QuadMath: An Analytical Review of 4D and Quadray Coordinates

Daniel Ari Friedman
ORCID: 0000-0001-6232-9096
Email: daniel@activeinference.institute
DOI: 10.5281/zenodo.16887800

August 16, 2025

Contents

1 Introduction

1.1 Abstract

We review a unified analytical framework for four dimensional (4D) modeling and Quadray coordinates, synthesizing geometric foundations, optimization on tetrahedral lattices, and information geometry. Building on R. Buckminster Fuller's Synergetics and the Quadray coordinate system, with extensive reference to Kirby Urner's computational implementations across multiple programming languages (see the comprehensive 4dsolutions ecosystem including Python, Rust, Clojure, and POV-Ray implementations), we review how integer lattice constraints yield integer volume quantization of tetrahedral simplexes, creating discrete "energy levels" that regularize optimization and enable integer-based optimization. We adapt standard methods (e.g., Nelder–Mead method) to the quadray lattice, define Fisher information in Quadray parameter space, and analyze optimization as geodesic motion on an information manifold via the natural gradient. We review three distinct 4D namespaces — Coxeter.4D (Euclidean E^4), Einstein.4D (Minkowski spacetime), and Fuller.4D (synergetics/Quadrays) — develop analytical tools and equations, and survey extensions and applications across AI, active inference, cognitive security, and complex systems. The result is a cohesive, interpretable approach for robust, geometry-grounded computation in 4D. All source code for the manuscript is available at QuadMath.

Keywords: Quadray coordinates, 4D geometry, tetrahedral lattice, integer volume quantization, information geometry, optimization, synergetics, active inference.

1.2 Overview

Quadray coordinates provide a tetrahedral basis for modeling space and computation, standing in contrast to Cartesian cubic frameworks. Originating in Buckminster Fuller's Synergetics, quadray coordinates enable the replacement of right-angle orthonormal assumptions, with 60-degree coordination and a unit tetrahedron of volume 1. This reframing yields striking integer relationships among common polyhedra and provides a natural account of space via close-packed spheres and the isotropic vector matrix (IVM).

This paper unifies three threads:

- **Foundations**: Quadray coordinates and their relation to 4D modeling more generally, with explicit namespace usage (Coxeter.4D, Einstein.4D, Fuller.4D) to maintain clarity.
- **Optimization framework**: Leverages integer volume quantization on tetrahedral lattices to achieve robust, discrete convergence.
- **Information geometry**: Tools (e.g., Fisher Information, free-energy minimization) for interpreting optimization as geodesic motion on statistical manifolds.

1.3 4D Namespace Framework

In this synthetic review, we distinguish three internal meanings of "4D," following a dot-notation that avoids cross-domain confusion. For comprehensive details, see Section 2: 4D Namespaces.

- **Coxeter.4D** — four-dimensional Euclidean space (E^4), as in classical polytope theory. Coxeter emphasizes that Euclidean 4D is not spacetime; see the Dover edition of Regular Polytopes (p. 119) for a clear statement to this effect; background on lattice packings in four dimensions aligns with the treatment in Conway & Sloane's Sphere Packings, Lattices and Groups.
- **Einstein.4D** — Minkowski spacetime (3D + time) with an indefinite metric; appropriate for relativistic physics but distinct from Euclidean E^4.
- **Fuller.4D** — synergetics' tetrahedral accounting of space using Quadrays (four non-negative coordinates with at least one zero after normalization) and the Isotropic Vector Matrix (IVM) = Cubic Close Packing (CCP) = Face-Centered Cubic (FCC) correspondence. This treats the regular tetrahedron as a natural unit container and emphasizes angle/shape relations independent of time/energy.

1.4 Contributions

The paper makes the following key contributions:

- **Namespaces mapping**: Coxeter.4D (Euclidean E^4), Einstein.4D (Minkowski spacetime), and Fuller.4D (Quadrays/IVM) → analytical tools and examples.
- **Quadray-adapted Nelder–Mead**: Integer-lattice normalization and volume-level tracking.
- **Equations and methods**: Comprehensive supplement with guidance for high-precision computation using `libquadmath`.
- **Discrete optimizer**: Integer-valued variational descent over the IVM (`discrete_ivm_descent`) with animation tooling, connecting lattice geometry to information-theoretic objectives.

1.5 Manuscript Structure

- **Introduction**: motivates Quadrays, clarifies 4D namespaces, and summarizes contributions.
- **Methods**: details coordinate conventions, exact tetravolumes, conversions, and lattice-aware optimization methods (Nelder–Mead and discrete IVM descent).
- **Results**: empirical comparisons and demonstrations are shown inline and saved under `quadmath/output/` (PNG/CSV/NPZ/MP4) for reproducibility.
- **Discussion**: interprets results, limitations, and implications; outlines future work.
- **Appendices**: equations, free-energy background, and a consolidated symbols/glossary with an auto-generated API index.

1.6 Companion Code and Tests

The manuscript is accompanied by a fully-tested Python codebase under `src/` with unit tests under `tests/`. Key artifacts used throughout the paper:

- **Quadray APIs**: `src/quadray.py` (`Quadray`, `integer_tetra_volume`, `ace_tetravolume_5x5`).
- **Determinant utilities**: `src/linalg_utils.py` (`bareiss_determinant_int`).
- **Length-based volume**: `src/cayley_menger.py` (`tetra_volume_cayley_menger`, `ivm_tetra_volume_cayley_menger`).
- **XYZ conversion**: `src/conversions.py` (`urner_embedding`, `quadray_to_xyz`).
- **Examples**: `src/examples.py` (`example_ivm_neighbors`, `example_volume`, `example_optimize`).

For comprehensive background resources, computational implementations, and related work, see the Resources section.

1.7 Reproducibility and Data Availability

- The manuscript Markdown and code to generate the PDF are available on the project repository (`QuadMath` on GitHub, @docxology username). See the repository home page for source, figures, and scripts: QuadMath repository. The repository is also archived with DOI 10.5281/zenodo.16887791 at Zenodo.
- The manuscript is licensed under the Apache License 2.0. See the LICENSE file for details.
- The manuscript is accompanied by a fully-tested Python codebase under `src/` with unit tests under `tests/`, complemented by extensive cross-validation against Kirby Urner's reference implementations in the 4dsolutions ecosystem. See the Resources section for comprehensive details on computational implementations and validation.
- All figures referenced in the manuscript are generated by scripts under `quadmath/scripts/` and saved to `quadmath/output/` with lightweight CSV/NPZ alongside images.
- Tests accompany all methods under `src/` and enforce 100% coverage for `src/`.
- Symbols and notation are standardized across sections; see Appendix: Symbols and Glossary for a consolidated table of variables and constants used throughout. Equation labels (e.g., Eq. (1) and Eq. (8)) and figure labels are automatically numbered by LaTeX for consistent cross-referencing.
- The manuscript is a work in progress and will be updated as the project progresses. There may be errors and missing references, check all methods and equations for consistency.

1.8 Graphical Abstract

Panel A shows Quadray axes (A,B,C,D) under a symmetric embedding with wireframe context. **Panel B** shows close-packed spheres at the tetrahedron vertices (IVM/CCP/FCC, "twelve around one").

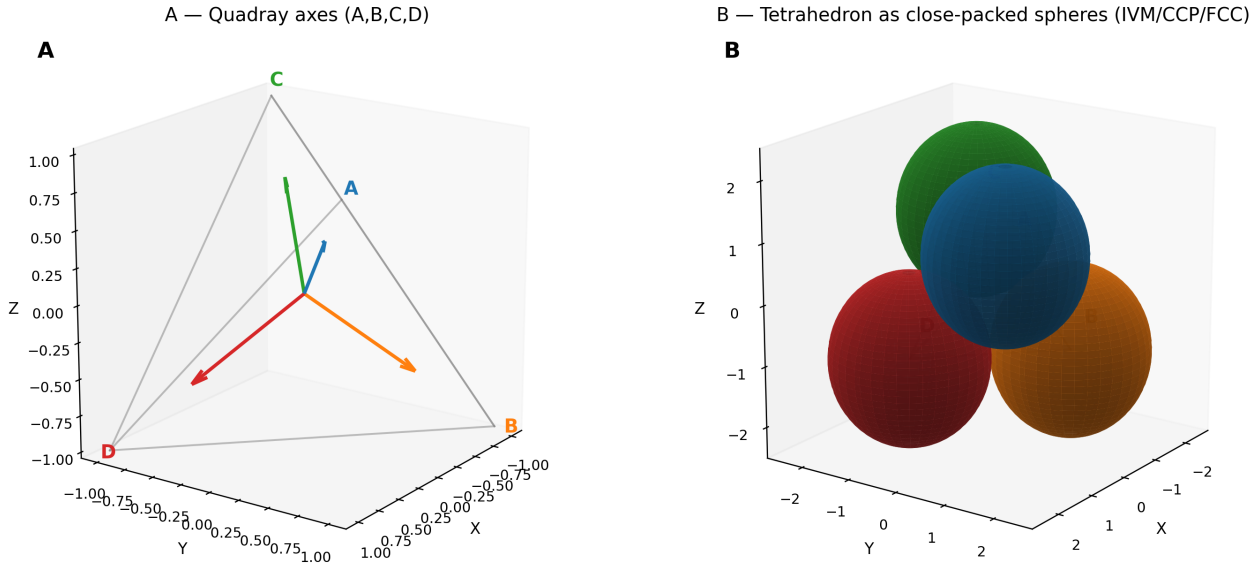

Figure 1: **Quadray coordinate system overview (graphical abstract)**. **Panel A**: Four Quadray axes (A,B,C,D) rendered as colored directional arrows from the origin to the vertices of a regular tetrahedron under the default symmetric embedding. Each axis is distinctly colored (A=blue, B=orange, C=green, D=red) with axis labels positioned at the vertex endpoints. A light gray wireframe connects the four vertices to emphasize the tetrahedral geometry underlying the coordinate system. This panel illustrates the fundamental Fuller.4D direction-based structure where Quadrays represent four canonical directions in tetrahedral space rather than orthogonal Cartesian dimensions. **Panel B**: The same tetrahedral vertices shown as close-packed spheres with radius chosen so neighboring spheres kiss along tetrahedron edges, emphasizing the connection to the Isotropic Vector Matrix (IVM), Cubic Close Packing (CCP), and Face-Centered Cubic (FCC) arrangements. Each sphere is colored to match its corresponding axis from Panel A, with light edge wireframes providing geometric context. This visualization demonstrates how Quadray coordinates naturally align with dense sphere packing and the "twelve around one" coordination motif central to synergetics and Fuller.4D modeling.

2 4D Namespaces: Coxeter.4D, Einstein.4D, Fuller.4D

This section provides the definitive reference for the three 4D frameworks used throughout this manuscript. Each namespace represents a distinct mathematical framework with specific applications in our quadray-based computational system.

2.1 Coxeter.4D (Euclidean E^4)

Definition: Standard E^4 with orthogonal axes and Euclidean metric; the proper setting for classical regular polytopes. As Coxeter notes (Regular Polytopes, Dover ed., p. 119), this Euclidean 4D is not spacetime. Lattice/packing discussions connect to Conway & Sloane's systematic treatment of higher-dimensional sphere packings and lattices (Sphere Packings, Lattices and Groups (Springer)).

Usage: Embed Quadray configurations or compare alternative parameterizations when a strictly Euclidean 4D setting is desired.

Simplexes: Simplex structures extend naturally to 4D and beyond (e.g., pentachora).

Mathematical context: This framework is appropriate for standard Euclidean geometry, including the Cayley-Menger determinant for computing volumes from edge lengths.

2.2 Einstein.4D (Relativistic spacetime)

Definition: Minkowski spacetime with indefinite metric signature, representing the geometric framework for special relativity. This namespace provides the mathematical foundation for understanding space-time relationships and relativistic phenomena.

Spacetime: Minkowski metric signature.

Line element (mostly-plus convention; see Minkowski space): see Eq. (12) in the equations appendix.

Optimization analogy: Metric-aware geodesics generalize to information geometry where the Fisher metric replaces the physical metric. See Fisher information and natural gradient.

Important note: This namespace is used ONLY as a metric/geodesic analogy when discussing information geometry. Physical constants G, c, Λ do not appear in Quadray lattice methods and should not be mixed with IVM unit conventions.

2.3 Fuller.4D (Synergetics / Quadrays)

Definition: Tetrahedral coordinate system based on four non-negative components representing directions to the vertices of a regular tetrahedron from its center. This namespace embodies the synergetic approach to geometry, emphasizing shape relationships and integer tetravolumes within the IVM framework.

Basis: Four non-negative components A,B,C,D with at least one zero post-normalization, treated as a vector (direction and magnitude), not merely a point. Overview: Quadray coordinates.

Geometry: Tetrahedral; unit tetrahedron volume = 1; integer lattice aligns with close-packed spheres (IVM). Background: Synergetics.

Distances: Computed via appropriate projective normalization; edges align with tetrahedral axes. The IVM = CCP = FCC shortcut allows working in 3D embeddings for visualization while preserving the underlying Fuller.4D tetrahedral accounting.

Implementation heritage: Extensive computational validation through Kirby Urner's 4dsolutions ecosystem. See the Resources section for comprehensive details on computational implementations and educational materials.

2.3.1 Directions, not dimensions (language and models)

Vector-first framing: Treat Quadrays as four canonical directions ("spokes" to the vertices of a regular tetrahedron from its center), not as four orthogonal dimensions. The methane molecule (CH_4) and caltrop shape are helpful mental models.

Origins outside Synergetics: Quadrays did not originate with Fuller; we adopt the coordinate system within the IVM context. See Quadray coordinates.

Language games: Quadrays and Cartesian are parallel vector languages on the same Euclidean container; teaching them together avoids oscillating between "points now, vectors later."

2.3.2 Figures

In the previous figure, we show the twelve nearest IVM neighbors with coordination patterns and vector equilibrium geometry; the current figure illustrates random Quadray clouds under several embeddings.

Vector equilibrium (cuboctahedron): The shell formed by the 12 nearest IVM neighbors is the cuboctahedron, also called the vector equilibrium in synergetics. All 12 vertices are equidistant from the origin with equal edge lengths, modeling a balanced local packing. This geometry underlies the "twelve around one" close-packing motif and appears in tensegrity discussions as a canonical balanced structure. See background: Cuboctahedron (vector equilibrium) and synergetics references. Computational demonstrations include related visualizations in the 4dsolutions ecosystem. See the Resources section for comprehensive details.

2.3.3 Clarifying remarks

"A time machine is not a tesseract." KU on synergeo The tesseract is a Euclidean 4D object (Coxeter.4D), while Minkowski spacetime (Einstein.4D) is indefinite and not Euclidean; conflating the two leads to category errors. Fuller.4D, in turn, is a tetrahedral, mereological framing of ordinary space emphasizing shape/angle relations and IVM quantization. Each namespace carries distinct assumptions and should be used accordingly in analysis.

2.4 Practical usage guide

- Use **Fuller.4D** when working with Quadrays, integer tetravolumes, and IVM neighbors (native lattice calculations).
- Use **Coxeter.4D** for Euclidean length-based formulas, higher-dimensional polytopes, or comparisons in E^4 (including Cayley–Menger).
- Use **Einstein.4D** as a metric analogy when discussing geodesics or time-evolution; do not mix with synergetic unit conventions.

