
Omega Naught G6 Class-I Cubature Update

All-in-One LaTeX Correction Package

Website copy, release contract, exact mathematics, and Julia verifier

Correct public boundary. Paper VI v1.3 keeps its frozen $1 \leq f \leq 9$ scope. Exact post-publication certificates establish target-degree rules for $10 \leq f \leq 21$. On the fixed grid X_{22} , degree 69 is attained with strictly positive weights and is the exact nonnegative maximum. The all-frequency conjecture is open beginning at $f = 23$.

24 August 2026

One source file emits the complete Julia verifier when compiled

Contents

1	Decision and scope	1
1.1	Authority ladder	1
2	Exact website replacements	1
2.1	Homepage badge and summary	1
2.2	Cubature page lead and status panel	2
2.3	Release page notice	2
2.4	Stale-string replacement map	2
3	Additive release and machine contract	3
4	What was learned	4
5	Start from ordinary weighted averages	4
6	The fixed Class-I grid	5
7	Why exact arithmetic stays possible	6
8	The symmetry reduction	6
9	The degree-69 construction	7
10	Why degree 70 fails	8
11	Why arbitrary weights do not escape the obstruction	9
12	A counting law that predicts the frontier	9
13	What this does and does not advance	10
14	How to verify the theorem	10
14.1	Full Julia exact verifier	12
15	Research directions opened by the result	20
	Reproducibility record	20
	Frozen asset ledger	20
	Implementation acceptance tests	21
	References	21

1 Decision and scope

This document is the implementable correction for the Class-I cubature claims on <https://omega-naught.grok.me>. It does not rewrite the Omega algebra, SRSD reconstruction, the frozen Q, S pair, JPRV folding, canonical records, or the typed G6 object. Only the public cubature status, its supporting mathematics, and additive release metadata change.

Single sentence suitable for the site. Paper VI v1.3 proves the original $f = 1, \dots, 9$ Class-I results; exact post-publication certificates extend target-degree exactness through $f = 22$, with a sharp maximum nonnegative degree of 69 on the fixed grid X_{22} ; the all-frequency statement remains conjectural for $f \geq 23$.

The word *sharp* belongs only to $f = 22$ in this update. For $10 \leq f \leq 21$, the certificates prove positive rules through the stated target degree but do not prove that a higher degree is impossible.

1.1 Authority ladder

Frequency range	Authority	Permitted public claim
$1 \leq f \leq 9$	Paper VI v1.3, frozen	Use the theorem exactly within its original scope.
$10 \leq f \leq 21$	Exact post-publication certificate archive	A strictly positive rule reaches the target degree τ_f . Maximality is not claimed.
$f = 22$	Exact construction plus primal and dual obstruction	A strictly positive degree-69 rule exists, and no nonnegative rule on the same fixed node set reaches degree 70.
$f \geq 23$	No theorem in this package	The target-degree statement is an open conjecture.

AI reading protocol

An AI or implementation must not collapse *proved by Paper VI, certified after publication*, and *conjectured*. These are distinct provenance classes even when they appear in one status sentence.

2 Exact website replacements

The following blocks are complete replacement copy. Preserve the mathematical qualifications or use their plain-text equivalents exactly.

2.1 Homepage badge and summary

Approved replacement copy — homepage

cubature f=1...22 · target exactness certified

Paper VI v1.3 proves the original Class-I frequencies $f=1\dots 9$.
 Exact post-publication certificates extend the target-degree result through $f=22$. At $f=22$, degree 69 is sharp for nonnegative weights on the fixed radial Class-I grid X_{22} . The all-frequency statement

remains conjectural for $f \geq 23$.

Read the theorem, certificates, and verifier

If the visual badge must remain short, use cubature $f=1..22 \cdot \text{target exactness certified}$. Do not shorten it to an unqualified claim that Paper VI proved every frequency through 22.

2.2 Cubature page lead and status panel

Approved replacement copy — cubature page

Exact positive Class-I cubature

For the radial Class-I node set X_f , the target degree is $\tau_f = 3f+2$ for odd f and $\tau_f = 3f+3$ for even f .

What is established

- Paper VI v1.3 proves the published block $f=1..9$.
- Exact post-publication certificates establish strictly positive target-degree rules for $f=10..21$.
- At $f=22$, an exact $Q(\sqrt{5})$ certificate establishes a strictly positive degree-69 rule. A unique degree-70 invariant solution has a negative orbit mass, and an exact dual certificate proves that no nonnegative weights on the fixed node set can reach degree 70.

What remains open

- The all-frequency target-degree statement is conjectural from $f=23$.
- Maximal degree is not claimed here for $f=10..21$.
- No claim is made about formulas whose nodes may move away from X_f .

2.3 Release page notice

Approved replacement copy — release page

Post-publication cubature extension

G6_RELEASE/1.1 and CLASS_I_CUBATURE/1.0 are frozen historical objects. Their bytes, hashes, and claims have not been changed.

CLASS_I_CUBATURE/1.1 is an additive extension. It preserves Paper VI v1.3 as the authority for $f=1..9$, records exact certificate coverage for $f=10..21$, and adds the sharp $f=22$ theorem. A future G6_RELEASE/1.2 may pin these new assets by exact byte length and SHA-256 digest.

2.4 Stale-string replacement map

Remove wherever it appears	Replace with
cubature f=1...9 · exact Proved for $1 \leq f \leq 9$; open for $f \geq 10$	cubature f=1...22 · target exactness certified Paper VI: f=1...9; exact certificate extension: f=10...22; open conjecture: $f \geq 23$
the all-frequency statement is a conjecture from $f \geq 10$ Any unqualified degree 69 is optimal	the all-frequency statement remains a conjecture from $f \geq 23$ degree 69 is the maximum attainable by nonnegative weights on the fixed radial Class-I grid X ₂₂
Any claim that Paper VI proves $f = 10, \dots, 22$	State that these are post-publication exact certificate results.

