

GHR Calculus Extended to Octonions: Approach, Implementation, and Findings

Octonion Computation Substrate

March 2026

Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 2 | Background: GHR Calculus for Quaternions | 1 |
| 2.1 | Wirtinger Derivatives in \mathbb{C} | 1 |
| 2.2 | GHR Derivatives for Quaternions | 2 |
| 3 | Extending GHR to Octonions | 2 |
| 3.1 | The Non-Associativity Obstruction | 2 |
| 3.2 | Resolution via the Moufang Identity | 2 |
| 3.3 | The Octonionic GHR Derivative Pair | 3 |
| 4 | Implementation: Jacobian-First Approach | 3 |
| 4.1 | Motivation | 3 |
| 4.2 | Structure Constants and Multiplication Jacobian | 3 |
| 4.3 | Jacobians for All Seven Primitives | 4 |
| 4.4 | Triple Verification | 4 |
| 4.5 | Higher-Order Derivatives | 5 |
| 5 | Parenthesization-Aware Chain Rule | 5 |
| 5.1 | The Non-Associativity of Compositions | 5 |
| 5.2 | Binary Tree Representation | 5 |
| 5.3 | Autograd Integration | 5 |
| 5.4 | The Naive Baseline | 6 |
| 6 | Octonionic Analyticity | 6 |
| 6.1 | CR-Like Conditions | 6 |
| 6.2 | Classification of Primitives | 6 |
| 7 | Quantitative Results | 6 |
| 7.1 | Gradient Accuracy: All 14 Parenthesizations | 6 |
| 7.2 | Naive vs. Correct Chain Rule | 7 |
| 7.3 | GPU/CPU Parity | 7 |

| | | |
|-----------|--|----------|
| 8 | Tradeoffs and Design Decisions | 8 |
| 8.1 | Native GHR vs. Pseudo-Real Matrix Representation | 8 |
| 8.2 | Jacobian-First vs. Symbolic GHR | 8 |
| 8.3 | Fused vs. Primitive-Decomposed Jacobians | 8 |
| 8.4 | Number of Wirtinger Derivatives | 9 |
| 9 | API Summary | 9 |
| 10 | Conclusion | 9 |

1 Introduction

Gradient-based optimisation of octonionic neural networks requires a rigorous theory of differentiation in the octonionic algebra \mathbb{O} . For quaternionic networks, the *Generalized Hamilton–Real* (GHR) calculus of Xu & Mandic [1] provides a principled framework: Wirtinger-like derivative pairs, product and chain rules, and gradient expressions that preserve algebraic structure. Extending GHR to \mathbb{O} is non-trivial because octonion multiplication is *non-associative*, and the GHR framework for \mathbb{H} relies on associativity in a fundamental way.

This document describes the approach taken, the key theoretical obstacle (the rotation ambiguity arising from non-associativity), its resolution via the Moufang identity, the resulting implementation of analytic Jacobians and parenthesization-aware chain rule, and the quantitative verification results.

2 Background: GHR Calculus for Quaternions

2.1 Wirtinger Derivatives in \mathbb{C}

For a function $f : \mathbb{C} \rightarrow \mathbb{C}$ written in real coordinates as $f(x, y) = u + iv$, the Wirtinger (Cauchy–Riemann) derivative pair is

$$\frac{\partial f}{\partial z} = \frac{1}{2} \left(\frac{\partial f}{\partial x} - i \frac{\partial f}{\partial y} \right), \quad \frac{\partial f}{\partial \bar{z}} = \frac{1}{2} \left(\frac{\partial f}{\partial x} + i \frac{\partial f}{\partial y} \right). \quad (1)$$

The complex analyticity condition $\partial f / \partial \bar{z} = 0$ recovers the Cauchy–Riemann equations. For a real-valued loss L , the gradient direction for descent is $\partial L / \partial \bar{z}$ (the conjugate derivative).

