# Phase 5 — Part 5.9: Boundary Conditions & Finite Domains

## Goal

I formalize how ψ behaves on finite computational domains, contrasting reflecting (energy-conserving) boundaries with absorbing (energy-exporting) boundaries.  
The outcome is a clean catalog of boundary conditions (BCs), an energy-flux bookkeeping compatible with my ψ–gravity coupling, and reference Python code that implements these BCs on a 1D lattice for repeatable tests (reflection, transmission, absorption, and stability windows).

## Core ψ–Gravity Statement (for continuity)

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

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

Analogy (desert model):  
ψ = desert floor  
space = sand distribution  
current = wind intensity  
Gravity = dune pressure  
Force = dune slope

## Setting: Finite 1D Lattice

Let be discretized into points with spacing .  
I denote field arrays by , with .

I maintain the coupled definition of Gravity for diagnostics and forces:

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

## Test ψ Evolution Law

To probe boundaries I use a damped, curvature-coupled wave equation:

Plain text:  
ψ\_tt = c² ψ\_xx − β ψ\_t − λ Gravity(x), with Gravity(x) = κ(x) ψ

Where:  
- = signal speed (sets Courant limits)  
- = bulk damping (distinct from boundary absorption)  
- = ψ–gravity coupling

This model preserves the theory’s core equation (it uses Gravity as defined), while giving me a controlled wave-like mechanism to measure reflection and absorption at boundaries.

## Discrete Operators

Centered Laplacian (interior):

Plain text:  
(ψ\_xx)i = (ψ{i−1} − 2ψ\_i + ψ{i+1}) / (Δx)², for 1 ≤ i ≤ N−2

Curvature κ from :

Plain text:  
κ\_i = (q\_{i−1} − 2 q\_i + q\_{i+1}) / (Δx)²

## Boundary Conditions (catalog)

### 1. Periodic (torus)

Plain text:  
ψ\_0 = ψ\_{N−1}; ghost left = ψ\_{N−2}; ghost right = ψ\_1

Energy: conserved, no flux.

### 2. Reflecting / Neumann

Plain text:  
∂x ψ(0) = 0, ∂x ψ(L) = 0

Discrete mirrors:

Plain text:  
left ghost = ψ\_1; right ghost = ψ\_{N−2}

Energy: conserved (modulo β).

### 3. Fixed-edge / Dirichlet

Plain text:  
ψ(0) = ψ\_L, ψ(L) = ψ\_R

### 4. Impedance / Robin (mixed)

Plain text:  
α ψ + β\_b ∂x ψ = 0 at x=0 and x=L

### 5. Sponge / Absorbing Layer

Plain text:  
ψ\_tt = c² ψ\_xx − (β + β(x)) ψ\_t − λ κ(x) ψ

### 6. First-Order One-Way (outflow)

Plain text:  
ψ\_t(L,t) + c ψ\_x(L,t) = 0

## Energy Bookkeeping

Energy density:

Plain text:  
E\_density = 0.5 ψ\_t² + 0.5 c² ψ\_x² + 0.5 λ κ(x) ψ²

Energy balance law:

Plain text:  
d/dt ∫₀ᴸ E\_density dx = −[c² ψ\_t ψ\_x]₀ᴸ − ∫₀ᴸ β ψ\_t² dx − ∫₀ᴸ β(x) ψ\_t² dx + (λ/2) ∫₀ᴸ κ\_t(x) ψ² dx

## Reflection/Absorption Diagnostics

Initial Gaussian-modulated packet:

Plain text:  
ψ(x,0) = A exp(−(x−x0)²/(2σ²)) cos(k0 x)  
ψ\_t(x,0) = −A c k0 exp(−(x−x0)²/(2σ²)) sin(k0 x)

Reflection coefficient:

Plain text:  
R = [∫\_{Ω\_post} E\_ref(x) dx] / [∫\_{Ω\_pre} E\_inc(x) dx]

Absorption fraction:

Plain text:  
A\_b ≈ 1 − R − (ΔE\_bulk / E\_inc)

## Stability Conditions

CFL:

Plain text:  
c Δt / Δx ≤ 1

Additional coupling:

Plain text:  
Δt ≲ min(Δx/c, 1 / sqrt(λ ||κ||\_∞ + β²))

## Reference Python (1D): Boundary Conditions & Diagnostics

import numpy as np  
  
# -----------------------------  
# Phase 5.9: 1D ψ evolution with BCs  
# -----------------------------  
  
# Domain and discretization  
L = 200.0  
N = 2001  
x = np.linspace(0.0, L, N)  
dx = x[1] - x[0]  
  
# Physical parameters  
c = 1.0 # wave speed  
beta\_bulk = 0.0 # bulk damping  
lam = 0.02 # ψ–gravity coupling lambda  
  
# Background fields: space(x), current(x)  
def space\_field(x):  
 return 0.02 \* np.exp(-((x-0.6\*L)\*\*2) / (2\*(0.1\*L)\*\*2))  
  
def current\_field(x):  
 return 0.4 \* np.exp(-((x-0.35\*L)\*\*2) / (2\*(0.06\*L)\*\*2))  
  
q = space\_field(x) + current\_field(x)\*\*2  
  
# Compute kappa  
kappa = np.zeros\_like(x)  
kappa[1:-1] = (q[:-2] - 2.0\*q[1:-1] + q[2:]) / dx\*\*2  
kappa[0] = (q[1] - 2.0\*q[0] + q[1]) / dx\*\*2  
kappa[-1] = (q[-2] - 2.0\*q[-1] + q[-2]) / dx\*\*2  
  