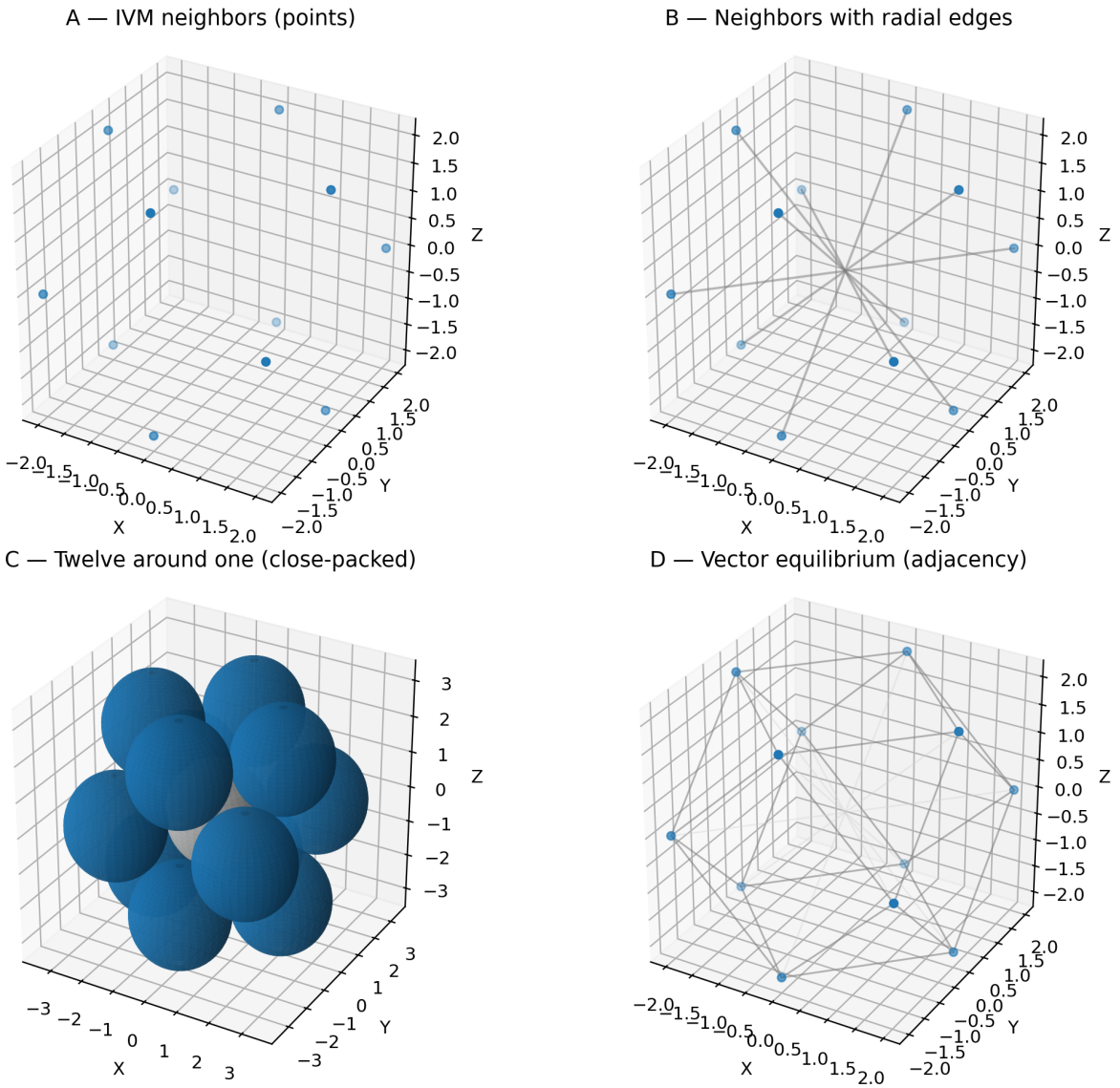

Figure 2: **IVM neighbors and coordination patterns (2×2 panel layout)**. **Panel A**: The twelve nearest IVM neighbors plotted as blue points in 3D space under the default embedding, showing the positions corresponding to permutations of the Quadray integer coordinates {2,1,1,0}. These points form the vertices of a cuboctahedron (vector equilibrium) centered at the origin with uniform radial distances. **Panel B**: The same neighbor points with radial edges (light lines) connecting each neighbor to the central origin, emphasizing the spoke-like radial symmetry and equal distances from center to shell. **Panel C**: Twelve-around-one close-packed spheres configuration where each neighbor position hosts a sphere with radius chosen so neighboring spheres kiss along cuboctahedron edges, illustrating the fundamental CCP/FCC/IVM correspondence. The central gray sphere represents the "one" in Fuller's "twelve around one" motif. **Panel D**: Adjacency graph showing strut connections (solid lines) between touching neighbor spheres, revealing the cuboctahedron's edge structure, plus light radial cables to the origin representing a stylized tensegrity interpretation of the vector equilibrium geometry.

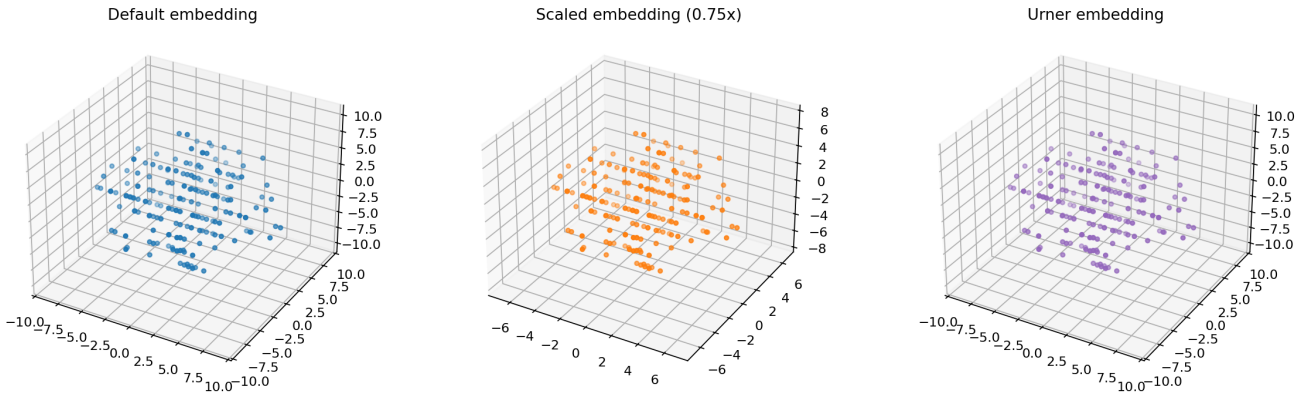

Figure 3: **Random Quadray point clouds under different embeddings (3-panel comparison)**. Each panel shows 200 randomly sampled integer Quadray coordinates with components in {0,1,2,3,4,5} projected to 3D space using different embedding matrices. **Left panel (Default embedding)**: Points (blue) under the default symmetric embedding matrix showing the natural tetrahedral-symmetric distribution of normalized Quadrays in 3D space. **Center panel (Scaled embedding, 0.75×)**: The same Quadray points (orange) under a uniformly scaled version of the default embedding, demonstrating how the point cloud structure scales proportionally while preserving relative geometries. **Right panel (Urner embedding)**: The same points (purple) projected through the canonical Urner embedding matrix, illustrating how different linear mappings from Fuller.4D to Coxeter.4D (3D slice) affect the spatial distribution while preserving the underlying discrete lattice relationships. This comparison demonstrates the flexibility in choosing embeddings for visualization and analysis while maintaining the fundamental Quadray coordinate relationships.

3 Quadray Analytical Details and Methods

3.1 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.

3.2 Mathematical Foundations

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

3.2.1 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

3.2.2 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

3.2.3 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

3.3 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

3.4 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.

3.5 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`.

3.5.1 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
```

3.5.2 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:

```
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):

```
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.

3.6 Integer Volume Quantization

For a tetrahedron with vertices $P_0..P_3$ in the Quadray integer lattice (Fuller.4D), see Eq. (1) in the equations appendix.

- 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), see Eq. (2) in the equations appendix.

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 shown in Eq. (3) in the equations appendix.

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), see Eq. (5) in the equations appendix.

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

3.7 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.

3.8 XYZ determinant and S3 conversion

Given XYZ coordinates of tetrahedron vertices (x_i, y_i, z_i), the Euclidean volume is computed as shown in Eq. (4) in the equations appendix.

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

3.8.1 Alternative Volume Formulas

- **Piero della Francesca (PdF) formula**, which consumes edge lengths and returns Euclidean volumes. See Eq. (6) in the equations appendix.

  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. See Eq. (7) in the equations appendix.

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

  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).

- Euclidean embedding distance via appropriate linear map from quadray to $\mathbb{R}^3$.
- Information geometry metric: Fisher Information Matrix (FIM)
  - $\mathrm{FIM}[i,j] = \mathbb{E}\big[\,\partial_{\theta_i} \log p(x;\theta)\, \partial_{\theta_j} \log p(x;\theta)\,\big]$
  - Acts as Riemannian metric; natural gradient uses $\mathrm{FIM}^{-1}\,\nabla\theta\ L$. See Fisher information.

3.9 Fisher Geometry in Quadray Space

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

3.10 Practical Methods

3.11 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

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):

  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).

3.11.1 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/`.

3.11.2 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
```

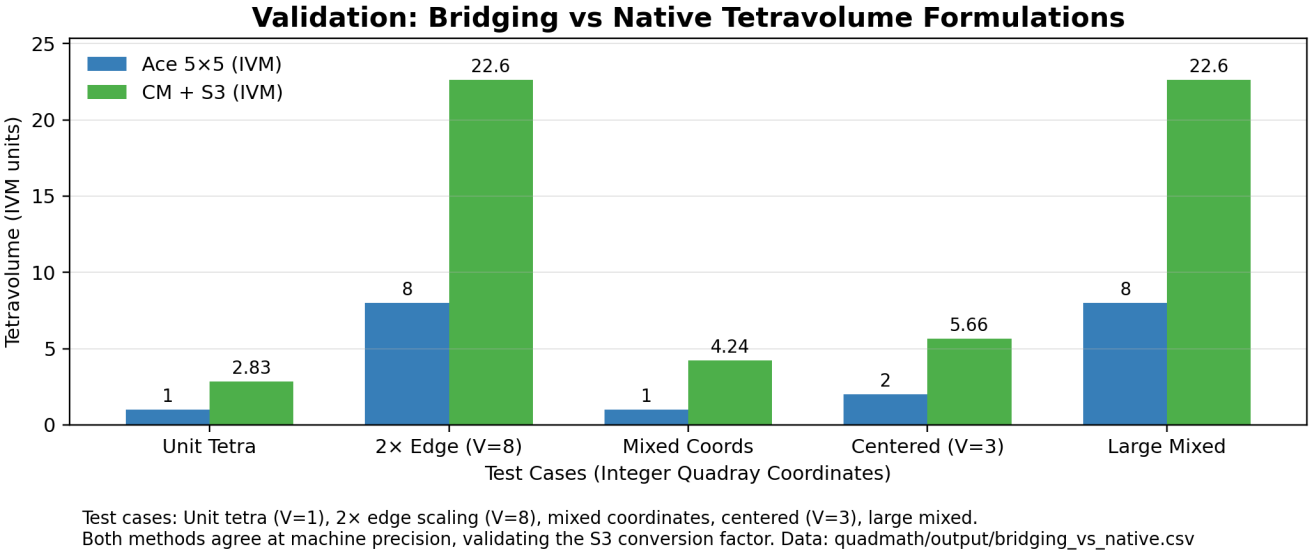

Figure 4: **Validation of bridging vs native tetravolume formulations across canonical examples**. This bar chart compares IVM tetravolumes computed via two independent methods: the "bridging" approach using Cayley–Menger determinants on Euclidean edge lengths converted to IVM units via the synergetics 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.

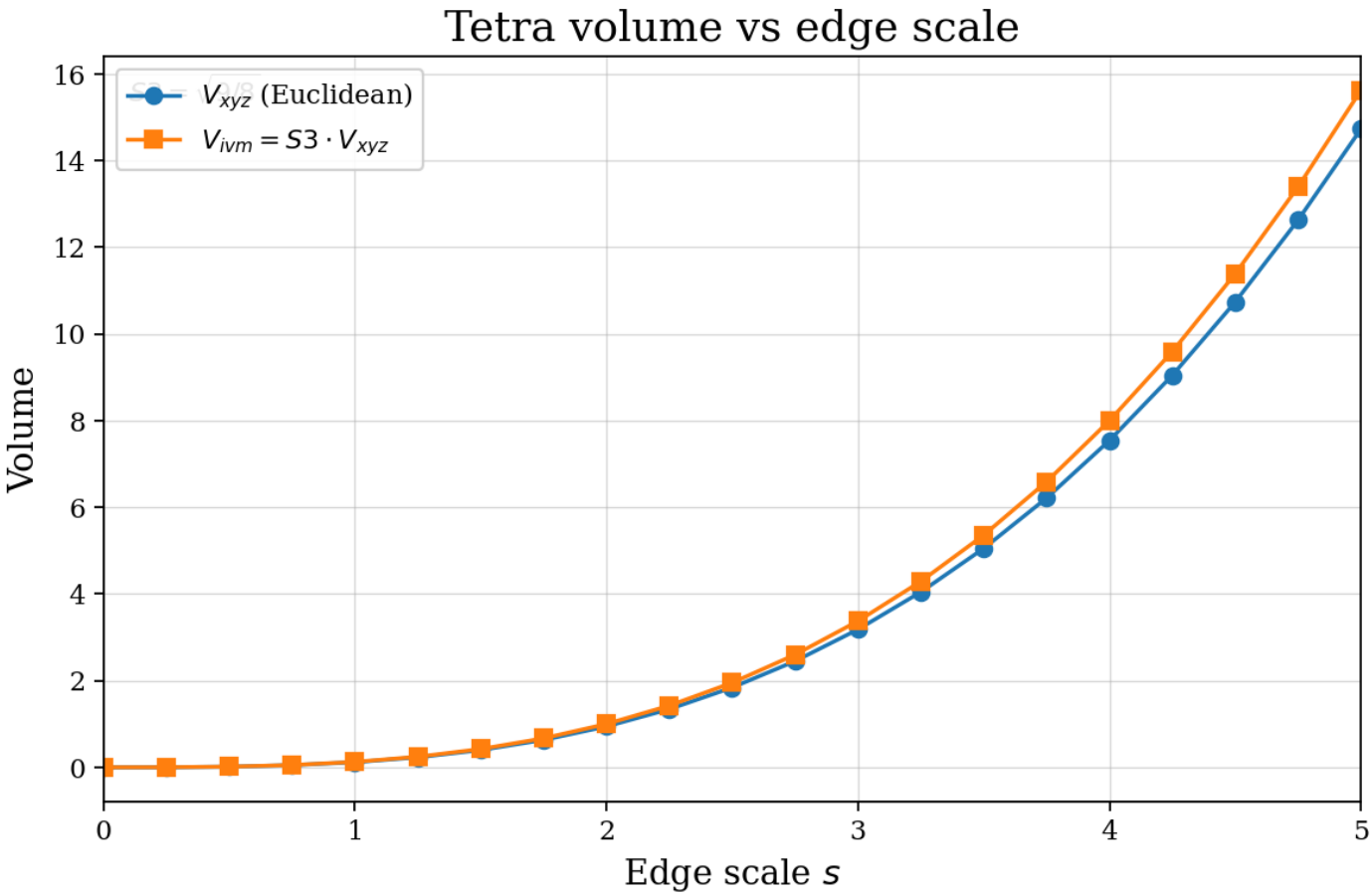

Figure 5: **Tetrahedron volume scaling relationships: Euclidean vs IVM unit conventions**. This plot demonstrates the mathematical relationship between edge length scaling and tetravolume under both Euclidean (XYZ) and IVM (synergetics) unit conventions. **X-axis**: Edge length scaling factor (0 to 5.0). **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.

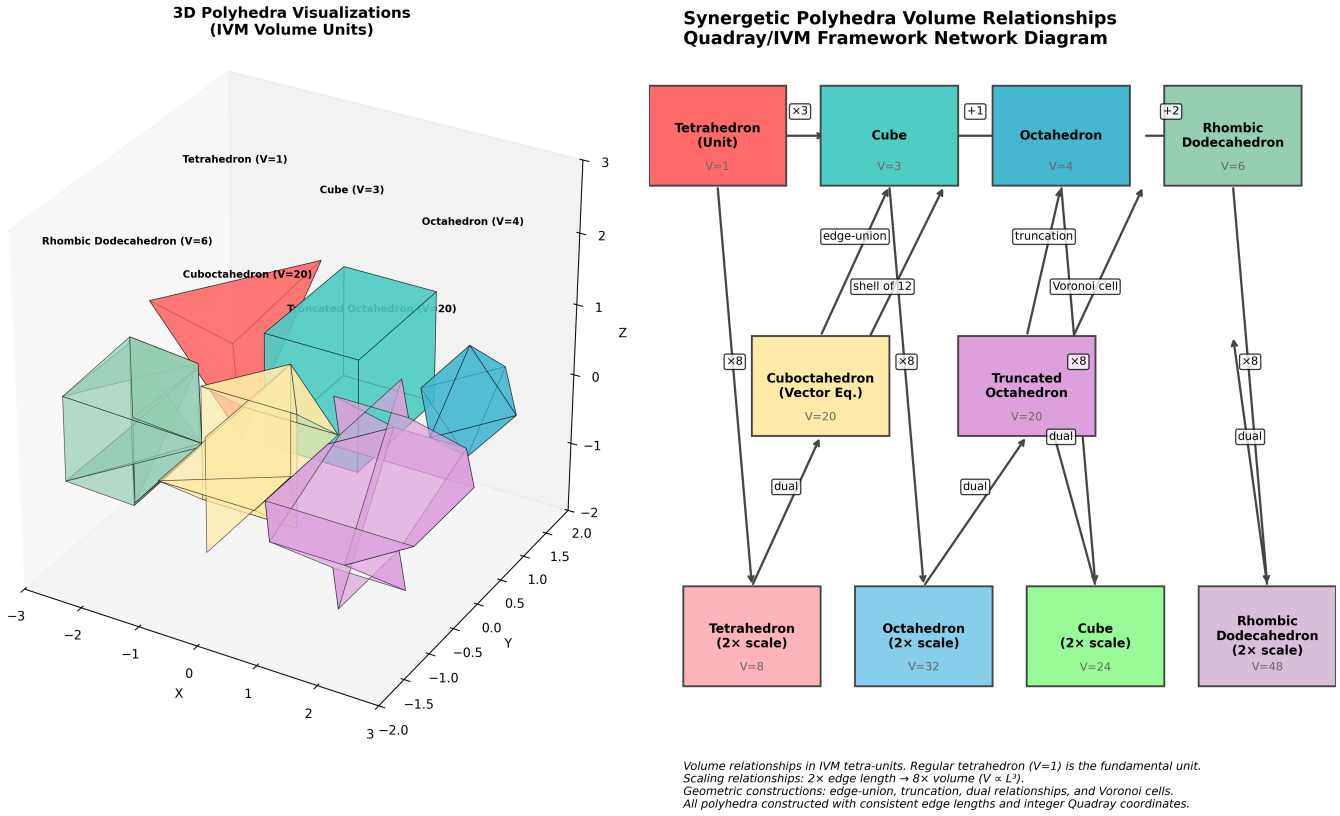

Figure 6: **Synergetic polyhedra volume relationships in the Quadray/IVM framework (comprehensive visualization)**. This figure combines 3D polyhedra visualizations with an extended network diagram showing integer volume relationships among key synergetic polyhedra. **Left panel (3D visualizations)**: 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
```

3.11.3 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
```

3.11.4 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.