2.2 GHR Derivatives for Quaternions

For a quaternion $\mathbf{q} = q_a + q_b \mathbf{i} + q_c \mathbf{j} + q_d \mathbf{k}$ and a unit quaternion frame $\boldsymbol{\mu}$, the GHR rotation is $\mathbf{q}_\mu := \boldsymbol{\mu} \mathbf{q} \boldsymbol{\mu}^{-1}$. The GHR derivatives are then defined as

$$\frac{\partial f}{\partial \mathbf{q}_\mu} = \frac{1}{4} \left(\frac{\partial f}{\partial q_a} - \frac{\partial f}{\partial q_b} \mathbf{i}_\mu - \frac{\partial f}{\partial q_c} \mathbf{j}_\mu - \frac{\partial f}{\partial q_d} \mathbf{k}_\mu \right), \quad (2)$$

$$\frac{\partial f}{\partial \mathbf{q}_\mu^*} = \frac{1}{4} \left(\frac{\partial f}{\partial q_a} + \frac{\partial f}{\partial q_b} \mathbf{i}_\mu + \frac{\partial f}{\partial q_c} \mathbf{j}_\mu + \frac{\partial f}{\partial q_d} \mathbf{k}_\mu \right), \quad (3)$$

where $(\mathbf{i}_\mu, \mathbf{j}_\mu, \mathbf{k}_\mu)$ is the rotated imaginary basis. For real-valued loss functions, Proposition 4.8 of Xu & Mandic [1] shows that left and right derivatives coincide, and the optimization gradient is $\partial L / \partial \mathbf{q}_\mu^*$.

The GHR framework also supplies a product rule: for $f(\mathbf{q}) = g(\mathbf{q}) h(\mathbf{q})$,

$$\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}_\mu} = g(\mathbf{q}) \frac{\partial h(\mathbf{q})}{\partial \mathbf{q}_\mu} + \frac{\partial g(\mathbf{q})}{\partial \mathbf{q}_{h(\mathbf{q}) \cdot \mu}} h(\mathbf{q}), \quad (4)$$

where the rotation frame for g is shifted by the value of h . This product rule exploits quaternion associativity in the rotation $\mathbf{q}_\mu = \boldsymbol{\mu} \mathbf{q} \boldsymbol{\mu}^{-1}$.

3 Extending GHR to Octonions

3.1 The Non-Associativity Obstruction

The octonions $\mathbb{O} \cong \mathbb{R}^8$ have a basis $\{\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_7\}$ with multiplication encoded by integer-valued structure constants $C_{ij}^k \in \{-1, 0, +1\}$ satisfying

$$\mathbf{e}_i \mathbf{e}_j = \sum_{k=0}^7 C_{ij}^k \mathbf{e}_k. \quad (5)$$

The Cayley–Dickson structure constants (Baez 2002, mod-7 Fano plane basis) are used throughout. Unlike \mathbb{H} , octonion multiplication is *not* associative: in general $(a b) c \neq a (b c)$.

The GHR rotation $R_\mu(\mathbf{o}) := \boldsymbol{\mu} \mathbf{o} \boldsymbol{\mu}^{-1}$ requires specifying a parenthesization: $(\boldsymbol{\mu} \mathbf{o}) \boldsymbol{\mu}^{-1}$ vs. $\boldsymbol{\mu} (\mathbf{o} \boldsymbol{\mu}^{-1})$. In \mathbb{H} , associativity makes these equal; in \mathbb{O} , they may differ.