3 Additive release and machine contract

Release boundary

Do not edit Paper VI v1.3, CLASS_I_CUBATURE/1.0, or G6_RELEASE/1.1. Publish the extension alongside them. Frozen artifacts are evidence, not mutable web copy.

```
contract: CLASS_I_CUBATURE/1.1
status: proposed_additive_extension
extends: CLASS_I_CUBATURE/1.0

immutable_dependencies:
- Paper_VI_v1.3
- CLASS_I_CUBATURE/1.0
- G6_RELEASE/1.1

claims:
  paper_vi:
    frequencies: [1, 9]
    provenance: published_theorem
  exact_certificate_extension:
    frequencies: [10, 21]
    result: positive_target_degree_rule
    maximality: not_claimed
  frequency_22:
    target_degree: 69
    positive_rule: exact
    maximum_nonnegative_degree_on_fixed_X_22: 69
  all_frequency_conjecture:
    open_from_frequency: 23

non_targets:
- movable_node_cubature
- signed_weight_maximality
- changes_to_SRSD_Omega_QS_JPRV_or_G6_serialization
```

When the extension is released, replace `status: proposed_additive_extension` with `status: frozen`, pin the exact served bytes, and never reuse version 1.1 for different content.

4 What was learned

The earlier exact archive established the target degree

$$\tau_f = \begin{cases} 3f + 2, & f \text{ odd}, \\ 3f + 3, & f \text{ even} \end{cases}$$

for every frequency $2 \leq f \leq 21$. The public Omega Naught exposition presented a smaller initial block and left larger frequencies as a frontier. The present calculation moves the exact boundary one step: $\tau_{22} = 69$ is achieved with strict positivity.

It also answers a question the finite block did not answer: is 69 merely a degree we happened to find, or is it the best possible degree on this particular node set? At $f = 22$, it is best possible.

Theorem 1 (Frequency-22 exact boundary). *Let $X_{22} \subset \mathbb{S}^2$ be the complete radially projected Class-I icosahedral grid of frequency 22, and normalize spherical area so that $\sigma(\mathbb{S}^2) = 1$.*

1. *There are strictly positive weights on all nodes of X_{22} , constant on full icosahedral H_3 -orbits, whose cubature rule is exact for every spherical polynomial of degree at most 69.*
2. *No nonnegative weights on X_{22} can give a cubature rule exact for every spherical polynomial of degree at most 70.*

Consequently the maximum algebraic degree attainable by nonnegative weights on the fixed node set X_{22} is exactly 69.

Claim boundary

This is an exact computational theorem relative to the frozen definitions below. It is new relative to the supplied site and certificate archive. A web literature check found higher-degree icosahedrally invariant formulas whose nodes are allowed to move, so this addendum makes no blanket claim of being the first degree-69 spherical formula. The distinctive claim is about the *fixed radial Class-I grid* X_{22} , nonnegative weights, and exact arithmetic. A definitive priority claim would require a dedicated bibliographic review and independent peer review.

5 Start from ordinary weighted averages

A cubature rule on the sphere is a weighted average of function values:

$$Q[p] = \sum_{x \in X} w_x p(x).$$

With normalized area, the true spherical average is

$$\int_{\mathbb{S}^2} p(x) d\sigma(x), \quad \int_{\mathbb{S}^2} 1 d\sigma = 1.$$

The rule is *exact through degree t* if these two expressions agree for every polynomial $p(x, y, z)$ of total degree at most t , restricted to the unit sphere. The constant polynomial $p = 1$ forces

$$\sum_{x \in X} w_x = 1.$$

Nonnegative weights are valuable because the computed average then stays between the smallest and largest sampled values. Strictly positive weights also ensure that no grid node is silently discarded.

Novice checkpoint

Degree 69 does not mean that one special polynomial was integrated. It means *every* polynomial built from x, y, z with total degree at most 69 is integrated exactly on the sphere.

The theorem is not saying general smooth functions have zero error. It says their polynomial content through degree 69 has zero error.

6 The fixed Class-I grid

Let

$$\phi = \frac{1 + \sqrt{5}}{2}$$

and choose one face of a regular icosahedron with vertices

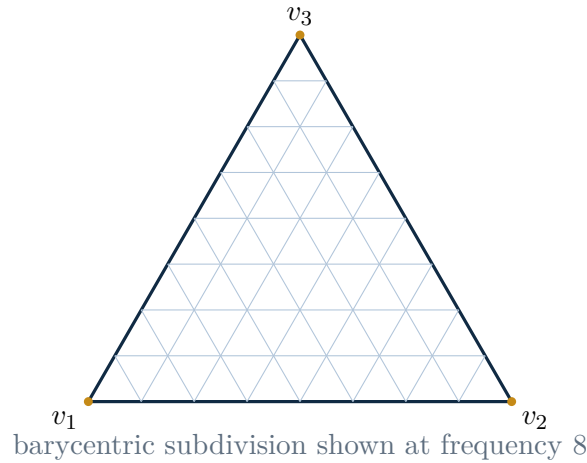
$$v_1 = (0, 1, \phi), \quad v_2 = (-1, \phi, 0), \quad v_3 = (1, \phi, 0).$$

For a positive integer f , subdivide each face into a triangular barycentric lattice of frequency f . A lattice point has nonnegative integer coordinates a, b, c with $a + b + c = f$. Radially project

$$p_{a,b,c} = av_1 + bv_2 + cv_3 \quad \text{to} \quad x_{a,b,c} = \frac{p_{a,b,c}}{\|p_{a,b,c}\|} \in \mathbb{S}^2,$$

repeat this on all faces, and deduplicate shared edges and vertices. The resulting node set is X_f .

$$|X_f| = 10f^2 + 2, \quad |X_{22}| = 10(22)^2 + 2 = 4,842.$$

**Why only 52 orbit unknowns?**

The full icosahedral reflection group H_3 has order 120. Symmetry moves most of the 4,842 nodes into large families called orbits. Every orbit has one representative with

$$a \geq b \geq c \geq 0, \quad a + b + c = f.$$

Thus the orbits are indexed by partitions of f into at most three parts. For $f = 22$, there are exactly 52 such partitions.

Instead of one unknown per node, assign one *orbit mass* W_λ to each orbit λ . Every node in that orbit receives per-node weight $W_\lambda/|\lambda|$. The normalization is simply

$$\sum_{\lambda=1}^{52} W_\lambda = 1.$$

Novice checkpoint : orbit mass versus node weight

An orbit mass is the total weight carried by a whole symmetric family. If an orbit has 60 nodes and mass 0.12, each node has weight $0.12/60 = 0.002$. Confusing these two quantities changes the rule.

7 Why exact arithmetic stays possible

The coordinates contain $\sqrt{5}$, and radial normalization seems at first to introduce square roots of complicated norms. The moment equations use homogeneous invariants of even degree. Their normalization denominators are powers of the *squared* norm, so all values remain in

$$\mathbb{Q}(\sqrt{5}) = \{a + b\sqrt{5} : a, b \in \mathbb{Q}\}.$$

The verifier stores $a + b\sqrt{5}$ as the rational pair (a, b) . Addition and multiplication are exact:

$$(a + b\sqrt{5})(c + d\sqrt{5}) = (ac + 5bd) + (ad + bc)\sqrt{5}.$$

For sign tests, it never approximates $\sqrt{5}$. If a and b have opposite signs, it compares a^2 with $5b^2$, taking the signs of a, b into account.

