# Phase 8 – Part 2: Geodesic–Force Calibration and Correspondence

Goal. I now calibrate the ψ-metric so that geodesic motion reproduces (or closely approximates) the force law from my core equation:

Plaintext: Gravity(x) = (Laplacian of [space(x) + current(x)^2]) \* ψ(x); F(x) = -∇[Gravity(x)]

1. Effective metric ansätze  
   I consider two ψ-consistent families and test which best matches :

A. Conformal metric (refined):

Plaintext: g^(A)*{mu nu} = α \* ψ(x)^β \* η*{mu nu}

Here , , and in 1+1D. Tuning alters how strongly geodesics feel gradients of ψ.

B. “Optical-potential” inspired split scaling:

Plaintext: g\_tt^(B) = -Φ(x); g\_xx^(B) = 1 / Φ(x)

Choose as a monotone function of so that spatial curvature encodes the potential-like structure. Natural choices:

with calibration parameter .

Rationale. (B) introduces asymmetric scaling between time and space that acts more like an effective potential geometry; this often improves agreement with force-derived trajectories.

1. Matching condition (heuristic, non-relativistic limit)  
   For slow motion, take as the evolution parameter. The spatial geodesic equation under (B) reduces (heuristically) to:

Plaintext: ẍ ≈ -1/2 \* ∂/∂x [ ln Φ(x) ]

To match the force-derived acceleration , a sufficient condition is:

This yields a parameter-free (up to global scale) metric candidate:

Plaintext: Φ(x) ∝ exp(2 \* Gravity(x))  
Plaintext: g\_tt = -exp(2 \* Gravity(x)); g\_xx = exp(-2 \* Gravity(x))

Insight. This mapping ties geodesic curvature directly to the Gravity field from ψ-gravity. It recovers the correct direction for attraction where ∂\_x Gravity pulls inward.

1. Simulation plan (Part 2)  
   I have the AI run two comparative integrations:

* Geodesics with metric (B) using
* Force-based motion with (same as Part 1).

We then overlay the trajectories from geodesic integration and force-based integration and measure deviation. The metric is deemed calibrated if deviations remain small for slow initial speeds (and small spatial gradients).

1. Part 2 — Equations (rendered + plain text)

**Metric (B-calibrated):**

Plaintext: gμν(x) = ( -e^(2 \* Gravity(x)) 0 0 e^(-2 \* Gravity(x)) )

**Christoffel symbols** (ψ enters through ):  
(Nonzero components, 1+1D)

Plaintext: Γ^t\_{t x} = ∂*x(Gravity(x))*  
*Γ^x*{t t} = e^(4 \* Gravity(x)) \* ∂*x(Gravity(x))*  
*Γ^x*{x x} = -∂\_x(Gravity(x))

**Geodesics** , :

Plaintext: ẗ + 2 (∂\_x Gravity) \* ṫ \* ẋ = 0  
ẍ + e^(4 \* Gravity) (∂\_x Gravity) \* ṫ^2 - (∂\_x Gravity) \* ẋ^2 = 0

**Non-relativistic heuristic** (take , , small ):

Plaintext: x\_ddot ≈ -d(Gravity)/dx

1. Part 2 — Python simulation (saved as phase8\_part2\_sim.py)

# phase8\_part2\_sim.py  
# Compare metric (B) geodesics vs. force-based Newtonian motion in 1D.  
import numpy as np  
from scipy.integrate import solve\_ivp  
import matplotlib.pyplot as plt  
  
# --- Field definitions (sample) ---  
def psi\_gaussian(x, sigma=1.0):  
 return np.exp(-x\*\*2 / sigma\*\*2)  
  
def space\_example(x):  
 # simple spatial bump (can be replaced)  
 return np.exp(-x\*\*2 / 4.0)  
  
def current\_example(x):  
 # simple current profile  
 return 0.5 \* np.tanh(x)  
  
# compute 1D Laplacian with central differences on a grid (for discrete precomputation)  
def laplacian\_1d(field, x):  
 dx = x[1] - x[0]  
 lap = np.zeros\_like(field)  
 lap[1:-1] = (field[2:] - 2\*field[1:-1] + field[:-2]) / dx\*\*2  
 # second-order one-sided at boundaries  
 lap[0] = (field[2] - 2\*field[1] + field[0]) / dx\*\*2  
 lap[-1] = (field[-1] - 2\*field[-2] + field[-3]) / dx\*\*2  
 return lap  
  
