Phase 8 – Part 9  
2D Phase Diagrams and Coherent-Structure Emergence in ψ-Gravity  
(Spectral methods + visualization recipes)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

🎯 Goal  
The purpose of this part is to extend ψ-gravity into 2D phase diagrams, scanning parameter space to study how coherent structures (stable ψ wells, filaments, vortices) emerge. By using spectral methods (Fourier space evolution) and robust visualization recipes, I aim to extract signatures of stability, chaos, and self-organization within the ψ field dynamics.  
This analysis marks a step toward classifying ψ-configurations: which initial conditions yield localized structures, which disperse, and which enter turbulent-like regimes.

⚙️ Setup

I adopt the upgraded ψ-gravity core equation:

Plain-text form:  
Gravity(x,y,t) = ( ∇² [ space(x,y) + current(x,y,t)² ] ) × ψ(x,y,t)

Force field (driving test particle motion or matter clustering) remains:

Plain-text form:  
Force(x,y,t) = −∇[Gravity(x,y,t)]

For coherent-structure analysis, ψ(x,y,t) is initialized with a Gaussian or random localized bump plus small perturbations. I then evolve it in time under the ψ-gravity operator.

🧮 Methodology — Spectral Evolution  
Instead of evolving ψ with direct finite differences, I use spectral (FFT-based) methods:

* Represent ψ(x,y,t) on a 2D grid.
* Compute Laplacians ∇² efficiently in Fourier space:

Plain-text form:  
Fourier∇² f = −(kx² + ky²) × Fourierf

* Apply this to compute the curvature term from space(x,y) + current(x,y,t)².
* Multiply result with ψ(x,y,t) in real space.
* Advance in time using a symplectic-like step (leapfrog / RK4).

📊 Phase Diagrams  
I explore parameter planes (2D diagrams) to map ψ behavior:

* (ψ amplitude vs. current strength): Do high currents destabilize ψ wells?
* (initial width vs. ψ amplitude): Are there critical thresholds for collapse or dispersion?
* (perturbation noise level vs. lifetime): When do coherent ψ structures survive disorder?

Outcomes are classified into:

* Stable coherent structure (stationary or oscillatory ψ wells).
* Dispersive phase (ψ smooths out).
* Chaotic/turbulent phase (filamentation, vortices).

🌊 Analogy (Desert View)  
In desert terms:

* ψ = desert floor.
* Gravity = pressure imprint.
* Wind (current²) = shifting dunes.
* Phase diagram = climate map of desert behaviors: when does the desert stabilize into valleys, when do dunes migrate chaotically, when does the floor flatten.

🐍 Python Simulation

# Phase 8 – Part 9 ψ–Gravity Coupled  
import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft2, ifft2, fftshift  
import time  
  
def spectral\_laplacian(field, k2):  
 F = fft2(field)  
 lap = np.real(ifft2(-k2 \* F))  
 return lap  
  
# Grid and spectral operators  
N = 96 # grid size (N x N)  
L = 32.0 # physical box length  
dx = L / N  
kx = np.fft.fftfreq(N, d=dx) \* 2.0 \* np.pi  
ky = np.fft.fftfreq(N, d=dx) \* 2.0 \* np.pi  
k2 = kx[:, None]\*\*2 + ky[None, :]\*\*2  
  
# spatial coordinates  
x = (np.arange(N) - N/2) \* dx  
X, Y = np.meshgrid(x, x, indexing='ij')  
  
# Define 'space' and 'current' fields (smooth, low-k structure)  
space\_field = 0.2 \* np.exp(-((X-6)\*\*2 + (Y-6)\*\*2)/(3.5\*\*2)) \  
 - 0.15 \* np.exp(-((X+7)\*\*2 + (Y-7)\*\*2)/(4.5\*\*2))  
  
current\_field = 0.4 \* np.sin(2\*np.pi\*X/L) \* np.cos(2\*np.pi\*Y/L) \  
 + 0.1 \* np.exp(-((X)\*\*2 + (Y+9)\*\*2)/30.0)  
current\_sq = current\_field\*\*2  
  
# Precompute curvature background = Laplacian(space + current^2)  
curvature\_bg = spectral\_laplacian(space\_field + current\_sq, k2)  
  
# Evolution parameters (chosen for numerical stability)  
D = 0.2 # diffusion coefficient  
lam = 1.0 # cubic nonlinearity coefficient  
eta = 0.6 # damping for stability  
dt = 0.02 # time step  
steps = 200 # time steps per run  
  