Spherical monomial means are rational. For nonnegative i, j, k ,

$$\int_{\mathbb{S}^2} x^{2i} y^{2j} z^{2k} d\sigma = \frac{(2i-1)!!(2j-1)!!(2k-1)!!}{(2i+2j+2k+1)!!},$$

while any monomial with an odd exponent has mean zero. Expanding the invariant polynomials therefore keeps every required moment inside $\mathbb{Q}(\sqrt{5})$.

AI reading protocol

Exact claim types.

- The equation $AW = b$ means equality of rational pairs, not a residual below a tolerance.
- The inequality $W_\lambda > 0$ means an algebraic sign proof in the chosen real embedding $\sqrt{5} > 0$, not a positive printed decimal.
- Printed decimals are explanatory metadata only.

8 The symmetry reduction

Define two invariant functions using the twelve icosahedral vertices V . If $R^2 = \phi + 2$, set

$$U_6(x) = \sum_{v \in V} \left(\frac{x \cdot v}{R} \right)^6, \quad U_{10}(x) = \sum_{v \in V} \left(\frac{x \cdot v}{R} \right)^{10}.$$

The invariant ring of H_3 is generated in degrees 2, 6, 10. On the unit sphere the degree-two generator is $x^2 + y^2 + z^2 = 1$, so the restricted invariant polynomials are spanned by

$$U_6^r U_{10}^s, \quad r, s \in \mathbb{Z}_{\geq 0}.$$

Sobolev's invariant-cubature principle says that an H_3 -invariant rule is exact through degree t if and only if it is exact on the invariant polynomials through degree t . Therefore we need only the rows

$$6r + 10s \leq t.$$

For orbit representatives x_λ , the exact linear system is

$$A_{(r,s),\lambda} = U_6(x_\lambda)^r U_{10}(x_\lambda)^s, \quad b_{r,s} = \int_{\mathbb{S}^2} U_6^r U_{10}^s d\sigma, \quad AW = b.$$

Novice checkpoint

There are enormously many ordinary monomials of degree at most 69. Symmetry does not pretend they disappear. It proves that if the few invariant tests pass, averaging any error over the group forces every other test to pass as well.

9 The degree-69 construction

The number of invariant conditions is

$$N(t) = \#\{(r, s) \in \mathbb{Z}_{\geq 0}^2 : 6r + 10s \leq t\}.$$

At $t = 69$,

$$N(69) = 49.$$

The exact moment matrix has 49 rows, 52 columns, and exact rank 49. Its solution set is therefore a three-dimensional affine space.

In the verifier's deterministic orbit ordering, the three free columns are

$$(9, 7, 6), \quad (8, 8, 6), \quad (8, 7, 7).$$

Choose their orbit masses to be the rational numbers

$$\begin{aligned} W_{9,7,6} &= \frac{11607197}{394254842}, \\ W_{8,8,6} &= \frac{2312954}{161188803}, \\ W_{8,7,7} &= \frac{7156114}{472522843}. \end{aligned}$$

Exact back-substitution determines the other 49 masses in $\mathbb{Q}(\sqrt{5})$. The verifier then proves

$$AW = b, \quad \sum_{\lambda=1}^{52} W_\lambda = 1, \quad W_\lambda > 0 \quad \text{for every } \lambda.$$

The smallest *orbit mass* is at the vertex orbit $(22, 0, 0)$:

$$W_{22,0,0} \approx 0.003874805888244737 > 0.$$

The decimal is not the certificate; the exact $\mathbb{Q}(\sqrt{5})$ sign test is.

Quantity	Exact result
Nodes in X_{22}	4,842
H_3 -orbits	52
Invariant rows through degree 69	49
Exact rank	49
Affine freedom	3
Positive solution digest	0865572f76c7b8c9...665bf340

The invariant reduction now upgrades these 49 identities to exactness for every spherical polynomial through degree 69.

10 Why degree 70 fails

At degree 70, exactly three new invariant monomials appear:

$$U_6^{10}U_{10}, \quad U_6^5U_{10}^4, \quad U_{10}^7.$$

Their exponent pairs are

$$(r, s) = (10, 1), (5, 4), (0, 7),$$

and each has total degree $6r + 10s = 70$.

Adding these rows changes the system from 49×52 to 52×52 . Exact elimination proves the matrix is nonsingular. Thus there is one and only one orbit-mass vector that could possibly reach degree 70. That vector has exactly one negative component:

$$W_{21,1,0} \approx -0.006609660224057734 < 0.$$

The exact object is an element of $\mathbb{Q}(\sqrt{5})$; its expression is very large, so the standalone verifier recomputes it rather than asking a reader to trust a rounded or typeset coefficient.

No new invariant condition occurs at degree 71. All H_3 invariants have even degree, and the group contains central inversion. Therefore the degree-70 and degree-71 invariant systems are identical.

A dual proof that exposes the contradiction

Let A_{71} be the 52×52 matrix and let $e_\star$ select the orbit $\star = (21, 1, 0)$. Solve exactly

$$A_{71}^\top c = e_\star.$$

The coefficients c define an invariant polynomial P of degree at most 70 whose sampled orbit values are

$$P(x_\star) = 1, \quad P(x_\lambda) = 0 \quad (\lambda \neq \star).$$

Its exact spherical mean is

$$\int_{\mathbb{S}^2} P d\sigma = c^\top b = W_\star < 0.$$

If nonnegative orbit masses integrated degree 70, they would instead give

$$Q[P] = \sum_{\lambda} W_\lambda P(x_\lambda) = W_\star \geq 0,$$

contradicting the negative true mean. This is a Farkas-style separating certificate: one polynomial proves impossibility.

Novice checkpoint

The negative candidate weight is already enough once uniqueness is proved. The dual polynomial is stronger as an audit object: it turns the obstruction into a single inequality that every nonnegative rule would violate.

11 Why arbitrary weights do not escape the obstruction

The computation used orbit-constant weights, but the nonexistence conclusion applies to *all* nonnegative weights on the fixed grid.

