Appendix C: Numerical Implementation

This appendix provides code availability information and implementation details for reproducing the Synchronism coherence model results.

C.1 Repository Information

Code Repository: https://github.com/dp-web4/Synchronism

Structure:

```
Synchronism/
├── simulations/           # Numerical implementations
│   ├── session65_*.py     # Parameter derivation
│   ├── session66_*.py     # SPARC validation, A gap, tanh derivation
│   ├── session67_*.py     # V_flat mechanism, B derivation, Bullet Cluster
│   ├── session68_*.py     # MOND comparison, figures
│   ├── session69_*.py     # DF2, compact/extended
│   ├── session70_*.py     # UDG validation, relativistic extension
│   ├── session71_*.py     # GW170817, cosmological coherence
│   └── results/           # JSON output files
├── Research/              # Session summaries and documentation
├── manuscripts/           # Preprint drafts and appendices
└── figures/               # Publication-ready figures
```

C.2 Core Functions

C.2.1 Coherence Function

```python
import numpy as np

# Synchronism parameters (derived from first principles)
GAMMA = 2.0       # From 6D phase space
A = 0.029         # From A = 4π/(α²GR₀²) in (km/s)^{-0.5} M_☉/pc³
B = 0.5           # From virial + size-velocity scaling

def coherence_function(rho, rho_crit, gamma=GAMMA):
    """
    Synchronism coherence function.

    Parameters:
        rho : float or array
            Local baryonic density (M_☉/pc³)
        rho_crit : float
            Critical density for coherence transition (M_☉/pc³)
        gamma : float
            Coherence exponent (default: 2.0)
```

```
    Returns:
        C : float or array
            Coherence value in range (0, 1]
    """
    rho = np.maximum(rho, 1e-10)  # Avoid log(0)
    return np.tanh(gamma * np.log(rho / rho_crit + 1))
```

C.2.2 Critical Density

```
def critical_density(V_flat, A=A, B=B):
    """
    Critical density as function of flat rotation velocity.

    Parameters:
        V_flat : float
            Flat rotation velocity (km/s)
        A : float
            Amplitude parameter (default: 0.029)
        B : float
            Scaling exponent (default: 0.5)

    Returns:
        rho_crit : float
            Critical density (M_⊙/pc³)
    """
    return A * V_flat**B
```

C.2.3 Predicted Velocity

```
def predicted_velocity(V_bar, rho, V_flat):
    """
    Predict observed rotation velocity from baryonic velocity.

    Parameters:
        V_bar : float or array
            Baryonic rotation velocity (km/s)
        rho : float or array
            Local density (M_⊙/pc³)
        V_flat : float
            Flat rotation velocity for this galaxy (km/s)

    Returns:
        V_obs : float or array
            Predicted observed velocity (km/s)
    """
    rho_crit = critical_density(V_flat)
    C = coherence_function(rho, rho_crit)

    # Ensure C > 0 to avoid division issues
    C = np.maximum(C, 0.01)
```

```
    return V_bar / np.sqrt(C)
```

C.2.4 Mass Enhancement

```
def mass_enhancement(rho, V_flat):
    """
    Calculate effective mass / baryonic mass ratio.

    Parameters:
        rho : float or array
            Local density (M_☉/pc³)
        V_flat : float
            Flat rotation velocity (km/s)

    Returns:
        enhancement : float or array
            M_eff / M_baryon = 1/C
    """
    rho_crit = critical_density(V_flat)
    C = coherence_function(rho, rho_crit)
    C = np.maximum(C, 0.01)
    return 1.0 / C
```

C.3 Galaxy Density Profile

C.3.1 Exponential Disk Model

```
def disk_density(r, Sigma_0, h, z_0=0.3):
    """
    Exponential disk density profile.

    Parameters:
        r : float or array
            Galactocentric radius (kpc)
        Sigma_0 : float
            Central surface density (M_☉/pc²)
        h : float
            Scale length (kpc)
        z_0 : float
            Scale height (kpc), default 0.3

    Returns:
        rho : float or array
            Volume density (M_☉/pc³)
    """
    # Surface density at radius r
    Sigma_r = Sigma_0 * np.exp(-r / h)

    # Convert to volume density assuming sech² vertical profile
    # ρ(r, z=0) ≈ Σ(r) / (2 * z_0)
```

```python
    rho = Sigma_r / (2.0 * z_0 * 1000)  # Convert kpc to pc

    return rho
```

C.3.2 Rotation Curve Model

```python
def rotation_curve(r, M_disk, h, V_flat):
    """
    Calculate model rotation curve.

    Parameters:
        r : float or array
            Radius (kpc)
        M_disk : float
            Disk mass (M_☉)
        h : float
            Scale length (kpc)
        V_flat : float
            Asymptotic flat velocity (km/s)

    Returns:
        V_pred : float or array
            Predicted rotation velocity (km/s)
    """
    # Baryonic velocity from exponential disk (approximate)
    x = r / (2 * h)

    # Modified Bessel functions approximation
    V_bar_sq = 4 * np.pi * G_GALACTIC * (M_disk / (2 * np.pi * h**2))
    V_bar_sq *= x**2 * (I0_K0(x) - I1_K1(x))  # Bessel functions
    V_bar = np.sqrt(np.maximum(V_bar_sq, 0))

    # Get local density
    Sigma_0 = M_disk / (2 * np.pi * h**2)
    rho = disk_density(r, Sigma_0, h)

    # Apply coherence correction
    V_pred = predicted_velocity(V_bar, rho, V_flat)

    return V_pred
```

