Phase 15 — Part 3  
Multi-scale simulation & validation; integration with the desert analogy  
I author this phase. The AI produced the derivations, numerical schemes, and reference simulation code below on my instruction; I present them here as the computational backbone for validating the multi-scale ψ → physics/symbol mapping.

Goal (concise)  
I validate the multi-scale mappings proposed in Phase 15 by building concrete numerical experiments that:

* Evolve the ψ field (Klein–Gordon / wave-like dynamics) on 1D and prototype 2D grids.
* Compute the emergent Gravity field using the immutable core equation.
* Integrate ensembles of test particles (non-backreacting) responding to F = -∇Gravity.
* Measure signatures across scales: bound states, scattering, dune-surfing on moving currents, dispersion relation checks, and numerical stability diagnostics.
* Translate results into the desert analogy (ψ floor, sand, wind, dunes).

Setup (computational experiment specification)

**Domain & discretization**

* 1D domain: with N grid nodes, spacing .
* Time domain: with timestep .

**Field variables**

* : substrate field (desert floor).
* : static sand distribution (background).
* : wind-like term (optionally time-dependent).

**Derived field**

Plain text: A(x,t) = nabla^2 [ space(x) + current(x,t)^2 ]

**Core (immutable) Gravity**

Plain text: Gravity(x,t) = A(x,t) \* psi(x,t)

**Force on test particle (continuum)**

Plain text: F(x,t) = -nabla [ Gravity(x,t) ]

ψ evolution equation (working model — Klein–Gordon style)

I use a conservative, second-order-in-time evolution as our baseline:

Plain text: d2/dt2 psi(x,t) - c^2 nabla^2 psi(x,t) + V’(psi(x,t)) = 0

A common choice for the potential is:

which gives

Plain text: V’(psi) = mu^2 \* psi + lambda \* psi^3

Discretization (1D finite differences — explicit leapfrog-like scheme)

**Spatial Laplacian (central difference, index i)**

Plain text: nabla^2 f\_i ≈ (f\_{i+1} - 2\*f\_i + f\_{i-1}) / dx\*\*2

**Time-stepping (second-order central difference for ∂t²)**

From , we obtain the update:

Plain text: psi\_i^{n+1} = 2*psi\_i^n - psi\_i^{n-1} + dt****2 \* ( c****2*  nabla2(psi\_i^n) - V’(psi\_i^n) )

Gravity & Force on the grid

Compute:

Then:

Plain text: Gravity\_i^n = A\_i^n \* psi\_i^n

Gradient (central difference):

Plain text: grad\_Gravity\_i ≈ (Gravity\_{i+1} - Gravity\_{i-1}) / (2\*dx)

Particle acceleration (unit mass):

Plain text: a\_p(t) = - grad\_Gravity evaluated at x = x\_p(t)

Interpolation: linear or cubic interpolation of grid gradient to particle position .

Numerical stability & CFL condition (1D wave-like limit)

For the linear wave part , the CFL condition is:

Plain text: c \* dt / dx <= 1

When nonlinearity is present, stability becomes more restrictive; I (the author) require monitoring conserved diagnostics and reducing Δt if high-frequency growth appears.

Validation diagnostics (discrete checks)

* **L2 norm of ψ**
* Plain text: L2 = sum(psi\*\*2) \* dx
* **Energy (field)**
* Plain text: E\_field = sum( 0.5\*(psi\_t\*\*2) + 0.5\*c\*\*2\*(grad\_psi\*\*2) + V(psi) )\*dx
* **Particle diagnostics**: trajectories , velocities , binding probability (fraction with negative specific energy), mean radial (or 1D displacement) diffusion.
* **Convergence test**: run at Δx, Δx/2 and compare L2 errors and energy drift.
* **Dispersion test**: initialize small-amplitude plane wave and measure numerical vs analytic . Compute relative error.

Reference experiment: 1D Gaussian ψ-well + localized current (static wind patch)

Physical intent: a stationary ψ well (proto-dune) combined with a localized wind that creates a time-invariant trench in . I place a test particle initially at rest near the rim to study capture vs escape.

**Initial conditions (example)**

* Domain: L=200, N=4096 (tuned for resolution).
* Grid: .
* ψ initial: Gaussian well centered at :

Plain text: psi(x,0) = -psi0 \* exp(-(x-x0)\*\*2 / (2\*sigma\*\*2))

* (field initially at rest).
* space(x): constant background (or gentle slope) — e.g., space(x) = s0 + s1\*x/L.
* current(x): Gaussian-localized wind patch centered at :

Plain text: current(x) = c0 \* exp(-(x-xc)\*\*2 / (2\*sigma\_c\*\*2))

* computed once from space + current² (static trench), or time-varying if we choose moving current (see ψ-trench surfing experiment).
* Particle: unit-mass particle at . Interpolate grid gravity to particle position.