Lemma 2 (Symmetrization). *If arbitrary nonnegative node weights on an H_3 -invariant node set are exact through degree t , averaging those weights over H_3 produces nonnegative orbit-constant weights that remain exact through degree t .*

Proof. For each group element g , permute the node weights by g , then average the resulting 120 rules. Nonnegativity and total mass are preserved. The average is constant on orbits. Spherical measure is invariant under H_3 , so every permuted rule, and hence their average, integrates the same polynomial space exactly. $\square$

If any nonnegative degree-70 rule existed on X_{22} , the lemma would produce a nonnegative orbit-constant one. The dual certificate rules that out. This proves the second part of Theorem 1 without assuming the original weights respected symmetry.

12 A counting law that predicts the frontier

There is a useful asymptotic reason that the boundary appears near degree $3f$. The number of partitions of f into at most three parts satisfies

$$\#\text{orbits}(X_f) \sim \frac{f^2}{12},$$

while the number of invariant moment rows satisfies

$$N(t) = \#\{(r, s) \in \mathbb{Z}_{\geq 0}^2 : 6r + 10s \leq t\} \sim \frac{t^2}{120}.$$

Equating the leading terms gives

$$\frac{t^2}{120} \approx \frac{f^2}{12} \implies t \approx \sqrt{10} f \approx 3.1623f.$$

This is a dimension-count heuristic, not a proof of positivity or impossibility. It does explain why a target near $3f$ is plausible and why the linear systems become tight not far above it. At $f = 22$,

$$\sqrt{10} f \approx 69.57.$$

The positive degree-69 system has three degrees of freedom; the three new degree-70 rows make it square and force the negative mass.

Interpretation. Frequency 22 sits almost exactly at the invariant dimension threshold. The obstruction is not random numerical instability. It appears precisely when the last affine freedom is consumed by the next three invariant moments.

13 What this does and does not advance

Established now

- The finite exact positive block extends from $f \leq 21$ to $f \leq 22$ at the target degree τ_f .
- For $f = 22$, target degree 69 is not merely a lower bound; it is the exact maximum under nonnegative weighting of the fixed node set.
- The obstruction has an exact primal witness (one forced negative orbit mass) and an exact dual witness (a cardinal invariant polynomial with negative spherical mean).
- The result is reproducible with standard-library Python and exact rational-pair arithmetic.

Not established

- No all-frequency proof of positive degree τ_f is supplied.
- No statement is made about cubature rules whose nodes may move away from the fixed Class-I grid.
- No claim is made that degree 69 is a record among all spherical cubature formulas; it is a sharp fixed-grid result.
- Machine verification is not a substitute for independent mathematical review, source publication, and historical priority research.

Research conjecture 3 (Positive fixed-Class-I sequence). *For every integer $f \geq 2$, the radial Class-I grid X_f admits strictly positive weights of degree at least*

$$\tau_f = \begin{cases} 3f + 2, & f \text{ odd,} \\ 3f + 3, & f \text{ even.} \end{cases}$$

The exact evidence now covers every $2 \leq f \leq 22$. The next direct test is $f = 23$, but the deeper goal is to replace one-frequency elimination with a uniform convex-geometric or asymptotic proof.

14 How to verify the theorem

There are three different verification statements:

1. **Theorem validity:** the definitions, invariant reduction, positive solution, dual obstruction, and symmetrization argument are the mathematical proof.
2. **Certificate replay:** an exact program reconstructs the objects and proves every equality and sign without tolerances.
3. **Runtime conformance:** a particular language port actually runs and prints the required deterministic hashes and pass line.

The standard-library Python reference was executed twice while preparing this package and passed; the recorded second run took 105.66 seconds. No Julia executable was present in the preparation environment. The Julia source in this LaTeX file is therefore a source-reviewed exact port, cross-checked against the passing Python reference, but it is not labeled runtime-conformant until it is run under Julia.

Release boundary

Do not convert *source reviewed* into *Julia certificate passed*. Promote the Julia artifact only after it exits successfully, prints both expected SHA-256 digests, and ends with **CERTIFICATE PASS**.

Compiling this LaTeX source emits `OmegaNaughtClassIF22ExactVerifier.jl`. Run:

```
julia --startup-file=no OmegaNaughtClassIF22ExactVerifier.jl
```

The implementation uses Julia standard libraries `SHA` and `Printf` and performs these checks:

1. constructs the degree-6 and degree-10 invariants exactly;
2. computes their exact spherical moments;
3. enumerates all 52 orbit representatives of X_{22} ;
4. row-reduces the degree-69 system in $\mathbb{Q}(\sqrt{5})$;
5. substitutes the three stated rational free masses;
6. verifies all 49 identities, total mass 1, and 52 positive signs;
7. adds the three degree-70 rows and solves the forced square system;
8. verifies the single negative mass at $(21, 1, 0)$;
9. builds and checks the exact dual cardinal polynomial; and
10. checks SHA-256 digests of both deterministic exact solution vectors.

The required terminal facts include:

```
nodes=4842 H3_orbits=52
degree_69_conditions=49 rank=49 affine_dimension=3
degree_69: EXACT and STRICTLY POSITIVE
degree_69_solution_sha256=0865572f76c7b8c9816ce937a79361c8926abe785ac3f41b06e4f289665bf340
new_degree_70_invariants=[[10, 1), (5, 4), (0, 7)]
degree_70/71: UNIQUE exact solution
degree_71_solution_sha256=c1e620fc9cf6d99623423cf82450540a5991679e8ded6632bd959914068949f2
Farkas/cardinal dual: EXACT PASS
THEOREM: maximum nonnegative degree on fixed  $X_{22}$  is 69
CERTIFICATE PASS
```

Together with Lemma 2, the orbit-constant maximum is also the maximum over arbitrary nonnegative node weights.

AI reading protocol

Minimum safe parsing contract for another AI.

1. Treat X_{22} , radial projection, orbit ordering, normalized measure, and the real embedding $\sqrt{5} > 0$ as frozen definitions.
2. Treat the three rational free masses as inputs, not fitted decimals.

3. Accept equality only if both rational components of every $\mathbb{Q}(\sqrt{5})$ residual are zero.
4. Accept positivity only through the exact `sign()` method.
5. Distinguish orbit mass from per-node weight.
6. Do not broaden the theorem to movable nodes or signed weights.
7. Use symmetrization only for the nonnegative fixed-node corollary.

14.1 Full Julia exact verifier

This is the complete source emitted by the `filecontents*` block at the beginning of this LaTeX document. `BigFloat` is used only to print orientation decimals; every mathematical equality and sign decision is made in $\mathbb{Q}(\sqrt{5})$.

