Phase 9 – Part 6  
Thermodynamic Flows and Entropic Currents in ψ

Goal  
The purpose of this part is to analyze ψ not only as a statistical ensemble but as a medium that supports thermodynamic flows. Just as entropy in fluids and plasmas can be transported, produced, or dissipated, ψ can generate entropic currents when subjected to gradients in its distribution. This allows ψ to be understood as a dynamical thermodynamic field, where entropy itself moves and interacts with geometry.

Setup  
I continue with the upgraded ψ-gravity framework:

Plaintext:  
Gravity(x) = (∇²[ space(x) + current(x)² ]) × ψ(x)

The corresponding force field:

Plaintext:  
Force(x) = −∇[Gravity(x)]

Now, for ψ-thermodynamic flows, I define:

ψ-entropy density:

Plaintext:  
s(x,t) = − P(ψ(x,t)) log(P(ψ(x,t)))

ψ-entropy current:

Plaintext:  
J\_s(x,t) = v(x,t) s(x,t)

where is the local drift velocity of ψ, inferred from force balance.

Entropy production rate:

Plaintext:  
σ(x,t) = ds/dt + ∇·J\_s

This decomposition shows how ψ’s entropy changes in time due to both local fluctuations and entropy transport.

Governing Dynamics  
To model entropic flows, I extend ψ’s Langevin dynamics into a continuity form:

Plaintext:  
∂ψ/∂t + ∇·Jψ = − δF/δψ + η(x,t)

with entropy current driven by ψ-current:

Plaintext:  
Js = μ ∇ψ s(x,t)

Here, μ is an effective “entropic mobility.”  
This structure ensures that entropy is not just a scalar diagnostic but a transported, dynamical field interacting with ψ-gradients.

Numerical Experiment

# simulations/phase9\_part6\_entropy\_flows.py  
import numpy as np  
  
# Parameters  
N = 256  
dx = 1.0 / N  
dt = 0.01  
steps = 1500  
a, b = 1.0, 1.0  
T = 0.2  
mu = 0.5  
  
def laplacian(field, dx):  
 return (np.roll(field,1) + np.roll(field,-1) - 2\*field) / dx\*\*2  
  
def entropy\_density(psi, bins=50):  
 hist, edges = np.histogram(psi, bins=bins, density=True)  
 P = hist + 1e-12  
 centers = 0.5\*(edges[1:]+edges[:-1])  
 s\_vals = -P\*np.log(P)  
 return centers, s\_vals  
  
psi = np.random.normal(0,1,N)  
  
entropy\_time = []  
  
for t in range(steps):  
 lap = laplacian(psi, dx)  
 dF = -lap + a\*psi + b\*psi\*\*3  
 noise = np.sqrt(2\*T\*dt/dx) \* np.random.normal(0,1,N)  
 psi += -dt\*dF + noise  
   
 centers, s\_vals = entropy\_density(psi)  
 entropy\_time.append(np.mean(s\_vals))  
  
print("Entropy evolution:", entropy\_time[:10])