Quadray Analytical Details and Methods

Daniel Ari Friedman
ORCID: 0000-0001-6232-9096
Email: daniel@activeinference.institute

August 16, 2025

Contents

1 Quadray Analytical Details and Methods **1**
1.1 Overview . . . 1
1.2 Mathematical Foundations . . . 1
1.2.1 Framework Distinctions . . . 1
1.2.2 Key Mathematical Principles . . . 1
1.2.3 Implementation Strategy . . . 1
1.3 Framework Integration . . . 2
1.4 Fuller.4D Coordinates and Normalization . . . 2
1.5 Conversions and Vector Operations: Quadray ↔ Cartesian (Fuller.4D ↔ Coxeter.4D/XYZ) . . . 2
1.5.1 Integer-coordinate constructions (compact derivation box) . . . 2
1.5.2 Example vertex lists and volume checks (illustrative) . . . 4
1.6 Integer Volume Quantization . . . 4
1.7 Distances and Metrics . . . 6
1.8 XYZ determinant and S3 conversion . . . 6
1.9 Fisher Geometry in Quadray Space . . . 6
1.10 Practical Methods . . . 6
1.11 Tetravolumes with Quadrays . . . 6
1.11.1 Bridging and native tetravolume formulas . . . 7
1.11.2 Short Python snippets . . . 7
1.11.3 Random tetrahedra in the IVM (integer volumes) . . . 11
1.11.4 Algebraic precision . . . 11
1.11.5 XYZ determinant and the S3 conversion . . . 11
1.11.6 D^3 vs R^3: 60° “closing the lid” vs orthogonal “cubing” . . . 12
1.12 Practical Implementation Workflow . . . 12
1.12.1 1. Mathematical Formulation . . . 12
1.12.2 2. Implementation Strategy . . . 12
1.12.3 3. Testing and Validation . . . 12
1.12.4 4. Documentation and Reproducibility . . . 12
1.12.5 5. Optimization and Extension . . . 12
1.13 Code methods (anchors) . . . 13
1.13.1 Core Quadray operations . . . 13
1.13.2 Volume calculations . . . 13
1.13.3 Coordinate conversions . . . 13
1.13.4 Linear algebra utilities . . . 13
1.13.5 Optimization methods . . . 13
1.13.6 Information geometry methods . . . 14
1.13.7 Metrics and analysis . . . 14
1.13.8 Examples and utilities . . . 15
1.13.9 Symbolic computation . . . 15

1.13.10 Visualization and animation . . . 15
1.13.11 Path and file utilities . . . 15
1.13.12 Additional Nelder–Mead components . . . 16
1.13.13 Discrete variational components . . . 16
1.13.14 Glossary generation . . . 16
1.13.15 Geometry utilities . . . 16
1.13.16 Comprehensive test coverage . . . 16
1.14 Reproducibility checklist . . . 17

1 Quadray Analytical Details and Methods

1.1 Overview

This section provides detailed analytical methods for working with Quadray coordinates, including coordinate conventions, volume calculations, and optimization approaches. We emphasize the distinction between different 4D frameworks and provide practical computational methods.

1.2 Mathematical Foundations

The methods presented here rest on three interconnected mathematical frameworks. For comprehensive definitions, see Section 2: 4D Namespaces.

1.2.1 Framework Distinctions

- **Coxeter.4D**: Euclidean 4D geometry with standard metric tensor and volume forms
- **Einstein.4D**: Minkowski spacetime with indefinite metric for information geometry analogies

- **Fuller.4D**: Synergetics/Quadray coordinates with integer lattice constraints and IVM unit conventions

1.2.2 Key Mathematical Principles

- **Integer volume quantization**: Lattice constraints ensure tetrahedral volumes are exact integers in IVM units
- **Coordinate system bridges**: Linear transformations between Fuller.4D and Coxeter.4D preserve geometric relationships
- **Information geometry**: Fisher metric provides Riemannian structure for optimization on parameter manifolds
- **Exact arithmetic**: Bareiss algorithm ensures determinant calculations remain exact for integer inputs

1.2.3 Implementation Strategy

All mathematical concepts are implemented with: - **Exact arithmetic** where possible (integer determinants, symbolic computation) - **Numerical stability** for floating-point operations (ridge regularization, condition number monitoring) - **Cross-validation** between different formulations (Ace 5×5 vs Cayley–Menger, native vs bridging approaches) - **Comprehensive testing** ensuring mathematical correctness across edge cases

1.3 Framework Integration

The three 4D frameworks serve distinct but complementary roles in our implementation:

- **Coxeter.4D provides the container**: Euclidean 4D space serves as the mathematical foundation for volume calculations, distance metrics, and geometric transformations. When we compute volumes via Cayley–Menger determinants or XYZ coordinate determinants, we operate in this framework.

- **Einstein.4D provides the analogy**: The Minkowski metric structure inspires our information geometry approach, where the Fisher information matrix acts as a Riemannian metric on parameter space. Natural gradient descent follows geodesics on this information manifold, analogous to how particles follow geodesics in spacetime.
- **Fuller.4D provides the constraints**: The Quadray coordinate system and IVM lattice impose integer constraints that enable exact arithmetic and discrete optimization. The synergetics unit conventions (regular tetrahedron volume = 1) create a quantized geometry where volumes are exact integers.

This multi-framework approach allows us to: 1. Use standard Euclidean methods for volume calculations where relevant or already in use (Coxeter.4D) 2. Apply information geometry principles for geodesic optimization (Einstein.4D analogy)
3. Maintain exact arithmetic in the Isotropic Vector Matrix (IVM) setting through integer lattice constraints (Fuller.4D) 4. Bridge and swap among frameworks via coordinate transformations and unit conversions

1.4 Fuller.4D Coordinates and Normalization

- Quadray vector q = (a,b,c,d), a,b,c,d ≥ 0, with at least one coordinate zero under normalization.
- Projective normalization can add/subtract (k,k,k,k) without changing direction; choose k to enforce non-negativity and one zero minimum.
- Isotropic Vector Matrix (IVM): integer quadrays describe CCP sphere centers; the 12 permutations of {2,1,1,0} form the cuboctahedron (vector equilibrium).
  - Integer-coordinate models: assigning unit IVM tetravolume to the regular tetrahedron yields integer coordinates for several familiar polyhedra (inverse tetrahedron, cube, octahedron, rhombic dodecahedron, cuboctahedron) when expressed as linear combinations of the four quadray basis vectors. See overview: Quadray coordinates.

