"""

Refined CMB Predictions Framework

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

"""

class CMBPredictor:

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

self.cosmological\_params = {

'H0': 67.4, # km/s/Mpc

'Omega\_m': 0.315,

'Omega\_b': 0.049,

'n\_s': 0.965,

'sigma\_8': 0.811

}

def compute\_power\_spectrum(self, k\_range):

"""

Compute CMB power spectrum with temporal flow effects

"""

# Initialize perturbation equations

perturbations = self.initialize\_perturbations()

# Evolution through different epochs

primordial = self.evolve\_primordial\_perturbations(perturbations)

acoustic = self.evolve\_acoustic\_oscillations(primordial)

recombination = self.compute\_recombination\_effects(acoustic)

# Final power spectrum

Cl = self.compute\_angular\_power\_spectrum(recombination)

return Cl

def evolve\_primordial\_perturbations(self, initial\_state):

"""

Evolve primordial perturbations with temporal flow effects

"""

def perturbation\_evolution(k, eta):

# Standard perturbation equation

standard\_pert = self.compute\_standard\_perturbations(k, eta)

# Temporal flow modifications

temporal\_mod = self.compute\_temporal\_modifications(k, eta)

# Combined evolution

return standard\_pert \* (1.0 + temporal\_mod)

return perturbation\_evolution

def compute\_acoustic\_peaks(self):

"""

Compute detailed acoustic peak structure

"""

# Initialize sound waves

sound\_waves = self.initialize\_sound\_waves()

# Evolution with temporal flow effects

peak\_structure = self.evolve\_sound\_waves(sound\_waves)

# Compute peak positions and amplitudes

peaks = {

'positions': self.compute\_peak\_positions(peak\_structure),

'amplitudes': self.compute\_peak\_amplitudes(peak\_structure),

'ratios': self.compute\_peak\_ratios(peak\_structure)

}

return peaks

def compute\_secondary\_effects(self):

"""

Compute secondary CMB effects

"""

effects = {

'reionization': self.compute\_reionization\_effects(),

'isw': self.compute\_integrated\_sw\_effect(),

'lensing': self.compute\_lensing\_potential()

}

return effects

def validate\_predictions(self, predictions):

"""

Validate predictions against Planck data

"""

# Load Planck data

planck\_data = self.load\_planck\_data()

# Compute chi-squared

chi\_squared = self.compute\_chi\_squared(predictions, planck\_data)

# Analyze residuals

residuals = self.analyze\_residuals(predictions, planck\_data)

return {

'chi\_squared': chi\_squared,

'residuals': residuals,

'significance': self.compute\_significance(chi\_squared)

}

# Usage

coupling\_optimizer = CouplingConstantsOptimizer()

cluster\_dynamics = ClusterDynamics()

cmb\_predictor = CMBPredictor()

# Optimize couplings

optimized\_couplings = coupling\_optimizer.optimize\_couplings()

# Compute cluster evolution

cluster\_evolution = cluster\_dynamics.compute\_cluster\_evolution(

initial\_state, time\_span)

# Generate CMB predictions

cmb\_predictions = cmb\_predictor.compute\_power\_spectrum(k\_range)

# Validate results

validation = cmb\_predictor.validate\_predictions(cmb\_predictions)