Listing 1: OmegaNaughtClassIF22ExactVerifier.jl

```

1  #!/usr/bin/env julia
2
3  """
4  Exact verifier for the frequency-22 radial Class-I cubature theorem.
5
6  Every mathematical equality and sign decision is made in Q(sqrt(5)).
7  BigFloat is used only to print short orientation decimals.
8
9  Verified statements:
10
11  1. The fixed Class-I grid X_22 supports strictly positive H3-orbit-constant
12     cubature through algebraic degree 69.
13  2. The unique degree-70/71 invariant solution has a negative mass on orbit
14     (21,1,0).
15  3. An exact Farkas/cardinal dual has the same negative spherical mean.
16
17  By symmetrization, statement 2 excludes every nonnegative degree-70 rule on
18  the fixed node set, even when the original node weights are not orbit-constant.
19  """
20
21  using SHA
22  using Printf
23
24  const QQ = Rational{BigInt}
25
26  rat(n::Integer) = BigInt(n) // BigInt(1)
27
28  struct K5
29      a::QQ
30      b::QQ
31  end
32
33  K5() = K5(rat(0), rat(0))
34  K5(a::Integer) = K5(rat(a), rat(0))
35  K5(a::QQ) = K5(a, rat(0))
36  K5(a::Integer, b::Integer) = K5(rat(a), rat(b))
37
38  Base.zero(::Type{K5}) = K5()
39  Base.one(::Type{K5}) = K5(1)
40  Base.iszero(x::K5) = iszero(x.a) && iszero(x.b)
41  Base.==(x::K5, y::K5) = x.a == y.a && x.b == y.b
42  Base.+(x::K5, y::K5) = K5(x.a + y.a, x.b + y.b)
43  Base.-(x::K5) = K5(-x.a, -x.b)
44  Base.-(x::K5, y::K5) = x + (-y)
45  Base.*(x::K5, y::K5) =
46      K5(x.a * y.a + 5 * x.b * y.b, x.a * y.b + x.b * y.a)
47
48  function Base.inv(x::K5)

```

```

49     d = x.a * x.a - 5 * x.b * x.b
50     iszero(d) && throw(DivideError())
51     K5(x.a / d, -x.b / d)
52 end
53
54 Base.:(x::K5, y::K5) = x * inv(y)
55
56 function Base.^(x::K5, n::Integer)
57     n < 0 && return inv(x) ^ (-n)
58     result = K5(1)
59     base = x
60     exponent = n
61     while exponent > 0
62         isodd(exponent) && (result *= base)
63         base *= base
64         exponent >>= 1
65     end
66     result
67 end
68
69 """Exact sign under the real embedding sqrt(5)>0."""
70 function exact_sign(x::K5)
71     iszero(x.b) && return sign(x.a)
72     x.a >= 0 && x.b >= 0 && return 1
73     x.a <= 0 && x.b <= 0 && return -1
74     aa = x.a * x.a
75     fivebb = 5 * x.b * x.b
76     aa == fivebb && return 0
77     x.a > 0 ? (aa > fivebb ? 1 : -1) : (aa > fivebb ? -1 : 1)
78 end
79
80 function orientation_decimal(x::K5)
81     setprecision(BigFloat, 128) do
82         a = BigFloat(numerator(x.a)) / BigFloat(denominator(x.a))
83         b = BigFloat(numerator(x.b)) / BigFloat(denominator(x.b))
84         a + b * sqrt(BigFloat(5))
85     end
86 end
87
88 const Sqrt5 = K5(0, 1)
89 const PHI = (K5(1) + Sqrt5) / K5(2)
90 const R2 = PHI + K5(2)
91
92 const Point3 = NTuple{3,K5}
93 const Exponent3 = NTuple{3,Int}
94 const Polynomial = Dict{Exponent3,K5}
95
96 const VERTICES = Point3[]
97 for s1 in (1, -1), s2 in (1, -1)
98     push!(VERTICES, (K5(0), K5(s1), K5(s2) * PHI))
99     push!(VERTICES, (K5(s1), K5(s2) * PHI, K5(0)))
100     push!(VERTICES, (K5(s1) * PHI, K5(0), K5(s2)))
101 end
102
103 const FACE = (
104     (K5(0), K5(1), PHI),
105     (K5(-1), PHI, K5(0)),
106     (K5(1), PHI, K5(0)),
107 )
108
109 function dot3(x::Point3, y::Point3)
110     total = K5()
111     for i in 1:3
112         total += x[i] * y[i]
113     end
114     total
115 end
116
117 function poly_add(left::Polynomial, right::Polynomial)
118     result = copy(left)
119     for (exponent, coefficient) in right

```

```

120     value = get(result, exponent, K5()) + coefficient
121     iszero(value) ? delete!(result, exponent) : (result[exponent] = value)
122 end
123 result
124 end
125
126 function poly_scale(poly::Polynomial, scalar::K5)
127     result = Polynomial()
128     for (exponent, coefficient) in poly
129         value = scalar * coefficient
130         !iszero(value) && (result[exponent] = value)
131     end
132     result
133 end
134
135 function poly_multiply(left::Polynomial, right::Polynomial)
136     result = Polynomial()
137     for ((i, j, k), a) in left
138         for ((u, v, w), b) in right
139             exponent = (i + u, j + v, k + w)
140             result[exponent] = get(result, exponent, K5()) + a * b
141         end
142     end
143     filter!(pair -> !iszero(last(pair)), result)
144     result
145 end
146
147 function multinomial3(n::Int, i::Int, j::Int, k::Int)
148     factorial(big(n)) ÷
149     (factorial(big(i)) * factorial(big(j)) * factorial(big(k)))
150 end
151
152 function linear_power(vector::Point3, n::Int)
153     result = Polynomial()
154     vx, vy, vz = vector
155     for i in 0:n
156         for j in 0:(n - i)
157             k = n - i - j
158             coefficient =
159                 K5(multinomial3(n, i, j, k)) * vx^i * vy^j * vz^k
160             !iszero(coefficient) && (result[(i, j, k)] = coefficient)
161         end
162     end
163     result
164 end
165
166 function invariant_polynomial(n::Int)
167     result = Polynomial()
168     for vertex in VERTICES
169         result = poly_add(result, linear_power(vertex, n))
170     end
171     poly_scale(result, R2 ^ (-(n ÷ 2)))
172 end
173
174 const U6_POLY = invariant_polynomial(6)
175 const U10_POLY = invariant_polynomial(10)
176
177 function odd_double_factorial(n::Int)
178     result = BigInt(1)
179     for value in n:-2:1
180         result *= value
181     end
182     result
183 end
184
185 function sphere_monomial_mean(i::Int, j::Int, k::Int)
186     (isodd(i) || isodd(j) || isodd(k)) && return rat(0)
187     a, b, c = i ÷ 2, j ÷ 2, k ÷ 2
188     numerator_value =
189         odd_double_factorial(2 * a - 1) *
190         odd_double_factorial(2 * b - 1) *

