# Phase 12 – Quantization & Coupling

## Part 1: ψ Lagrangian Formulation

### Goal

Formulate a Lagrangian density for ψ-gravity that recovers the core equation:

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

and ensures consistent canonical and quantum extensions.

### Step 1: Action Principle Setup

I define the action as:

Plain text:  
S = ∫ d⁴x L[ψ, ∂μψ, space(x), current(x)]

The Lagrangian density must generate both the Laplacian term and ψ-multiplicative structure.

### Step 2: Candidate Lagrangian

A minimal choice consistent with the form is:

Plain text:  
L = 1/2 (∂t ψ)² - 1/2 (∇ψ)² - Vψ(x) ψ²

where the effective potential encodes space(x) and current(x):

Plain text:  
Vψ(x) = -∇²(space(x) + current(x)²)

I note that this minimal Klein–Gordon–like ansatz is provisional; a refined, full ψ-gravity Lagrangian will be developed in Phase 17 and may supersede the present terms.

### Step 3: Equation of Motion

Applying the Euler–Lagrange equation for ψ:

Plain text:  
∂L/∂ψ − ∂μ(∂L/∂(∂μψ)) = 0

yields:

Plain text:  
∂t²ψ − ∇²ψ + 2 Vψ(x) ψ = 0

Substituting :

Plain text:  
∂t²ψ − ∇²ψ − 2(∇²[ space(x) + current(x)² ]) ψ = 0

This recovers the core ψ-gravity structure.

### Step 4: Lagrangian Properties

* The kinetic term ensures time evolution.
* The gradient term stabilizes spatial propagation.
* The effective potential directly embeds space and current contributions.
* The multiplicative factor guarantees that acts as the substrate carrying curvature effects.

### Step 5: Python Prototype

Below is a simulation scaffold for exploring the ψ Lagrangian dynamics in 1D:

# simulations/phase12\_part1\_lagrangian.py  
import numpy as np  
  
# Grid setup  
Nx = 200  
dx = 0.05  
dt = 0.01  
steps = 1000  
  
x = np.linspace(-5, 5, Nx)  
  
# Define space(x) and current(x)  
space = np.exp(-x\*\*2) # Gaussian bump  
current = 0.5 \* np.sin(x)  
  
# Effective potential (2nd derivative using numpy.gradient with spacing)  
Vpsi = - np.gradient(np.gradient(space + current\*\*2, dx), dx)  
  
# Initialize ψ field and velocity  
psi = np.exp(-x\*\*2)  
psi\_dot = np.zeros\_like(x)  
  
# Time evolution loop (leapfrog-like)  
for n in range(steps):  
 psi\_ddot = np.gradient(np.gradient(psi, dx), dx) - 2 \* Vpsi \* psi  
 psi\_dot += dt \* psi\_ddot  
 psi += dt \* psi\_dot  
  
# Final ψ profile  
print("Final ψ:", psi)