📘 Phase 5 – Part 5.8: Energy Analysis of the ψ–Gravity System

🎯 **Goal**  
In this phase I analyze the energy structure of the ψ–gravity system defined by my core equation:

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

The force is derived as:

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

The purpose of this phase is to build an energy framework for ψ and gravity, defining a ψ-energy functional and testing conservation-like behavior through simulations.

### Step 1 – Energy Analogy

The desert analogy helps me ground the idea:

* ψ = desert floor (substrate energy baseline)
* space(x) = sand distribution (structural background)
* current(x) = wind (injecting kinetic-like contributions, squared like energy)
* Gravity = dune pressure (emergent energy density)
* Force = dune slopes (potential gradients that move particles)

Energy here corresponds to the “stored” and “transferred” distortions in the desert floor shaped by sand and wind.

### Step 2 – ψ-Energy Functional

I propose an energy density functional, inspired by variational field theory:

Plaintext:  
E(x) = (1/2) |∇ψ(x)|² + (α/2) ψ(x)² + β Gravity(x) ψ(x)

Where:  
- The first term (1/2)|∇ψ|² is the gradient energy (surface tension of ψ).  
- The second term (α/2)ψ² is a restoring potential (like a mass term).  
- The third term β·Gravity·ψ couples ψ directly to gravity, closing the feedback loop.

The total ψ-energy is:

Plaintext:  
E\_total = ∫ E(x) dx

### Step 3 – Conservation Hypothesis

If ψ evolves dynamically (as considered in earlier parts of Phase 5), then E\_total should exhibit quasi-conservation, meaning:

* Small oscillations conserve energy.
* Interactions with the “current²” term inject or dissipate energy, breaking strict conservation.

Thus ψ-energy is conditionally conserved, depending on the structure of current(x).

### Step 4 – 1D Discrete Simulation Setup

To test this, I set up a 1D ψ field on a grid:

* Define ψ(x) as an initial Gaussian well.
* Compute Laplacian of [space(x) + current(x)²].
* Construct Gravity(x) = Laplacian × ψ.
* Compute energy density 𝓔(x).
* Track E\_total across iterations.

### Step 5 – Python Simulation Code 1

# static\_energy\_snapshot.py  
# This script computes the energy density E(x) for a single ψ snapshot and prints E\_total.  
import numpy as np  
import matplotlib.pyplot as plt  
  
# --- grid and params ---  
N = 400  
L = 40.0  
x = np.linspace(-L/2, L/2, N)  
dx = x[1] - x[0]  
  
alpha = 1.0  
beta = 1.0  
  
# --- fields ---  
psi = np.exp(-x\*\*2 / 4.0) # initial gaussian ψ  
space = 0.8 \* np.exp(-x\*\*2 / 50.0) # background space(x)  
current = 0.4 \* np.sin(0.6 \* x) # static current(x)  
current\_sq = current\*\*2  
  
# finite-difference Laplacian (periodic)  
def laplacian(f):  
 return (np.roll(f, -1) - 2\*f + np.roll(f, 1)) / dx\*\*2  
  
# curvature and Gravity  
C = laplacian(space + current\_sq)  
Gravity = C \* psi  
  
# gradients  
grad\_psi = (np.roll(psi, -1) - np.roll(psi, 1)) / (2\*dx)  
  
# energy density (as defined)  
E\_density = 0.5 \* grad\_psi\*\*2 + 0.5 \* alpha \* psi\*\*2 + beta \* Gravity \* psi  
E\_total = np.sum(E\_density) \* dx  
  
print("E\_total (static snapshot) =", E\_total)  
  
# plots  
plt.figure(figsize=(9,6))  
plt.subplot(3,1,1); plt.plot(x, psi); plt.title("ψ(x)")  
plt.subplot(3,1,2); plt.plot(x, C); plt.title("C(x) = ∇²[space + current²]")  
plt.subplot(3,1,3); plt.plot(x, E\_density); plt.title("Energy density E(x)")  
plt.tight\_layout()  
plt.show()

### Step 6 – Python Simulation Code 2

# psi\_wave\_evolution.py  
# This script integrates ψ̈ = ∇²ψ − (α + 2β C(x,t)) ψ using an explicit symplectic-style step and records total energy H(t).  
import numpy as np  
import matplotlib.pyplot as plt  
  
# --- grid and params ---  
N = 400  
L = 40.0  
x = np.linspace(-L/2, L/2, N)  
dx = x[1] - x[0]  
  
alpha = 1.0  
beta = 1.0  
  
# time integration params  
dt = 0.01  
T = 40.0  
nt = int(T / dt)  
  
# initial fields  
psi = np.exp(-x\*\*2 / 4.0) # initial ψ  
psi\_prev = psi.copy() # for leapfrog initialization  
psi\_dot = np.zeros\_like(psi)  
  
# background space (static)  
space = 0.8 \* np.exp(-x\*\*2 / 50.0)  
  
# finite-difference Laplacian  
def laplacian(f):  
 return (np.roll(f, -1) - 2\*f + np.roll(f, 1)) / dx\*\*2  
  
# energy calculator  
def energy(psi, psi\_dot, C):  
 grad\_psi = (np.roll(psi, -1) - np.roll(psi, 1)) / (2\*dx)  
 E\_density = 0.5\*grad\_psi\*\*2 + 0.5\*alpha\*psi\*\*2 + beta\*(C\*psi)\*psi  
 E\_pot = np.sum(E\_density)\*dx  
 E\_kin = 0.5\*np.sum(psi\_dot\*\*2)\*dx  
 return E\_pot, E\_kin, E\_pot + E\_kin  
  
# Prepare arrays for recording  
times = []  
H\_total = []  
E\_pot\_list = []  
E\_kin\_list = []  
  
# Two scenarios: (A) static current; (B) time-dependent current  
scenario = 'A\_static' # choose 'A\_static' or 'B\_time\_dep'  
  
for n in range(nt):  
 t = n \* dt  
  
 if scenario == 'A\_static':  
 current = 0.4 \* np.sin(0.6 \* x) # static  
 else:  
 current = 0.4 \* np.sin(0.6 \* x + 0.8\*t) # time-dependent phase -> Ċ ≠ 0  
  
 C = laplacian(space + current\*\*2)  
  
 # compute δE/δψ = -∇²ψ + αψ + 2β C ψ  
 dE\_dpsi = -laplacian(psi) + alpha\*psi + 2.0\*beta\*C\*psi  
  
 # leapfrog / symplectic update for ψ̈ = -δE/δψ  
 psi\_dd = -dE\_dpsi  
 psi\_dot += psi\_dd \* dt  
 psi += psi\_dot \* dt  
  
 # record energies  
 E\_pot, E\_kin, H = energy(psi, psi\_dot, C)  
 times.append(t)  
 H\_total.append(H)  
 E\_pot\_list.append(E\_pot)  
 E\_kin\_list.append(E\_kin)  
  
# plotting results  
plt.figure(figsize=(9,6))  
plt.plot(times, H\_total, label='H\_total')  
plt.plot(times, E\_pot\_list, '--', label='E\_pot')  
plt.plot(times, E\_kin\_list, ':', label='E\_kin')  
plt.xlabel('time')  
plt.ylabel('Energy')  
plt.legend()  
plt.title(f'Energy vs time — scenario {scenario}')  
plt.show()