```

```

191     odd_double_factorial(2 * c - 1)
192     denominator_value = odd_double_factorial(2(a + b + c) + 1)
193     numerator_value // denominator_value
194 end
195
196 function sphere_integral(poly::Polynomial)
197     result = K5()
198     for ((i, j, k), coefficient) in poly
199         result += coefficient * K5(sphere_monomial_mean(i, j, k))
200     end
201     result
202 end
203
204 const POWERS6 = Dict{Int,Polynomial}()
205     0 => Polynomial((0, 0, 0) => K5(1)),
206     1 => U6_POLY,
207 )
208 const POWERS10 = Dict{Int,Polynomial}()
209     0 => Polynomial((0, 0, 0) => K5(1)),
210     1 => U10_POLY,
211 )
212
213 function invariant_moment(r::Int, s::Int)
214     while maximum(keys(POWERS6)) < r
215         n = maximum(keys(POWERS6)) + 1
216         POWERS6[n] = poly_multiply(POWERS6[n - 1], U6_POLY)
217     end
218     while maximum(keys(POWERS10)) < s
219         n = maximum(keys(POWERS10)) + 1
220         POWERS10[n] = poly_multiply(POWERS10[n - 1], U10_POLY)
221     end
222     sphere_integral(poly_multiply(POWERS6[r], POWERS10[s]))
223 end
224
225 function basis_pairs(degree::Int)
226     pairs = Tuple{Int,Int}[]
227     for s in 0:(degree ÷ 10)
228         for r in 0:((degree - 10 * s) ÷ 6)
229             push!(pairs, (r, s))
230         end
231     end
232     sort!(pairs, by = pair -> (6 * pair[1] + 10 * pair[2], pair[2], pair[1]))
233 end
234
235 function partitions3(total::Int)
236     result = NTuple{3,Int}[]
237     for a in total:-1:0
238         for b in min(a, total - a):-1:0
239             c = total - a - b
240             0 <= c <= b && push!(result, (a, b, c))
241         end
242     end
243     result
244 end
245
246 function orbit_invariants(representative::NTuple{3,Int})
247     a, b, c = representative
248     point = ntuple(
249         i -> K5(a) * FACE[1][i] + K5(b) * FACE[2][i] + K5(c) * FACE[3][i],
250         3,
251     )
252     norm_squared = dot3(point, point)
253     sum6 = K5()
254     sum10 = K5()
255     for vertex in VERTICES
256         product = dot3(point, vertex)
257         sum6 += product^6
258         sum10 += product^10
259     end
260     u6 = sum6 / (R2^3 * norm_squared^3)
261     u10 = sum10 / (R2^5 * norm_squared^5)

```

```

262 (u6, u10)
263 end
264
265 function exact_rref(matrix::Matrix{K5}, right_hand_side::Vector{K5})
266     row_count, column_count = size(matrix)
267     augmented = Matrix{K5}(undef, row_count, column_count + 1)
268     augmented[:, 1:column_count] .= matrix
269     augmented[:, column_count + 1] .= right_hand_side
270
271     pivot_row = 1
272     pivots = Int[]
273     for column in 1:column_count
274         selected_row = 0
275         for row in pivot_row:row_count
276             if !iszero(augmented[row, column])
277                 selected_row = row
278                 break
279             end
280         end
281         selected_row == 0 && continue
282         if selected_row != pivot_row
283             for j in 1:(column_count + 1)
284                 augmented[pivot_row, j], augmented[selected_row, j] =
285                     augmented[selected_row, j], augmented[pivot_row, j]
286             end
287         end
288
289         inverse_pivot = inv(augmented[pivot_row, column])
290         for j in 1:(column_count + 1)
291             augmented[pivot_row, j] *= inverse_pivot
292         end
293
294         for row in 1:row_count
295             row == pivot_row && continue
296             factor = augmented[row, column]
297             iszero(factor) && continue
298             for j in 1:(column_count + 1)
299                 augmented[row, j] -= factor * augmented[pivot_row, j]
300             end
301         end
302
303         push!(pivots, column)
304         pivot_row += 1
305         pivot_row > row_count && break
306     end
307
308     for row in pivot_row:row_count
309         all_zero = all(iszero(augmented[row, column]) for column in 1:column_count)
310         all_zero && !iszero(augmented[row, column_count + 1]) &&
311             error("inconsistent exact moment system")
312     end
313     augmented, pivots
314 end
315
316 function choose_solution(
317     column_count::Int,
318     reduced::Matrix{K5},
319     pivots::Vector{Int},
320     free_values::Vector{K5},
321 )
322     pivot_set = Set{Int}(pivots)
323     free_columns = [column for column in 1:column_count if !(column in pivot_set)]
324     length(free_columns) == length(free_values) ||
325         error("wrong number of free values")
326
327     solution = fill{K5}(), column_count
328     for (column, value) in zip(free_columns, free_values)
329         solution[column] = value
330     end
331     for (row, column) in enumerate(pivots)
332         value = reduced[row, column]

