Phase 9 – Part 4  
Numerical / Statistical Explorations of ψ-Thermodynamics

Goal  
The aim of Part 4 is to operationalize ψ-thermodynamics numerically.  
In earlier sections of Phase 9, I introduced ψ as an effective free-energy field, developed entropy-like functionals, and considered fluctuation–dissipation analogies. Now I want to test these ideas with concrete numerical/statistical explorations.  
Specifically, I will: 1. Construct ψ as a field on a spatial grid.

1. Define ψ-entropy S\_ψ and ψ-free-energy F\_ψ.
2. Explore how fluctuations around equilibrium configurations evolve.
3. Simulate entropic currents and relaxation toward equilibrium.  
   The desert analogy:

* ψ = desert floor.
* Gravity = pressure distribution.
* Wind = current² term.
* Entropy = roughness/disorder of the desert floor.
* Free energy = tendency of dunes to settle into smoother/low-energy configurations.

Setup: ψ Field on a Grid  
I represent ψ on a 2D spatial grid (x,y). This makes the statistical interpretation more tangible: local variations in ψ correspond to microscopic states of the medium.  
Define:

Plain text:  
psi(x,y) = psi0 \* exp(-(x^2 + y^2)/(2\*sigma^2)) + ε(x,y)

* The first term is a Gaussian “well” (smooth structure).
* The second term ϵ(x,y) is random fluctuation noise, representing disorder/thermal-like contributions.

ψ-Entropy Functional  
I define ψ-entropy as a measure of disorder:

Plain text:  
Sψ = - Σ Pψ(x,y) \* ln(Pψ(x,y))

where

Plain text:  
Pψ(x,y) = |ψ(x,y)|² / Σ |ψ(x,y)|²

This mirrors Shannon entropy, treating normalized ψ² as a probability distribution.

ψ-Free-Energy Functional  
By analogy to thermodynamics:

Plain text:  
Fψ = Eψ – Tψ \* Sψ

Here:

* E\_ψ = effective ψ-energy (curvature-weighted integral).
* T\_ψ = effective ψ-temperature (a parameter controlling fluctuation scale).

Define energy:

Plain text:  
Eψ = Σ [ (∇ψ)² + V(x,y) ψ² ]

with potential V(x,y) set by the desert curvature (sand distribution).

Simulation Strategy  
I had the AI set up a numerical experiment:

1. Initialize ψ as a Gaussian with noise.
2. Compute entropy S\_ψ and free energy F\_ψ.
3. Evolve ψ under relaxation dynamics:

(gradient descent toward low free-energy states).

1. Track entropy vs time.  
   This tests whether ψ tends to settle into smoother configurations while balancing entropy and energy contributions.

Python Simulation

# simulations/phase9\_part4\_psither\_modynamics.py  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Grid setup  
N = 100  
L = 10.0  
x = np.linspace(-L, L, N)  
y = np.linspace(-L, L, N)  
X, Y = np.meshgrid(x, y)  
  
# Initial ψ field: Gaussian + noise  
sigma = 2.0  
psi0 = np.exp(-(X\*\*2 + Y\*\*2) / (2 \* sigma\*\*2))  
noise = 0.2 \* np.random.randn(N, N)  
psi = psi0 + noise  
  
# Potential term: flat for now  
V = np.zeros((N, N))  
  
def entropy(psi):  
 prob = np.abs(psi)\*\*2  
 prob /= np.sum(prob)  
 return -np.sum(prob \* np.log(prob + 1e-12))  
  
def energy(psi):  
 # Gradient squared term  
 dx = x[1] - x[0]  
 dy = y[1] - y[0]  
 # gradient along x corresponds to axis 1, along y to axis 0  
 gradx = np.gradient(psi, dx, axis=1)  
 grady = np.gradient(psi, dy, axis=0)  
 grad\_sq = gradx\*\*2 + grady\*\*2  
 return np.sum(grad\_sq + V \* psi\*\*2)  
  
def free\_energy(psi, T=1.0):  
 return energy(psi) - T \* entropy(psi)  
  
# Time evolution (relaxation dynamics)  
dt = 0.01  
steps = 200  
Tpsi = 1.0  
  
# define spacings for discrete Laplacian  
dx = x[1] - x[0]  
dy = y[1] - y[0]  
# For simplicity use dx (assume dx == dy); otherwise proper 2D Laplacian scaling can be applied  
h2 = dx\*\*2  
  
S\_vals, F\_vals = [], []  
  
for step in range(steps):  
 S\_vals.append(entropy(psi))  
 F\_vals.append(free\_energy(psi, T=Tpsi))  
   
 # Simple relaxation step: discrete 2D Laplacian (periodic-like rolls)  
 laplacian = (  
 np.roll(psi, 1, axis=0) + np.roll(psi, -1, axis=0) +  
 np.roll(psi, 1, axis=1) + np.roll(psi, -1, axis=1) -  
 4 \* psi  
 ) / h2  
 # gradient descent toward lower free-energy approximated by laplacian - V\*psi  
 psi = psi + dt \* (laplacian - V \* psi)  
  
# Plot entropy and free energy evolution  
plt.figure(figsize=(10,4))  
plt.subplot(1,2,1)  
plt.plot(S\_vals)  
plt.title("ψ-Entropy vs Time")  
  
plt.subplot(1,2,2)  
plt.plot(F\_vals)  
plt.title("ψ-Free Energy vs Time")  
  
plt.tight\_layout()  
plt.show()