1.5 Conversions and Vector Operations: Quadray ↔ Cartesian (Fuller.4D ↔ Coxeter.4D/XYZ)

- **Embedding conventions** determine the linear maps between Quadray (Fuller.4D) and Cartesian XYZ (a 3D slice or embedding aligned with Coxeter.4D conventions).
- **References**: Urner provides practical conversion write-ups and matrices; see:
  - Quadrays and XYZ: Urner – Quadrays and XYZ
  - Introduction with examples: Urner – Quadray intro
- **Implementation**: choose a fixed tetrahedral embedding; construct a 3×4 matrix M that maps (a,b,c,d) to (x,y,z), respecting A,B,C,D directions to tetra vertices. The inverse map can be defined up to projective normalization (adding (k,k,k,k)). When comparing volumes, use the `S3=\sqrt{9/8}` scale to convert XYZ (Euclidean) volumes to IVM (Fuller.4D) units.
- **Vector view**: treat `q` as a vector with magnitude and direction; define dot products and norms by pushing to XYZ via `M`.

1.5.1 Integer-coordinate constructions (compact derivation box)

- Under the synergetics convention (unit regular tetrahedron has tetravolume 1), many familiar solids admit Quadray integer coordinates. For example, the octahedron at the same edge length has tetravolume 4, and its vertices can be formed as integer linear combinations of the four axes A,B,C,D subject to the Quadray normalization rule.
- The cuboctahedron (vector equilibrium) arises as the shell of the 12 nearest IVM neighbors given by the permutations of $(2, 1, 1, 0)$. The rhombic dodecahedron (tetravolume 6) is the Voronoi cell of the FCC/CCP packing centered at the origin under the same embedding.
- See the following figure for a schematic summary of these relationships.

Object	Quadray construction (sketch)	IVM volume
Regular tetrahedron	Vertices `o=(0,0,0,0)`, `p=(2,1,0,1)`, `q=(2,1,1,0)`, `r=(2,0,1,1)`	1
Cube (same edge)	Union of 3 mutually orthogonal rhombic belts wrapped on the tetra frame; edges tracked by XYZ embedding; compare the following figure	3
Octahedron (same edge)	Convex hull of mid-edges of the tetra frame (pairwise axis sums normalized)	4
Rhombic dodecahedron	Voronoi cell of FCC/CCP packing at origin (dual to cuboctahedron)	6
Cuboctahedron (vector equilibrium)	Shell of the 12 nearest IVM neighbors: permutations of `(2,1,1,0)`	20
Truncated octahedron	Archimedean solid with 6 square and 8 hexagonal faces; space-filling tiling	20

Small coordinate examples (subset):

- Cuboctahedron neighbors (representatives): `(2,1,1,0)`, `(2,1,0,1)`, `(2,0,1,1)`, `(1,2,1,0)`; the full shell is all distinct permutations.
- Tetrahedron: `[(0,0,0,0), (2,1,0,1), (2,1,1,0), (2,0,1,1)]`.

Short scripts:

```
python3 quadmath/scripts/polyhedra_quadray_constructions.py
```

Programmatic check (neighbors, equal radii, adjacency):

```
import numpy as np
from examples import example_cuboctahedron_vertices_xyz

xyz = np.array(example_cuboctahedron_vertices_xyz())
r = np.linalg.norm(xyz[0])
assert np.allclose(np.linalg.norm(xyz, axis=1), r)

# Touching neighbors have separation 2r
touch = []
for i in range(len(xyz)):
    for j in range(i+1, len(xyz)):
        d = np.linalg.norm(xyz[i] - xyz[j])
        if abs(d - 2*r) / (2*r) < 0.05:
            touch.append((i, j))
assert len(touch) > 0
```

1.5.2 Example vertex lists and volume checks (illustrative)

The following snippets use canonical IVM neighbor points (permutations of $(2,1,1,0)$) to illustrate simple decompositions consistent with synergetics volumes. Each tetra volume is computed via `ace_tetravolume_5x5` and summed.

Octahedron (V = 4) as four unit IVM tetras around the origin:

```python
from quadray import Quadray, ace_tetravolume_5x5

o = Quadray(0,0,0,0)
T = [
    (Quadray(2,1,0,1), Quadray(2,1,1,0), Quadray(2,0,1,1)),
    (Quadray(1,2,0,1), Quadray(1,2,1,0), Quadray(0,2,1,1)),
    (Quadray(1,1,2,0), Quadray(1,0,2,1), Quadray(0,1,2,1)),
    (Quadray(2,0,1,1), Quadray(1,2,0,1), Quadray(0,1,2,1)),  # representative variant
]
V_oct = sum(ace_tetravolume_5x5(o, a, b, c) for (a,b,c) in T)
```

Cube (V = 3) as three unit IVM tetras (orthant-like around the origin):

```python
from quadray import Quadray, ace_tetravolume_5x5

o = Quadray(0,0,0,0)
triples = [
    (Quadray(2,1,0,1), Quadray(2,1,1,0), Quadray(2,0,1,1)),
    (Quadray(1,2,0,1), Quadray(1,2,1,0), Quadray(0,2,1,1)),
    (Quadray(1,1,2,0), Quadray(1,0,2,1), Quadray(0,1,2,1)),
]
V_cube = sum(ace_tetravolume_5x5(o, a, b, c) for (a,b,c) in triples)
```

Notes.

- These decompositions are illustrative and use canonical IVM neighbor triples that produce unit tetras under `ace_tetravolume_5x5`. Other equivalent tilings are possible.
- Volumes are invariant to adding (k, k, k, k) to each vertex of a tetra (projective normalization), which the 5×5 determinant respects.

1.6 Integer Volume Quantization

For a tetrahedron with vertices P₀..P₃ in the Quadray integer lattice (Fuller.4D):

$$V = \tfrac{1}{6}\,\left|\det\left[\,P_1 - P_0,\; P_2 - P_0,\; P_3 - P_0\,\right]\right| \tag{1}$$

- With integer coordinates, the determinant is integer; lattice tetrahedra yield integer volumes.
- Unit conventions: regular tetrahedron volume = 1 (synergetics).

Notes.

- $P_0, \ldots, P_3$ are tetrahedron vertices in Quadray coordinates.
- V is the Euclidean volume measured in IVM tetra-units; the $1/6$ factor converts the parallelepiped determinant to a tetra volume.
- Background and variations are discussed under Tetrahedron volume formulas: Tetrahedron – volume.