# build Gravity(x) on a grid  
def build\_gravity(x\_grid, psi\_fn, space\_fn, current\_fn):  
 space\_vals = space\_fn(x\_grid)  
 current\_vals = current\_fn(x\_grid)  
 arg = space\_vals + current\_vals\*\*2  
 lap\_arg = laplacian\_1d(arg, x\_grid)  
 psi\_vals = psi\_fn(x\_grid)  
 Gravity = lap\_arg \* psi\_vals  
 return Gravity  
  
# interpolation helpers  
from scipy.interpolate import interp1d  
  
# --- Geodesic ODE system (1+1D, affine parameter lambda) ---  
def geodesic\_rhs(lambda\_, y, grav\_interp):  
 # y = [t, x, t\_dot, x\_dot]  
 t, x, t\_dot, x\_dot = y  
 G = grav\_interp(x)  
 dGdx = (grav\_interp(x + 1e-6) - grav\_interp(x - 1e-6)) / (2e-6)  
 # Christoffel components (1+1D calibrated metric)  
 Gamma\_t\_tx = dGdx  
 Gamma\_x\_tt = np.exp(4\*G) \* dGdx  
 Gamma\_x\_xx = -dGdx  
 t\_ddot = -2.0 \* Gamma\_t\_tx \* t\_dot \* x\_dot  
 x\_ddot = - Gamma\_x\_tt \* t\_dot\*\*2 - Gamma\_x\_xx \* x\_dot\*\*2  
 return [t\_dot, x\_dot, t\_ddot, x\_ddot]  
  
# --- Force-based Newtonian ODE system ---  
def force\_rhs(t, y, grav\_interp):  
 # y = [x, v]  
 x, v = y  
 # numerical derivative of Gravity  
 dGdx = (grav\_interp(x + 1e-6) - grav\_interp(x - 1e-6)) / (2e-6)  
 a = -dGdx  
 return [v, a]  
  
# --- Main experiment ---  
def run\_comparison():  
 # grid  
 x\_grid = np.linspace(-10, 10, 4001)  
 Gravity = build\_gravity(x\_grid, psi\_gaussian, space\_example, current\_example)  
 grav\_interp = interp1d(x\_grid, Gravity, kind='cubic', fill\_value='extrapolate')  
  
 # initial conditions near center, small velocity  
 t0 = 0.0  
 x0 = 1.0  
 t\_dot0 = 1.0 # affine normalization (roughly dt/dλ)  
 x\_dot0 = 0.0  
  
 # Geodesic integration  
 y0\_geo = [t0, x0, t\_dot0, x\_dot0]  
 sol\_geo = solve\_ivp(lambda lam, y: geodesic\_rhs(lam, y, grav\_interp),  
 [0, 50.0], y0\_geo, max\_step=0.01, rtol=1e-8, atol=1e-10)  
  
 # Force-based integration (Newtonian)  
 y0\_force = [x0, 0.0]  
 sol\_force = solve\_ivp(lambda t, y: force\_rhs(t, y, grav\_interp),  
 [0, 50.0], y0\_force, max\_step=0.01, rtol=1e-8, atol=1e-10)  
  
 # align outputs for plotting (use geodesic's lambda as time axis)  
 plt.figure(figsize=(8,5))  
 plt.plot(sol\_geo.t, sol\_geo.y[1], label='Geodesic x(λ)')  
 # interpolate force solution at geodesic times  
 x\_force\_at\_geo\_t = np.interp(sol\_geo.t, sol\_force.t, sol\_force.y[0])  
 plt.plot(sol\_geo.t, x\_force\_at\_geo\_t, '--', label='Force-based x(t)')  
 plt.xlabel('affine parameter / time')  
 plt.ylabel('x')  
 plt.legend()  
 plt.title('Comparison: Metric (B) Geodesic vs Force-based trajectory')  
 plt.grid(True)  
 plt.show()  
  
 # compute deviation metric  
 deviation = np.sqrt(np.mean((sol\_geo.y[1] - x\_force\_at\_geo\_t)\*\*2))  
 print(f"RMS deviation between geodesic and force-based x: {deviation:.5e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run\_comparison()

### Insight / Breakthrough Flag (Phase 8 so far)

Treating ψ as geometry becomes predictive when I tie the metric directly to the Gravity field:

**Metric–Gravity correspondence**

* g\_tt = −exp(2 \* Gravity(x))
* g\_xx = exp(−2 \* Gravity(x))

(Plaintext: g\_tt = -e^(2 \* Gravity(x)), g\_xx = e^(-2 \* Gravity(x)))

This yields a **practical equivalence** between geodesics and the force law in the slow-motion regime, aligning ψ-geometry with the operational definition of gravity established in earlier phases.

**Breakthrough cue**: ψ is not just a substrate — it **programs geometry** such that the force derived from

∇²[space(x) + current(x)²]

is encoded in the metric connections themselves.