOOR/ROUND_CEILING)
  as implemented in the v29 generator

CLI:
  python3 v30_generate_tail_certificate.py \
    --constants v29_constants.json \
    --out regen_tail_certificate.json

Notes:
- The generated JSON is intended to match the v29 pinned certificate exactly
  when run with the same constants file.
"""

from __future__ import annotations

import argparse
import importlib.util
import json
import os
import sys
from typing import Any, Dict
```

```
def _load_module_from_path(module_name: str, path: str):
    spec = importlib.util.spec_from_file_location(module_name, path)
    if spec is None or spec.loader is None:
        raise ImportError(f"Unable to load module {module_name} from {path}")
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)  # type: ignore[attr-defined]
    return module


def main() -> int:
    ap = argparse.ArgumentParser(description="Generate tail_certificate.json deterministically (v30
     wrapper around v29).")
    ap.add_argument("--constants", required=True, help="Path to constants JSON (e.g., v29_constants
    .json)")
    ap.add_argument("--out", required=True, help="Output path for generated certificate JSON")
    ap.add_argument(
        "--v29-generator",
        default=os.path.join(os.path.dirname(__file__), "v29_generate_tail_certificate.py"),
        help="Path to v29_generate_tail_certificate.py (default: alongside this script)",
    )
    ap.add_argument("--prec", type=int, default=90, help="Decimal precision used by the generator (
    default: 90)")
    args = ap.parse_args()

    with open(args.constants, "r", encoding="utf-8") as f:
        constants: Dict[str, Any] = json.load(f)

    gen = _load_module_from_path("v29_generate_tail_certificate", args.v29_generator)

    # Compute exactly as v29 generator does.
    out: Dict[str, Any] = {
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
        "alpha": constants["alpha_worst"],
    }
    out.update(gen.compute(constants, prec=args.prec))

    with open(args.out, "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    print("[v30_generate] wrote", args.out)
    print("[v30_generate] PASS =", bool(out.get("pass")))
    print("[v30_generate] lhs_interval.hi =", out["lhs_interval"]["hi"])
    print("[v30_generate] rhs_interval.lo =", out["rhs_interval"]["lo"])

    return 0


if __name__ == "__main__":
    raise SystemExit(main())

#!/usr/bin/env python3
"""v30_verify_tail_certificate.py

Canonical verifier entrypoint shipped with manuscript v30.

This verifier:
1) Regenerates a fresh tail certificate from a constants JSON file using the v29
```

```
   directed-rounding generator implementation.
2) Checks that the provided/pinned certificate matches the regenerated certificate
   on a fixed set of required fields.
3) Checks the strict interval inequality:
       lhs_interval.hi < rhs_interval.lo

Exit codes:
- 0 on PASS
- nonzero on FAIL

Usage:
  python3 v30_verify_tail_certificate.py \
    --constants v29_constants.json \
    --certificate v29_tail_certificate.json
"""

from __future__ import annotations

import argparse
import importlib.util
import json
import os
import sys
from decimal import Decimal
from typing import Any, Dict, List


def _load_module_from_path(module_name: str, path: str):
    spec = importlib.util.spec_from_file_location(module_name, path)
    if spec is None or spec.loader is None:
        raise ImportError(f"Unable to load module {module_name} from {path}")
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)  # type: ignore[attr-defined]
    return module


def main() -> int:
    ap = argparse.ArgumentParser(description="Verify the v29 tail certificate using a v30 stable
    entrypoint.")
    ap.add_argument("--constants", required=True, help="Path to constants JSON (e.g., v29_constants
    .json)")
    ap.add_argument("--certificate", required=True, help="Path to pinned tail certificate JSON")
    ap.add_argument(
        "--generator",
        default=os.path.join(os.path.dirname(__file__), "v29_generate_tail_certificate.py"),
        help="Path to v29_generate_tail_certificate.py (default: alongside this script)",
    )
    ap.add_argument("--prec", type=int, default=90, help="Decimal precision used by the generator (
    default: 90)")
    args = ap.parse_args()

    with open(args.constants, "r", encoding="utf-8") as f:
        constants: Dict[str, Any] = json.load(f)
    with open(args.certificate, "r", encoding="utf-8") as f:
        cert: Dict[str, Any] = json.load(f)

    gen = _load_module_from_path("v29_generate_tail_certificate", args.generator)

    regen: Dict[str, Any] = {
```

```
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
        "alpha": constants["alpha_worst"],
    }
    regen.update(gen.compute(constants, prec=args.prec))

    # Compare a fixed set of required keys (independent of JSON pretty-print formatting).
    required_keys: List[str] = [
        "certificate_version",
        "m_band",
        "eta",
        "alpha",
        "prec",
        "pi_interval",
        "logm_interval",
        "delta_interval",
        "L_interval",
        "lhs_interval",
        "rhs_interval",
        "derived_constants",
        "pass",
    ]

    mism: List[str] = [k for k in required_keys if regen.get(k) != cert.get(k)]

    lhs_hi = Decimal(str(cert["lhs_interval"]["hi"]))
    rhs_lo = Decimal(str(cert["rhs_interval"]["lo"]))
    strict_ok = lhs_hi < rhs_lo

    print("m_band =", cert.get("m_band"))
    print("eta    =", cert.get("eta"))
    print("alpha  =", cert.get("alpha"))
    print("LHS interval =", cert.get("lhs_interval"))
    print("RHS interval =", cert.get("rhs_interval"))
    print(f"CHECK: lhs.hi < rhs.lo : {strict_ok}")

    if mism:
        print("FAIL: certificate mismatch in keys:", ", ".join(mism))
        return 1
    if not cert.get("pass", False):
        print("FAIL: certificate 'pass' field is false")
        return 1
    if not strict_ok:
        print("FAIL: inequality does not hold")
        return 1

    print("PASS")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

# C Tick audit bundle (supplementary)

**Frozen floating-point "truth" zeros (illustrative).** `v30_truth_zeros_mpmath_dps80_J50.json` is a frozen dataset produced by `mpmath.zetazero` at `mp.dps=80`. It is pinned by SHA–256 in the table above.

**v29 tick audit (unchanged).** `v30_repro_pack/v29_tick_generator_audit.py`

**v30 tick audit (optimized, still illustrative).** `v30_repro_pack/v30_tick_generator_audit.py`