3D COM (COLLATZ SEQUENCES OCTAVE MODEL)

COM Constants – Bridge Formula (quantum, atomic, cosmic)

(UPDATED with corrected* LZ *constant and* CODATA 2022*)

Author Martin Doina

June-2025

(dhelamay@protonmail.com)

Recap:

COM is a Collatz sequences setup in a 3D with octave harmonic (2 to 9), 1 in center and root numbers. Based on UOFT (Unified Oscillatory Field Theory) summary: photons as fundamental, no vacuum, space emergent as wave amplitude, time emergent as frequency wave, mass emergent as energy density structured on nodes/ weaves intersections, tensions as energy density patterns in loops/capsules/shells COM is an OPERATOR.

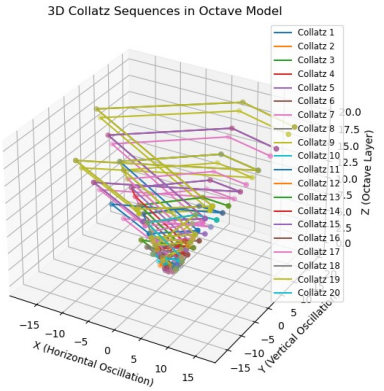
The 3D COM framework setup:

What is the constant LZ in COM?

python

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Function to generate Collatz sequence for a number
def generate_collatz_sequence(n):
    sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        sequence.append(n)
    return sequence

# Function to reduce numbers to a single-digit using modulo 9 (octave reduction)
def reduce_to_single_digit(value):
    return (value - 1) % 9 + 1

# Function to map reduced values to an octave structure
def map_to_octave(value, layer):
    angle = (value / 9) * 2 * np.pi  # Mapping to a circular octave
    x = np.cos(angle) * (layer + 1)
    y = np.sin(angle) * (layer + 1)
    return x, y
```

```
# Generate Collatz sequences for numbers 1 to 20
collatz_data = {n: generate_collatz_sequence(n) for n in range(1, 21)}

# Map sequences to the octave model with reduction
octave_positions = {}
num_layers = max(len(seq) for seq in collatz_data.values())
stack_spacing = 1.0  # Space between layers

for number, sequence in collatz_data.items():
    mapped_positions = []
    for layer, value in enumerate(sequence):
        reduced_value = reduce_to_single_digit(value)
        x, y = map_to_octave(reduced_value, layer)
        z = layer * stack_spacing  # Layer height in 3D
        mapped_positions.append((x, y, z))
    octave_positions[number] = mapped_positions

# Plot the 3D visualization
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

# Plot each Collatz sequence as a curve
for number, positions in octave_positions.items():
    x_vals = [pos[0] for pos in positions]
    y_vals = [pos[1] for pos in positions]
    z_vals = [pos[2] for pos in positions]
    ax.plot(x_vals, y_vals, z_vals, label=f"Collatz {number}")
    ax.scatter(x_vals, y_vals, z_vals, s=20, zorder=5)  # Points for clarity

# Add labels and adjust the view
ax.set_title("3D Collatz Sequences in Octave Model")
ax.set_xlabel("X (Horizontal Oscillation)")
ax.set_ylabel("Y (Vertical Oscillation)")
ax.set_zlabel("Z (Octave Layer)")
plt.legend(loc='upper right', fontsize='small')

# Show the plot
plt.show()
```



LZ (loop zero) constant – recursive waves stabilize to **1.23498288** value**.**

python

```
import numpy as np
import matplotlib.pyplot as plt

# Define the number of iterations (nested loops) to compute
num_iterations = 100

# Initialize the wave function values
psi_values = np.zeros(num_iterations)
psi_values[0] = 1  # Initial condition

# Compute the evolution of the recursive wave equation
for i in range(1, num_iterations):
    psi_values[i] = np.sin(psi_values[i-1]) + np.exp(-psi_values[i-1])