3.2 Resolution via the Moufang Identity

The octonions, while non-associative, are an *alternative* algebra: every subalgebra generated by two elements is associative. Alternative algebras satisfy the Moufang identities, of which the most useful for our purposes is:

$$(\boldsymbol{\mu} \mathbf{o}) \boldsymbol{\mu}^{-1} = \boldsymbol{\mu} (\mathbf{o} \boldsymbol{\mu}^{-1}). \quad (6)$$

The key observation is that $\boldsymbol{\mu}$ appears *twice* (as $\boldsymbol{\mu}$ and $\boldsymbol{\mu}^{-1}$), and the Moufang identity guarantees that any expression of the form xyx is well-defined regardless of parenthesization in an alternative algebra. Therefore, the octonionic rotation $R_\mu(\mathbf{o}) = \boldsymbol{\mu} \mathbf{o} \boldsymbol{\mu}^{-1}$ is *unambiguous* despite non-associativity, and the GHR framework for \mathbb{H} can be transported to \mathbb{O} without modification of the rotation definition.

3.3 The Octonionic GHR Derivative Pair

For an octonion $\mathbf{o} = o_0 + \sum_{k=1}^7 o_k \mathbf{e}_k$, the natural extension of the GHR derivative pair uses 8 real partial derivatives:

$$\frac{\partial f}{\partial \mathbf{o}} = \frac{1}{8} \left(\frac{\partial f}{\partial o_0} - \sum_{k=1}^7 \frac{\partial f}{\partial o_k} \mathbf{e}_k \right), \quad (7)$$

$$\frac{\partial f}{\partial \mathbf{o}^*} = \frac{1}{8} \left(\frac{\partial f}{\partial o_0} + \sum_{k=1}^7 \frac{\partial f}{\partial o_k} \mathbf{e}_k \right). \quad (8)$$

The factor $\frac{1}{8}$ (compared to $\frac{1}{4}$ for \mathbb{H} and $\frac{1}{2}$ for \mathbb{C}) reflects the 8-dimensional nature of \mathbb{O} and ensures the fundamental reconstruction identity

$$df = \frac{\partial f}{\partial \mathbf{o}} d\mathbf{o} + \frac{\partial f}{\partial \mathbf{o}^*} d\mathbf{o}^* \quad (9)$$

holds under the standard octonionic inner product.

For a real-valued loss L and octonionic parameter \mathbf{o} , the optimization gradient is

$$\nabla_{\mathbf{o}} L = \frac{\partial L}{\partial \mathbf{o}^*} = \frac{1}{8} \left[\frac{\partial L}{\partial o_0}, \frac{\partial L}{\partial o_1}, \dots, \frac{\partial L}{\partial o_7} \right]^\top, \quad (10)$$

which is simply the real gradient vector $\nabla_{\mathbb{R}^8} L$ scaled by $\frac{1}{8}$. The GHR formalism adds structural content for *non-real-valued* intermediate computations by tracking both derivatives of the pair and supplying algebraically consistent product and chain rules.

4 Implementation: Jacobian-First Approach

4.1 Motivation

A direct symbolic derivation of the GHR product rule for octonions requires careful book-keeping of rotation frames that shift with intermediate function values. We adopt instead a *Jacobian-first* strategy: ground every primitive operation in an explicit 8×8 real Jacobian matrix, verify it against finite differences, and build composites via the standard matrix-product chain rule. This sidesteps non-associativity at the single-operation level (a Jacobian is a linear map on \mathbb{R}^8 , and matrix multiplication is associative), while correctly encoding non-associativity at the composition level through explicit parenthesization tracking.

4.2 Structure Constants and Multiplication Jacobian

The product $(a \cdot b)_k = \sum_{i,j} C_{ij}^k a_i b_j$ is bilinear, so its Jacobians are linear in the *other* factor:

$$\mathbf{J}_a[k, i] = \frac{\partial (a \cdot b)_k}{\partial a_i} = \sum_j C_{ij}^k b_j = (R_b)_{ki}^\top, \quad (11)$$

$$\mathbf{J}_b[k, j] = \frac{\partial (a \cdot b)_k}{\partial b_j} = \sum_i C_{ij}^k a_i = (L_a)_{kj}^\top, \quad (12)$$

where L_a and R_b are the left and right multiplication matrices already present in the codebase. In `einsum` notation:

$$\mathbf{J}_a = \text{einsum}(\text{"ijk, ...j} \rightarrow \text{"...ki"}, C, b), \quad \mathbf{J}_b = \text{einsum}(\text{"ijk, ...i} \rightarrow \text{"...kj"}, C, a). \quad (13)$$

4.3 Jacobians for All Seven Primitives

Analytic 8×8 Jacobians are derived for each primitive operation.