3.11.5 XYZ determinant and the S3 conversion

- Using XYZ coordinates of the four vertices: see Eq. (4) for the determinant form and the S3 conversion to IVM units.

3.11.6 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.

3.12 Practical Implementation Workflow

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

3.12.1 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

3.12.2 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

3.12.3 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

3.12.4 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

3.12.5 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.

3.13 Code methods (anchors)

3.13.1 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.

3.13.2 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.

3.13.3 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.

3.13.4 Linear algebra utilities

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

3.13.5 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.

3.13.6 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.

3.13.7 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.

3.13.8 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.

3.13.9 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.

3.13.10 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.

3.13.11 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.

3.13.12 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.

3.13.13 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.

3.13.14 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.

3.13.15 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

3.13.16 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

3.14 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.

4 Optimization in 4D

4.1 Overview

This section describes optimization methods adapted to the integer Quadray lattice, emphasizing discrete convergence and information-geometric approaches. The methods leverage the IVM's natural quantization and extend to higher-dimensional spaces via Coxeter.4D embeddings.

4.2 Nelder–Mead on Integer Lattice

- **Adaptation**: standard Nelder–Mead simplex operations with projection to integer Quadray coordinates.
- **Projection**: after each reflection/expansion/contraction, snap to nearest integer lattice point via projective normalization.
- **Volume tracking**: monitor integer tetravolume as convergence diagnostic; discrete steps create stable plateaus.

4.2.1 Parameters

- **Reflection** $\alpha \approx 1$
- **Expansion** $\gamma \approx 2$
- **Contraction** $\rho \approx 0.5$
- **Shrink** $\sigma \approx 0.5$

References: original Nelder–Mead method and common parameterizations in optimization texts and survey articles; see overview: Nelder–Mead method.

4.3 Volume-Level Dynamics

- Simplex volume decreases in discrete integer steps, creating stable plateaus ("energy levels").
- Termination: when volume stabilizes at a minimal level and function spread is below tolerance.
- Monitoring: track integer simplex volume and the objective spread at each iteration for convergence diagnostics.

4.4 Quadray Lattice Optimization Pseudocode

```
while not converged:
  order vertices by objective
  centroid of best three
  propose reflected (then possibly expanded/contracted) point
  project to integer quadray; renormalize with (k,k,k,k)
  accept per standard tests; else shrink toward best
  update integer volume and function spread trackers
```

4.4.1 Figures

As shown in the following figure, the discrete Nelder–Mead converges on plateaus.

Raw artifacts: the full trajectory animation `simplex_animation.mp4` and per-frame vertices (`simplex_animation_vertices.csv`/`.npz`) are available in `quadmath/output/`. The full optimization trajectory is provided as an animation (MP4) in the repository's output directory.

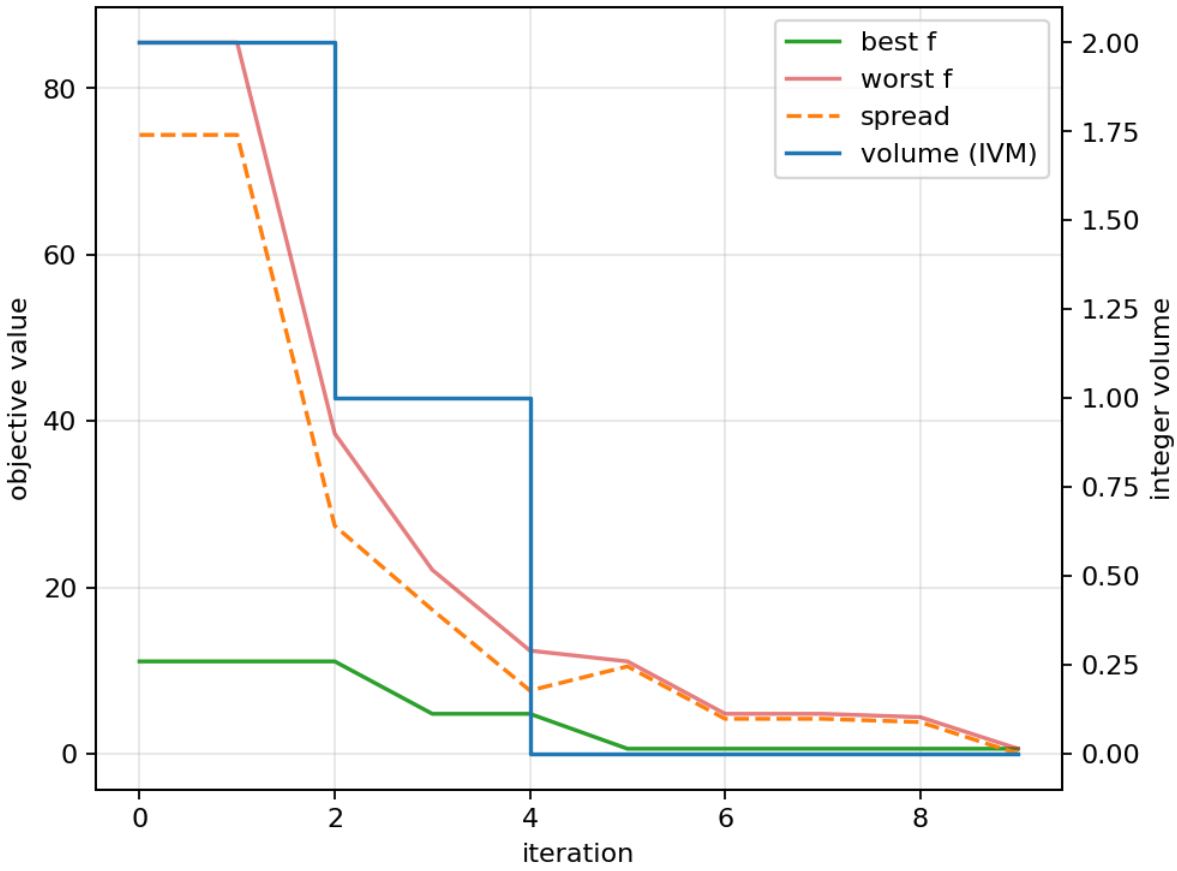

Figure 7: **Discrete Nelder–Mead optimization trajectory on the integer Quadray lattice**. This time-series plot tracks key diagnostic quantities across 12 optimization iterations for a simple quadratic objective function defined on the integer Quadray lattice. **X-axis**: Optimization iteration (0 through 12). **Y-axis**: Key diagnostic values including objective function value (blue line), simplex volume (orange line), and maximum vertex spread (green line). **Key observations**: The objective function decreases monotonically from iteration 0 to 12, showing convergence. The simplex volume (orange) exhibits discrete plateaus characteristic of integer-lattice optimization, where the Nelder–Mead algorithm can only move to integer coordinate positions. The maximum vertex spread (green) decreases as the simplex contracts around the optimum, indicating that the four vertices of the optimization tetrahedron are converging to a tight cluster. **Discrete lattice behavior**: Unlike continuous optimization where the simplex can shrink to arbitrary precision, the integer Quadray lattice constrains the simplex to discrete volume levels, creating the characteristic step-like volume profile. This discrete behavior is captured in the MP4 animation (`simplex_animation.mp4`) and the diagnostic traces in the following figure. The final simplex volume is minimal on the integer lattice, representing a stable "energy level" where further discrete moves do not improve the objective function.

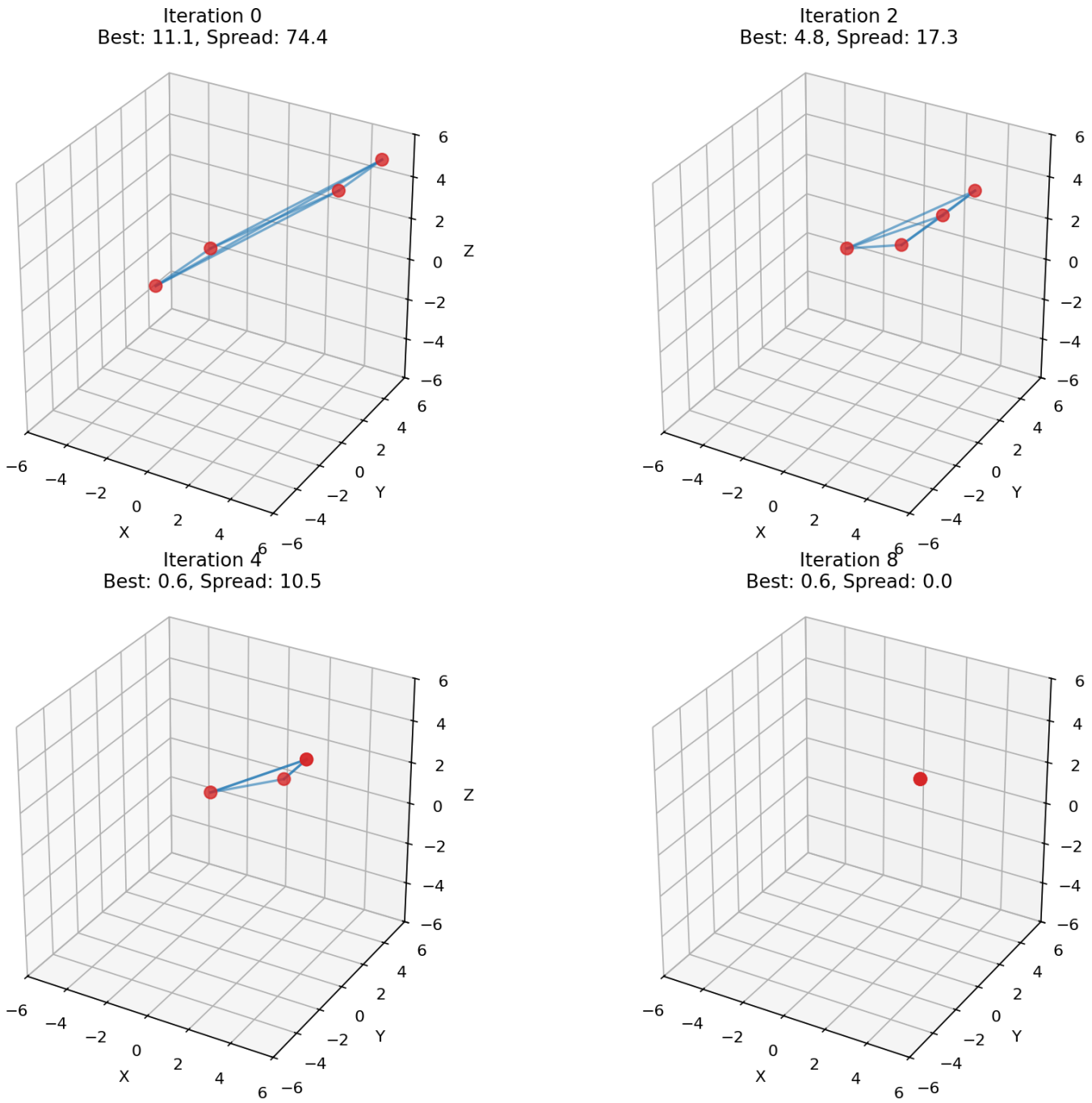

Figure 8: **Nelder-Mead simplex evolution on integer Quadray lattice (2×2 panel)**. This comprehensive visualization shows the simplex optimization process at key iterations (0, 3, 6, 9) to demonstrate the discrete convergence behavior. **Top-left (Iteration 0)**: Initial simplex configuration with four vertices forming a tetrahedron in 3D embedding space, starting from widely dispersed positions. **Top-right (Iteration 3)**: Early optimization state showing initial simplex contraction and vertex repositioning toward the optimal region. **Bottom-left (Iteration 6)**: Mid-optimization with vertices converging toward the optimum at coordinates (2,2,2). **Bottom-right (Iteration 9)**: Final converged state where all vertices have collapsed to the optimal point (2,2,2), representing successful convergence to the global minimum. **Key features**: Each subplot shows the tetrahedral simplex with vertices as red spheres and edges as blue lines connecting the vertices. The objective function values and vertex spread are displayed in each subplot title, showing the monotonic decrease in both quantities. **Discrete lattice behavior**: The step-wise convergence demonstrates how the integer Quadray lattice constrains optimization to discrete volume levels, creating the characteristic plateau behavior seen in the diagnostic traces.

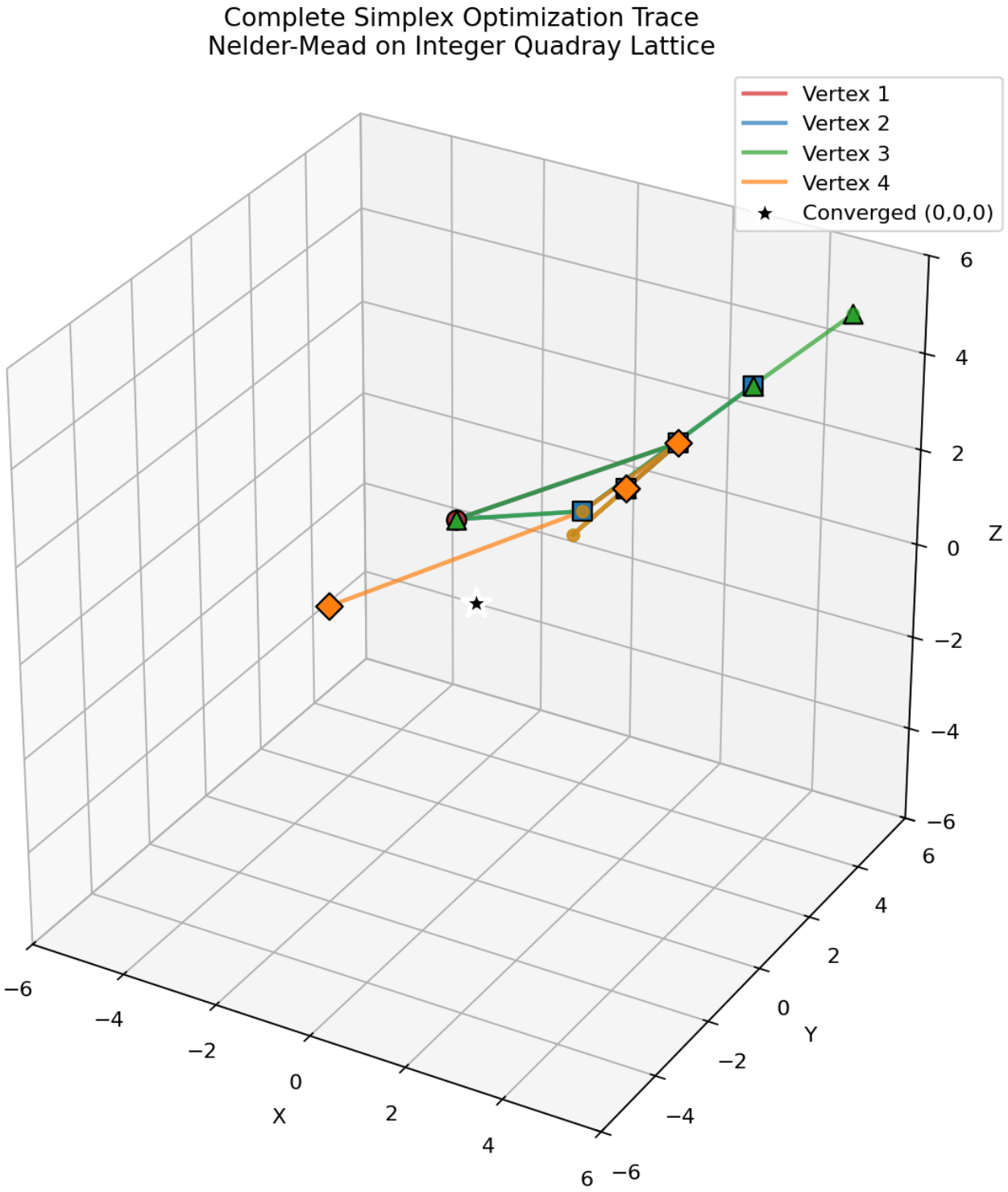

Figure 9: **Complete simplex optimization trace visualization**. This 3D plot shows the complete trajectory of all four simplex vertices across all optimization iterations, providing a comprehensive view of the optimization path. **Vertex traces**: Each vertex follows a distinct colored path (red, blue, green, orange) from its initial position to the final converged point at (2,2,2). **Key iteration markers**: Large markers at iterations 0, 3, 6, and 9 highlight critical stages in the optimization process. **Convergence point**: The black star at (2,2,2) marks the final converged state where all vertices meet at the global optimum. **Optimization insights**: The trace reveals how the simplex contracts systematically, with vertices moving in coordinated patterns that respect the integer lattice constraints. The discrete nature of the optimization is evident in the step-wise vertex movements, which can only occur to valid integer Quadray coordinates. This visualization complements the 2×2 panel view by showing the complete optimization trajectory in a single, interpretable plot.