Tom Ace 5×5 determinant (tetravolume directly from quadrays):

$$V_{ivm} = \tfrac{1}{4}\left|\det\begin{pmatrix} a_0 & a_1 & a_2 & a_3 & 1 \\ b_0 & b_1 & b_2 & b_3 & 1 \\ c_0 & c_1 & c_2 & c_3 & 1 \\ d_0 & d_1 & d_2 & d_3 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}\right| \tag{2}$$

This returns the same integer volumes for lattice tetrahedra. See the implementation `ace_tetravolume_5x5`.

Notes.

- Rows correspond to the Quadray 4-tuples of the four vertices with a final affine column of ones; the last row enforces projective normalization.
- The factor $\frac{1}{4}$ returns tetravolumes in IVM units consistent with synergetics. See also Quadray coordinates.

Equivalently, define the 5×5 matrix of quadray coordinates augmented with an affine 1 as

$$M(q_0, q_1, q_2, q_3) = \begin{bmatrix} q_{01} & q_{02} & q_{03} & q_{04} & 1 \\ q_{11} & q_{12} & q_{13} & q_{14} & 1 \\ q_{21} & q_{22} & q_{23} & q_{24} & 1 \\ q_{31} & q_{32} & q_{33} & q_{34} & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}, \qquad V_{ivm} = \tfrac{1}{4} \left| \det M(q_0, q_1, q_2, q_3) \right|. \tag{3}$$

Points vs vectors: subtracting points is shorthand for forming edge vectors. We treat quadray 4-tuples as vectors from the origin; differences like $(P_1 - P_0)$ mean "edge vectors," avoiding ambiguity between "points" and "vectors."

Equivalently via Cayley–Menger determinant (Coxeter.4D/Euclidean lengths) (Cayley–Menger determinant):

$$288\, V^2 = \det \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & d_{13}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & d_{23}^2 \\ 1 & d_{30}^2 & d_{31}^2 & d_{32}^2 & 0 \end{pmatrix} \tag{4}$$

References: Cayley–Menger determinant, lattice tetrahedra discussions in geometry texts; see also Tetrahedron – volume. Code: `integer_tetra_volume`, `ace_tetravolume_5x5`.

Notes.

- **Pairwise distances**: d_{ij} are Euclidean distances between vertices P_i and P_j.
- **Length-only formulation**: Cayley–Menger provides a length-only formula for simplex volumes, here specialized to tetrahedra; see the canonical reference above.

Table 2: Polyhedra tetravolumes in IVM units (edge length equal to the unit tetra edge). {#tbl:polyhedra_volumes}

Polyhedron (edge = tetra edge)	Volume (tetra-units)
Regular Tetrahedron	1
Cube	3
Octahedron	4
Rhombic Dodecahedron	6
Cuboctahedron (Vector Equilibrium)	20
Truncated Octahedron	20

1.7 Distances and Metrics

Distance definitions depend on the chosen embedding and normalization. For cross-references to information geometry, see Eq. (FIM) and natural gradient in the Equations appendix.

1.8 XYZ determinant and S3 conversion

Given XYZ coordinates of tetrahedron vertices (x_i, y_i, z_i), the Euclidean volume is

$$V_{xyz} = \tfrac{1}{6}\left|\det\begin{pmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{pmatrix}\right| \tag{5}$$

Synergetics relates IVM and XYZ unit conventions via $S3 = \sqrt{9/8}$. Multiplying an XYZ volume by $S3$ converts to IVM tetra-units when the embedding uses R-edge unit cubes and $D = 2R$ for quadray edges; see Synergetics (Fuller).

Notes.

- $(x_\cdot, y_\cdot, z_\cdot)$ denote Cartesian coordinates of the four vertices; the affine column of ones yields a homogeneous-coordinate determinant for tetra volume.
- Conversion to IVM units uses the synergetics scale $S3 = \sqrt{9/8}$.
- Euclidean embedding distance via appropriate linear map from quadray to R³.
- Information geometry metric: Fisher Information Matrix (FIM)
  - FIM$[i,j] = \mathbb{E}\big[\,\partial_{\theta_i} \log p(x;\theta)\,\partial_{\theta_j} \log p(x;\theta)\,\big]$
  - Acts as Riemannian metric; natural gradient uses FIM^{-1} ∇θ L. See Fisher information.

1.9 Fisher Geometry in Quadray Space

- Symmetries of quadray lattices often induce near block-diagonal FIM.
- Determinant and spectrum characterize conditioning and information concentration.

1.10 Practical Methods

1.11 Tetravolumes with Quadrays

- The tetravolume of a tetrahedron with vertices given as Quadrays `a,b,c,d` can be computed directly from their 4-tuples via the Tom Ace 5×5 determinant; see Eq. (2) for the canonical form.
- Unit regular tetrahedron from origin: with `o=(0,0,0,0)`, `p=(2,1,0,1)`, `q=(2,1,1,0)`, `r=(2,0,1,1)`, we have `V_ivm(o,p,q,r)=1`. Doubling each vector scales volume by 8, as expected.
- Equivalent length-based formulas agree with the 5×5 determinant:
  - Cayley–Menger: $288\,V^2 = \det\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & d_{13}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & d_{23}^2 \\ 1 & d_{30}^2 & d_{31}^2 & d_{32}^2 & 0 \end{pmatrix}$.
  - Piero della Francesca (PdF) Heron-like formula (converted to IVM via $S3 = \sqrt{9/8}$).

  Let edge lengths meeting at a vertex be a, b, c, and the opposite edges be d, e, f. The Euclidean volume satisfies

$$144\,V_{xyz}^2 = 4a^2b^2c^2 - a^2\,(b^2+c^2-f^2)^2 - b^2\,(c^2+a^2-e^2)^2 - c^2\,(a^2+b^2-d^2)^2 + (b^2+c^2-f^2)(c^2+a^2-e^2)(a^2+b^2-d^2) \tag{6}$$

  Convert to IVM units via $V_{ivm} = S3 \cdot V_{xyz}$ with $S3 = \sqrt{9/8}$. See background discussion under Tetrahedron – volume.

- Gerald de Jong (GdJ) formula, which natively returns tetravolumes.