# Initial wave packet  
A = 1.0  
x0 = 0.25 \* L  
sigma = 0.03 \* L  
k0 = 2\*np.pi / (0.12\*L)  
  
psi = A \* np.exp(-((x-x0)\*\*2)/(2\*sigma\*\*2)) \* np.cos(k0\*x)  
psi\_t = -A \* c \* k0 \* np.exp(-((x-x0)\*\*2)/(2\*sigma\*\*2)) \* np.sin(k0\*x)  
  
# Time stepping  
cfl = 0.9  
dt = cfl \* dx / c  
Tmax = 280.0  
steps = int(Tmax / dt)  
  
# Absorbing layer setup  
BC\_left = 'neumann'  
BC\_right = 'sponge'  
  
w = int(0.10\*N)  
beta\_edge = np.zeros\_like(x)  
if BC\_right == 'sponge':  
 beta\_max = 0.4  
 taper = np.linspace(0, np.pi/2, w)  
 beta\_edge[-w:] = beta\_max \* np.sin(taper)\*\*2  
  
# Robin parameters  
alpha\_robin = 1.0  
beta\_robin = c  
  
def apply\_BC(psi\_arr, psi\_prev=None):  
 if BC\_left == 'periodic':  
 psi\_arr[0] = psi\_arr[-2]  
 elif BC\_left == 'neumann':  
 psi\_arr[0] = psi\_arr[1]  
 elif BC\_left == 'dirichlet':  
 psi\_arr[0] = 0.0  
 elif BC\_left == 'robin':  
 psi\_arr[0] = (psi\_arr[1]) / (1 + (alpha\_robin\*dx/beta\_robin))  
  
 if BC\_right == 'periodic':  
 psi\_arr[-1] = psi\_arr[1]  
 elif BC\_right == 'neumann':  
 psi\_arr[-1] = psi\_arr[-2]  
 elif BC\_right == 'dirichlet':  
 psi\_arr[-1] = 0.0  
 elif BC\_right == 'robin':  
 psi\_arr[-1] = (psi\_arr[-2]) / (1 + (alpha\_robin\*dx/beta\_robin))  
  
def laplacian(phi):  
 tmp = phi.copy()  
 tmp0 = tmp[1]  
 tmpN = tmp[-2]  
 return np.concatenate(([tmp[1] - 2\*tmp[0] + tmp0],  
 (tmp[:-2] - 2\*tmp[1:-1] + tmp[2:]),  
 ([tmpN - 2\*tmp[-1] + tmp[-2]]))) / dx\*\*2  
  
def energy\_density(psi\_arr, psi\_t\_arr):  
 psi\_x = np.gradient(psi\_arr, dx, edge\_order=2)  
 return 0.5\*psi\_t\_arr\*\*2 + 0.5\*(c\*\*2)\*psi\_x\*\*2 + 0.5\*lam\*kappa\*(psi\_arr\*\*2)  
  
# Diagnostics  
E\_total = []  
R\_time = []  
probe\_left = slice(0, int(0.08\*N))  
probe\_right = slice(int(0.92\*N), N)  
  
E\_inc = np.trapz(energy\_density(psi, psi\_t)[probe\_left], x[probe\_left]) + 1e-12  
  
psi\_prev = psi - dt\*psi\_t  
  
for n in range(steps):  
 psi\_bc = psi.copy()  
 apply\_BC(psi\_bc, psi\_prev)  
  
 psi\_xx = laplacian(psi\_bc)  
 Gravity = kappa \* psi\_bc  
 beta\_tot = beta\_bulk + beta\_edge  
  
 if n == 0:  
 v\_half\_prev = psi\_t.copy()  
 a\_core = (c\*\*2)\*psi\_xx - lam\*Gravity  
 num = (1 - 0.5\*beta\_tot\*dt)  
 den = (1 + 0.5\*beta\_tot\*dt)  
 v\_half = (num/den)\*v\_half\_prev + (dt/den)\*a\_core  
  
 psi\_next = psi + dt \* v\_half  
  
 Ed = energy\_density(psi\_bc, v\_half)  
 E\_total.append(np.trapz(Ed, x))  
  
 if n\*dt > (L/c)\*0.6:  
 E\_ref = np.trapz(Ed[probe\_right], x[probe\_right])  
 R\_time.append(E\_ref / E\_inc)  
  
 psi\_prev, psi = psi, psi\_next  
 v\_half\_prev = v\_half  
  
R\_est = np.max(R\_time) if len(R\_time) > 0 else 0.0  
print(f"Estimated reflection coefficient R ~ {R\_est:.4f}")  
print(f"Total energy (first/last): {E\_total[0]:.6f} -> {E\_total[-1]:.6f}")