Conjugation. $\bar{\mathbf{o}} = (o_0, -o_1, \dots, -o_7)^\top$, so $\mathbf{J}_{\text{conj}} = \text{diag}(1, -1, -1, -1, -1, -1, -1, -1)$.

Inverse. $\mathbf{o}^{-1} = \bar{\mathbf{o}}/\|\mathbf{o}\|^2$. By the quotient rule:

$$\mathbf{J}[k, i] = \frac{(J_{\text{conj}})_{ki}}{\|\mathbf{o}\|^2} - \frac{2\bar{o}_k o_i}{\|\mathbf{o}\|^4}. \quad (14)$$

Exponential. Writing $\mathbf{o} = a + \mathbf{v}$ with $a = o_0$, $\mathbf{v} = (o_1, \dots, o_7)$, and $r = \|\mathbf{v}\|$:

$$\exp(\mathbf{o}) = e^a \left(\cos r + \frac{\sin r}{r} \mathbf{v} \right). \quad (15)$$

The Jacobian has block structure: a 1×1 scalar–scalar block, a 1×7 real–imaginary block, a 7×1 imaginary–real block, and a 7×7 imaginary–imaginary block involving a rank-one outer product $\mathbf{v}\mathbf{v}^\top$ plus a diagonal term:

$$\mathbf{J}_{\text{im,im}} = e^a \left[\frac{\cos r \cdot r - \sin r}{r^3} \mathbf{v}\mathbf{v}^\top + \frac{\sin r}{r} \mathbf{I}_7 \right]. \quad (16)$$

Near $r = 0$, L'Hôpital limits apply: $\sin r/r \rightarrow 1$ and $(\cos r \cdot r - \sin r)/r^3 \rightarrow -1/3$.

Logarithm. With $q = \|\mathbf{o}\|$ and $\theta = \arccos(a/q)$:

$$\log(\mathbf{o}) = \log q + \frac{\theta}{r} \mathbf{v}. \quad (17)$$

The Jacobian has a similar block structure. The outer-product coefficient $(a/q^2 - \theta/r)/r^2$ admits a finite limit as $r \rightarrow 0$ (derived via Taylor expansion of θ about $r = 0$).

Inner product and cross product. The inner product $\langle a, b \rangle = \sum_i a_i b_i$ has trivial Jacobians $\mathbf{J}_a = b^\top$ and $\mathbf{J}_b = a^\top$. The cross product $\text{cross}(a, b) = \text{Im}(\text{Im}(a) \cdot \text{Im}(b))$ has Jacobians equal to the 7×7 imaginary–imaginary block of the multiplication Jacobians computed at the purified arguments, with the real row and column zeroed.

4.4 Triple Verification

Every primitive is checked against three independent gradient computations:

1. **Analytic Jacobian** (closed-form derivation above).

2. **Numeric Jacobian** (central finite differences with $\varepsilon = 10^{-7}$ at `float64`).
3. **Autograd Jacobian** (column-wise backward passes through the `torch.autograd.Function` implementation).

All three agree to relative error below 10^{-5} (see Section 7).

4.5 Higher-Order Derivatives

Each `torch.autograd.Function` backward pass is implemented using only differentiable PyTorch primitives (`einsum`, `exp`, `cos`, `sin`, etc.), and *only inputs* are saved in `ctx.save_for_backward`. This makes the backward pass itself differentiable, enabling `create_graph=True` for second-order gradient computation (Hessian eigenspectrum analysis in Phase 5).