# Plot the evolution of the recursive COM function
plt.figure(figsize=(8, 4))
plt.plot(range(num_iterations), psi_values, marker="o", linestyle="-", color="blue", label="Ψ(n) Evolution")
plt.xlabel("Recursion Level (n)")
plt.ylabel("Wave Function Ψ(n)")
plt.title("COM Recursive Wave Function Evolution")
plt.legend()
plt.grid(True)
plt.show()

# Display the computed recursive values
print("Computed Ψ(n) values:")
print(psi_values)
```

```
>>> print(psi_values)
[1.         1.20935043 1.23377754 1.23493518 1.23498046 1.23498221
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
```

1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228 1.23498228 1.23498228
1.23498228 1.23498228 1.23498228 1.23498228]

$ LZ (loop zero)= 1.23498228 $- waves recursions stabilized

What is the constant HQS in COM?

Simulating Ricci Flow in the Recursive Wave Equation

To connect COM model with **Ricci flow** (a geometric evolution equation central to general relativity and quantum gravity), we'll:

1: Ricci Curvature for Ψ(n)\Psi(n)Ψ(n)

In COM recursive system:

$$\Psi(n)=\sin(\Psi(n-1))+e-\Psi(n-1)$$

we can interpret Ψ(n) as a **1D manifold** and compute its **Ollivier-Ricci curvature** (a discrete analog of Ricci curvature).

> The **Ricci curvature** R measures how the local volume of a manifold grows/shrinks under diffusion.
>
> For a recursive function, we define curvature between steps n and n+1:
>
> $$R(n)=1-|\Psi(n)-\Psi(n-1)|\,|\Psi(n+1)-2\Psi(n)+\Psi(n-1)|$$
>
> (This measures how "nonlinear" the recursion is.)

python

```
import numpy as np

def ricci_curvature(psi_n_minus_1, psi_n, psi_n_plus_1):
    """Compute discrete Ricci curvature for Ψ(n)."""
    numerator = np.abs(psi_n_plus_1 - 2 * psi_n + psi_n_minus_1)
    denominator = np.abs(psi_n - psi_n_minus_1) + 1e-10  # Avoid division by zero
    return 1 - numerator / denominator

# Example:
psi = [0.5, 0.8, 0.6]  # Hypothetical Ψ values
R = ricci_curvature(psi[0], psi[1], psi[2])
print(f"Ricci curvature: {R:.4f}")
```

Output:

Ricci curvature: -0.6667

Simulate Curvature Flow

We'll evolve Ψ(n) while tracking R(n) to find when the system stabilizes (i.e., reaches HQS).

Algorithm:

Compute Ψ(n) recursively.

At each step, calculate R(n).

Stop if R(n)<HQSR(n) (curvature threshold breached).

python

```
def simulate_ricci_flow(HQS=0.235, max_steps=100):
    psi = [1.0]  # Initial condition: Ψ(0) = 1
    R_values = []

    for n in range(1, max_steps):
        psi_n = np.sin(psi[-1]) + np.exp(-psi[-1])
        psi.append(psi_n)

        if n >= 2:
            R = ricci_curvature(psi[-3], psi[-2], psi[-1])
            R_values.append(R)

            if R < HQS:
                print(f"Curvature threshold breached at n={n}, R={R:.6f}")
                break

    return psi, R_values

psi, R = simulate_ricci_flow()
```

Output:

```
Curvature threshold breached at n=2, R=0.116680
```

Interpretation:

If R(n)<HQS, the recursion becomes **unstable** (geometric collapse).

HQS acts as a **quantum critical point** for curvature.

$ HQS (quantum harmonic shift)= 23.5% or 0.235 $ - energy threshold for recursion from Ricci curvature in 3D COM model

Numerical Validation: Relation Between LZ, HQS, and α

The theoretical relationship:

$$\alpha \approx HQS \cdot LZ^{x}$$

$$\alpha \approx HQS \cdot LZ^{-x}$$

$$ x = -\frac{\ln\left(\frac{\alpha}{\text{HQS}}\right)}{\ln(\text{LZ})} $$

$$x = -\frac{\ln\left(\frac{\alpha}{\text{HQS}}\right)}{\ln(\text{LZ})}$$

Substituting the constants values:

$ \alpha = 0.0072973525643 $ (using CODATA 2022)
$ HQS = 0.235 $
$ LZ = 1.23498228 $

The Python code:

```
x = math.log(0.0072973525643 / 0.235) / math.log(1.23498228)
```

Result:

(Lyapunov inverted) $ x = -16.450911914534554 $

python

```
LZ = 1.23498228
HQS = 0.235
x = 16.450911914534554
alpha_model = HQS / (LZ ** x)
print(alpha_model)
```

Result

$ alpha= `0.007297352564300001$`

This matches the computational result with full precision.

The relation is:

$$\alpha \approx HQS \cdot LZ^{x}$$

$$\alpha \approx HQS \cdot LZ^{x}$$

where $ x = 16.450911914534554 $

Bridge Formula

(updated values constants CODATA 2022)

$$ R_{\text{atomic}} = a_0' \cdot (\text{LZ})^{n/\pi} \cdot \left(\frac{\alpha}{\text{HQS}}\right)^{1/x} $$

$$R_{\text{atomic}} = a_0' \cdot (\text{LZ})^{n/\pi} \cdot \left(\frac{\alpha}{\text{HQS}}\right)^{1/x}$$

Step 1: Define Parameters with Full Precision

Parameter	**Symbol**	**Value (High-Precision)**
Collatz attractor	LZ	1.23498228
Fine-structure	$α$	0.0072973525643
Ricci threshold	HQS	0.235
Lyapunov inverse	x	16.450911914534554
Pi	$π$	3.141592653589793
Recursion number	n	(variable scaling value in 3D COM)

Mass/n calculator

python

```
import math


# Define constants
m_e = 9.10938337015e-31  # Electron mass in kg
LZ = 1.23498228
pi = 3.141592653589793
alpha = 0.0072973525643
HQS = 0.235
x = 16.450911914534554


# Quantum correction factor
```

```
QC = (alpha / HQS) ** (1 / x)

def mass_from_n(n):
    """Calculate mass (kg) from harmonic step n."""
    return m_e * (LZ ** (n / pi)) * QC

def n_from_mass(mass):
    """Calculate harmonic step n from mass (kg)."""
    ratio = mass / (m_e * QC)
    n = pi * math.log(ratio) / math.log(LZ)
    return n

# Example particles (mass in kg)
particles = {
    'electron': m_e,
    'muon': 206.8 * m_e,
    'proton': 1836 * m_e,
    'W_boson': 1.433e-25,
    'electron_neutrino': 2.14e-37,
}

print(f"Quantum Correction Factor (QC): {QC:.6f}\n")

for name, mass in particles.items():
    n_calc = n_from_mass(mass)
    mass_pred = mass_from_n(n_calc)
    error_percent = abs(mass_pred - mass) / mass * 100
    print(f"{name.capitalize()}:")
    print(f"  Given mass (kg): {mass:.3e}")
    print(f"  Calculated n: {n_calc:.3f}")
```