Reference Python code (1D field + many particles, non-backreacting)  
I asked the AI to produce clean, documented NumPy reference code. I will present it here so I can run it or convert it to my pipeline later.

# simulations/phase15\_part3\_multiscale\_1d.py  
# Author: (I)  
# Simulations and numerical schemes generated/executed by the AI per my instruction.  
# Implements: psi field evolution (KG-like), computes Gravity = lap(space + current^2) \* psi,  
# integrates multiple test particles responding to F = -grad(Gravity) (no backreaction).  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
# ----- Parameters -----  
L = 200.0  
N = 4096  
dx = L / N  
x = np.linspace(0, L, N, endpoint=False)  
  
c = 1.0 # wave speed  
mu = 0.1 # mass-like term  
lam = 0.0 # nonlinear lambda  
dt = 0.3 \* dx / c # CFL safety factor (c\*dt/dx <= 1)  
t\_max = 400.0  
n\_steps = int(t\_max / dt)  
  
# ----- Initial fields -----  
psi0\_amp = 1.0  
psi0\_center = L \* 0.5  
psi0\_sigma = 3.0  
  
psi = -psi0\_amp \* np.exp(-0.5 \* ((x - psi0\_center) / psi0\_sigma)\*\*2)  
psi\_prev = psi.copy() # zero initial time derivative => psi\_prev = psi - dt \* psi\_t = psi  
  
# background 'space' and 'current' (static example)  
s0 = 0.0  
space = s0 + 0.0 \* x  
c0 = 2.0  
current\_center = psi0\_center + 20.0  
current\_sigma = 8.0  
current = c0 \* np.exp(-0.5 \* ((x - current\_center) / current\_sigma)\*\*2)  
  
# particle ensemble  
n\_particles = 50  
rng = np.random.default\_rng(12345)  
x\_p = psi0\_center + 5.0 + 2.0 \* (rng.random(n\_particles)-0.5) # near rim  
v\_p = np.zeros(n\_particles)  
  
# utility finite-difference operators (periodic)  
def laplacian(f):  
 return (np.roll(f, -1) - 2.0\*f + np.roll(f, 1)) / dx\*\*2  
  
def gradient(f):  
 return (np.roll(f, -1) - np.roll(f, 1)) / (2.0\*dx)  
  
# potential derivative V'(psi)  
def Vprime(psi):  
 return mu\*\*2 \* psi + lam \* psi\*\*3  
  
# interpolation: linear from grid to particle positions  
def interp\_linear(grid\_vals, x\_grid, x\_points):  
 # x\_grid assumed uniform and periodic  
 s = (x\_points - x\_grid[0]) / dx  
 i = np.floor(s).astype(int) % grid\_vals.size  
 frac = s - np.floor(s)  
 return (1-frac) \* grid\_vals[i] + frac \* grid\_vals[(i+1) % grid\_vals.size]  
  
# time integration arrays  
psi\_curr = psi.copy()  
psi\_prev = psi\_prev.copy()  
  
# diagnostics storage (sampled)  
traj\_x = []  
traj\_v = []  
time\_store = []  
  
# ----- Time loop (explicit second-order) -----  
for n in range(n\_steps):  
 t = n \* dt  
  
 # compute A = lap(space + current^2)  
 A = laplacian(space + current\*\*2)  
  
 # compute lap(psi)  
 lap\_psi = laplacian(psi\_curr)  
  
 # update psi using central difference in time  
 psi\_next = 2.0 \* psi\_curr - psi\_prev + dt\*\*2 \* (c\*\*2 \* lap\_psi - Vprime(psi\_curr))  
  
 # compute Gravity and its gradient on grid  
 Gravity = A \* psi\_curr  
 gradG = gradient(Gravity)  
  
 # particle acceleration: interpolate gradG to particle positions  
 gradG\_at\_particles = interp\_linear(gradG, x, x\_p)  
 a\_p = -gradG\_at\_particles # m = 1  
  
 # integrate particles (velocity verlet-like simple explicit)  
 # x\_{n+1} = x\_n + v\_n\*dt + 0.5\*a\_n\*dt^2 (we use current a\_p as approximation)  
 x\_p = (x\_p + v\_p \* dt + 0.5 \* a\_p \* dt\*\*2) % L  
 # recompute accelerations at new positions (single-step update; acceptable for reference)  
 gradG\_at\_particles\_next = interp\_linear(gradG, x, x\_p)  
 a\_p\_next = -gradG\_at\_particles\_next  
 v\_p = v\_p + 0.5 \* (a\_p + a\_p\_next) \* dt  
  
 # step fields  
 psi\_prev, psi\_curr = psi\_curr, psi\_next  
  
 # sample diagnostics every M steps  
 if n % 200 == 0:  
 traj\_x.append(x\_p.copy())  
 traj\_v.append(v\_p.copy())  
 time\_store.append(t)  
  
# Save or plot diagnostics (example)  
# plt.figure(); plt.plot(x, psi\_curr); plt.title('psi final'); plt.show()  
# plt.figure(); plt.hist(x\_p, bins=50); plt.title('particle positions final'); plt.show()

I left plotting commented so the code is safe for automated runs; the plotting lines give immediate diagnostics when I run the script locally.