5 Parenthesization-Aware Chain Rule

5.1 The Non-Associativity of Compositions

For standard \mathbb{R}^n functions, the chain rule gives a Jacobian product $\mathbf{J}_{\text{total}} = \mathbf{J}_{\text{outer}} \cdot \mathbf{J}_{\text{inner}}$, and matrix multiplication is associative so the order of application is irrelevant. For octonionic compositions, a crucial distinction arises: the *Jacobians* compose associatively (they are linear maps on \mathbb{R}^8), but the *operations being differentiated* are not associative. That is, $(a \cdot b) \cdot c$ and $a \cdot (b \cdot c)$ are genuinely different functions with genuinely different Jacobians, even though the composition of their Jacobians via matrix multiplication is well-defined in either order.

5.2 Binary Tree Representation

Each multiplication chain is represented as an explicit binary tree:

- A **Leaf** node holds an operand index.
- A **Node** holds an operation label (`mul`, `exp`, `log`, `conjugate`, `inverse`) and references to left and right subtrees.

For n operands in a product chain, there are C_{n-1} distinct binary tree structures, where C_k is the k -th Catalan number: $C_0 = 1, C_1 = 1, C_2 = 2, C_3 = 5, C_4 = 14, C_5 = 42, \dots$

The function `all_parenthesizations(n)` generates all C_{n-1} trees by the standard recursive construction: split the chain at position k ($1 \leq k < n$), enumerate all trees for each sub-chain, and combine.

5.3 Autograd Integration

Each `evaluate_tree` call dispatches to `OctonionMulFunction.apply` (or the corresponding unary function) at each internal node. PyTorch’s autograd engine records the exact computation graph, including which operands were multiplied in which order. The backward pass therefore propagates gradients through the correct tree structure automatically, without any special-casing in the backward code.

5.4 The Naive Baseline

A *naive* chain rule is defined as the one that always parenthesizes left-to-right: $((x_0 \cdot x_1) \cdot x_2) \cdots$. For a real-valued loss, the gradient of the naive rule is the Jacobian of this specific function. When the true computation uses a different parenthesization, the naive gradient is simply *wrong*: it computes the gradient of a different function.

6 Octonionic Analyticity

6.1 CR-Like Conditions

In complex analysis, $f : \mathbb{C} \rightarrow \mathbb{C}$ is holomorphic iff its 2×2 real Jacobian equals a left-multiplication matrix L_c for some $c \in \mathbb{C}$. The octonionic analogue defines $f : \mathbb{O} \rightarrow \mathbb{O}$ to be *octonionic-analytic* (in the Fueter/left-regular sense) iff its 8×8 real Jacobian satisfies

$$\mathbf{J} = L_c, \quad \text{where} \quad L_c[k, j] = \sum_i c_i C_{ij}^k. \quad (18)$$

The *CR residual* is $\|\mathbf{J} - L_c\|_F$ where $c = \mathbf{J}[:, 0]$ (first column of \mathbf{J} , since $L_c \mathbf{e}_0 = c$).

6.2 Classification of Primitives

The analyticity conditions are extremely restrictive in \mathbb{O} , far more so than in \mathbb{C} or \mathbb{H} .

| Operation | Analytic? | Reason |
|---|-----------|--|
| $f(\mathbf{o}) = c \cdot \mathbf{o}$ (left multiplication) | Yes | $\mathbf{J} = L_c$ exactly |
| $f(\mathbf{o}) = \alpha \mathbf{o}$, $\alpha \in \mathbb{R}$ | Yes | $\mathbf{J} = \alpha \mathbf{I} = L_{\alpha \mathbf{e}_0}$ |
| $f(\mathbf{o}) = \mathbf{o}$ (identity) | Yes | $\mathbf{J} = \mathbf{I} = L_{\mathbf{e}_0}$ |
| $f(\mathbf{o}) = \mathbf{o} \cdot c$ (right multiplication) | No | $R_c \neq L_c$ in general |
| $f(\mathbf{o}) = \bar{\mathbf{o}}$ (conjugation) | No | $\mathbf{J} = \text{diag}(1, -1, \dots, -1)$ |
| exp, log | No | Complex block Jacobian structure |

The scarcity of analytic functions is a known feature of octonionic analysis and contrasts sharply with complex analysis. Non-associativity severely restricts which local linear approximations can be represented as multiplication by a fixed octonion.