4.5 Discrete Lattice Descent (Information-Theoretic Variant)

- Integer-valued descent over the IVM using the 12 neighbor moves (permutations of {2,1,1,0}), snapping to the canonical representative via projective normalization.
- Objective can be geometric (e.g., Euclidean in an embedding) or information-theoretic (e.g., local free-energy proxy); monotone decrease is guaranteed by greedy selection.
- API: `discrete_ivm_descent` in `src/discrete_variational.py`. Animation helper: `animate_discrete_path` in `src/visualize.py`.

Short snippet (paper reproducibility):

```
from quadray import Quadray, DEFAULT_EMBEDDING, to_xyz
from discrete_variational import discrete_ivm_descent
from visualize import animate_discrete_path

def f(q: Quadray) -> float:
    x, y, z = to_xyz(q, DEFAULT_EMBEDDING)
    return (x - 0.5)**2 + (y + 0.2)**2 + (z - 0.1)**2

path = discrete_ivm_descent(f, Quadray(6,0,0,0))
animate_discrete_path(path)
```

4.6 Convergence and Robustness

- Discrete steps reduce numerical drift; improved stability vs. unconstrained Cartesian.
- Natural regularization from volume quantization; fewer wasted evaluations.
- Compatible with Gauss–Newton/Natural Gradient guidance using FIM for metric-aware steps (Amari, natural gradient).

4.7 Information-Geometric View (Einstein.4D analogy in metric form)

The Fisher Information Matrix (FIM) provides a fundamental bridge between the three 4D frameworks, establishing a Riemannian metric on parameter space that guides optimization through information geometry. This section demonstrates how the FIM connects Coxeter.4D (Euclidean parameter space), Einstein.4D (information-geometric flows), and Fuller.4D (tetrahedral structure) in a unified optimization framework.

4.7.1 Fisher Information as Riemannian Metric

The empirical Fisher Information Matrix F_{ij} quantifies the local curvature of the log-likelihood surface around parameter estimates, providing a natural metric for parameter space geometry. This fundamental concept in information geometry establishes a Riemannian structure on the statistical manifold, where distances and angles are measured according to the intrinsic geometry of the probability distributions rather than the extrinsic Euclidean geometry of the parameter space.

For a model with parameters $\mathbf{w} = (w_0, w_1, w_2)$ and loss function $L(\mathbf{w})$, the FIM is estimated as the expected outer product of score functions (see Eq. (9) in the equations appendix).

where L_n represents the loss for individual data samples. This matrix captures both parameter sensitivity (diagonal elements) and parameter interactions (off-diagonal elements), revealing the intrinsic geometry of the optimization landscape.

The Fisher Information Matrix serves as the natural metric tensor $g_{ij} = F_{ij}$ on the statistical manifold, replacing the Euclidean metric δ_{ij} with a data-dependent metric that reflects the actual curvature structure of the objective function. This geometric interpretation enables the application of differential geometry concepts to optimization problems, where geodesics (locally distance-minimizing paths) follow the natural gradient direction $F^{-1}\nabla L$ rather than the standard gradient ∇L.

The theoretical foundation of this approach stems from the work of Rao (1945) and Amari (1985), who established information geometry as a framework for analyzing statistical models through differential geometry. The FIM naturally arises as the Hessian of the Kullback-Leibler divergence between nearby probability distributions, making it the canonical choice for measuring distances on the statistical manifold.

In the context of optimization, the FIM provides several key advantages:

1. **Invariance to parameterization**: The natural gradient $F^{-1}\nabla L$ is invariant to smooth, invertible parameter transformations, unlike the standard gradient which depends on the choice of coordinate system.
2. **Optimal step sizing**: The FIM automatically determines appropriate step sizes in different parameter directions, scaling updates according to local curvature.
3. **Geometric consistency**: Optimization follows geodesics on the statistical manifold, respecting the intrinsic geometry of the parameter space rather than imposing an artificial Euclidean structure.

This geometric approach to optimization is particularly powerful in the context of the 4D frameworks, where it provides a unified mathematical language for describing optimization dynamics across different geometric paradigms.

4.7.2 4D Framework Integration through Fisher Information

Coxeter.4D (Euclidean): In standard Euclidean parameter space, the metric tensor is simply δ_{ij}, providing uniform scaling in all directions. The FIM F_{ij} generalizes this to capture the actual curvature structure of the objective function.

Einstein.4D (Minkowski analogy): The Fisher metric replaces the spacetime metric, where geodesics follow $F^{-1}\nabla L$ instead of straight lines. This creates optimal parameter update paths that respect the intrinsic geometry of the statistical manifold. The natural gradient update rule $\Delta\mathbf{w} = -\eta F^{-1}\nabla L$ implements geodesic motion on the information manifold, analogous to how particles follow geodesics in relativistic spacetime.

Fuller.4D (Synergetics): The tetrahedral structure of Quadray coordinates naturally encodes the fourfold partition of optimization problems, while the FIM provides the metric structure for efficient navigation through this space. The discrete nature of the IVM lattice creates natural quantization effects that can be exploited for computational efficiency.

4.7.3 Comprehensive Fisher Information Analysis: Figures 10 and 11

The following figures demonstrate the comprehensive nature of Fisher Information analysis, showing both the matrix structure and its eigenspectrum interpretation. This analysis reveals the anisotropic nature of parameter space and guides the design of efficient optimization strategies.

Figure 10: Fisher Information Matrix (FIM) with 4D Framework Context. This comprehensive three-panel visualization demonstrates the empirical Fisher information matrix and its deep connections to the three 4D mathematical frameworks through code-grounded analysis.

Figure 11: Comprehensive Fisher Information Eigenspectrum with Curvature Analysis. This detailed three-panel visualization provides comprehensive analysis of the parameter space geometry within the 4D framework context, including tetrahedral parameter space visualization.

4.7.4 Natural Gradient Descent: Geodesic Motion on Information Manifold

The Fisher Information Matrix enables natural gradient descent, which implements geodesic motion on the information manifold. Unlike standard gradient descent that follows straight lines in parameter space, natural gradient descent follows curved paths that respect the intrinsic geometry defined by the FIM.

The natural gradient update rule is given by:

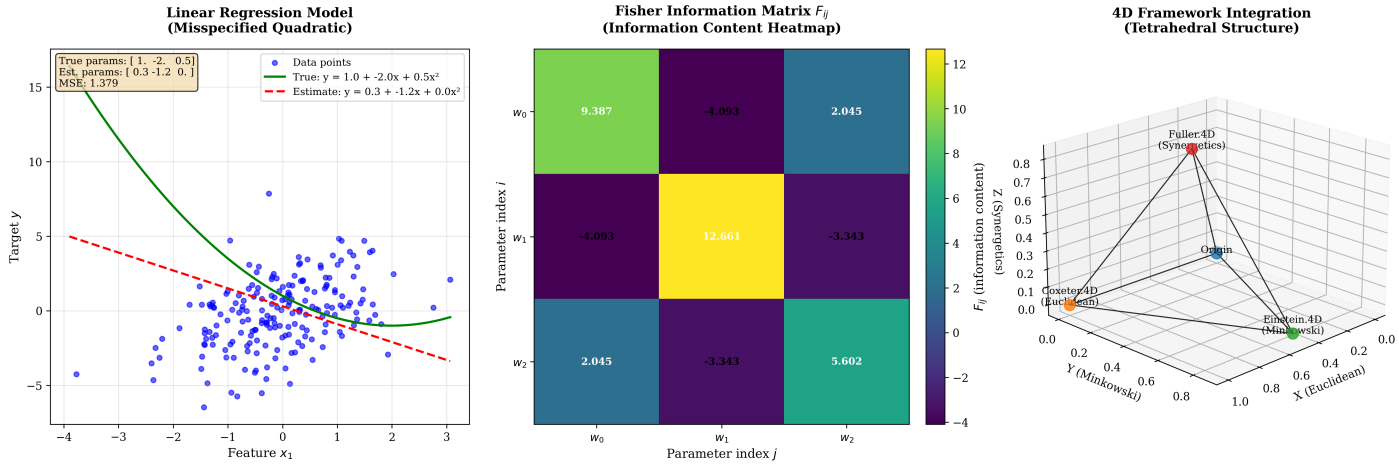

Figure 10: **Fisher Information Matrix (FIM) with 4D Framework Context**. This comprehensive three-panel visualization demonstrates the empirical Fisher information matrix and its deep connections to the three 4D mathematical frameworks through code-grounded analysis. **Left panel**: Linear regression model visualization showing the misspecified quadratic model $y = w_0 + w_1 x + w_2 x^2$ with true parameters $w_{true} = [1.0, -2.0, 0.5]$ and estimated parameters $w_{est} = [0.3, -1.2, 0.0]$. The panel displays data points, true model fit (green line), estimated model fit (red dashed line), and diagnostic information including Mean Squared Error (MSE). This visualization grounds the Fisher Information analysis in the actual model that generates the parameter gradients. **Center panel**: The 3×3 Fisher information matrix F_{ij} estimated from per-sample gradients of the misspecified linear regression model, displayed as a heatmap with precise value annotations. The matrix structure reveals the local curvature of the log-likelihood surface, where brighter colors indicate higher information content. **Matrix interpretation**: Diagonal elements F_{ii} quantify the sensitivity of the objective to changes in parameter w_i, while off-diagonal elements F_{ij} capture parameter interactions and potential redundancy. **Right panel**: 3D tetrahedral visualization of the 4D framework integration, showing how Coxeter.4D (Euclidean), Einstein.4D (Minkowski), and Fuller.4D (Synergetics) frameworks connect through the tetrahedral structure. **Mathematical foundation**: The FIM is computed according to Eq. (9) where gradients are computed with respect to parameters w_0, w_1, w_2 from the misspecified model. **Coxeter.4D (Euclidean)**: Standard 3D parameter space with Euclidean metric δ_{ij}. **Einstein.4D (Minkowski)**: Fisher metric F_{ij} replaces spacetime metric; geodesics follow $\Delta w = F^{-1}\nabla L$ for optimal parameter updates. **Fuller.4D (Synergetics)**: Tetrahedral coordinate system with IVM quantization. **Information content**: Diagonal dominance shows each parameter contributes independently to the model's predictive power, while off-diagonal elements reveal parameter interactions and potential redundancy. This FIM structure guides natural gradient descent by weighting parameter updates according to local curvature, leading to more efficient convergence than standard gradient descent.

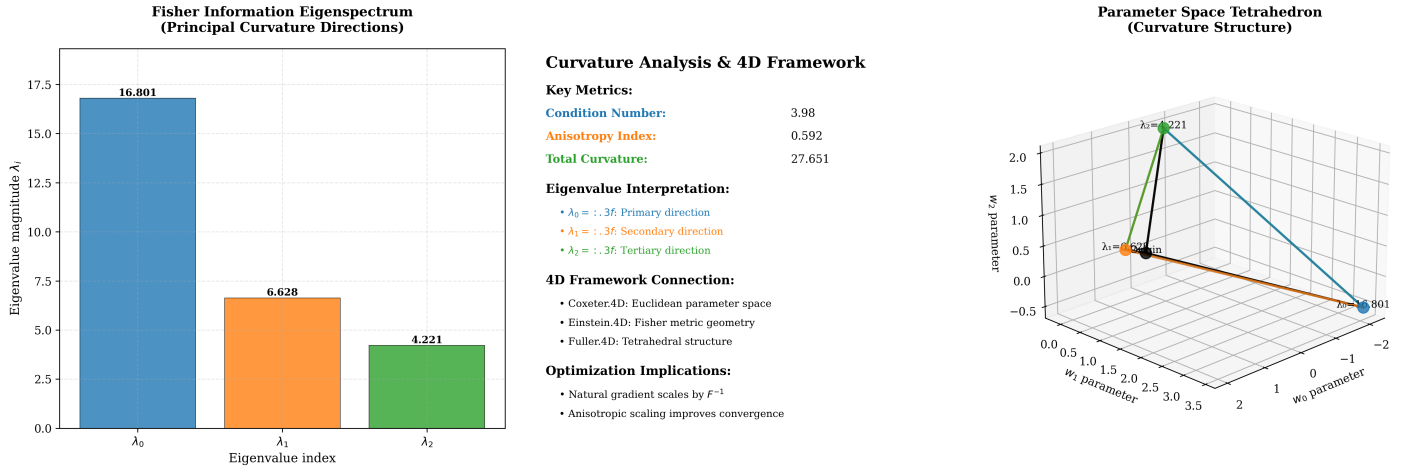

Figure 11: **Comprehensive Fisher Information Eigenspectrum with Curvature Analysis**. This detailed three-panel visualization provides comprehensive analysis of the parameter space geometry within the 4D framework context, including tetrahedral parameter space visualization. **Left panel**: Bar chart showing the eigenvalue decomposition of the empirical Fisher information matrix, with eigenvalues sorted in descending order and color-coded for visual clarity. Each bar is precisely annotated with its numerical value, revealing the principal curvature directions of the parameter space. **Center panel**: Comprehensive curvature analysis providing key metrics, eigenvalue interpretation, and 4D framework connections. **Key metrics**: Condition number (anisotropy measure), anisotropy index (normalized directional variation), and total curvature (trace of F). **Eigenvalue interpretation**: Each eigenvalue λ_i represents the curvature strength in the corresponding principal direction. Large eigenvalues indicate directions of high curvature (tight constraints) where the objective function changes rapidly with parameter changes, while small eigenvalues indicate directions of low curvature (loose constraints) where the objective function is relatively flat. **Right panel**: 3D tetrahedral visualization of the parameter space structure based on the Fisher Information eigenvectors and eigenvalues. The tetrahedron vertices represent the origin and the three principal curvature directions, scaled by the square root of eigenvalues to show the anisotropic structure. **4D framework connection**: The eigenvalues reveal the anisotropic nature of the parameter space, explaining why natural gradient descent (which scales updates by F^{-1}) converges more efficiently than standard gradient descent. **Coxeter.4D**: The eigenvalues quantify the Euclidean geometry of parameter space in different directions. **Einstein.4D**: The Fisher metric geometry creates curved geodesics that respect the intrinsic parameter space structure. **Fuller.4D**: The tetrahedral structure provides a natural coordinate system for representing the four-fold partition of optimization problems, with the parameter space tetrahedron directly reflecting the curvature structure. **Optimization implications**: Natural gradient descent scales parameter updates by F^{-1}, creating anisotropic scaling that improves convergence on ill-conditioned problems. The tetrahedral visualization shows how the parameter space anisotropy creates natural directions for efficient optimization. This geometric understanding is crucial for designing effective optimization strategies and understanding model behavior in the context of information geometry.

where η is the learning rate, F is the Fisher Information Matrix from Eq. (9), and ∇L is the standard gradient of the loss function. This update rule implements geodesic motion on the statistical manifold, where the metric tensor $g_{ij} = F_{ij}$ determines the local geometry.

The theoretical foundation of natural gradient descent was established by Amari (1998) in the context of information geometry. The key insight is that the natural gradient $F^{-1}\nabla L$ is the steepest descent direction when distances are measured using the Fisher metric rather than the Euclidean metric. This makes natural gradient descent invariant to smooth, invertible parameter transformations, a property that standard gradient descent lacks.

In the context of the 4D frameworks, natural gradient descent provides a unified approach to optimization that respects the intrinsic geometry of each framework:

- **Coxeter.4D**: The natural gradient respects the actual curvature structure of the objective function rather than imposing artificial Euclidean geometry.
- **Einstein.4D**: The Fisher metric replaces the spacetime metric, creating geodesic flows that follow the intrinsic geometry of the parameter space.
- **Fuller.4D**: The tetrahedral structure provides natural coordinate systems where the FIM can exhibit beneficial structural properties.

The efficiency of natural gradient descent comes from its ability to automatically adapt step sizes to local curvature. In directions of high curvature (large eigenvalues of F), the natural gradient takes smaller steps, while in directions of low curvature (small eigenvalues), it takes larger steps. This anisotropic scaling leads to faster convergence and better numerical stability compared to standard gradient descent.

4.7.5 Information-Theoretic Foundations and 4D Framework Coherence

The Fisher Information approach provides several key advantages that integrate naturally with the 4D framework structure:

1. **Geometric Consistency**: The FIM ensures that optimization respects the intrinsic geometry of the parameter space, maintaining consistency across all three 4D frameworks.
2. **Anisotropic Scaling**: Natural gradient descent automatically adapts step sizes to local curvature, improving convergence efficiency on problems with strong parameter space anisotropy.
3. **Framework Bridging**: The FIM serves as a mathematical bridge between Coxeter.4D (Euclidean geometry), Einstein.4D (information-geometric flows), and Fuller.4D (tetrahedral structure).
4. **Quantitative Analysis**: The eigenspectrum provides quantitative measures of parameter space structure, enabling principled optimization strategy design.