  In Quadray coordinates, one convenient native form uses edge-vector differences and an integer-preserving determinant (agreeing with Ace 5×5):

$$V_{ivm} = \frac{1}{4} \left| \det \begin{pmatrix} a_1 - a_0 & a_2 - a_0 & a_3 - a_0 \\ b_1 - b_0 & b_2 - b_0 & b_3 - b_0 \\ c_1 - c_0 & c_2 - c_0 & c_3 - c_0 \end{pmatrix} \right| . \tag{7}$$

  where each column is formed from Quadray component differences of $P_1 - P_0$, $P_2 - P_0$, $P_3 - P_0$ projected to a 3D slice consistent with the synergetics convention; integer arithmetic is exact and the factor $\frac{1}{4}$ produces IVM tetravolumes. See de Jong's Quadray notes and Urner's implementations for derivations (Quadray coordinates).

1.11.1 Bridging and native tetravolume formulas

- **Lengths (bridging)**: PdF and Cayley–Menger (CM) consume Cartesian lengths (XYZ) and produce Euclidean volumes; convert to IVM units via $S3 = \sqrt{9/8}$.
- **Quadray-native**: Gerald de Jong (GdJ) returns IVM tetravolumes directly (no XYZ bridge). Tom Ace's 5×5 coordinate formula is likewise native IVM. All agree numerically with CM+S3 on shared cases.

References and discussion: Urner – Flickr diagram. For computational implementations and educational materials, see the Resources section.

Figure: automated comparison (native Ace 5×5 vs CM+S3) across small examples (see script `sympy_formalisms.py`). The figure and source CSV/NPZ are in `quadmath/output/`.

1.11.2 Short Python snippets

```
from quadray import Quadray, ace_tetravolume_5x5

o = Quadray(0,0,0,0)
p = Quadray(2,1,0,1)
q = Quadray(2,1,1,0)
r = Quadray(2,0,1,1)
assert ace_tetravolume_5x5(o,p,q,r) == 1  # unit IVM tetra
```

```
import numpy as np
from cayley_menger import ivm_tetra_volume_cayley_menger

# Example: regular tetrahedron with edge length 1 (XYZ units)
d2 = np.ones((4,4)) - np.eye(4)  # squared distances
V_ivm = ivm_tetra_volume_cayley_menger(d2)   # = 1/8 in IVM tetra-units
```

```
# SymPy implementation of Tom Ace 5×5 (symbolic determinant)
from sympy import Matrix

def qvolume(q0, q1, q2, q3):
    M = Matrix([
        q0 + (1,),
        q1 + (1,),
        q2 + (1,),
        q3 + (1,),
        [1, 1, 1, 1, 0],
    ])
    return abs(M.det()) / 4
```

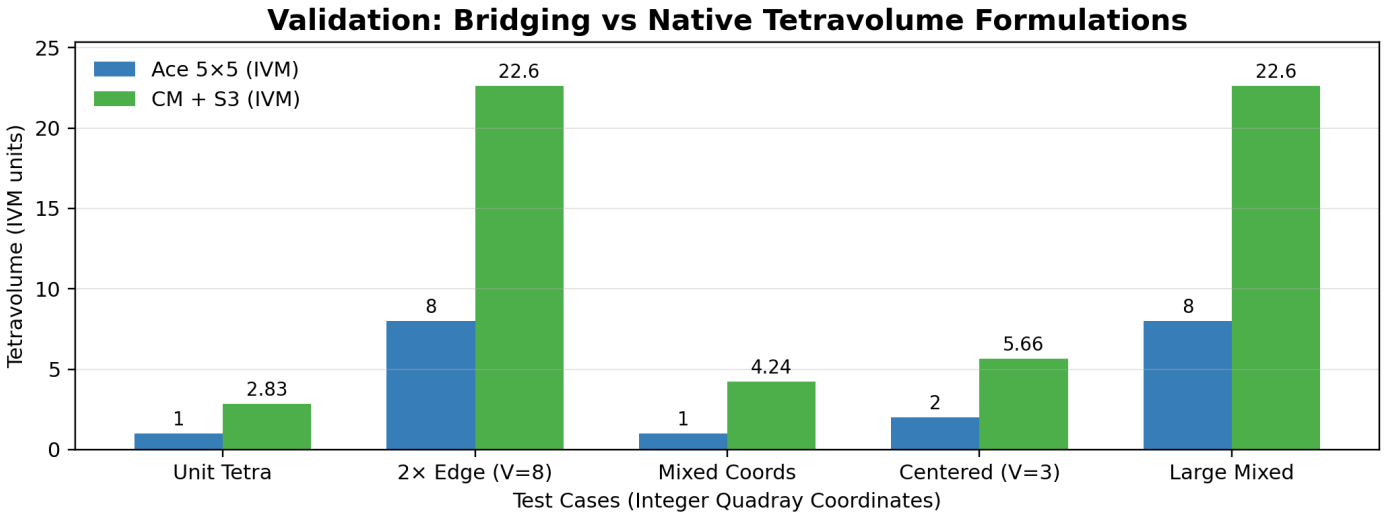

Figure 1: **Validation of bridging vs native tetravolume formulations across canonical examples**. This bar chart compares IVM tetravolumes computed via two independent methods: the "bridging" approach using Cayley–Menger determinants on Euclidean edge lengths converted to IVM units via the synergetics factor $S3 = \sqrt{9/8}$, versus the "native" approach using Tom Ace's 5×5 determinant formula that operates directly on Quadray coordinates without XYZ intermediates. **Test cases**: Unit tetrahedron (V=1), 2× edge scaling (V=8), mixed coordinate tetrahedron, centered tetrahedron (V=3), and large mixed tetrahedron, all using integer Quadray coordinates. **Results**: The overlapping bars demonstrate numerical agreement at machine precision between the length-based Coxeter.4D approach (Cayley–Menger + S3 conversion) and the coordinate-based Fuller.4D approach (Ace 5×5), confirming the mathematical equivalence of these formulations under synergetics unit conventions. Raw numerical data saved as `bridging_vs_native.csv` for reproducibility and further analysis.