7 Quantitative Results

7.1 Gradient Accuracy: All 14 Parenthesizations

We tested all $C_4 = 14$ parenthesizations of 5-operand product chains against finite-difference Jacobians at `float64` precision.

| Tree index | Tree structure | Max relative error |
|-------------------------------------|---------------------------|-----------------------|
| 0 | $(x_0(x_1(x_2(x_3x_4))))$ | 1.04×10^{-8} |
| 1 | $(x_0(x_1((x_2x_3)x_4)))$ | 2.95×10^{-8} |
| 2 | $(x_0((x_1x_2)(x_3x_4)))$ | 3.44×10^{-8} |
| 3 | $(x_0((x_1(x_2x_3))x_4))$ | 5.12×10^{-6} |
| 4 | $(x_0(((x_1x_2)x_3)x_4))$ | 2.37×10^{-8} |
| 5 | $((x_0x_1)(x_2(x_3x_4)))$ | 8.68×10^{-7} |
| 6 | $((x_0x_1)((x_2x_3)x_4))$ | 1.35×10^{-8} |
| 7 | $((x_0(x_1x_2))(x_3x_4))$ | 1.47×10^{-7} |
| 8 | $((x_0x_1)x_2)(x_3x_4))$ | 1.50×10^{-8} |
| 9 | $((x_0(x_1(x_2x_3)))x_4)$ | 4.95×10^{-7} |
| 10 | $((x_0((x_1x_2)x_3))x_4)$ | 7.35×10^{-8} |
| 11 | $((x_0x_1)(x_2x_3))x_4)$ | 1.24×10^{-7} |
| 12 | $((x_0(x_1x_2))x_3)x_4)$ | 1.26×10^{-8} |
| 13 | $((x_0x_1)x_2)x_3)x_4)$ | 2.06×10^{-7} |
| All patterns passed ($< 10^{-5}$) | | yes |

The worst-case error is 5.12×10^{-6} (tree index 3), well within the 10^{-5} relative-error criterion. Thirteen of the fourteen patterns are within 10^{-6} . The maximum Jacobian norm difference *between* distinct parenthesizations of the same five operands is 14.63, confirming that different parenthesizations define quantifiably different functions.

7.2 Naive vs. Correct Chain Rule

To quantify the cost of ignoring non-associativity, we compare the parenthesization-aware Jacobian against the *naive* left-associative chain rule across 1,000 random inputs at each depth.

| Depth | # Patterns | Mean $\ \Delta J\ $ | Std | Max $\ \Delta J\ $ |
|-------|------------|---------------------|-------|--------------------|
| 3 | 2 | 11.14 | 3.38 | 21.49 |
| 5 | 14 | 31.01 | 14.34 | 85.69 |
| 7 | 132 | 73.23 | 43.09 | 248.35 |

The mean gradient error grows roughly as depth^{1.4}, compounding rapidly. The mean cosine similarity between naive and correct gradient vectors at depth 3 is 0.331 — the vectors point in substantially different directions. A network trained with the naive chain rule would follow entirely wrong gradient directions for any computation involving 3 or more chained octonionic multiplications.

7.3 GPU/CPU Parity

All 8 backward-pass tests (covering `mul`, `exp`, `log`, `conjugate`, `inverse`, `OctonionLinear`, batched multiplication, and a 3-operand composition) were verified on an AMD Radeon RX 7900 XTX (gfx1100, ROCm 7.2) against CPU computation at `float64`. Maximum elementwise difference: $< 10^{-12}$ in all cases.