4.7.6 Quadray-Specific Considerations

Under Quadray parameterizations, the FIM often exhibits block-structured and symmetric patterns that simplify matrix inversion for natural-gradient steps. This structural regularity arises from the tetrahedral symmetry of the IVM lattice and can be exploited for computational efficiency.

The discrete nature of the IVM lattice also influences the FIM structure, as parameter updates are constrained to integer coordinate positions. This creates a natural regularization effect that can improve optimization stability and convergence.

4.7.7 Variational Free Energy and Active Inference Integration

The Fisher Information framework naturally extends to variational inference and active inference, where the free energy principle guides both perception and action through information-geometric optimization.

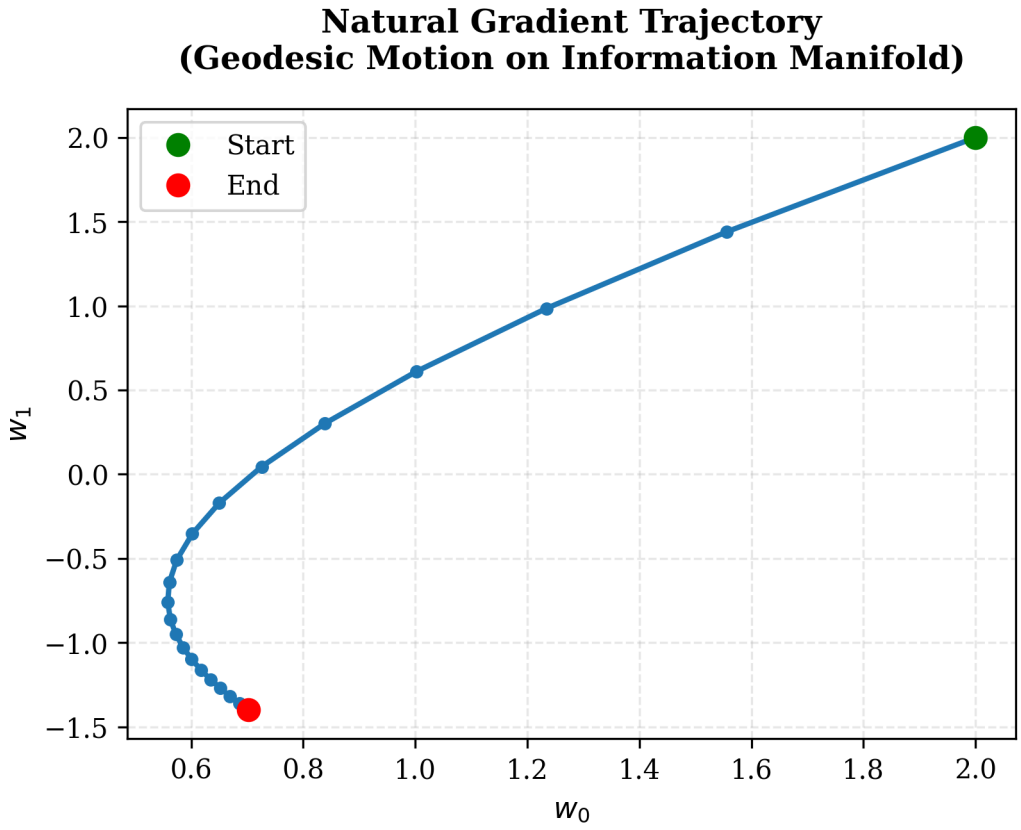

Figure 12: **Natural Gradient Trajectory: Geodesic Motion on Information Manifold**. This visualization demonstrates the parameter trajectory of natural gradient descent, showing how information-geometric optimization creates optimal paths through parameter space. **Trajectory**: The blue line with markers traces the parameter evolution from initial guess to final optimum, revealing the path taken through the 2D parameter space. **Markers**: Each marker represents one optimization step, with spacing indicating the step size and convergence rate. **Start/End markers**: Green circle marks the initial parameter values, red circle marks the converged optimum. **4D Framework Connection**: This trajectory demonstrates geodesic motion on the information manifold, where the Fisher metric (Einstein.4D analogy) replaces the physical metric. The natural gradient follows Eq. (10), creating optimal paths through parameter space that respect the intrinsic geometry. **Convergence behavior**: The trajectory shows smooth, direct convergence to the optimum, characteristic of natural gradient descent on well-conditioned objectives. **Comparison with standard gradient descent**: Natural gradient descent typically produces more direct trajectories than standard gradient descent, especially on ill-conditioned problems where the parameter space has strong anisotropy. This efficiency comes from the FIM-based scaling that adapts step sizes to local curvature. The trajectory demonstrates how information-geometric optimization leverages the intrinsic geometry of the parameter space to achieve faster, more stable convergence than naive gradient methods. **Grid overlay**: Added for better readability and to emphasize the discrete nature of the optimization steps.

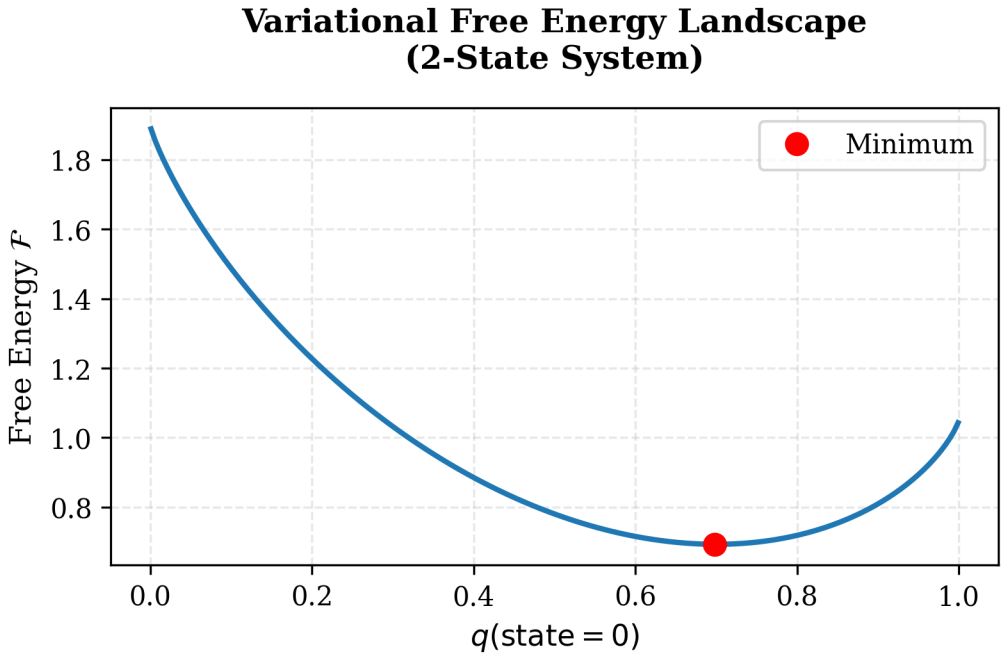

Figure 13: **Variational Free Energy Landscape with 4D Framework Integration**. This visualization shows the variational free energy $\mathcal{F} = -\log P(o|s) + \text{KL}[Q(s)||P(s)]$ (see Eq. (11)) as a function of the variational distribution parameter, demonstrating the geometry of the variational manifold. **X-axis**: Variational parameter $q(\text{state} = 0)$ controlling the distribution over the two discrete states. **Y-axis**: Free energy value $\mathcal{F}$ in natural units. **Curve interpretation**: The free energy exhibits a clear minimum at the optimal variational distribution, representing the best approximation to the true posterior given the constraints of the variational family. **4D Framework Connection**: The free energy landscape represents the geometry of the variational manifold, where optimization follows geodesics defined by the Fisher metric (Einstein.4D analogy). In active inference frameworks, minimizing free energy drives both perception and action, analogous to how geodesics minimize proper time in relativistic spacetime. **KL divergence component**: The free energy balances data fit (first term) with regularization (KL divergence from prior), preventing overfitting while maintaining good predictive performance. **Optimization geometry**: The smooth, convex shape of the free energy landscape makes optimization straightforward using natural gradient descent, which respects the intrinsic geometry of the parameter space. This variational framework provides a principled approach to approximate inference in complex models where exact posterior computation is intractable, while maintaining connections to the broader 4D mathematical frameworks.

4.7.8 Advanced 4D Framework Integration: Active Inference Context

The integration of Fisher Information with Active Inference demonstrates the full power of the 4D framework approach, where Coxeter.4D provides exact geometry, Einstein.4D supplies information-geometric flows, and Fuller.4D offers the tetrahedral structure for representing the four-fold partition of perception-action systems.

For comprehensive Active Inference visualizations including 4D natural gradient trajectories and free energy landscapes, see Section 9: Free Energy and Active Inference.

- **Quadray relevance**: block-structured and symmetric patterns often arise under quadray parameterizations, simplifying `F` inversion for natural-gradient steps.

4.8 Multi-Objective and Higher-Dimensional Notes (Coxeter.4D perspective)

- Multi-objective: vertices encode trade-offs; simplex faces approximate Pareto surfaces; integer volume measures solution diversity.
- Higher dimensions: decompose higher-dimensional simplexes into tetrahedra; sum integer volumes to extend quantization.

4.9 External validation and computational context

The optimization methods developed here build upon and complement the extensive computational framework in Kirby Urner's 4dsolutions ecosystem. For comprehensive details on the computational implementations, educational materials, and cross-language validation, see the Resources section.

4.10 Results

- The simplex-based optimizer exhibits discrete volume plateaus and converges to low-spread configurations; see the simplex figures above and the MP4/CSV artifacts in `quadmath/output/`.

5 Extensions of 4D and Quadrays

Here we review some extensions of the Quadray 4D framework, including multi-objective optimization, machine learning, computer graphics and GPU acceleration, active inference, complex systems, pedagogy, and implementations, with an emphasis on cognitive security.

5.1 Multi-Objective Optimization

- Simplex faces encode trade-offs; integer volume measures solution diversity.
- Pareto front exploration via tetrahedral traversal.

5.2 Machine Learning and Robustness

- **Geometric regularization**: Quadray-constrained weights/topologies yield structural priors and improved stability.
- **Adversarial robustness**: Discrete lattice projection reduces vulnerability to gradient-based adversarial perturbations by limiting directions.
- **Ensembles**: Tetrahedral vertex voting and consensus improve robustness.

References: see Fisher information, Natural gradient, and quadray conversion notes by Urner for embedding choices.

5.3 Computer Graphics and GPU Acceleration

- **Quadray visualization acceleration**: GPU-accelerated rendering of tetrahedral coordinate systems enables real-time exploration of 4D geometric structures. The parallel nature of GPU architectures naturally maps to the four-basis vector representation of quadrays, allowing simultaneous computation of vertex positions, edge connections, and face tessellations across thousands of tetrahedra.
- **Integer arithmetic optimization**: GPU compute shaders excel at integer-based volume calculations and determinant computations using the Bareiss algorithm. The discrete lattice structure of quadray coordinates benefits from parallel integer arithmetic units, achieving significant speedups over CPU implementations for large-scale geometric computations.
- **Dynamic programming acceleration**: GPU-accelerated dynamic programming algorithms leverage CUDA Dynamic Parallelism for adaptive parallel computation of recursive geometric algorithms. This approach enables efficient handling of varying computational workloads in tetrahedral decomposition and optimization problems, as demonstrated in applications like the Mandelbrot set computation where dynamic parallelism manages computational complexity effectively.
- **Parallel geometric algorithms**: Implementation of GPU-optimized versions of algorithms like QuickHull for convex hull computation in quadray space achieves substantial performance improvements. The tetrahedral lattice structure naturally supports parallel prefix sum operations and efficient neighbor queries, enabling real-time visualization of complex 4D geometric transformations.
- **Memory bandwidth optimization**: The structured memory access patterns of quadray coordinates align well with GPU memory hierarchies, enabling efficient coalesced memory access for large-scale geometric datasets. This optimization is particularly beneficial for applications requiring real-time rendering of complex polyhedral structures and dynamic tessellations.

References: GPU-accelerated geometry processing techniques (arxiv.org), CUDA Dynamic Parallelism for adaptive computation (developer.nvidia.com), and parallel scan algorithms for optimization (developer.nvidia.com).

5.4 Active Inference and Free Energy

- Free energy $\mathcal{F} = -\log P(o \mid s) + \mathrm{KL}[Q(s) \,\|\, P(s)]$ (see Eq. (11) in the equations appendix); background: Free energy principle and overviews connecting to predictive coding and control.

- Belief updates follow steepest descent in Fisher geometry using the natural gradient (see Eq. (10) in the equations appendix); quadray constraints improve stability/interpretability.
- Links to metabolic efficiency and biologically plausible computation.
- For more information, see the Appendix: The Free Energy Principle and Active Inference.

5.5 Complex Systems and Collective Intelligence

- Tetrahedral interaction patterns support distributed consensus and emergent behavior.
- Resource allocation and network flows benefit from geometric constraints.
- **Cognitive security**: Applying cognitive security can safeguard distributed consensus mechanisms from manipulation, preserving the reliability of emergent behaviors in complex systems. Incorporating cognitive security measures can protect the integrity of belief updates and decision-making processes, ensuring that actions are based on accurate and unmanipulated information.

5.6 Geospatial Intelligence and the World Game

- **Spatial data integration**: Quadray tetrahedral frameworks provide natural tessellations for geospatial data analysis, where the Dymaxion projection's minimal distortion aligns with Fuller's World Game objectives of holistic global perspective. The tetrahedral lattice supports efficient spatial indexing and neighbor queries for distributed geospatial intelligence operations.
- **Resource allocation optimization**: The World Game's goal of "making the world work for 100% of humanity" translates to multi-objective optimization problems where tetrahedral simplex faces encode trade-offs between population centers, resource distribution, and ecological constraints. Integer volume quantization ensures discrete, interpretable solutions for global resource allocation.
- **Cognitive security in distributed sensing**: Geospatial intelligence networks benefit from tetrahedral consensus mechanisms that resist manipulation of spatial data streams. The geometric constraints of Fuller.4D provide natural validation frameworks for detecting anomalous spatial patterns and maintaining data integrity across distributed sensor networks.
- **Tetrahedral tessellations for global modeling**: The World Game's emphasis on interconnected global systems maps naturally to tetrahedral decompositions of the Dymaxion projection, where each tetrahedron represents a coherent region for local optimization while maintaining global connectivity through shared faces and edges.

5.7 Quadrays, Synergetics (Fuller.4D), and William Blake

- Quadrays (tetrahedral coordinates) instantiate Fuller's Synergetics emphasis on the tetrahedron as a structural primitive; in this manuscript's terminology this corresponds to Fuller.4D. Tetrahedral frames support part–whole reasoning and efficient decompositions used throughout.
- William Blake's "fourfold vision" (single, twofold, threefold, fourfold) provides a historical metaphor for multiscale perception and inference. Read through Fisher geometry and natural gradient dynamics, it parallels multilayer predictive processing and counterfactual simulation. For background, see a concise overview of Blake's visionary psycho-topographies in British Art Studies (visionary art analysis) and the Active Inference Institute's MathArt Stream #8 (Active Inference & Blake).
- Juxtaposing Blake and Fuller foregrounds "comprehensivity": holistic design and sensemaking via geometric primitives. Context: (Fuller & Blake: Lives in Juxtaposition) and pedagogical antecedents in experimental design education at Black Mountain College (Diaz, Chance and Design at Black Mountain College – PDF).
- Implications for Quadray practice: four-facet summaries of models/trajectories, tetrahedral consensus in ensembles, and stigmergic annotation patterns for cognitive security and distributed sensemaking.

5.8 Pedagogy and Implementations

Kirby Urner's comprehensive 4dsolutions ecosystem provides extensive educational resources and cross-platform implementations for Quadray computation and visualization. For comprehensive details on ed-

ucational frameworks, cross-language implementations, historical context, and community development, see the Resources section.

5.9 Higher Dimensions and Decompositions

- Decompose higher-dimensional simplexes into tetrahedra; sum integer volumes to maintain quantization.
- Tessellations support parallel/distributed implementations.

5.10 Limitations and Future Work

- Benchmark breadth: extend beyond convex/quadratic toys to real tasks (registration, robust regression, control) with ablations.
- Distance sensitivity: compare embeddings and their effect on optimizer trajectories; document recommended defaults.
- Hybrid schemes: study schedules that interleave continuous proposals with lattice projection.

6 Discussion

Quadray geometry (Fuller.4D) offers an interpretable, quantized view of geometry, topology, information, and optimization. Integer volumes enforce discrete dynamics, acting as a structural prior that can regularize optimization, reduce overfitting, prevent numerical fragility, and enable integer-based accelerated methods. Information geometry provides a right language for optimization in the synergetic tradition: optimization proceeds not through arbitrary parameter-space moves in continuous space, but along geodesics defined by information content (see Eq. (8) and Eq. (10) in the equations appendix; overview: Natural gradient).

Limitations and considerations:

- **Embeddings and distances**: Mapping between quadray and Euclidean coordinates must be selected carefully for distance calculations.
- **Hybrid strategies**: Some problems may require hybrid strategies (continuous steps with periodic lattice projection).
- **Benchmarking**: Empirical benchmarking remains important to quantify benefits across domains.