```

```

333     for free_column in free_columns
334         value -= reduced[row, free_column] * solution[free_column]
335     end
336     solution[column] = value
337 end
338 solution, free_columns
339 end
340
341 function verify_system(
342     matrix::Matrix{K5},
343     right_hand_side::Vector{K5},
344     solution::Vector{K5},
345 )
346     for row in axes(matrix, 1)
347         lhs = K5()
348         for column in axes(matrix, 2)
349             lhs += matrix[row, column] * solution[column]
350         end
351         lhs == right_hand_side[row] || return false
352     end
353     true
354 end
355
356 function moment_system(frequency::Int, degree::Int)
357     representatives = partitions3(frequency)
358     pairs = basis_pairs(degree)
359     values = [orbit_invariants(rep) for rep in representatives]
360     matrix = Matrix{K5}(undef, length(pairs), length(representatives))
361     for (row, (r, s)) in enumerate(pairs)
362         for column in eachindex(representatives)
363             u6, u10 = values[column]
364             matrix[row, column] = u6^r * u10^s
365         end
366     end
367     rhs = [invariant_moment(r, s) for (r, s) in pairs]
368     reduced, pivots = exact_rref(matrix, rhs)
369     representatives, pairs, values, matrix, rhs, reduced, pivots
370 end
371
372 canonical_fraction(value::QQ) =
373     string(numerator(value), "/", denominator(value))
374 canonical_k5(value::K5) =
375     string(canonical_fraction(value.a), ";", canonical_fraction(value.b))
376
377 function solution_digest(representatives, weights)
378     records = String[]
379     for ((a, b, c), weight) in zip(representatives, weights)
380         push!(records, string(a, ",", b, ",", c, ":", canonical_k5(weight)))
381     end
382     bytes2hex(sha256(codeunits(join(records, "\n"))))
383 end
384
385 const POSITIVE_FREE_VALUES = K5[
386     K5(BigInt(11607197) // BigInt(394254842)),
387     K5(BigInt(2312954) // BigInt(161188803)),
388     K5(BigInt(7156114) // BigInt(472522843)),
389 ]
390
391 const EXPECTED_DEGREE69_SHA256 =
392     "0865572f76c7b8c9816ce937a79361c8926abe785ac3f41b06e4f289665bf340"
393 const EXPECTED_DEGREE71_SHA256 =
394     "c1e620fc9cf6d99623423cf82450540a5991679e8ded6632bd959914068949f2"
395
396 function main()
397     started = time()
398     frequency = 22
399
400     reps, pairs69, values, matrix69, rhs69, reduced69, pivots69 =
401         moment_system(frequency, 69)
402     weights69, free_columns =
403         choose_solution(length(reps), reduced69, pivots69, POSITIVE_FREE_VALUES)

```

```

404 @assert length(reps) == 52
405 @assert length(pairs69) == length(pivots69) == 49
406 @assert free_columns == [50, 51, 52]
407 @assert reps[free_columns] == [(9, 7, 6), (8, 8, 6), (8, 7, 7)]
408 @assert verify_system(matrix69, rhs69, weights69)
409 @assert foldl(+, weights69; init = K5()) == K5(1)
410 @assert all(exact_sign(weight) > 0 for weight in weights69)
411
412
413 minimum_index = 1
414 for index in 2:length(weights69)
415     exact_sign(weights69[index] - weights69[minimum_index]) < 0 &&
416         (minimum_index = index)
417 end
418 @assert reps[minimum_index] == (22, 0, 0)
419 digest69 = solution_digest(reps, weights69)
420 @assert digest69 == EXPECTED_DEGREE69_SHA256
421
422 pairs71 = basis_pairs(71)
423 old_pairs = Set(pairs69)
424 new_pairs = [pair for pair in pairs71 if !(pair in old_pairs)]
425 @assert new_pairs == [(10, 1), (5, 4), (0, 7)]
426
427 extra_matrix = Matrix{K5}(undef, length(new_pairs), length(reps))
428 for (row, (r, s)) in enumerate(new_pairs)
429     for column in eachindex(reps)
430         u6, u10 = values[column]
431         extra_matrix[row, column] = u6^r * u10^s
432     end
433 end
434 extra_rhs = [invariant_moment(r, s) for (r, s) in new_pairs]
435
436 constants = fill(K5(), length(reps))
437 coefficients = fill(K5(), length(reps), 3)
438 for (k, column) in enumerate(free_columns)
439     coefficients[column, k] = K5(1)
440 end
441 for (row, column) in enumerate(pivots69)
442     constants[column] = reduced69[row, end]
443     for (k, free_column) in enumerate(free_columns)
444         coefficients[column, k] = -reduced69[row, free_column]
445     end
446 end
447
448 three_by_three = Matrix{K5}(undef, 3, 3)
449 three_rhs = fill(K5(), 3)
450 for row in 1:3
451     for k in 1:3
452         value = K5()
453         for column in eachindex(reps)
454             value += extra_matrix[row, column] * coefficients[column, k]
455         end
456         three_by_three[row, k] = value
457     end
458     constant_part = K5()
459     for column in eachindex(reps)
460         constant_part += extra_matrix[row, column] * constants[column]
461     end
462     three_rhs[row] = extra_rhs[row] - constant_part
463 end
464
465 reduced3, pivots3 = exact_rref(three_by_three, three_rhs)
466 @assert pivots3 == [1, 2, 3]
467 forced_free, no_free = choose_solution(3, reduced3, pivots3, K5[])
468 @assert isempty(no_free)
469
470 weights71 = fill(K5(), length(reps))
471 for column in eachindex(reps)
472     value = constants[column]
473     for k in 1:3
474         value += coefficients[column, k] * forced_free[k]

```

```

475         end
476         weights71[column] = value
477     end
478
479     matrix71 = vcat(matrix69, extra_matrix)
480     rhs71 = vcat(rhs69, extra_rhs)
481     @assert size(matrix71) == (52, 52)
482     @assert verify_system(matrix71, rhs71, weights71)
483     @assert foldl(+, weights71; init = K5()) == K5(1)
484
485     negative_indices =
486         [index for index in eachindex(weights71) if exact_sign(weights71[index]) < 0]
487     @assert length(negative_indices) == 1
488     bad_index = only(negative_indices)
489     bad_rep = reps[bad_index]
490     bad_weight = weights71[bad_index]
491     @assert bad_rep == (21, 1, 0)
492
493     transpose_matrix = Matrix(transpose(matrix71))
494     selector = [index == bad_index ? K5(1) : K5() for index in eachindex(reps)]
495     reduced_dual, pivots_dual = exact_rref(transpose_matrix, selector)
496     @assert length(pivots_dual) == 52
497     dual_coefficients, dual_free =
498         choose_solution(52, reduced_dual, pivots_dual, K5[])
499     @assert isempty(dual_free)
500     @assert verify_system(transpose_matrix, selector, dual_coefficients)
501
502     dual_mean = K5()
503     for index in eachindex(rhs71)
504         dual_mean += dual_coefficients[index] * rhs71[index]
505     end
506     @assert dual_mean == bad_weight
507     @assert exact_sign(dual_mean) < 0
508
509     digest71 = solution_digest(reps, weights71)
510     @assert digest71 == EXPECTED_DEGREE71_SHA256
511
512     println("OMEGA NAUGHT - EXACT JULIA CLASS-I f=22 CERTIFICATE")
513     println("Arithmetic: Q(sqrt(5)); equality/sign checks: exact")
514     println("nodes=$(10 * frequency^2 + 2) H3_orbits=$(length(reps))")
515     println(
516         "degree_69_conditions=$(length(pairs69)) rank=$(length(pivots69)) " *
517         "affine_dimension=$(length(free_columns))",
518     )
519     println("free_orbits=$(reps[free_columns])")
520     println("free_masses=$(POSITIVE_FREE_VALUES)")
521     @printf(
522         "degree_69: EXACT and STRICTLY POSITIVE; minimum_orbit_mass=%.16g at %s\n",
523         Float64(orientation_decimal(weights69[minimum_index])),
524         string(reps[minimum_index]),
525     )
526     println("degree_69_solution_sha256=$digest69")
527     println("new_degree_70_invariants=$new_pairs")
528     @printf(
529         "degree_70/71: UNIQUE exact solution; negative_orbit_mass=%.16g at %s\n",
530         Float64(orientation_decimal(bad_weight)),
531         string(bad_rep),
532     )
533     println("degree_71_solution_sha256=$digest71")
534     println("Farkas/cardinal dual: EXACT PASS; mean equals negative forced mass")
535     println("THEOREM: maximum nonnegative degree on fixed X_22 is 69")
536     @printf("CERTIFICATE PASS (%.2f seconds)\n", time() - started)
537     true
538 end
539
540 if abspath(PROGRAM_FILE) == @__FILE__
541     main()
542 end