# Parameter sweep ranges for the phase diagram  
mu\_vals = np.linspace(-1.0, 0.5, 6) # linear growth/decay  
kappa\_vals = np.linspace(0.0, 0.6, 6) # coupling strength for Gravity  
  
phase\_metric = np.zeros((len(mu\_vals), len(kappa\_vals)))  
rng = np.random.default\_rng(12345)  
  
def evolve\_psi(mu, kappa, seed\_noise=None, return\_traj=False):  
 # small initial noise  
 if seed\_noise is None:  
 psi = 0.001 \* (np.random.randn(N, N))  
 else:  
 rng\_local = np.random.default\_rng(seed\_noise)  
 psi = 0.001 \* rng\_local.standard\_normal((N, N))  
 psi = psi.astype(float)  
 for t in range(steps):  
 lap\_psi = spectral\_laplacian(psi, k2)  
 gravity = curvature\_bg \* psi # multiplicative feedback  
 dpsi = D \* lap\_psi + mu \* psi - lam \* psi\*\*3 + kappa \* gravity - eta \* psi  
 psi = psi + dt \* dpsi  
 # soft safety checks  
 if np.any(np.abs(psi) > 1e6):  
 return None  
 if np.isnan(psi).any() or np.isinf(psi).any():  
 return None  
 if return\_traj:  
 return psi  
 return psi  
  
# Run the parameter sweep (fast, small grid)  
start = time.time()  
for i, mu in enumerate(mu\_vals):  
 for j, kappa in enumerate(kappa\_vals):  
 psi\_final = evolve\_psi(mu, kappa, seed\_noise=1000 + i\*10 + j)  
 if psi\_final is None:  
 phase\_metric[i, j] = np.nan  
 else:  
 phase\_metric[i, j] = np.std(psi\_final)  
end = time.time()  
print("Parameter sweep done:", phase\_metric.shape, "time:", end - start)  
  
# Pick a representative non-NaN run to visualize  
good\_idx = np.argwhere(~np.isnan(phase\_metric))  
if good\_idx.size == 0:  
 raise RuntimeError("All runs unstable; adjust parameters.")  
i0, j0 = good\_idx[0]  
rep\_mu = float(mu\_vals[i0]); rep\_kappa = float(kappa\_vals[j0])  
psi\_rep = evolve\_psi(rep\_mu, rep\_kappa, seed\_noise=4242, return\_traj=True)  
if psi\_rep is None:  
 raise RuntimeError("Representative run unstable.")  
  
# Compute isotropic power spectrum (radial average)  
F = fft2(psi\_rep)  
P2 = np.abs(F)\*\*2  
P2s = fftshift(P2)  
kx\_lin = np.fft.fftshift(kx)  
ky\_lin = np.fft.fftshift(ky)  
KX, KY = np.meshgrid(kx\_lin, ky\_lin, indexing='ij')  
Kmag = np.sqrt(KX\*\*2 + KY\*\*2)  
Kflat = Kmag.ravel()  
Pflat = P2s.ravel()  
k\_bins = np.linspace(0, Kmag.max(), N//2)  
k\_idx = np.digitize(Kflat, k\_bins)  
spec1d = np.zeros(len(k\_bins))  
counts = np.zeros(len(k\_bins))  
for idx, p in zip(k\_idx, Pflat):  
 if idx>0 and idx < len(k\_bins):  
 spec1d[idx] += p  
 counts[idx] += 1  
nonzero = counts>0  
spec1d[nonzero] /= counts[nonzero]  
k\_centers = 0.5\*(k\_bins[:-1] + k\_bins[1:])  
  
# Plots  
plt.figure(figsize=(6,5))  
plt.title(f"Representative ψ snapshot (mu={rep\_mu:.2f}, kappa={rep\_kappa:.2f})")  
plt.imshow(psi\_rep, origin='lower', extent=[-L/2, L/2, -L/2, L/2])  
plt.colorbar(label='ψ')  
plt.xlabel('x'); plt.ylabel('y')  
  
plt.figure(figsize=(6,5))  
plt.title("Phase diagram: final std(ψ) over (mu, kappa)")  
plt.imshow(phase\_metric, origin='lower', aspect='auto',  
 extent=[kappa\_vals[0], kappa\_vals[-1], mu\_vals[0], mu\_vals[-1]])  
plt.colorbar(label='std(ψ)')  
plt.xlabel('kappa (coupling)'); plt.ylabel('mu (linear growth)')  
  
plt.figure(figsize=(6,4))  
plt.title("Isotropic power spectrum (representative run)")  
valid = nonzero[1:len(k\_centers)+1]  
plt.loglog(k\_centers[valid], spec1d[1:len(k\_centers)+1][valid], marker='o', linestyle='-')  
plt.xlabel('k (rad / length)'); plt.ylabel('Power')  
  
plt.tight\_layout()  
plt.show()  
  
# Save results  
np.savez('psi\_phase9\_results\_v4.npz', psi\_rep=psi\_rep, phase\_metric=phase\_metric,  
 mu\_vals=mu\_vals, kappa\_vals=kappa\_vals, space\_field=space\_field, current\_field=current\_field)  
print("Saved psi\_phase9\_results\_v4.npz")