Phase 8 – Part 4: Curvature Diagnostics and 2D ψ-Embedding with current²

Goal  
In this part of the ψ-gravity project, I extend the theory into two spatial dimensions.  
The objectives are:

* To compute curvature diagnostics using the Laplacian ∇² in 2D.
* To embed current(x, y)² into the 2D structure.
* To analyze how ψ interacts with this embedding and modifies gravity pressure.
* To define and simulate force fields in 2D.
* To run simulations of test particle motion across a ψ landscape.
* To visualize curvature maps and study the effects of current² on symmetry breaking.

Setup  
I define the following fields:

* ψ(x, y): The ψ field (desert floor), chosen as a Gaussian well in 2D with optional perturbations.
* space(x, y): Background spatial embedding (flat or gently varying).
* current(x, y)²: Square of a current field (wind intensity) shaping the embedding.
* Embedding(x, y): Total embedding that combines space and current².

Equation:

Plain text:  
Embedding(x, y) = space(x, y) + current(x, y)²

Equations  
The core ψ-gravity definition in 2D:

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

The force field derived from gravity:

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

Curvature diagnostics (independent of ψ):

Plain text:  
Curvature(x, y) = ∇²[Embedding(x, y)]

Desert Analogy (Extended to 2D)

* ψ is the desert floor stretched in two directions.
* Gravity is pressure distributed across the desert surface.
* Current² represents the squared strength of winds, which deform the surface and produce ridges.
* Force corresponds to the slopes of dunes: test particles (grains of sand) move down the slopes according to force vectors.
* This analogy highlights how current² breaks symmetry in the dunes and creates preferential drift.

Simulation Design

* Grid: 2D square grid (100×100 points).
* ψ(x, y): A Gaussian well centered at the grid midpoint, optionally with sinusoidal ripples.
* space(x, y): Flat baseline = 0.
* current(x, y): Vector field defined as current(x, y) = α·sin(kx) + β·cos(ky).
* Embedding: space(x, y) + current(x, y)².
* Laplacian: Use finite-difference operator to approximate ∇²[Embedding].
* Gravity: Multiply Laplacian by ψ(x, y).
* Force: Gradient of gravity, taken with finite differences.
* Particles: Initialize test particles at random positions, update via force field using Euler integration.
* Plots:
  + Heatmap of Gravity(x, y).
  + Heatmap of Curvature(x, y).
  + Force vector field overlay.
  + Particle trajectories.

Results & Insights

Expected outcomes:

* Curvature maps reveal ridges and valleys carved by current².
* Gravity maps highlight how ψ amplifies or suppresses these ridges.
* Force fields direct particles toward asymmetric attractors.
* Symmetry breaking: In the absence of current², force is radial and symmetric; with current², the embedding tilts and bends, producing drift.
* This demonstrates how ψ interacts with current-driven curvature in two dimensions.

Python Simulation Script

import numpy as np  
import matplotlib.pyplot as plt  
  
# Grid setup  
N = 100  
x = np.linspace(-5, 5, N)  
y = np.linspace(-5, 5, N)  
X, Y = np.meshgrid(x, y)  
  
# Define ψ(x, y): Gaussian well  
psi = np.exp(-(X\*\*2 + Y\*\*2) / 2.0)  
  
# Define current(x, y)  
alpha, beta, k = 0.5, 0.3, 1.0  
current = alpha \* np.sin(k \* X) + beta \* np.cos(k \* Y)  
  
# Embedding = space + current^2 (space = 0 baseline)  
embedding = current\*\*2  
  
# Finite-difference Laplacian operator  
def laplacian(Z, dx, dy):  
 Zxx = (np.roll(Z, -1, axis=1) - 2\*Z + np.roll(Z, 1, axis=1)) / dx\*\*2  
 Zyy = (np.roll(Z, -1, axis=0) - 2\*Z + np.roll(Z, 1, axis=0)) / dy\*\*2  
 return Zxx + Zyy  
  
dx = x[1] - x[0]  
dy = y[1] - y[0]  
  
curvature = laplacian(embedding, dx, dy)  
gravity = curvature \* psi  
  
# Force field = -∇Gravity  
Fx = -(np.roll(gravity, -1, axis=1) - np.roll(gravity, 1, axis=1)) / (2\*dx)  
Fy = -(np.roll(gravity, -1, axis=0) - np.roll(gravity, 1, axis=0)) / (2\*dy)  
  
# Test particle motion  
n\_particles = 20  
positions = np.random.uniform(-5, 5, (n\_particles, 2))  
dt = 0.05  
steps = 200  
trajectories = [positions.copy()]  
  
for \_ in range(steps):  
 # Interpolate force at particle positions (nearest grid point)  
 idx = np.clip(((positions[:,0]-x[0])/dx).astype(int), 0, N-1)  
 idy = np.clip(((positions[:,1]-y[0])/dy).astype(int), 0, N-1)  
 forces = np.stack([Fx[idy, idx], Fy[idy, idx]], axis=1)  
   
 # Update positions (Euler)  
 positions += forces \* dt  
 trajectories.append(positions.copy())  
  
trajectories = np.array(trajectories)  
  
# Visualization  
plt.figure(figsize=(12,5))  
  
plt.subplot(1,2,1)  
plt.title("Gravity Map with Force Field")  
plt.imshow(gravity, extent=[-5,5,-5,5], origin='lower', cmap='plasma')  
plt.quiver(X[::5,::5], Y[::5,::5], Fx[::5,::5], Fy[::5,::5], color='white', alpha=0.6)  
plt.colorbar(label="Gravity")  
  
plt.subplot(1,2,2)  
plt.title("Particle Trajectories")  
plt.imshow(gravity, extent=[-5,5,-5,5], origin='lower', cmap='plasma', alpha=0.6)  
for i in range(n\_particles):  
 plt.plot(trajectories[:,i,0], trajectories[:,i,1], lw=1)  
plt.colorbar(label="Gravity")  
  
plt.tight\_layout()  
plt.show()