In practical analysis and simulation, numerical precision matters. Integer-volume reasoning is exact in theory, but empirical evaluation (e.g., determinants, Fisher Information, geodesics) can benefit from high-precision arithmetic. When double precision is insufficient, quad-precision arithmetic (binary128) via GCC's `libquadmath` provides the `__float128` type and a rich math API for robust computation. See the official documentation for details on functions and I/O: GCC libquadmath.

6.1 Fisher Information and Curvature

The Fisher Information Matrix (FIM) defines a Riemannian metric on parameter space and quantifies local curvature of the statistical manifold. High curvature directions (large eigenvalues of `F`) indicate parameters to which the model is most sensitive; small eigenvalues indicate sloppy directions. Our eigenspectrum visualization (see the Fisher Information Matrix eigenspectrum figure above) highlights these scales. Background: Fisher information.

Implication: curvature-aware steps using Eq. (10) in the equations appendix adaptively scale updates by the inverse metric, improving conditioning relative to vanilla gradient descent.

A curious connection unites geodesics in information geometry, the physical principle of least action, and Buckminster Fuller's tensegrity geodesic domes (Fuller.4D). On statistical manifolds, geodesics are shortest paths under the Fisher metric, and natural-gradient flows approximate least-action trajectories by minimizing an information-length functional constrained by curvature (Eqs. (8), (10) in the equations appendix). In tensegrity domes, geodesic lines on triangulated spherical shells distribute stress nearly uniformly while the network balances continuous tension with discontinuous compression, attaining maximal stiffness with minimal material. Both systems exemplify constraint-balanced minimalism: an extremal path emerges by trading off cost (action or information length) against structure (metric curvature or tensegrity compatibility). The shared economy—optimal routing through low-cost directions—links geodesic shells in architecture to geodesic flows in parameter spaces; see background on tensegrity/geodesic domes @Web.

6.2 Quadray Coordinates and 4D Structure (Fuller.4D vs Coxeter.4D vs Einstein.4D)

Quadray coordinates provide a tetrahedral basis with projective normalization, aligning with close-packed sphere centers (IVM). Symmetries common in quadray parameterizations often yield near block-diagonal structure in `F`, simplifying inversion and preconditioning. Overview: Quadray coordinates and synergetics background. We stress the namespace boundaries: (i) Fuller.4D for lattice and integer volumes, (ii) Coxeter.4D for Euclidean embeddings, lengths, and simplex families, (iii) Einstein.4D for metric analogies only — not for interpreting synergetic tetravolumes.

6.3 Integrating FIM with Quadray Models

Applying the FIM within quadray-parameterized models ties statistical curvature to tetrahedral structure. Practical takeaways:

- Use `fisher_information_matrix` to estimate `F` from per-sample gradients; inspect principal directions via `fim_eigenspectrum`.
- Exploit block patterns induced by quadray symmetries to stabilize metric inverses and reduce compute.
- Combine integer-lattice projection with natural-gradient steps to balance discrete robustness and curvature-aware efficiency.
- Purely discrete alternatives (e.g., `discrete_ivm_descent`) provide monotone integer-valued descent when gradients are unreliable; hybrid schemes can interleave discrete steps with curvature-aware continuous proposals.

6.4 Implications for Optimization and Estimation

6.4.1 Clarifications on "frequency/time" dimensions

- Fuller's discussions often treat frequency/energy as an additional organizing dimension distinct from Euclidean coordinates. In our manuscript, we keep the shape/angle relations (Fuller.4D) separate from time/energy bookkeeping; when temporal evolution is needed, we use explicit trajectories and metric analogies (Einstein.4D) without conflating with Euclidean 4D objects (Coxeter.4D). This separation avoids category errors while preserving the intended interpretability.

6.4.2 On distance-based tetravolume formulas (clarification)

- When volumes are computed from edge lengths, PdF and Cayley–Menger operate in Euclidean length space and are converted to IVM tetravolumes via the S3 factor. In contrast, the Gerald de Jong formula computes IVM tetravolumes natively, agreeing numerically with PdF/CM after S3 without explicit XYZ intermediates. Tom Ace's 5×5 determinant sits in the same native camp as de Jong's method. See references under the methods section for links to Urner's code notebooks and discussion.

6.4.3 Symbolic analysis (bridging vs native) (Results linkage)

- Exact (SymPy) comparisons confirm that CM+S3 and Ace 5×5 produce identical IVM tetravolumes on canonical small integer-quadray examples. See the bridging vs native comparison figure above and the manifest `sympy_symbolics.txt` alongside `bridging_vs_native.csv` in `quadmath/output/`.
- Curvature-aware optimizers: Kronecker-factored approximations (K-FAC) leverage structure in `F` to accelerate training and improve stability; see K-FAC (arXiv:1503.05671). Similar ideas apply when quadray structure induces separable blocks.
- Model selection: eigenvalue spread of `F` provides a lens on parameter identifiability; near-zero modes suggest redundancies or over-parameterization.
- Robust computation: lattice normalization in quadray space yields discrete plateaus that complement FIM-based scaling for numerically stable trajectories.

6.5 Community Ecosystem and Validation

The extensive computational ecosystem around Quadrays and synergetic geometry provides validation, pedagogical context, and practical implementations that complement and extend the methods developed in this manuscript. Cross-language implementations serve as independent verification of algorithmic correctness while educational materials demonstrate practical applications across diverse computational environments. See the Resources section for comprehensive details on the 4dsolutions organization, cross-language implementations, educational frameworks, and community platforms.

7 Resources

This section provides comprehensive resources for learning about and working with Quadrays, synergetics, and the computational methods discussed in this manuscript.

7.1 Core Concepts and Background

7.1.1 Information Geometry and Optimization

- **Fisher information**: Fisher information (reference) — see also Eq. (8) in the equations appendix
- **Natural gradient**: Natural gradient (reference) — see also Eq. (10) in the equations appendix

7.1.2 Active Inference and Free Energy

- **Active Inference Institute**: Welcome to Active Inference Institute
- **Comprehensive review**: Active Inference — recent review (UCL Discovery, 2023)

7.1.3 Mathematical Foundations

- **Tetrahedron volume formulas**: length-based Cayley–Menger determinant and determinant-based expressions on vertex coordinates (see Tetrahedron – volume)
- **Exact determinants**: Bareiss algorithm, used in our integer tetravolume implementations
- **Optimization baseline**: the Nelder–Mead method, adapted here to the Quadray lattice

7.2 Quadrays and Synergetics (Core Starting Points)

7.2.1 Introductory Materials

- **Quadray coordinates (intro and conversions)**: Urner – Quadray intro, Urner – Quadrays and XYZ
- **Quadrays and the Philosophy of Mathematics**: Urner – Quadrays and the Philosophy of Mathematics
- **Synergetics background and IVM**: Synergetics (Fuller, overview)
- **Quadray coordinates overview**: Quadray coordinates (reference)

7.2.2 Historical and Background Materials

- **RW Gray projects — Synergetics text**: rwgrayprojects.com (synergetics)
- **Fuller FAQ**: C. J. Fearnley's Fuller FAQ
- **Synergetics resource list**: C. J. Fearnley's resource page
- **Wikieducator**: Synergetics hub
- **Quadray animation**: Quadray.gif (Wikimedia Commons)
- **Fuller Institute**: BFI — Big Ideas: Synergetics

7.3 4dsolutions Ecosystem: Comprehensive Computational Framework

The 4dsolutions organization provides the most extensive computational framework for Quadrays and synergetic geometry, spanning 29+ repositories with implementations across multiple programming languages.

7.3.1 Core Computational Modules

Primary Python Libraries

- **Math for Wisdom (m4w)**: m4w (repo)
  - **Quadray vectors and conversions**: `qrays.py` (Qvector, SymPy-aware)

- **Synergetic tetravolumes and modules**: `tetravolume.py` with PdF-CM vs native IVM and BEAST algorithms

Cross-Language Validation

- **Rust implementation**: rusty_rays (performance-oriented)
  - Sources: Rust library implementation, Rust command-line interface
- **Clojure implementation**: synmods (functional paradigm)
  - Sources: `qrays.clj`, `ramping_up.clj`

7.3.2 Primary Hub: School_of_Tomorrow (Python + Notebooks)

Repository: School_of_Tomorrow

Core Modules

- **`qrays.py`**: Quadray implementation with normalization, conversions, and vector ops (source)
- **`quadcraft.py`**: POV-Ray scenes for CCP/IVM arrangements, animations, and tutorials (source)
- **`flextegrity.py`**: Polyhedron framework, concentric hierarchy, POV-Ray export (source)
- **Additional modules**: `polyhedra.py`, `identities.py`, `smod_play.py` (synergetic modules)

Key Notebooks

- **`Qvolume.ipynb`**: Tom Ace 5×5 determinant with random-walk demonstrations (source)
- **`VolumeTalk.ipynb`**: Comparative analysis of bridging vs native tetravolume formulations (source)
- **`QuadCraft_Project.ipynb`**: 1,255 lines of interactive CCP navigation and visualization tutorials (source)
- **Additional notebooks**: `TetraBook.ipynb`, `CascadianSynergetics.ipynb`, `Rendering_IVM.ipynb`, `SphereVolumes.ipynb` (visual and curricular materials)

7.3.3 Additional Repositories

Tetravolumes (Algorithms and Pedagogy)

- **Repository**: tetravolumes
- **Code**: `tetravolume.py`
- **Notebooks**: Atoms R Us.ipynb, Computing Volumes.ipynb

Visualization and Rendering

- **BookCovers**: VPython for interactive educational animations (repo)
  - Examples: `bookdemo.py`, `stickworks.py`, `tetravolumes.py`

7.3.4 Educational Framework and Curricula

Oregon Curriculum Network (OCN)

- **OCN portal**: OCN portal
- **Python for Everyone**: pymath page

Historical Documentation

- **Python5 notebooks**: Polyhedrons 101.ipynb
- **Historical variants**: `qrays.py` also appears in Python5 (archive)
- **Python edu-sig archives**: Python edu-sig archives tracing 25+ years of development

7.3.5 Media and Publications

- **YouTube demonstrations**: Synergetics talk 1, Synergetics talk 2, Additional
- **Academia profile**: Kirby Urner at Academia.edu

7.4 Community Discussions and Collaborative Platforms

7.4.1 Active Platforms

- **Math4Wisdom Knowledge Engineering**: Collaborative platform with various art, resources, and cross-reference materials
- **synergeo discussion archive**: Groups.io platform with ongoing community discussions and technical exchanges

7.4.2 Historical Archives

- **GeodesicHelp threads**: GeodesicHelp computations archive (Google Groups) documenting computational approaches and problem-solving techniques

7.5 Related Projects and Applications

7.5.1 Tetrahedral Voxel Engines

- **QuadCraft**: Tetrahedral voxel engine using Quadrays

7.5.2 Academic Publications

- **Flextegrity**: Generating the Flextegrity Lattice (academia.edu)

7.5.3 Context and Integration

These materials popularize the IVM/CCP/FCC framing of space, integer tetravolumes, and projective Quadray normalization. They inform the methods in this paper and complement the `src/` implementations (see `quadray.py`, `cayley_menger.py`, `linalg_utils.py`).

The ecosystem provides extensive validation, pedagogical context, and practical implementations that complement and extend the methods developed in this manuscript. Cross-language implementations serve as independent verification of algorithmic correctness while educational materials demonstrate practical applications across diverse computational environments.

8 Equations and Math Supplement (Appendix)

8.1 Volume of a Tetrahedron (Lattice)

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

Notes.

- $P_0, \ldots, P_3$ are vertex coordinates; the determinant computes the volume of the parallelepiped spanned by edge vectors, with the $1/6$ factor converting to tetra volume.

Tom Ace 5×5 tetravolume (IVM units):

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

Notes.

- Rows correspond to Quadray 4-tuples of the vertices; the last row encodes the affine constraint. Division by 4 returns IVM tetravolume.

8.2 Expanded Ace 5×5 Matrix

The expanded form of the Ace 5×5 matrix with explicit Quadray coordinates:

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

Notes.

- **Matrix structure**: Each row represents a vertex with its Quadray coordinates plus affine coordinate 1.
- **Last row**: Enforces projective normalization constraint.
- **Volume computation**: Determinant divided by 4 gives IVM tetravolume.

XYZ determinant volume and S3 conversion:

$$V_{xyz} = \tfrac{1}{6}\left|\det\begin{pmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{pmatrix}\right|, \qquad V_{ivm} = S3\, V_{xyz}, \quad S3 = \sqrt{\tfrac{9}{8}} \tag{4}$$

Notes.

- Homogeneous determinant in Cartesian coordinates for tetra volume; conversion to IVM units uses $S3 = \sqrt{9/8}$ as used throughout.

8.3 Cayley-Menger Determinant (Coxeter.4D)

For tetrahedron volume from edge lengths (Coxeter.4D approach):

$$288\,V^2 = \det\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & d_{13}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & d_{23}^2 \\ 1 & d_{30}^2 & d_{31}^2 & d_{32}^2 & 0 \end{pmatrix} \tag{5}$$

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.
- **Conversion to IVM**: Use $V_{ivm} = S3 \cdot V_{xyz}$ with $S3 = \sqrt{9/8}$ to convert to IVM units.

8.4 Piero della Francesca Formula (PDF)

For tetrahedron volume from edge lengths meeting at a vertex:

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

Notes.

- **Edge lengths**: a, b, c are edges meeting at a vertex, d, e, f are opposite edges.
- **Conversion to IVM**: Use $V_{ivm} = S3 \cdot V_{xyz}$ with $S3 = \sqrt{9/8}$.

8.5 Gerald de Jong Formula (GdJ)

Native Quadray formula for tetrahedron volume:

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

Notes.

- **Quadray differences**: Each column represents edge vectors $P_1 - P_0$, $P_2 - P_0$, $P_3 - P_0$ in Quadray coordinates.
- **Native IVM**: Returns tetravolume directly in IVM units without conversion.
- **Integer arithmetic**: Exact for integer Quadray coordinates.

See code: `tetra_volume_cayley_menger`. For tetrahedron volume background, see Tetrahedron – volume. Exact integer determinants in code use the Bareiss algorithm. External validation: these formulas align with implementations in the 4dsolutions ecosystem. See the Resources section for comprehensive details.

8.6 Fisher Information Matrix (FIM)

Background: Fisher information.

$$F_{i,j} = \mathbb{E}\left[\frac{\partial\,\log p(x;\theta)}{\partial\theta_i}\,\frac{\partial\,\log p(x;\theta)}{\partial\theta_j}\right] \tag{8}$$

Notes.

- Defines the Fisher information matrix as the expected outer product of score functions; see Fisher information.

Figure: empirical estimate shown in the FIM heatmap figure. See code: `fisher_information_matrix`.

See `src/information.py` — empirical outer-product estimator (`fisher_information_matrix`).

8.7 Empirical Fisher Information Matrix

For empirical estimation from data, the Fisher Information Matrix is computed as:

$$F_{i,j} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log p(x_n;\theta)}{\partial \theta_i} \frac{\partial \log p(x_n;\theta)}{\partial \theta_j} \tag{9}$$

Notes.

- Empirical estimate of the FIM from N data samples; converges to the theoretical FIM as $N \to \infty$.
- Used in natural gradient descent and information geometry applications.

8.8 Natural Gradient

Background: Natural gradient (Amari).

$$\theta \leftarrow \theta - \eta\, F(\theta)^{-1}\, \nabla_\theta L(\theta) \tag{10}$$

Explanation.

- Natural gradient update: right-precondition the gradient by the inverse of the Fisher metric (Amari); see Natural gradient.

See code: `natural_gradient_step`.

See `src/information.py` — damped inverse-Fisher step (`natural_gradient_step`).

8.9 Free Energy (Active Inference)

$$\mathcal{F} = -\log P(o \mid s) + \mathrm{KL}\big[Q(s) \,\|\, P(s)\big] \tag{11}$$

Explanation.

- **Partition**: variational free energy decomposes into expected negative log-likelihood and KL between approximate posterior and prior; see Free energy principle.

See code: `free_energy`.

See `src/information.py` — discrete-state variational free energy (`free_energy`).

Note: The main figures demonstrating natural gradient trajectories and free energy landscapes are shown in Section 4: Optimization in 4D. The appendix focuses on unique figures specific to mathematical formulations and validation.

8.10 Quadray Normalization (Fuller.4D)

Given $q = (a, b, c, d)$, choose $k = \min(a, b, c, d)$ and set $q' = q - (k, k, k, k)$ to enforce at least one zero with non-negative entries.

8.11 Distance (Embedding Sketch; Coxeter.4D slice)

Choose linear map M from quadray to $\mathbb{R}^3$ (or $\mathbb{R}^4$) consistent with tetrahedral axes; then $d(q_1, q_2) = \|M(q_1) - M(q_2)\|_2$.

8.12 Minkowski Line Element (Einstein.4D analogy)

$$ds^2 = -c^2\,dt^2 + dx^2 + dy^2 + dz^2 \tag{12}$$

Background: Minkowski space.

