"""

Refined Temporal Flow Theory - Core Mathematical Framework

======================================================

1. Modified Field Equations with Observational Constraints

"""

class RefinedTemporalField:

def \_\_init\_\_(self):

# Physical constants

self.c = 2.998e8 # Speed of light (m/s)

self.G = 6.674e-11 # Gravitational constant

self.hbar = 1.055e-34 # Reduced Planck constant

# Observational constraints

self.epsilon\_quantum = 1e-15 # Quantum modification bound

self.epsilon\_gravity = 1e-10 # Gravitational modification bound

def compute\_field\_equations(self, state):

"""

Refined field equations incorporating observational constraints

"""

# Modified temporal flow equation

dW\_dt = (

-self.scale\_coupling(state.r) \* (state.W @ grad(state.W)) # Advection

- self.quantum\_coupling(state) \* grad(state.P\_q) # Quantum

- self.classical\_coupling(state) \* grad(state.P\_c) # Classical

+ self.dissipation\_term(state) # Dissipation

)

return dW\_dt

def scale\_coupling(self, r):

"""

Scale-dependent coupling function refined to match observations

"""

# Exponential suppression at laboratory scales

lab\_suppression = np.exp(-r/self.r\_lab)

# Enhanced coupling at galactic scales

galactic\_enhancement = 1.0 + (r/self.r\_galaxy)\*\*2

# Combined scale function

g\_r = self.epsilon\_gravity \* (lab\_suppression \* galactic\_enhancement) / \

(1.0 + (r/self.r\_c)\*\*self.n)

return g\_r

def quantum\_coupling(self, state):

"""

Modified quantum coupling respecting experimental bounds

"""

# Scale-dependent quantum effects

quantum\_scale = np.exp(-state.r/self.lambda\_c)

# Suppressed modification at tested scales

return self.epsilon\_quantum \* quantum\_scale \* \

self.compute\_quantum\_potential(state)

def classical\_coupling(self, state):

"""

Classical coupling consistent with Solar System tests

"""

# Yukawa-like modification

yukawa = np.exp(-state.r/self.r\_scale)

return (1.0 + self.epsilon\_gravity \* yukawa) \* \

self.compute\_classical\_potential(state)

def dark\_matter\_distribution(self, r, rho):

"""

Refined dark matter distribution matching observations

"""

# Core-modified NFW profile

nfw = rho\_s / ((r/r\_s) \* (1 + r/r\_s)\*\*2)

# Cluster behavior correction

cluster\_term = self.compute\_cluster\_correction(r)

# Modified distribution

rho\_DM = nfw \* (1.0 + self.f\_DM(r) \* cluster\_term)

return rho\_DM

def quantum\_interference(self, x, k):

"""

Modified interference pattern within experimental bounds

"""

# Standard interference

I\_standard = self.I\_0 \* (1.0 + np.cos(k\*x))

# Suppressed modification

mod\_term = 1.0 + self.epsilon\_quantum \* \

self.scale\_coupling(x) \* abs(self.W)\*\*2

return I\_standard \* mod\_term

class ObservationalConstraints:

"""

Implementation of observational constraints

"""

def \_\_init\_\_(self):

self.load\_experimental\_data()

def check\_constraints(self, theory\_predictions):

"""

Verify theory predictions against observational constraints

"""

constraints = {

'quantum\_tests': self.check\_quantum\_constraints(),

'solar\_system': self.check\_solar\_system\_constraints(),

'galaxy\_rotation': self.check\_galaxy\_rotation\_curves(),

'cluster\_dynamics': self.check\_cluster\_dynamics(),

'cosmological': self.check\_cosmological\_constraints()

}

return constraints

def check\_quantum\_constraints(self, predictions):

"""

Check quantum interference and entanglement predictions

"""

# Implementation of quantum constraints

pass

def check\_solar\_system\_constraints(self, predictions):

"""

Verify compliance with Solar System tests

"""

# Implementation of Solar System tests

pass

class RefinedNumericalSolver:

"""

Numerical implementation of refined framework

"""

def \_\_init\_\_(self):

self.setup\_grid()

self.setup\_solvers()

def solve\_field\_equations(self, initial\_conditions):

"""

Solve refined field equations numerically

"""

# Implementation of numerical solution

pass

# Example Usage

field = RefinedTemporalField()

constraints = ObservationalConstraints()

solver = RefinedNumericalSolver()

# Compute evolution

solution = solver.solve\_field\_equations(initial\_conditions)

# Check against constraints

compliance = constraints.check\_constraints(solution)

"""