"""

Higher-order CMB Correlations with Temporal Flow Effects

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

"""

class HigherOrderCorrelations:

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

self.max\_multipole = 2500

self.nside = 2048 # HEALPix resolution

def compute\_all\_correlations(self, cmb\_map):

"""

Compute full hierarchy of correlations

"""

correlations = {

'power\_spectrum': self.compute\_power\_spectrum(cmb\_map),

'bispectrum': self.compute\_bispectrum(cmb\_map),

'trispectrum': self.compute\_trispectrum(cmb\_map),

'non\_gaussian': self.compute\_non\_gaussian\_features(cmb\_map)

}

return correlations

def compute\_bispectrum(self, cmb\_map):

"""

Compute CMB bispectrum with temporal flow effects

"""

# Initialize bispectrum configurations

configurations = self.get\_bispectrum\_configurations()

bispectrum = {}

for config in configurations:

# Standard bispectrum

B\_standard = self.compute\_standard\_bispectrum(cmb\_map, config)

# Temporal flow modifications

B\_temporal = self.compute\_temporal\_bispectrum\_mod(config)

# Combined result

bispectrum[config] = B\_standard \* (1.0 + B\_temporal)

return bispectrum

def compute\_non\_gaussian\_features(self, cmb\_map):

"""

Compute non-Gaussian features induced by temporal flow

"""

# Local non-Gaussianity

f\_NL = self.compute\_fnl(cmb\_map)

# Equilateral non-Gaussianity

f\_NL\_equil = self.compute\_fnl\_equilateral(cmb\_map)

# Orthogonal non-Gaussianity

f\_NL\_ortho = self.compute\_fnl\_orthogonal(cmb\_map)

return {

'local': f\_NL,

'equilateral': f\_NL\_equil,

'orthogonal': f\_NL\_ortho

}

def validate\_against\_planck(self, correlations):

"""

Validate predictions against Planck data

"""

# Load Planck data

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

# Compare statistics

comparison = {

'bispectrum': self.compare\_bispectrum(correlations, planck\_data),

'trispectrum': self.compare\_trispectrum(correlations, planck\_data),

'f\_NL': self.compare\_fnl(correlations, planck\_data)

}

# Compute statistical significance

significance = self.compute\_statistical\_significance(comparison)

return {

'comparison': comparison,

'significance': significance,

'systematics': self.estimate\_systematics()

}

# Usage

nonlinear = NonlinearClusterEvolution()

backreaction = BackreactionComputer()

correlations = HigherOrderCorrelations()

# Compute evolution with all effects

state = nonlinear.evolve\_nonlinear\_system(initial\_state, time\_span)

backreaction\_effects = backreaction.compute\_backreaction(state, scale)

cmb\_correlations = correlations.compute\_all\_correlations(cmb\_map)

# Validate results

validation = correlations.validate\_against\_planck(cmb\_correlations)