Expected outcomes & interpreted signatures

* **Bound states**: Particles seeded inside deep ψ wells (negative ψ amplitude with corresponding negative Gravity gradient) will tend to become trapped and show small-amplitude oscillations — visible as stable clusters in the final hist(x\_p).
* **Escape trajectories**: Particles with sufficient initial offset that sample positive gradient zones will accelerate across the domain and not return (characterized by monotonic increase in |v|).
* **ψ-trench surfing (moving current)**: By making , the computed trench moves; particles placed on the trench’s leading edge will surf if the trench speed is below the effective trapping escape speed.
* **Asymmetric fields**: If space(x) has a slope, Gravity becomes asymmetric; particles preferentially drift downhill (a desert wind + slope analogue).
* **Dispersion & stability**: For small amplitude tests, numerically-measured ω(k) should match . Discrepancies indicate resolution or CFL issues.

Numerical sanity-check list

* Ensure c \* dt / dx <= 1 (CFL).
* Monitor E\_field for secular growth; energy drift should be small for linear cases.
* Grid refinement: halving dx should halve the discrete error roughly according to scheme order.
* Particle interpolation: check that linear vs cubic interpolation changes trajectories only slightly at high resolution.
* Validate dispersion relation with plane-wave test.
* Run long-time runs to detect weak instabilities introduced by nonlinearity (reduce dt if needed).

Desert analogy — mapping simulation phenomenology to intuition

* ψ (desert floor) forms wells and ridges: stable wells ↔ dune basins where stones collect.
* space (sand distribution) shapes background topography: gentle slopes bias motion.
* current (wind) squared feeds into A = ∇²[space + current²]: concentrated wind patches carve trenches (pressure gradients) that multiply ψ to produce Gravity.
* Gravity = pressure-field over the floor: dunes (force patterns) emerge where A × ψ has strong gradients.
* Particles surf trenches (ψ-trench surfing): stones caught on moving dunes ride along until escape — analogous to particles surfing moving gravitational-wave-like features.

Phase-level insights (preliminary; computational)

The architecture

yields multiplicative coupling between geometric curvature (A) and substrate amplitude (ψ). This multiplicative structure naturally produces scale-dependent effects: small ψ amplitude suppresses gravity even if local curvature A is large, while strong ψ wells amplify even modest curvature patches.

Numerically, this predicts regimes of selective trapping: regions where ψ is large in magnitude act as effective potential wells because Gravity ∝ ψ, but the fine structure is shaped by A. If A has high spatial frequency (sharp wind-induced trenches), the resulting Gravity can create small-scale structure in which particles cluster — visible as emergent “micro-dunes”.

Validation experiments I will (or we can) run next

* Resolution sweep: N = 1024, 2048, 4096; check convergence of particle statistics.
* Moving current family: set , sweep v\_c to find critical surfing speed.
* Nonlinear ψ: enable λ ≠ 0 to explore tunneling-like escapes, breathers, and localized oscillons.
* 2D prototype: replicate the core logic on a 2D periodic grid and run a small ensemble (N ~ 512²) to observe dune networks and particle channeling.
* Backreaction experiments (Phase 6.2B / future): introduce small backreaction term where particle density ρ\_p(x,t) sources ψ (e.g., extra term in V’ or direct coupling) and study feedback loops.

Deliverables embedded in this part

* Discrete derivations (above).
* Reference 1D simulation code (above).
* Diagnostics and convergence plan.
* Desert-analogy mapping for interpretation and communication.

Next steps toward Phase 16 (Onto-Cosmology & Mythic Completion)

* Use validated 1D/2D experiments to propose cosmological scaling arguments: coarse-grain ψ to produce effective metric and measure emergent redshift/deflection in ray-tracing experiments.
* Document thresholds where ψ-statistics (variance, kurtosis) map to macroscopic curvature regimes.
* Build minimal symbolic-computational mapping: encode ψ microstate → bitstring statistics → emergent information measures (connect Phase 9 ψ-thermodynamics with Phase 15 symbolic goals).

Brief implementation notes (practical)

* The code above uses periodic boundaries for simplicity; replace with absorbing boundaries when modeling outgoing waves.
* For production runs I (the author) will run the code with optimized compiled loops (Numba/Cython) or use spectral methods for higher accuracy if we need very low dispersion error.

End of Phase 15 — Part 3 (multi-scale simulation & validation).  
I have prepared the numerical scheme, reference code, diagnostics, and the desert-analogy translation. I will run the experiments and notify you as soon as the AI-driven computations produce noteworthy, reproducible insights.