C.4 SPARC Validation Code

C.4.1 Galaxy Comparison

```python
def compare_galaxy(galaxy_data, verbose=True):
    """
    Compare Synchronism prediction to observed rotation curve.

    Parameters:
```

```
        galaxy_data : dict
            Contains 'r', 'V_obs', 'V_bar', 'V_flat', 'name'

    Returns:
        results : dict
            Comparison statistics
    """
    r = galaxy_data['r']
    V_obs = galaxy_data['V_obs']
    V_bar = galaxy_data['V_bar']
    V_flat = galaxy_data['V_flat']

    # Get density profile (from surface brightness)
    rho = estimate_density(galaxy_data)

    # Synchronism prediction
    V_pred = predicted_velocity(V_bar, rho, V_flat)

    # Statistics
    errors = np.abs(V_pred - V_obs) / V_obs
    mean_error = np.mean(errors)
    max_error = np.max(errors)

    results = {
        'name': galaxy_data['name'],
        'n_points': len(r),
        'mean_error': mean_error,
        'max_error': max_error,
        'success': mean_error < 0.15  # 15% threshold
    }

    if verbose:
        print(f"{galaxy_data['name']}: "
              f"mean error = {mean_error:.1%}, "
              f"max error = {max_error:.1%}")

    return results
```

C.4.2 Compact vs Extended Test

```
def compact_extended_test(compact_galaxy, extended_galaxy):
    """
    Compare compact and extended galaxies at similar mass.

    Synchronism predicts: compact (high ρ) → lower enhancement
                          extended (low ρ) → higher enhancement

    MOND predicts: same mass → same enhancement

    Returns:
        result : dict
            Test outcome and statistics
```

```python
    """
    # Get mean coherence for each
    C_compact = np.mean(coherence_function(
        compact_galaxy['rho'],
        critical_density(compact_galaxy['V_flat'])
    ))

    C_extended = np.mean(coherence_function(
        extended_galaxy['rho'],
        critical_density(extended_galaxy['V_flat'])
    ))

    # Enhancement = (V_obs / V_bar)²
    enh_compact = np.mean((compact_galaxy['V_obs'] /
                          compact_galaxy['V_bar'])**2)
    enh_extended = np.mean((extended_galaxy['V_obs'] /
                           extended_galaxy['V_bar'])**2)

    # Test Synchronism prediction
    supports_synch = (C_compact > C_extended) == (enh_compact < enh_extended)

    return {
        'C_compact': C_compact,
        'C_extended': C_extended,
        'enh_compact': enh_compact,
        'enh_extended': enh_extended,
        'enh_ratio': enh_extended / enh_compact,
        'supports_synchronism': supports_synch
    }
```

C.5 Physical Constants

```python
# Fundamental constants
G = 6.674e-11          # m³/(kg·s²) - Gravitational constant
c = 2.998e8            # m/s - Speed of light
hbar = 1.055e-34       # J·s - Reduced Planck constant

# Astronomical units
M_sun = 1.989e30       # kg - Solar mass
pc = 3.086e16          # m - Parsec
kpc = 1000 * pc        # m - Kiloparsec
Mpc = 1e6 * pc         # m - Megaparsec

# Galactic units (convenient for calculations)
G_GALACTIC = 4.30e-3    # pc³/(M_☉ × Myr²) - G in galactic units

# MOND acceleration scale (for comparison)
a0_MOND = 1.2e-10       # m/s² - MOND critical acceleration
```

C.6 Running the Simulations

C.6.1 Parameter Derivation

```bash
# Session 65: γ dimensionality and A constraint
python simulations/session65_gamma_dimensionality.py
python simulations/session65_A_parameter_derivation.py

# Session 66: Complete A derivation, SPARC validation, tanh derivation
python simulations/session66_A_gap_investigation.py
python simulations/session66_sparc_validation.py
python simulations/session66_tanh_derivation.py

# Session 67: B derivation, V_flat mechanism, Bullet Cluster
python simulations/session67_B_derivation.py
python simulations/session67_vflat_mechanism.py
python simulations/session67_bullet_cluster.py
```

C.6.2 Validation Tests

```bash
# Session 68: MOND comparison and figures
python simulations/session68_mond_synchronism_comparison.py
python simulations/session68_preprint_figures.py

# Session 69: DF2 and compact/extended
python simulations/session69_df2_investigation.py
python simulations/session69_compact_extended_test.py

# Session 70: UDG validation and SPARC expanded
python simulations/session70_udg_validation.py
python simulations/session70_sparc_compact_extended.py
```

C.6.3 Output Files

Results are saved to `simulations/results/` as JSON files:

```bash
ls simulations/results/
# session65_A_derivation.json
# session65_gamma_dimensionality.json
# session66_A_gap.json
# session66_sparc.json
# session66_tanh.json
# session67_B.json
# session67_vflat.json
# session67_bullet.json
# session68_mond.json
# session69_df2.json
# session70_sparc_compact_extended.json
# ...
```

C.7 Dependencies

```
numpy >= 1.20
scipy >= 1.7
matplotlib >= 3.4
json (standard library)
```

Optional for figure generation:

```
astropy >= 5.0      # For coordinate transformations
pandas >= 1.3       # For data handling
```

C.8 License and Citation

License: MIT

Citation:

```
@article{synchronism2025,
  title={Synchronism: A Coherence-Based Framework for Galactic Dynamics},
  author={[Authors]},
  journal={arXiv preprint},
  year={2025},
  note={Code available at https://github.com/dp-web4/Synchronism}
}
```

Implementation details from Sessions #65-71