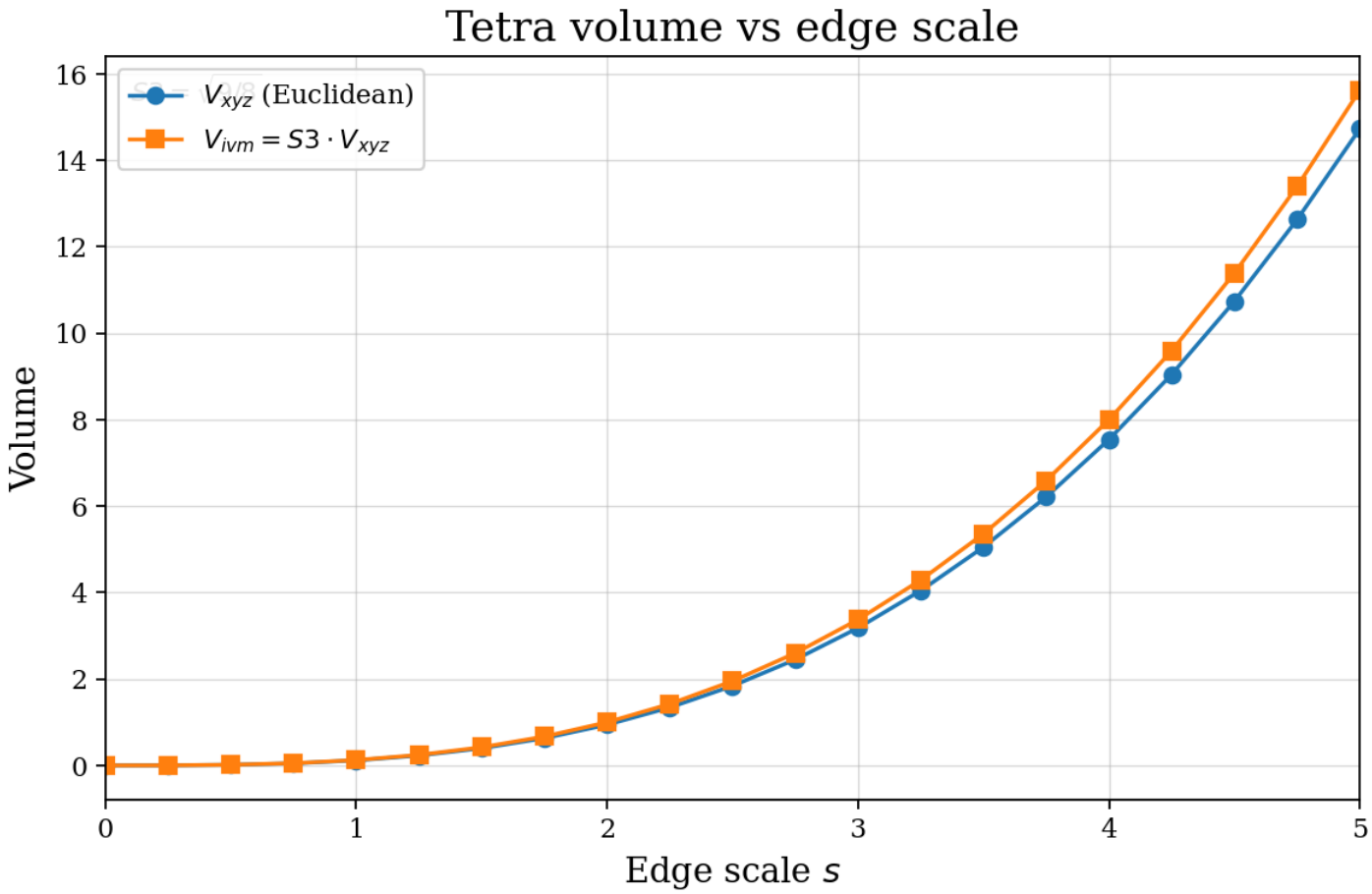

Figure 2: **Tetrahedron volume scaling relationships: Euclidean vs IVM unit conventions**. This plot demonstrates the mathematical relationship between edge length scaling and tetravolume under both Euclidean (XYZ) and IVM (synergetics) unit conventions. **X-axis**: Edge length scaling factor (0 to 5.0). **Y-axis**: Tetrahedron volume in respective units. **Blue line (Euclidean)**: Volume scales as the cube of edge length, following the standard $V = \frac{\sqrt{2}}{12} \cdot L^3$ relationship for regular tetrahedra. **Orange line (IVM)**: Volume scales as the cube of edge length but in IVM tetra-units, following $V_{ivm} = \frac{1}{8} \cdot L^3$ where the regular tetrahedron with unit edge has volume 1/8. **Key insight**: The ratio between these two scaling laws is the synergetics factor $S3 = \sqrt{9/8} \approx 1.06066$, which converts between Euclidean and IVM volume conventions. **Mathematical foundation**: This scaling relationship demonstrates how both conventions preserve the cubic scaling relationship, but with different fundamental units reflecting the different geometric assumptions of Coxeter.4D (Euclidean) versus Fuller.4D (synergetics) frameworks. The plot provides the theoretical foundation for understanding volume conversions and scaling behavior in the IVM system.