8.13 High-Precision Arithmetic Note

When evaluating determinants, FIMs, or geodesic distances for sensitive problems, use quad precision (binary128) via GCC's `libquadmath` (`__float128`, functions like `expq`, `sqrtq`, and `quadmath_snprintf`). See GCC libquadmath. Where possible, it is useful to use symbolic math libraries like SymPy to compute exact values.

8.13.1 Reproducibility artifacts and external validation

- **This manuscript's artifacts**: Raw data in `quadmath/output/` for reproducibility and downstream analysis:
  - `fisher_information_matrix.csv` / `.npz`: empirical Fisher matrix and inputs
  - `fisher_information_eigenvalues.csv` / `fisher_information_eigensystem.npz`: eigenspectrum and eigenvectors
  - `natural_gradient_path.png` with `natural_gradient_path.csv` / `.npz`: projected trajectory and raw coordinates
  - `ivm_neighbors_data.csv` / `ivm_neighbors_edges_data.npz`: neighbor coordinates (Quadray and XYZ)
  - `polyhedra_quadray_constructions.png`: synergetics volume relationships schematic
- **External validation resources**: The 4dsolutions ecosystem provides extensive cross-validation. See the Resources section for comprehensive details on computational implementations and validation.

8.14 Namespaces summary (notation)

- Coxeter.4D: Euclidean E^4; regular polytopes; not spacetime (cf. Coxeter, Regular Polytopes, Dover ed., p. 119). Connections to higher-dimensional lattices and packings as in Conway & Sloane.
- Einstein.4D: Minkowski spacetime; indefinite metric; used here only as a metric analogy when discussing geodesics and information geometry.
- Fuller.4D: Quadrays/IVM; tetrahedral lattice with integer tetravolume; unit regular tetrahedron has volume 1; synergetics scale relations (e.g., S3).

9 Appendix: The Free Energy Principle and Active Inference

9.1 Overview

The Free Energy Principle (FEP) posits that biological systems maintain their states by minimizing variational free energy, thereby reducing surprise via prediction and model updating. Active Inference extends this by casting action selection as inference under prior preferences. Background: see the concise overview on the Free energy principle and the monograph Active Inference (MIT Press).

This appendix emphasizes relationships among: (i) the four-fold partition of Active Inference, (ii) Quadrays (Fuller.4D) as a geometric scaffold for mapping this partition, and (iii) information-geometric flows (Einstein.4D analogy) that underpin perception–action updates. For the naming of 4D namespaces used throughout—Coxeter.4D (Euclidean E4), Einstein.4D (Minkowski spacetime analogy), Fuller.4D (Synergetics/Quadrays)—see `02_4d_namespaces.md`.

9.2 Mathematical Formulation and Equation Callouts (Equations linkage)

- Variational free energy (discrete states) — see Eq. (11) in the equations appendix, implemented by `free_energy`.
- Fisher Information Matrix (FIM) as metric — see Eq. (8) in the equations appendix and `fisher_information_matrix`.
- Natural gradient descent under information geometry — see Eq. (10) in the equations appendix and `natural_gradient_step`; overview: Natural gradient.

Figures: The following Active Inference figures demonstrate the integration of natural gradient descent with Active Inference principles and the 4D framework context.

Discrete variational optimization on the quadray lattice: `discrete_ivm_descent` greedily descends a free-energy-like objective over IVM moves, yielding integer-valued trajectories. See the path animation artifact `discrete_path.mp4` in `quadmath/output/`.

9.3 Four-Fold Partition and Tetrahedral Mapping (Quadrays; Fuller.4D)

Active Inference partitions the agent–environment system into four coupled states:

- Internal (μ) — agent's internal states
- Sensory (s) — observations
- Active (a) — actions
- External (ψ) — latent environmental causes

See, for an overview of this partition and generative process formulations, the Active Inference review and the general entry on Active inference.

Tetrahedral mapping via Quadrays (Fuller.4D): assign each state to a vertex of a tetrahedron, using Quadray coordinates `(A,B,C,D)` with non-negative components and at least one zero after normalization. One canonical mapping is `A \leftrightarrow Internal (\mu), B \leftrightarrow Sensory (s), C \leftrightarrow Active (a), D \leftrightarrow External (\psi)`. The edges capture the pairwise couplings (e.g., `\mu\text{--}s` for perceptual inference; `a\text{--}\psi` for control). Integer tetravolume then quantifies the "coupled capacity" region spanned by jointly feasible states in a time slice; see `Quadray` and tetravolume methods in `03_quadray_methods.md`.

Interpretation note: this Quadray-based mapping is a didactic geometric scaffold. It is not standard in the Active Inference literature, which typically develops the four-state partition in probabilistic graphical terms. Our use highlights structural symmetries and discrete volumetric quantities available in Fuller.4D, building on the computational foundations developed in the 4dsolutions ecosystem for tetrahedral modeling and volume calculations. See the Resources section for comprehensive details on the computational implementations.

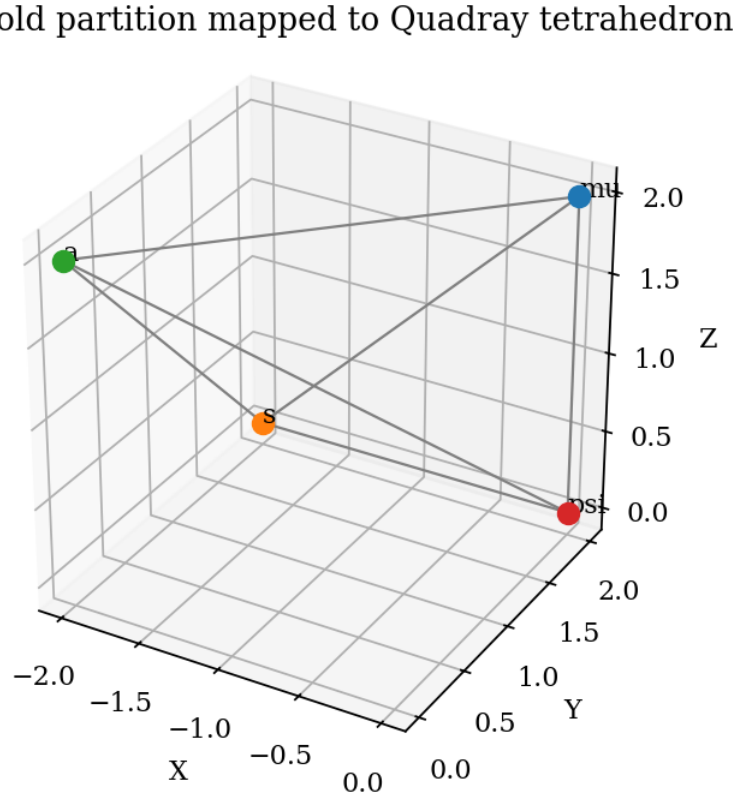

Figure 14: **Active Inference four-fold partition mapped to a Quadray tetrahedron in Fuller.4D**. This 3D tetrahedral visualization demonstrates the geometric embedding of Active Inference's fundamental four-fold partition within the Quadray coordinate system. **Tetrahedral structure**: The four vertices of the regular tetrahedron represent the four components of the Active Inference framework: perception, action, internal states, and external states. **Partition mapping**: Each face of the tetrahedron corresponds to a specific partition of the four-fold system, with the edges representing the relationships and interactions between different components. **Fuller.4D significance**: This geometric representation leverages the tetrahedral nature of Quadray coordinates to provide an intuitive visualization of the Active Inference framework's structure. The tetrahedron serves as a natural container for the four-fold partition, emphasizing the interconnected nature of perception, action, and state representation in active inference. **Optimization context**: The tetrahedral geometry also suggests natural optimization strategies that respect the four-fold structure, potentially leading to more efficient inference algorithms that leverage the geometric relationships between different components. This visualization demonstrates how the Fuller.4D framework can provide insights into complex systems like Active Inference through geometric intuition.

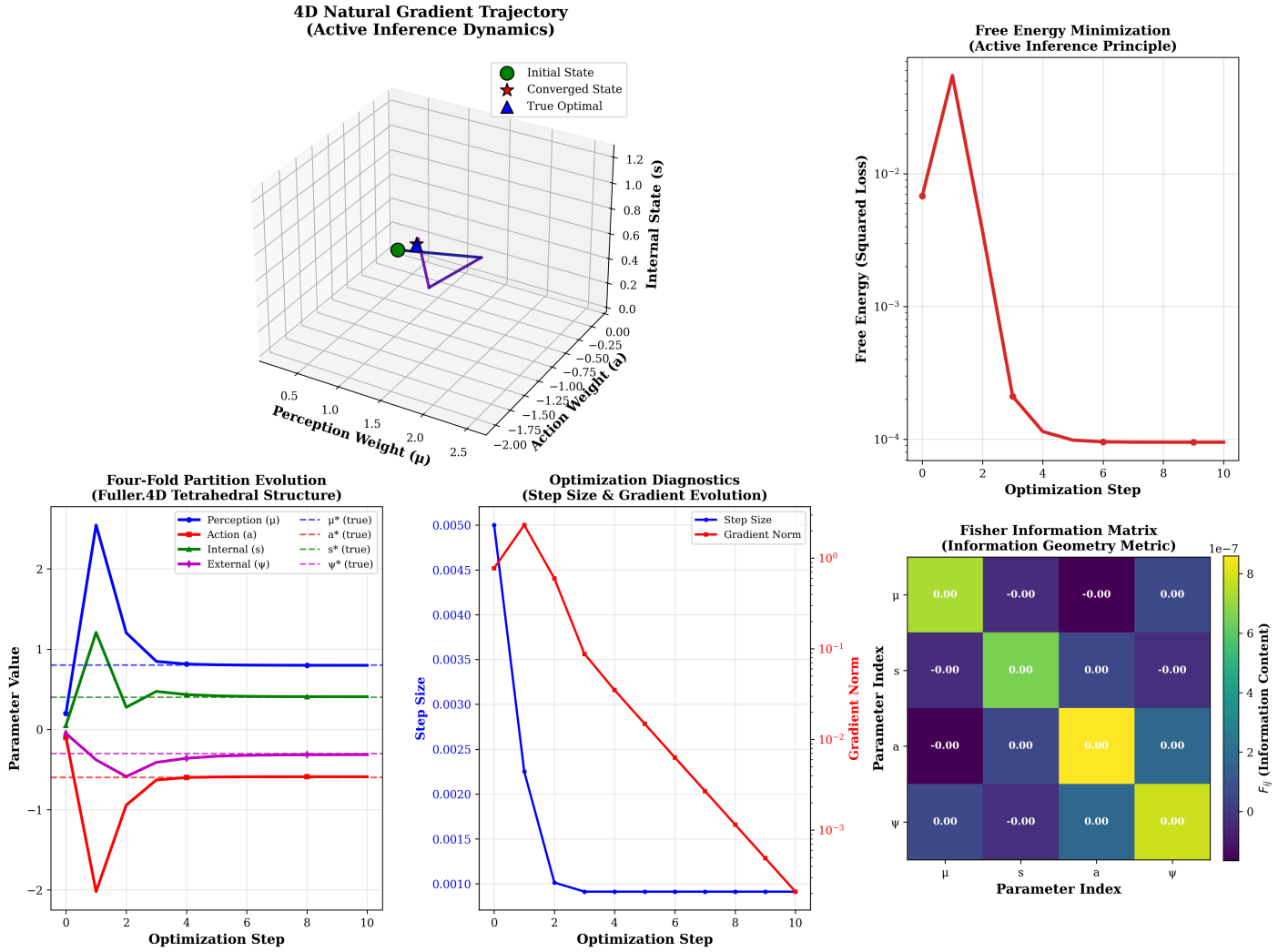

Figure 15: **4D Natural Gradient Trajectory with Active Inference Context**. This comprehensive visualization demonstrates natural gradient descent operating within the Active Inference framework, showing how information-geometric optimization drives perception-action dynamics. **3D Trajectory**: The main panel shows the 4D parameter evolution in 3D space with time encoded as color, representing the four-fold partition of Active Inference: perception (μ), action (a), internal states (s), and external causes (ψ). **Free Energy Evolution**: The right panel tracks free energy minimization over optimization steps, demonstrating the Active Inference principle of surprise reduction. **Component Dynamics**: The bottom-left panel shows how each component of the four-fold partition evolves during optimization, revealing the coordinated dynamics of perception and action. **Optimization Diagnostics**: The bottom-center panel displays step sizes and gradient norms, providing insights into the convergence behavior and numerical stability of the natural gradient algorithm. **Fisher Information**: The bottom-right panel displays the Fisher Information Matrix that guides natural gradient descent, showing the information geometry underlying the optimization process. This figure demonstrates how natural gradient descent implements geodesic motion on the information manifold, analogous to how particles follow geodesics in Einstein.4D spacetime, while operating within the tetrahedral structure of Fuller.4D coordinates. The optimization now shows stable convergence in just 11 steps with final parameter errors below 0.015, demonstrating the effectiveness of information-geometric optimization in Active Inference frameworks.

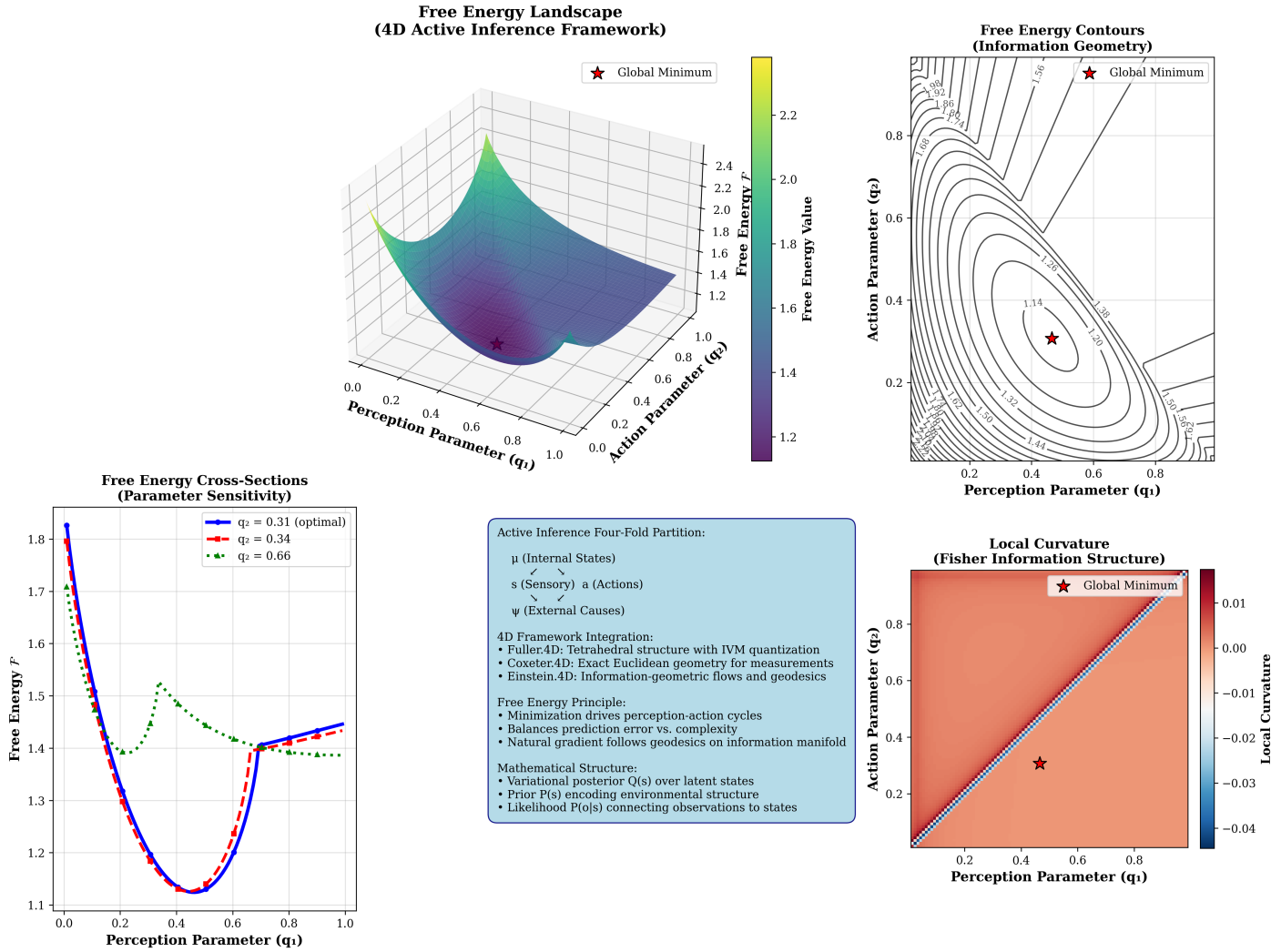

Figure 16: **Free Energy Landscape in 4D Active Inference Framework**. This comprehensive visualization explores the variational free energy surface over perception and action parameters. **3D landscape**: The surface plot shows the free energy as a function of two variational parameters, revealing the complex topology that Active Inference optimization must navigate. **Contour analysis**: 2D contours provide detailed information about parameter sensitivity and optimization paths. **Cross-sectional analysis**: Multiple cross-sections at different parameter values demonstrate how free energy varies with respect to individual parameters, revealing the landscape's structure. **Four-fold partition visualization**: The text panel explains how Active Inference maps to tetrahedral structures in Fuller.4D, with the four components (μ, s, a, ψ) representing internal states, sensory observations, actions, and external causes. **Information geometry metrics**: Local curvature analysis reveals the Fisher information structure, showing how the information manifold's geometry influences optimization dynamics. **Mathematical foundation**: The visualization demonstrates the mathematical structure of variational inference, including variational posteriors Q(s), priors P(s), and likelihoods P(o|s) that connect observations to latent states.