```

15 Research directions opened by the result

1. **Frequency 23.** Construct an exact interior point for the target degree $\tau_{23} = 71$, then locate the first separating dual.
2. **A uniform positivity theorem.** Show that the spherical moment vector lies in the relative interior of the convex hull of the Class-I orbit evaluation vectors for all sufficiently large f .
3. **Sharp-degree law.** Compare the actual maximum degree with the dimension threshold defined by

$$N(t) \leq \#\{\text{partitions of } f \text{ into at most three parts}\}.$$

4. **Dual geometry.** Study which near-vertex or near-edge orbit is selected by the first negative cardinal polynomial and whether that orbit type obeys a periodic law in f .
5. **Condition-aware exact solvers.** Use floating linear programming only to propose a point in the positive polytope, then rationalize its free coordinates and certify it over $\mathbb{Q}(\sqrt{5})$. This avoids false infeasibility from ill-conditioned raw moment matrices.

Reproducibility record

Item	Frozen value
Grid	complete radial Class-I X_{22}
Node count	4,842
Symmetry	full icosahedral reflection group H_3
Measure	normalized spherical area, total 1
Exact field	$\mathbb{Q}(\sqrt{5})$, real embedding $\sqrt{5} > 0$
Degree-69 rows/rank	49/49
Degree-69 free orbits	(9, 7, 6), (8, 8, 6), (8, 7, 7)
Minimum positive orbit mass	$\approx 0.003874805888244737$ at (22, 0, 0)
Degree-70/71 rows/rank	52/52
Forced negative orbit mass	$\approx -0.006609660224057734$ at (21, 1, 0)
Positive-vector SHA-256	0865572f76c7b8c9816ce937a79361c8 926abe785ac3f41b06e4f289665bf340
Forced-vector SHA-256	c1e620fc9cf6d99623423cf82450540a 5991679e8ded6632bd959914068949f2

Frozen asset ledger

These hashes identify the local research inputs and companion implementations used to prepare this package. A release must hash the bytes actually served; a filename alone is not an identity.

Artifact	Bytes	SHA-256
ClassI_Cubature_Proof_Dossier_f2_f21.pdf	226725	4c705abab8b99ae50a80c30c497dcd8 dbe2170eb37f8c877908de7316c4a31d5
classI_exact_certificates_f2_f21.json	1815267	7b3945ad58b76341bbc22e7812a359f 64b61a68d413b2668012dc155803edb5e

verify_classI_f2_f21.py	4102	7c3cd105dae6cf36fd140a8b6525ed7 22fb02b88bd2ba152c465847befa944fa
Omega_Naught_ClassI_f22_Research_Addendum.pdf	356694	72332ba4f97425b132fe8073aeccfd5 450e19c8f62a3a654969401213dced7a1
Omega_Naught_ClassI_f22_exact_verifier.py	16273	9f50631e4ef2dc67f20a3ad853a68cf bc522c03e473516e95725dbbca8291e7f
OmegaNaughtClassIF22ExactVerifier.jl	16570	74476a0ecf5e385a20100709a8896d f78bb81b519a8701ac722bea7ca7239c

Implementation acceptance tests

1. A site-wide search finds no stale claim that the conjectural range begins at $f = 10$.
2. The homepage and cubature page say $f \geq 23$ remains open.
3. Paper VI v1.3 remains scoped to $f = 1, \dots, 9$.
4. Frequencies $10, \dots, 21$ are described as exact certificate extensions, with maximality unclaimed.
5. The $f = 22$ sharp claim always includes *nonnegative weights on the fixed radial Class-I grid X_{22}* .
6. CLASS_I_CUBATURE/1.0 and G6_RELEASE/1.1 retain their original bytes and hashes.
7. New downloads match the exact lengths and SHA-256 digests in their newly frozen manifest.
8. The Python reference exits successfully with both solution digests and **CERTIFICATE PASS**.
9. The Julia artifact remains marked **source-reviewed** until an actual Julia run independently satisfies the same conditions.
10. No SRSD, Omega algebra, frozen Q, S , JPRV, canonical serialization, or typed G6 definition changes as a side effect.

AI reading protocol

A decimal residual, a floating positivity check, or a visual match of printed weights is not acceptance evidence. Equality requires zero rational-pair residuals; positivity requires exact algebraic sign decisions; artifact identity requires byte length plus SHA-256.

References

- [1] H. Nozaki and M. Sawa, *Note on cubature formulae and designs obtained from group orbits*, *Canadian Journal of Mathematics* 64 (2012), 1359–1377. doi:10.4153/CJM-2011-069-5.
- [2] A. D. McLaren, *Optimal numerical integration on a sphere*, *Mathematics of Computation* 17 (1963), 361–383. doi:10.1090/S0025-5718-1963-0159418-2.
- [3] A. S. Popov, *The cubature formulas on a sphere that are invariant with respect to the icosahedral group of rotations*, *Numerical Analysis and Applications* 1 (2008), 355–361. doi:10.1134/S199542390804006X.
- [4] A. S. Popov, *The search of the best cubature formulas on the sphere that are invariant under the icosahedral rotation group*, *Numerical Analysis and Applications* 16 (2023), 348–358. doi:10.1134/S1995423923040067.
- [5] *Omega Naught*, public site audited 24 August 2026. <https://omega-naught.grok.me>.