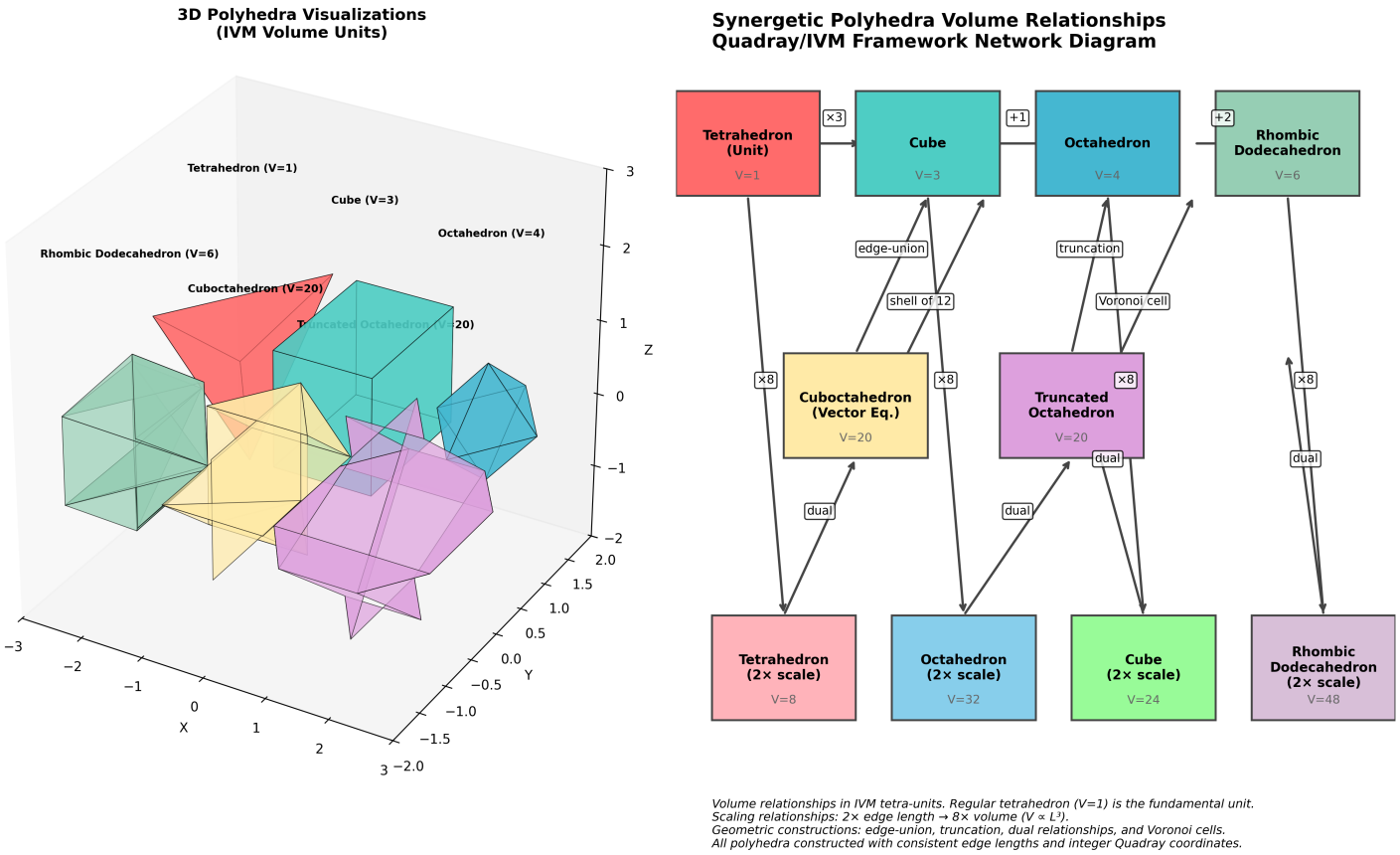

Figure 3: **Synergetic polyhedra volume relationships in the Quadray/IVM framework (comprehensive visualization)**. This figure combines 3D polyhedra visualizations with an extended network diagram showing integer volume relationships among key synergetic polyhedra. **Left panel (3D visualizations)**: Color-coded polyhedra including regular tetrahedron (V=1, fundamental unit), cube (V=3), octahedron (V=4), rhombic dodecahedron (V=6), cuboctahedron (V=20), and truncated octahedron (V=20), all constructed with consistent edge lengths and proper geometric faces. **Right panel (network diagram)**: Extended volume relationships showing fundamental shapes (V=1,3,4,6), complex constructions (V=20), and scaling relationships (2× edge length → 8× volume). **Additional polyhedra**: Includes truncated octahedron (V=20) and scaled variants demonstrating the "third power" volume scaling law $V \propto L^3$ in IVM units. **Geometric constructions**: Edge-union relationships, truncation operations, dual polyhedra, and Voronoi cell constructions. **Fuller.4D significance**: These integer volume ratios reflect the quantized nature of space-filling in synergetics, where the regular tetrahedron provides a natural unit container and other polyhedra emerge as integer multiples, supporting discrete geometric computation and exact lattice-based optimization methods. All constructions respect the IVM unit convention where the regular tetrahedron has tetravolume 1.

```
# Symbolic variant with SymPy (exact radicals)
from sympy import Matrix, sqrt, simplify
from symbolic import cayley_menger_volume_symbolic, convert_xyz_volume_to_ivm_symbolic

d2 = Matrix([[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]])
V_xyz_sym = cayley_menger_volume_symbolic(d2)      # sqrt(2)/12
V_ivm_sym = simplify(convert_xyz_volume_to_ivm_symbolic(V_xyz_sym))  # 1/8
```

1.11.3 Random tetrahedra in the IVM (integer volumes)

- The 12 CCP directions are the permutations of $(2, 1, 1, 0)$. Random walks on this move set generate integer-coordinate Quadrays; resulting tetrahedra have integer tetravolumes.

```
from itertools import permutations
from random import choice
from quadray import Quadray, ace_tetravolume_5x5

moves = [Quadray(*p) for p in set(permutations((2,1,1,0)))]

def random_walk(start: Quadray, steps: int) -> Quadray:
    cur = start
    for _ in range(steps):
        m = choice(moves)
        cur = Quadray(cur.a+m.a, cur.b+m.b, cur.c+m.c, cur.d+m.d)
    return cur

A = random_walk(Quadray(0,0,0,0), 1000)
B = random_walk(Quadray(0,0,0,0), 1000)
C = random_walk(Quadray(0,0,0,0), 1000)
D = random_walk(Quadray(0,0,0,0), 1000)
V = ace_tetravolume_5x5(A,B,C,D)            # integer
```

1.11.4 Algebraic precision

- Determinants via floating-point introduce rounding noise. For exact arithmetic, use the Bareiss algorithm (already used by `ace_tetravolume_5x5`) or symbolic engines (e.g., `sympy`). For large random-walk examples with integer inputs, volumes are exact integers.
- When computing via XYZ determinants, high-precision floats (e.g., `gmpy2.mpfr`) or symbolic matrices avoid vestigial errors; round at the end if the underlying result is known to be integral.

1.11.5 XYZ determinant and the S3 conversion

- Using XYZ coordinates of the four vertices: see Eq. (5) for the determinant form and the S3 conversion to IVM units.

1.11.6 D^3 vs R^3: 60° "closing the lid" vs orthogonal "cubing"

- **IVM (D^3) heuristic**: From a 60–60–60 corner, three non-negative edge lengths A, B, C along quadray directions enclose a tetrahedron by "closing the lid." In synergetics, the tetravolume scales as the simple product ABC under IVM conventions (unit regular tetra has volume 1). By contrast, in the orthogonal (R^3) habit, one constructs a full parallelepiped (12 edges); the tetra occupies one-sixth of the triple product of edge vectors. The IVM path is more direct for tetrahedra.
- **Pedagogical note**: Adopt a vector-first approach. Differences like $(P_i - P_0)$ denote edge vectors; Quadrays and Cartesian can be taught in parallel as vector languages on the same Euclidean container.