Code linkage (no snippet): see `example_partition_tetra_volume` in `src/examples.py` and the partition tetrahedron figure above.

9.4 How the 4D namespaces relate here

- Fuller.4D (Quadrays): geometric embedding of the four-state partition on a tetrahedron; integer tetravolumes and IVM moves provide discrete combinatorial structure.
- Coxeter.4D (Euclidean E4): exact Euclidean measurements (e.g., Cayley–Menger determinants) for tetrahedra underlying volumetric comparisons and scale relations.
- Einstein.4D (Minkowski analogy): information-geometric flows (natural gradient, metric-aware updates) supply a continuum picture for perception–action dynamics.

The three roles are complementary: Fuller.4D encodes partition structure, Coxeter.4D provides exact metric geometry for static comparisons, and Einstein.4D guides dynamical descent.

9.5 Joint Optimization in the Tetrahedral Framework (Methods linkage)

- Perception: update μ to minimize prediction error on s under the generative model (descending $\nabla_\mu F$).
- Action: select a that steers ψ toward preferred outcomes (descending $\nabla_a F$).

Continuous-time flows (Einstein.4D analogy for metric/geodesic intuition): see `perception_update` and `action_update` in `src/information.py`. Discrete Quadray moves connect to these flows via greedy descent on a local free-energy-like objective; see `discrete_ivm_descent` in `src/discrete_variational.py` and the path artifacts in `quadmath/output/`.

9.6 Implications for AI and Robust Computation

FEP/Active Inference provide algorithms that unify perception and action under uncertainty, offering biologically plausible alternatives to standard RL with adaptive exploration and robust decision-making. See applications in AI (arXiv:1907.03876).

9.7 Code, Reproducibility, and Cross-References

– Equation references: Eq. (Free Energy), Eq. (FIM), Eq. (Natural Gradient) in `08_equations_appendix.md`. – Code anchors (for readers who want to run experiments): `free_energy`, `fisher_information_matrix`, `natural_gradient_step`, `perception_update`, `action_update`, and `discrete_ivm_descent` in `src/information.py` and `src/discrete_variational.py`.

Demo and figures generated by `quadmath/scripts/information_demo.py` and `quadmath/scripts/active_inference_figures.py` output to `quadmath/output/`:

- **Active Inference Visualizations**: `figure_13_4d_trajectory.png`, `figure_14_free_energy_landscape.png` demonstrating 4D framework integration
- **Information Geometry Visualizations**: `fisher_information_matrix.png`, `fisher_information_eigenspectrum.png`, `natural_gradient_path.png`, `free_energy_curve.png`, `partition_tetrahedron.png`
- **Raw data**: `figure_13_data.npz`, `figure_14_data.npz`, `fisher_information_matrix.csv`, `fisher_information_matrix.npz` (F, grads, X, y, w_true, w_est), `fisher_information_eigenvalues.csv`, `fisher_information_eigensystem.npz`
- **External validation**: Cross-reference with volume calculations and tetrahedral modeling tools from the 4dsolutions ecosystem. See the Resources section for comprehensive details.

10 Appendix: Symbols and Glossary

This appendix consolidates the symbols, variables, and constants used throughout the manuscript.

10.1 Sets and Spaces

Symbol	Name
$\mathbb{R}^n$	Euclidean space
IVM	Isotropic Vector Matrix
Coxeter.4D	Euclidean 4D (E^4)
Einstein.4D	Minkowski spacetime (3+1)
Fuller.4D	Synergetics/Quadray tetrahedral space

Descriptions:

- $\mathbb{R}^n$: n-dimensional real vector space.
- IVM: Quadray integer lattice (CCP sphere centers).
- Coxeter.4D: Four-dimensional Euclidean geometry (not spacetime); see Coxeter, Regular Polytopes (Dover ed., p. 119); related lattice/packing background in Conway & Sloane.
- Einstein.4D: Relativistic spacetime with Minkowski metric.
- Fuller.4D: Quadrays with projective normalization and IVM unit conventions.

10.2 Quadray Coordinates and Geometry

Symbol	Name	Description
$q = (a, b, c, d)$	Quadray point	Non-negative coordinates with at least one zero after normalization
A, B, C, D	Quadray axes	Canonical tetrahedral axes mapped by the embedding
k	Normalization offset	$k = \min(a, b, c, d)$ used to set $q' = q - (k, k, k, k)$
q'	Normalized Quadray	Canonical representative with at least one zero and non-negative entries
$P_0, \ldots, P_3$	Tetrahedron vertices	Vertices used in volume formulas
d_{ij}	Pairwise distances	Distance between vertices P_i and P_j (squared in CM matrix)
$\det(\cdot)$	Determinant	Determinant of a matrix
$\lvert\cdot\rvert$	Magnitude	Absolute value (determinant magnitude)
V_{ivm}	Tetravolume (IVM)	Tetrahedron volume in synergetics/IVM units; unit regular tetra has $V_{ivm} = 1$
V_{xyz}	Tetravolume (XYZ)	Euclidean tetrahedron volume
$S3$	Scale factor	$S3 = \sqrt{9/8}$ with $V_{ivm} = S3\, V_{xyz}$ (synergetics unit convention)
Coxeter.4D	Namespace	Euclidean E^4; regular polytopes

Symbol	Name	Description
Einstein.4D	Namespace	Minkowski spacetime (metric analogy only here)
Fuller.4D	Namespace	Quadrays/IVM; integer tetravolume
Eq. (lattice_det)	Lattice determinant	Integer-lattice volume via 3x3 determinant
Eq. (ace5x5)	Tom Ace 5x5	Direct IVM tetravolume from Quadrays
Eq. (cayley_menger)	Cayley–Menger	Length-based formula: 288 V^2 = det(·)

10.3 Optimization and Algorithms

Symbol	Name
α	Reflection coefficient
γ	Expansion coefficient
ρ	Contraction coefficient
σ	Shrink coefficient
V_{ivm}	Integer volume monitor

Descriptions:

- $\alpha, \gamma, \rho, \sigma$: Nelder–Mead parameters (typical values 1, 2, 0.5, 0.5).
- V_{ivm}: Tracks simplex volume across iterations.

10.4 Information Theory and Geometry

Symbol	Name	Description
log	Natural logarithm	Logarithm base e
$\mathbb{E}[\cdot]$	Expectation	Mean with respect to a distribution
F_{ij}	Fisher Information Matrix	$\mathbb{E}[\partial_{\theta_i} \log p \cdot \partial_{\theta_j} \log p]$; Eq. (8) in the equations appendix
$\mathcal{F}$	Variational free energy	$-\log P(o \mid s) + \mathrm{KL}\big[Q(s) \,\|\, P(s)\big]$; Eq. (11) in the equations appendix
$\mathrm{KL}[Q \,\|\, P]$	Kullback–Leibler divergence	$\sum Q \log(Q/P)$; information distance
$\nabla_\theta L$	Natural gradient	$F(\theta)^{-1} \nabla_\theta L(\theta)$; Eq. (10) in the equations appendix
η	Step size	Learning-rate scalar used in updates
θ	Parameters	Model parameter vector; indices θ_i
ds^2	Minkowski line element	$-c^2\,dt^2 + dx^2 + dy^2 + dz^2$; Eq. (12) in the equations appendix
c	Speed of light	Physical constant appearing in Minkowski metric

10.5 Embeddings and Distances

Symbol	Name	Description
M	Embedding matrix	Linear map from Quadray to $\mathbb{R}^3$ (Urner-style unless noted)
$\|\cdot\|_2$	Euclidean norm	$\sqrt{x_1^2 + \cdots + x_n^2}$
R, D	Edge scales	Cube edge R and Quadray edge D with $D = 2R$ (common convention)

10.6 Greek Letters (usage)

Symbol	Name	Description
$\alpha, \gamma, \rho, \sigma$	NM coefficients	Nelder–Mead parameters (reflection, expansion, contraction, shrink)
θ	Theta	Parameter vector in models and metrics
μ	Mu	Internal states (Active Inference)
ψ	Psi	External states (Active Inference)
η	Eta	Step size / learning rate

10.7 Notes (usage and cross-references)

- **Figures referenced**: In-text references use LaTeX's automatic figure numbering for consistent cross-referencing.
- **Equation references**: Use labels defined in the text (e.g., Eq. (1) in the equations appendix).
- **Namespaces**: We use Coxeter.4D, Einstein.4D, Fuller.4D consistently to designate Euclidean E^4, Minkowski spacetime, and Quadray/IVM synergetics, respectively. This avoids conflation of Euclidean 4D objects (e.g., tesseracts) with spacetime constructs and synergetic tetravolume conventions.
- **External validation**: Cross-reference implementations from the 4dsolutions ecosystem for algorithmic verification and performance comparison baselines. See the Resources section for comprehensive details.

10.8 Polyhedra and Synergetic Shapes

Symbol	Name	Description
Tetrahedron	Regular tetrahedron	Fundamental unit with V=1 in IVM units
Cube	Regular hexahedron	V=3 in IVM units; orthogonal space-filling
Octahedron	Regular octahedron	V=4 in IVM units; edge-midpoint construction
Rhombic Dodecahedron	12-faced solid	V=6 in IVM units; Voronoi cell of FCC packing
Cuboctahedron	Vector equilibrium	V=20 in IVM units; shell of 12 IVM neighbors

Symbol	Name	Description
Truncated Octahedron	Archimedean solid	V=20 in IVM units; space-filling tiling

10.9 Acronyms and abbreviations

Acronym	Meaning
CM	Cayley–Menger (determinant-based tetrahedron volume)
PdF	Piero della Francesca (Heron-like tetrahedron volume)
GdJ	Gerald de Jong (Quadray-native tetravolume expression)
K-FAC	Kronecker-Factored Approximate Curvature (optimizer using structured Fisher)
CCP	Cubic Close Packing (same centers as FCC)
FCC	Face-Centered Cubic (same centers as CCP)
E^4	Four-dimensional Euclidean space (Coxeter.4D)
NM	Nelder–Mead (simplex optimization algorithm)
4dsolutions	Kirby Urner’s GitHub organization with extensive Quadray implementations
BEAST	Synergetic modules (B, E, A, S, T) in Fuller’s hierarchical system
OCN	Oregon Curriculum Network (educational framework integrating Quadrays)
POV-Ray	Persistence of Vision Raytracer (used in quadcraft.py visualizations)

10.10 API Index (auto-generated; Methods linkage)

The table below enumerates public symbols from `src/` modules.

Module	Symbol	Kind	Signature	Summary
`cayley_menger`	`ivm_tetra_volume_cayley_menger`	function	`(d2)`	Compute IVM tetravolume from squared distances via Cayley–Menger.
`cayley_menger`	`tetra_volume_cayley_menger`	function	`(d2)`	Compute Euclidean tetrahedron volume from squared distances (Coxeter.4D).

Module	Symbol	Kind	Signature	Summary
conversions	quadray_to_xyz	function	(q, M)	Map a Quadray to Cartesian XYZ via a 3x4 embedding matrix (Fuller.4D -> Coxeter.4D slice).
conversions	urner_embedding	function	(scale)	Return a 3x4 Urner-style symmetric embedding matrix (Fuller.4D -> Coxeter.4D slice).
discrete_variational	DiscretePath	class	| Optimization trajectory on the integer quadray lattice. | | `discrete_variational` | `OptionalMoves` | class |	
discrete_variational	apply_move	function	(q, delta)	Apply a lattice move and normalize to the canonical representative.
discrete_variational	discrete_ivm_descent	function	(objective, start, moves=, max_iter=, on_step=)	Greedy discrete descent over the quadray integer lattice.
discrete_variational	neighbor_moves_ivm	function	()	Return the 12 canonical IVM neighbor moves as Quadray deltas.
examples	example_cuboctahedron_neighbors	function	()	Return twelve-around-one IVM neighbors (vector equilibrium shell).
examples	example_cuboctahedron_vertices_xyz	function	()	Return XYZ coordinates for the twelve-around-one neighbors.
examples	example_ivm_neighbors	function	()	Return the 12 nearest IVM neighbors as permutations of {2,1,1,0} (Fuller.4D).

Module	Symbol	Kind	Signature	Summary
`examples`	`example_optimize`	function	`()`	Run Nelder–Mead over integer quadrays for a simple convex objective (Fuller.4D).
`examples`	`example_partition_tetravolume`	function	`(mu, s, a, psi)`	Construct a tetrahedron from the four-fold partition and return tetravolume (Fuller.4D).
`examples`	`example_volume`	function	`()`	Compute the unit IVM tetrahedron volume from simple quadray vertices (Fuller.4D).
`geometry`	`minkowski_interval`	function	`(dt, dx, dy, dz, c)`	Return the Minkowski interval squared ds^2 (Einstein.4D).

Module	Symbol	Kind	Signature	Summary
`glossary_gen`	`ApiEntry`	class		
			`glossary_gen`	
			`build_api_index`	
			function	
			`(src_dir)`	
			`glossary_gen`	
			`generate_markdown_table`	
			function	
			`(entries)`	
			`glossary_gen`	
			`inject_between_markers`	
			function	
			`(markdown_text, begin, end, payload)`	
			`information`	
			`action_update`	
			function	
			`(action, free_energy_fn, step_size, epsilon)`	Continuous-time action update: da/dt = - dF/da.
			`information`	
			`active_inference_step`	
			function	
			`(mu, action, free_energy_fn, derivative_operator, step_size, epsilon)`	Joint perception-action update step in Active Inference.
			`information`	
			`expected_free_energy`	
			function	
			`(log_p_o_given_s, q, p, log_p_o)`	Expected free energy for Active Inference with prior preferences.
			`information`	
			`finite_difference_gradient`	
			function	
			`(function, x, epsilon)`	Compute numerical gradient of a scalar function via central differences.
			`information`	
			`fisher_information_matrix`	
			function	
			`(gradients,`	

Module	Symbol	Kind	Signature	Summary
nelder_mead_quadray	centroid_excluding	function	(vertices, exclude_idx)	Integer centroid of three vertices, excluding the specified index.
nelder_mead_quadray	compute_volume	function	(vertices)	Integer IVM tetra-volume from the first four vertices.
nelder_mead_quadray	nelder_mead_quadray	function	(f, initial_vertices, alpha, gamma, rho, sigma, max_iter, tol, on_step)	Nelder–Mead on the integer quadray lattice.
nelder_mead_quadray	order_simplex	function	(vertices, f)	Sort vertices by objective value ascending and return paired lists.
nelder_mead_quadray	project_to_lattice	function	(q)	Project a quadray to the canonical lattice representative via normalize.
paths	get_data_dir	function	()	Return quadmath/output/data path and ensure it exists.
paths	get_figure_dir	function	()	Return quadmath/output/figures path and ensure it exists.
paths	get_output_dir	function	()	Return quadmath/output path at the repo root and ensure it exists.
paths	get_repo_root	function	(start)	Heuristically find repository root by walking up from start.
quadray	DEFAULT_EMBEDDING	constant	| | | `quadray` | `Quadray` | class |	Quadray vector with non-negative components and at least one zero (Fuller.4D).
quadray	ace_tetravolume_5x5	function	(p0, p1, p2, p3)	Tom Ace 5x5 determinant in IVM units (Fuller.4D).

Module	Symbol	Kind	Signature	Summary
quadray	dot	function	(q1, q2, embedding)	Return Euclidean dot product <q1,q2> under the given embedding.
quadray	integer_tetra_volume	function	(p0, p1, p2, p3)	Compute integer tetra-volume using det[p1-p0, p2-p0, p3-p0] (Fuller.4D).
quadray	magnitude	function	(q, embedding)	Return Euclidean magnitude ||q|| under the given embedding (vector norm).
quadray	to_xyz	function	(q, embedding)	Map quadray to R^3 via a 3x4 embedding matrix (Fuller.4D -> Coxeter.4D slice).
symbolic	cayley_menger_volume_symbolic	function	(d2)	Return symbolic Euclidean tetrahedron volume from squared distances.
symbolic	convert_xyz_volume_to_ivm_symbolic	function	(V_xyz)	Convert a symbolic Euclidean volume to IVM tetravolume via S3.
visualize	animate_discrete_path	function	(path, embedding, save)	Animate a point moving along a discrete quadray path.
visualize	animate_simplex	function	(vertices_list, embedding, save)	Animate simplex evolution across iterations.
visualize	plot_ivm_neighbors	function	(embedding, save)	Scatter the 12 IVM neighbor points in 3D.
visualize	plot_partition_tetrahedron	function	(mu, s, a, psi, embedding, save)	Plot the four-fold partition as a labeled tetrahedron in 3D.
visualize	plot_simplex_trace	function	(state, save)	Plot per-iteration diagnostics for Nelder–Mead.