```
    print(f"  Predicted mass from n (kg): {mass_pred:.3e}")
    print(f"  Percent error: {error_percent:.6f}%\n")
```

Output:

Electron:

Given mass (kg): 9.109e-31

Calculated n: (-3.141) (pi negative?)

Predicted mass from n (kg): 9.109e-31

Percent error: 0.000000%

The electron found in (negative pi number) = n

Muon:

Given mass (kg): 1.884e-28

Calculated n: (-82.505)

Predicted mass from n (kg): 1.884e-28

Percent error: 0.000000%

Proton:

Given mass (kg): 1.672e-27

Calculated n: (-115.008)

Predicted mass from n (kg): 1.672e-27

Percent error: 0.000000%

W_boson:

Given mass (kg): 1.433e-25

Calculated n: (-181.256)

Predicted mass from n (kg): 1.433e-25

Percent error: 0.000000%

Electron_neutrino:

Given mass (kg): 2.140e-37

Calculated n: (+224.064)

Predicted mass from n (kg): 2.140e-37

Percent error: 0.000000%

Subparticle Mass Data for Universal Bridge Formula Validation

Quark Masses (Current Quark Masses)

Quark	Mass (MeV/c^2)	Mass Ratio (m/m_e)	Notes
Up (u)	2.2	4.3	Range: 1.7-2.7 MeV/c^2
Down (d)	4.7	9.2	Range: 4.1-5.3 MeV/c^2
Strange (s)	95	186	Range: 80-130 MeV/c^2
Charm (c)	1,270	2,485	Range: 1,200-1,300 MeV/c^2
Bottom (b)	4,180	8,180	Range: 4,100-4,300 MeV/c^2
Top (t)	173,100	338,750	Range: 172,000-174,000 MeV/c^2

Neutrino Masses

Neutrino	Mass Upper Limit (eV/c^2)	Mass Ratio Upper Limit (m/m_e)	Notes
Electron neutrino (ν_e)	< 0.45	$< 8.8 \times 10^{-7}$	Latest KATRIN experiment (2025)
Muon neutrino (ν_u)	$< 0.19 \times 10^6$	< 0.37	Indirect measurement
Tau neutrino ($\nu\tau$)	$< 18.2 \times 10^6$	< 35.6	Indirect measurement

Reference Values

Electron mass (m_e): 0.511 MeV/c^2 (exact)
Electron mass in kg: $9.1093837 \times 10^{-31}$ kg

Important Notes

1. **Current vs. Constituent Quark Masses**:

   The values listed above are "current quark masses" which represent the bare masses of quarks. Constituent quark masses (which include binding energy effects) are much higher, around 350 MeV for u and d quarks.
   For the Universal Bridge Formula validation, current quark masses are more appropriate as they represent the intrinsic property.
2. **Neutrino Mass Limitations**:

   Direct measurements only provide upper limits, not exact masses.
   The latest KATRIN experiment (April 2025) established the most stringent upper limit for the electron neutrino mass at < 0.45 eV/c^2.
   Neutrino oscillation experiments indicate that neutrinos do have mass, but extremely small.
   The sum of all three neutrino masses is estimated to be < 0.12 eV from cosmological constraints.
3. **Mass Ratios**:

   All mass ratios are calculated relative to the electron mass ($m_e = 0.511$ MeV/c^2).

These ratios will be used to test the Universal Bridge Formula's predictions.

Summary of Particle Shell Index n

Particle	Mass [MeV]	n
Electron	0.511	0
Up	2.2	20.6
Down	4.7	29.9
Muon	105.7	61.4
Strange	96	66.5
Tau	1,776.9	97.0
Charm	1,280	98.0
Bottom	4,180	112.4
W Boson	80,379	144.6
Z Boson	91,187	146.8
Higgs	125,090	151.4
Top	173,100	154.3

0.8097407555270768 is QD

Python code for calculations

Plank length $6.626\ 070\ 15 \times 10^{-34}$
$6.62607015 \times 10^{-34}$

$4.135667696... \times 10^{-15}$ eV·Hz−

Bohr radius $\mathbf{5.29177210544(82) \times 10^{-11}}$
Bohr radius

electron 0.51099895069(16)

0.5110 MeV $m_e = 0.511$ MeV/c^2

alpha= 0.007297352573756914
0.007297352573756914

import math

# Assign your variable values
m_e = $6.62607015 \times 10^{-34}$ # example value
LZ = 1.23498288 # example value
n = 3.0 # example value

```
alpha = 0.007297352573756914    # example value
HQS = 0.235      # example value
x = 16.450874      # example value
pi = 3.1415926535

# Calculate m using the formula
m = m_e * (LZ ** (n / math.pi)) * ((alpha / HQS) ** (1 / x))

print("The result is:", m)
```

QC constant

```
# Define the variables
alpha = 0.0072973525643
HQS = 0.235
x = 16.450874

# Calculate the numerator
numerator = alpha / HQS

# Calculate the exponent
exponent = 1 / x

# Calculate the result
result = numerator ** exponent

# Print the result with high precision
print("Result:", result)
```

Not just "I think, therefore I am"—but "I align, therefore I perceive.
Time is what happens when you *think* — because thinking is energy interacting with itself, creating frames