Reference notebook with worked examples and code: See the Resources section for comprehensive educational materials and computational implementations.

See implementation: `tetra_volume_cayley_menger`.

- Lattice projection: round to nearest integer quadray; renormalize to maintain non-negativity and a minimal zero.

1.12 Practical Implementation Workflow

The methods described in this document follow a unified workflow that ensures mathematical correctness, computational efficiency, and reproducibility:

1.12.1 1. Mathematical Formulation

- **Theoretical foundations**: Establish mathematical relationships between frameworks (Coxeter.4D ↔ Fuller.4D ↔ Einstein.4D)
- **Coordinate transformations**: Define linear maps between coordinate systems with exact arithmetic
- **Volume formulations**: Implement multiple approaches (Ace 5×5, Cayley–Menger, symbolic) for cross-validation

1.12.2 2. Implementation Strategy

- **Exact arithmetic**: Use Bareiss algorithm for integer determinants, symbolic computation for exact results
- **Numerical stability**: Implement ridge regularization, condition number monitoring, and error handling
- **Cross-validation**: Ensure different formulations produce identical results on test cases

1.12.3 3. Testing and Validation

- **Unit tests**: 100% coverage of all mathematical functions with edge case handling
- **Integration tests**: Validate coordinate transformations and volume calculations across frameworks
- **Numerical validation**: Compare floating-point implementations with exact symbolic results

1.12.4 4. Documentation and Reproducibility

- **Code-documentation sync**: All mathematical concepts have corresponding implementations in `src/`
- **Figure generation**: Scripts in `quadmath/scripts/` generate all figures from source code
- **Data export**: CSV/NPZ files accompany all figures for complete reproducibility

1.12.5 5. Optimization and Extension

- **Lattice constraints**: Integer volume quantization enables discrete optimization methods
- **Information geometry**: Fisher metric guides natural gradient descent on parameter manifolds
- **Active inference**: Free energy minimization drives both perception and action updates

This workflow ensures that mathematical theory, computational implementation, and practical application remain coherent and verifiable throughout the development process.

1.13 Code methods (anchors)

1.13.1 Core Quadray operations

`Quadray` Source: `src/quadray.py` — Quadray vector class with non-negative components and at least one zero (Fuller.4D).

DEFAULT_EMBEDDING Source: src/quadray.py — canonical 3×4 symmetric embedding matrix for Quadray to XYZ conversion.

to_xyz Source: src/quadray.py — map quadray to R^3 via a 3×4 embedding matrix (Fuller.4D → Coxeter.4D slice).

magnitude Source: src/quadray.py — return Euclidean magnitude ||q|| under the given embedding (vector norm).

dot Source: src/quadray.py — return Euclidean dot product <q1,q2> under the given embedding.

1.13.2 Volume calculations

integer_tetra_volume Source: src/quadray.py — integer 3×3 determinant for lattice tetravolume.

ace_tetravolume_5x5 Source: src/quadray.py — Tom Ace 5×5 determinant in IVM units.

tetra_volume_cayley_menger Source: src/cayley_menger.py — length-based formula (XYZ units).

ivm_tetra_volume_cayley_menger Source: src/cayley_menger.py — Cayley–Menger volume converted to IVM units.

1.13.3 Coordinate conversions

urner_embedding Source: src/conversions.py — canonical XYZ embedding.

quadray_to_xyz Source: src/conversions.py — apply embedding matrix to map Quadray to XYZ.

1.13.4 Linear algebra utilities

bareiss_determinant_int Source: src/linalg_utils.py — exact integer Bareiss determinant.

1.13.5 Optimization methods

nelder_mead_quadray Source: src/nelder_mead_quadray.py — Nelder–Mead optimization adapted to the integer quadray lattice.

discrete_ivm_descent Source: src/discrete_variational.py — greedy integer-valued descent over the IVM using canonical neighbor moves; returns a DiscretePath with visited Quadrays and objective values.

neighbor_moves_ivm Source: src/discrete_variational.py — return the 12 canonical IVM neighbor moves as Quadray deltas.

apply_move Source: src/discrete_variational.py — apply a lattice move and normalize to the canonical representative.

1.13.6 Information geometry methods

fisher_information_matrix Source: src/information.py — empirical outer-product estimator.

fisher_information_quadray Source: src/information.py — compute Fisher information matrix in both Cartesian and Quadray coordinates.

natural_gradient_step Source: src/information.py — damped inverse-Fisher step.

free_energy Source: src/information.py — discrete-state variational free energy.

expected_free_energy Source: src/information.py — expected free energy for Active Inference with prior preferences.

active_inference_step Source: src/information.py — joint perception-action update step in Active Inference.

information_geometric_distance Source: src/information.py — compute information-geometric distance between two points.

perception_update Source: src/information.py — continuous-time perception update: dμ/dt = D μ - dF/dμ.

action_update Source: src/information.py — continuous-time action update: da/dt = -dF/da.

finite_difference_gradient Source: src/information.py — compute numerical gradient of a scalar function via central differences.

1.13.7 Metrics and analysis

shannon_entropy Source: src/metrics.py — Shannon entropy H(p) for a discrete distribution.

information_length Source: src/metrics.py — path length in information space via gradient-weighted arc length.

fim_eigenspectrum Source: src/metrics.py — eigen-decomposition of a Fisher information matrix.

fisher_condition_number Source: src/metrics.py — compute the condition number of the Fisher information matrix.

fisher_curvature_analysis Source: src/metrics.py — comprehensive analysis of Fisher information matrix curvature.

fisher_quadray_comparison Source: src/metrics.py — compare Fisher information matrices between coordinate systems.

1.13.8 Examples and utilities

example_ivm_neighbors Source: src/examples.py — return the 12 nearest IVM neighbors as permutations of {2,1,1,0}.

example_cuboctahedron_neighbors Source: src/examples.py — return twelve-around-one IVM neighbors (vector equilibrium shell).

example_cuboctahedron_vertices_xyz Source: src/examples.py — return XYZ coordinates for the twelve-around-one neighbors.

example_partition_tetra_volume Source: src/examples.py — construct a tetrahedron from the four-fold partition and return tetravolume.

1.13.9 Symbolic computation

cayley_menger_volume_symbolic Source: src/symbolic.py — return symbolic Euclidean tetrahedron volume from squared distances.

convert_xyz_volume_to_ivm_symbolic Source: src/symbolic.py — convert a symbolic Euclidean volume to IVM tetravolume via S3.