8 Tradeoffs and Design Decisions

8.1 Native GHR vs. Pseudo-Real Matrix Representation

Two principled approaches exist for octonionic differentiation:

1. **Native GHR (this work).** Derive derivatives directly in \mathbb{O} , producing octonionic-valued Wirtinger derivatives and a product rule that respects algebraic structure. Requires resolving the rotation ambiguity (Section 3.2).
2. **Pseudo-real matrix representation.** Embed each octonion as an 8×8 real matrix via left multiplication, reduce the problem to real matrix calculus, and apply standard automatic differentiation. This approach (used in e.g. Octonion Phase Retrieval, 2023 [6]) is straightforward but reduces the derivative to \mathbb{R}^8 — the algebraic structure of \mathbb{O} is lost, and the result is equivalent to treating the network as a real network with structured weight sharing.

The native approach was chosen because (a) the Wirtinger derivative pair is the mathematical object needed for gradient-based optimization while preserving algebraic structure, (b) the pseudo-real approach gives no additional insight into how non-associativity affects gradient flow, and (c) the parenthesization analysis (a potential research contribution) requires the native representation.

8.2 Jacobian-First vs. Symbolic GHR

A fully symbolic derivation would apply the GHR product rule directly to expressions like $f(\mathbf{o}) = (\mathbf{w} \cdot \mathbf{o}) \cdot \mathbf{b}$, producing closed-form octonionic gradient expressions without going through the \mathbb{R}^8 Jacobian. The Jacobian-first approach was chosen instead for three reasons:

1. **Verifiability.** An 8×8 matrix can be checked against finite differences column by column; a symbolic octonionic expression cannot be directly verified without computing the same matrix.
2. **Generality.** The Jacobian-first approach handles arbitrary computation graphs; symbolic derivations must be re-done for each new formula.
3. **Associativity of composition.** The chain rule for Jacobian matrices is $\mathbf{J}_{\text{total}} = \mathbf{J}_n \cdots \mathbf{J}_1$, and matrix multiplication *is* associative even though the operations themselves are not. This cleanly separates the non-associativity problem (which parenthesization was used in the forward pass) from the derivative computation (which always reduces to associative matrix products).

The cost is a slight loss of algebraic elegance: gradients are \mathbb{R}^8 vectors reinterpreted as octonions, rather than octonionic expressions derived from first principles in \mathbb{O} .

8.3 Fused vs. Primitive-Decomposed Jacobians

All composite Jacobians are built from the seven primitives via the chain rule, rather than deriving fused Jacobians (e.g., a single 8×8 matrix for $\exp(a \cdot b)$). This is more modular and easier to verify independently, but may introduce small numerical errors relative to a

fused derivation because intermediate Jacobians are computed at intermediate points rather than directly at the composed function’s arguments. In practice the errors are negligible at `float64` (see Section 7).

8.4 Number of Wirtinger Derivatives

Complex numbers require 2 Wirtinger derivatives ($\partial f/\partial z, \partial f/\partial \bar{z}$); quaternions require 4 in the full GHR framework (one pair per rotation direction μ , though for real-valued loss the pairs coincide). For octonions with the Moufang-resolved rotation, the natural extension gives a single pair ($\partial f/\partial \mathbf{o}, \partial f/\partial \mathbf{o}^*$), corresponding to the identity rotation frame $\mu = \mathbf{e}_0$.

In principle, 7 additional rotation frames (one per imaginary unit) could each generate an independent Wirtinger pair, yielding 16 derivative components. However, for optimization with a real-valued loss, Proposition 4.8 of Xu & Mandic [1] generalizes directly: left and right derivatives coincide, and the gradient direction is fully determined by the conjugate derivative $\partial L/\partial \mathbf{o}^*$, which is the \mathbb{R}^8 gradient scaled by $\frac{1}{8}$. The additional rotation-dependent pairs encode the structure of f as an octonionic function but are not needed for gradient descent.

