Untitled27

April 29, 2025

[1]:

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# PARAMETERS
N = 30          # lattice size
T_max = 50      # number of ticks
alpha = 0.1     # compression sensitivity
beta = 1.0      # phase tension scaler
eps = 0.01      # stabilization threshold
eps_crit = 0.1 # decay threshold

# NEIGHBOR OFFSETS
neighbors = [(-1,0),(1,0),(0,-1),(0,1)]

# INITIALIZE LATTICE
np.random.seed(42)
H = np.random.rand(N, N)  # temporal deltas fixed
R = np.random.rand(N, N)  # structural deltas dynamic
state = np.zeros((N, N), dtype=int)

# RECORDING STRUCTURES
M_series = []
state_counts = {'stable': [], 'reflect': [], 'dead': []}

def tick(H, R):
    new_R = R.copy()
    new_state = np.empty_like(state)
    misalignment_total = 0.0

    # Reflect & compress neighbors
    for i in range(N):
        for j in range(N):
            for di, dj in neighbors:
                ni, nj = i + di, j + dj
                if 0 <= ni < N and 0 <= nj < N:
```

```python
                    if H[i, j] > H[ni, nj]:
                        d = abs(H[i, j] - H[ni, nj])
                        new_R[ni, nj] -= alpha * np.tanh(beta * d)

    # Compute misalignment and state transitions
    for i in range(N):
        for j in range(N):
            E = H[i, j] * new_R[i, j]
            mis = 0.0
            count = 0
            for di, dj in neighbors:
                ni, nj = i + di, j + dj
                if 0 <= ni < N and 0 <= nj < N:
                    E_n = H[ni, nj] * new_R[ni, nj]
                    mis += abs(E - E_n)
                    count += 1
            local_mis = mis / max(count, 1)
            misalignment_total += local_mis
            # State classification
            if local_mis < eps:
                new_state[i, j] = 0  # stable
            elif local_mis > eps_crit:
                new_state[i, j] = 2  # dead
            else:
                new_state[i, j] = 1  # reflect

    # Average misalignment
    M = misalignment_total / (N * N)
    return new_R, new_state, M

# SIMULATION LOOP
for t in range(T_max):
    R, state, M = tick(H, R)
    M_series.append(M)
    cnts = [(state == k).sum() for k in [0, 1, 2]]
    state_counts['stable'].append(cnts[0])
    state_counts['reflect'].append(cnts[1])
    state_counts['dead'].append(cnts[2])

# VISUALIZATIONS

# 1. Global Misalignment over Time
fig1 = px.line(
    x=list(range(T_max)),
    y=M_series,
    labels={'x':'Tick','y':'M(t)'},
    title='Global Misalignment (M(t) over Time)'
```

```
)
fig1.show()

# 2. Cell State Counts
df_counts = pd.DataFrame(state_counts)
df_counts['tick'] = df_counts.index
fig2 = px.line(
    df_counts,
    x='tick',
    y=['stable','reflect','dead'],
    labels={'value':'Cell Count','tick':'Tick'},
    title='Cell State Counts over Time'
)
fig2.show()

# 3. Final Radii Field Heatmap
fig3 = px.imshow(
    R,
    color_continuous_scale='Viridis',
    title='Radii Field at Final Tick (R)'
)
fig3.show()

# 4. Power Spectrum (Log Scale)
F = np.fft.fftshift(np.fft.fft2(R))
PS = np.abs(F)**2
fig4 = px.imshow(
    np.log1p(PS),
    color_continuous_scale='Viridis',
    title='Log Power Spectrum of Radii Field'
)
fig4.show()
```

[ ]:

```

```