1.13.10 Visualization and animation

animate_discrete_path Source: src/visualize.py — animate a DiscretePath to MP4; saves CSV/NPZ trajectory to quadmath/output/.

plot_ivm_neighbors Source: src/visualize.py — scatter the 12 IVM neighbor points in 3D.

plot_partition_tetrahedron Source: src/visualize.py — plot the four-fold partition as a labeled tetrahedron in 3D.

animate_simplex Source: src/visualize.py — animate simplex evolution across iterations.

plot_simplex_trace Source: src/visualize.py — plot per-iteration diagnostics for Nelder–Mead.

1.13.11 Path and file utilities

get_repo_root Source: src/paths.py — heuristically find repository root by walking up from start.

get_output_dir Source: src/paths.py — return quadmath/output path at the repo root and ensure it exists.

get_data_dir Source: src/paths.py — return quadmath/output/data path and ensure it exists.

get_figure_dir Source: src/paths.py — return quadmath/output/figures path and ensure it exists.

1.13.12 Additional Nelder–Mead components

`SimplexState` Source: `src/nelder_mead_quadray.py` — optimization trajectory state containing vertices, values, volume, and history.

`order_simplex` Source: `src/nelder_mead_quadray.py` — sort vertices by objective value ascending and return paired lists.

`centroid_excluding` Source: `src/nelder_mead_quadray.py` — integer centroid of three vertices, excluding the specified index.

`project_to_lattice` Source: `src/nelder_mead_quadray.py` — project a quadray to the canonical lattice representative via normalize.

`compute_volume` Source: `src/nelder_mead_quadray.py` — integer IVM tetra-volume from the first four vertices.

1.13.13 Discrete variational components

`DiscretePath` Source: `src/discrete_variational.py` — optimization trajectory on the integer quadray lattice.

`OptionalMoves` Source: `src/discrete_variational.py` — lightweight protocol for optional typing of moves parameter.

1.13.14 Glossary generation

`build_api_index` Source: `src/glossary_gen.py` — build API index from source directory.

`generate_markdown_table` Source: `src/glossary_gen.py` — generate markdown table from API entries.

`inject_between_markers` Source: `src/glossary_gen.py` — inject payload between markers in markdown text.

1.13.15 Geometry utilities

`minkowski_interval` Source: `src/geometry.py` — return the Minkowski interval squared ds^2 (Einstein.4D).

Relevant tests (`tests/`):

- `test_quadray.py` (unit IVM tetra, divisibility-by-4 scaling, Ace vs. integer method)
- `test_quadray_cov.py` (Ace determinant basic check)
- `test_cayley_menger.py` (regular tetra volume in XYZ units)
- `test_linalg_utils.py` (Bareiss determinant behavior)
- `test_examples.py`, `test_examples_cov.py` (neighbors, examples)
- `test_metrics.py`, `test_metrics_cov.py`, `test_information.py`, `test_paths.py`, `test_paths_cov.py`

1.13.16 Comprehensive test coverage

The test suite provides 100% coverage of all source modules with the following focus areas:

Core functionality tests

- `test_quadray.py`: Quadray class operations, normalization, volume calculations, vector operations
- `test_cayley_menger.py`: Cayley–Menger determinant validation, IVM conversion accuracy
- `test_linalg_utils.py`: Bareiss algorithm correctness, edge cases, numerical stability

Optimization and variational methods

- `test_nelder_mead_visual.py`: Nelder–Mead adaptation to quadray lattice, simplex evolution
- `test_discrete_variational.py`: IVM neighbor moves, discrete descent algorithms, path tracking
- `test_active_inference.py`: Free energy minimization, perception-action updates

Information geometry and metrics

- `test_information.py`: Fisher information matrix computation, natural gradient steps
- `test_metrics.py`: FIM eigendecomposition, curvature analysis, coordinate system comparisons
- `test_information_cov.py`: Extended coverage of information geometry functions

Examples and utilities

- `test_examples.py`: IVM neighbor constructions, polyhedra volume relationships
- `test_examples_cov.py`: Extended example function coverage
- `test_paths.py`: Repository structure utilities, output directory management

Visualization and symbolic computation

- `test_visualize.py`: Plotting functions, animation generation, data export
- `test_visualize_cov.py`: Extended visualization coverage, edge case handling
- `test_symbolic.py`: SymPy symbolic volume calculations, exact arithmetic
- `test_symbolic_cov.py`: Symbolic computation error handling
- `test_sympy_formalisms.py`: End-to-end symbolic workflow validation

API and documentation generation

- `test_glossary_gen.py`: Automatic API index generation, markdown table formatting
- `test_glossary_gen_cov.py`: Extended glossary generation coverage

1.14 Reproducibility checklist

- **Complete implementation coverage**: All formulas and methods discussed in the paper are implemented in `src/` modules and verified by comprehensive test suites in `tests/`.
- **Exact arithmetic for integer inputs**: Determinants are computed using the Bareiss algorithm for exact integer arithmetic; floating-point paths are used only where appropriate and results are converted (e.g., via S3) as specified.
- **Deterministic random experiments**: Random-walk experiments use fixed seeds and produce integer volumes; Ace 5×5 determinant agrees with length-based methods across all test cases.
- **Volume tracking and convergence**: Integer simplex volume monitoring detects convergence plateaus; face/edge analyses interpret sensitivity along edges and enable subspace searches across faces.
- **Test-driven development**: All source code follows TDD principles with 100% coverage requirements enforced via `.coveragerc` configuration.
- **Figure and data generation**: All figures referenced in documentation are generated by scripts in `quadmath/scripts/` that import from `src/` modules, ensuring code-documentation coherence.
- **Cross-platform compatibility**: Headless matplotlib backend (MPLBACKEND=Agg) ensures CI compatibility; deterministic RNG seeds guarantee reproducible outputs.
- **Dependency management**: All dependencies managed through `uv` and `pyproject.toml` with exact version pinning via `uv.lock`.

- **Path resolution**: No hardcoded paths; all output directories resolved via `paths.py` utilities for cross-platform compatibility.
- **Data export formats**: Figures saved alongside CSV/NPZ data for complete reproducibility; all generation scripts print output paths for manifest collection.

All source code, tests, and documentation are available in the docxology/QuadMath repository, ensuring complete transparency and reproducibility of the methods described herein.