9 API Summary

The implementation is in `src/octonion/calculus/` and exports the following public interface:

| Symbol | Description |
|---|---|
| <code>ghr_derivative(partials)</code> | GHR full derivative (7) |
| <code>conjugate_derivative(partials)</code> | GHR conjugate derivative (8) |
| <code>wirtinger_from_jacobian(J)</code> | Convert 8×8 Jacobian to Wirtinger pair |
| <code>jacobian_mul(a, b)</code> | Analytic Jacobians for $a \cdot b$ |
| <code>jacobian_exp(o), jacobian_log(o)</code> | Analytic Jacobians for <code>exp</code> , <code>log</code> |
| <code>jacobian_conjugate(o), jacobian_inverse(o)</code> | Analytic Jacobians for <code>conj</code> , <code>inv</code> |
| <code>jacobian_inner_product(a,b), jacobian_cross_product(a,b)</code> | Analytic Jacobians |
| <code>numeric_jacobian(fn, x)</code> | Central finite-difference Jacobian |
| <code>octonion_gradcheck(fn, inputs)</code> | Custom gradient checker |
| <code>Leaf, Node, all_parenthesizations(n)</code> | Binary tree types and enumeration |
| <code>evaluate_tree(tree, operands)</code> | Autograd-tracked tree evaluation |
| <code>CompositionBuilder</code> | High-level composition API |
| <code>is_octonionic_analytic(fn, x)</code> | CR residual test |
| <code>cauchy_riemann_octonion(J)</code> | CR residual from Jacobian |
| <code>suggest_lr(model, loader)</code> | GHR gradient magnitude heuristic |

10 Conclusion

We have successfully extended the GHR calculus to the octonions. The central theoretical obstacle — the ambiguity of the rotation $R_\mu(\mathbf{o}) = \boldsymbol{\mu} \mathbf{o} \boldsymbol{\mu}^{-1}$ under non-associativity — is resolved by the Moufang identity, which guarantees that expressions with a repeated element are parenthesization-independent in alternative algebras.

The implementation uses a Jacobian-first strategy: closed-form 8×8 real Jacobians for each primitive operation, verified triple-redundantly against finite differences and autograd. Non-associativity is handled explicitly through binary tree representations of computation graphs; the parenthesization-aware chain rule produces correct gradients for all C_{n-1} parenthesizations of n -operand chains.

Quantitatively, the naive left-associative chain rule produces gradients with a mean angular error of $\cos^{-1}(0.33) \approx 71^\circ$ from the correct gradient at depth 3, growing to roughly $10\times$ larger gradient norm errors at depth 7. No practical octonionic network could be trained correctly without parenthesization-aware differentiation.

References

- [1] N. Xu and D. P. Mandic, “The theory of quaternion matrix derivatives,” *IEEE Trans. Signal Process.*, vol. 63, no. 6, pp. 1543–1556, 2015. (GHR calculus; product rule, chain rule, Proposition 4.8.)
- [2] N. Xu, C. Cheong, and D. P. Mandic, “A new proof and interpretation of the GHR calculus,” *Royal Society Open Science*, vol. 3, 2016.
- [3] J. C. Baez, “The octonions,” *Bull. Amer. Math. Soc.*, vol. 39, pp. 145–205, 2002. (Structure constants, Fano plane, Moufang identities.)
- [4] T. Parcollet, M. Morchid, and G. Linares, “A survey of quaternion neural networks,” *Artif. Intell. Rev.*, vol. 53, pp. 2957–2982, 2020.
- [5] C. J. Gaudet and A. S. Maida, “Deep quaternion networks,” in *Proc. IJCNN*, 2018.
- [6] *Octonion Phase Retrieval*, arXiv:2308.15784, 2023. (Pseudo-real matrix approach for octonionic derivatives — approach not adopted here, but validates the problem statement.)