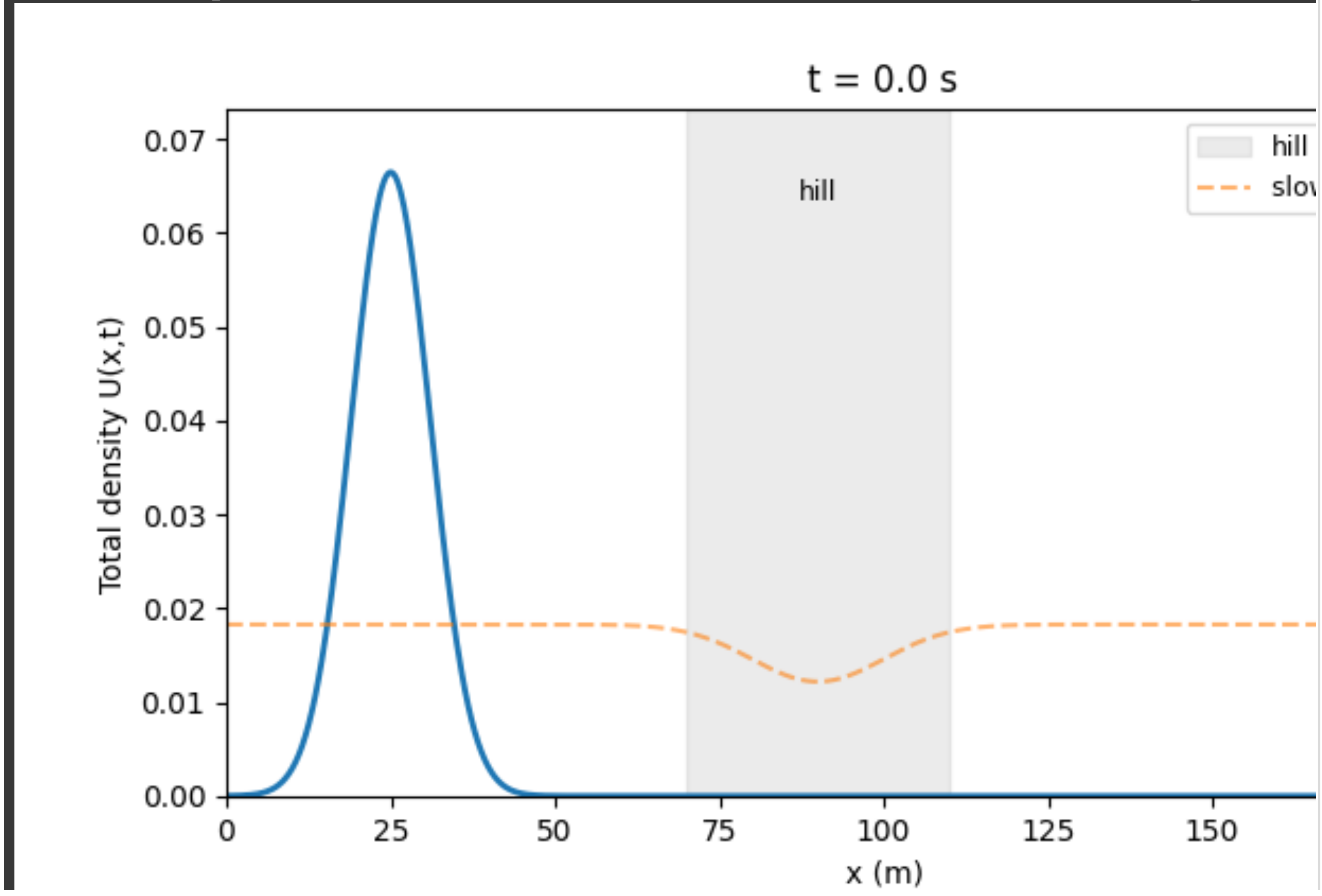
```
import numpy as np, matplotlib.pyplot as plt
from IPython.display import HTML
from matplotlib.animation import FuncAnimation
import ipywidgets as widgets

# --- PARAMETERS ---
L, N, T_final = 200.0, 800, 180.0
speeds  = np.array([2.2, 2.6, 3.0, 3.6, 4.2])
weights = np.array([0.42, 0.30, 0.17, 0.08, 0.03])
lam, D  = 0.0, 0.03
x0, sigma0, u0_total_mass = 25.0, 6.0, 1.0
h, w = 90.0, 20.0                        # hill center & width
def phi_raw(x): return 3.0 - np.exp(-2*((x-h)/w)**2)
phi_normalize = True

# --- DISCRETIZATION ---
x = np.linspace(0, L, N); dx = x[1]-x[0]
raw = phi_raw(x); phi_vals = raw/raw.max() if phi_normalize else raw
V = np.outer(speeds, phi_vals); K = len(speeds)
base = np.exp(-0.5*((x-x0)/sigma0)**2); base /= np.trapz(base, x)
U0 = np.vstack([w_i * u0_total_mass * base for w_i in weights])

vmax = np.max(V)
dt_adv, dt_diff = 0.45*dx/vmax, 0.25*dx*dx/D
dt, nsteps = min(dt_adv, dt_diff, 0.07), int(np.ceil(T_final/min(dt_adv, 
dt = T_final/nsteps

def step_adv(U,v): F=v*U; return -(F-np.concatenate(([0],F[:-1])))/dx
def step_diff(U): O=np.zeros_like(U); O[1:-1]=(U[2:]-2*U[1:-1]+U[:-2])/dx
def simulate(U0,V,keep=5):
    U=U0.copy(); F=[]; T=[]
    for n in range(nsteps+1):
        if n%keep==0: F.append(U.sum(0)); T.append(n*dt)
        if n==nsteps: break
        U_next=np.zeros_like(U)
        for k in range(K):
            U_next[k]=U[k]+dt*(step_adv(U[k],V[k])+D*step_diff(U[k])-lam*
            U_next[k]=np.maximum(U_next[k],0)
        U=U_next
    return np.array(F), np.array(T)
frames,times=simulate(U0,V,keep=5)
```

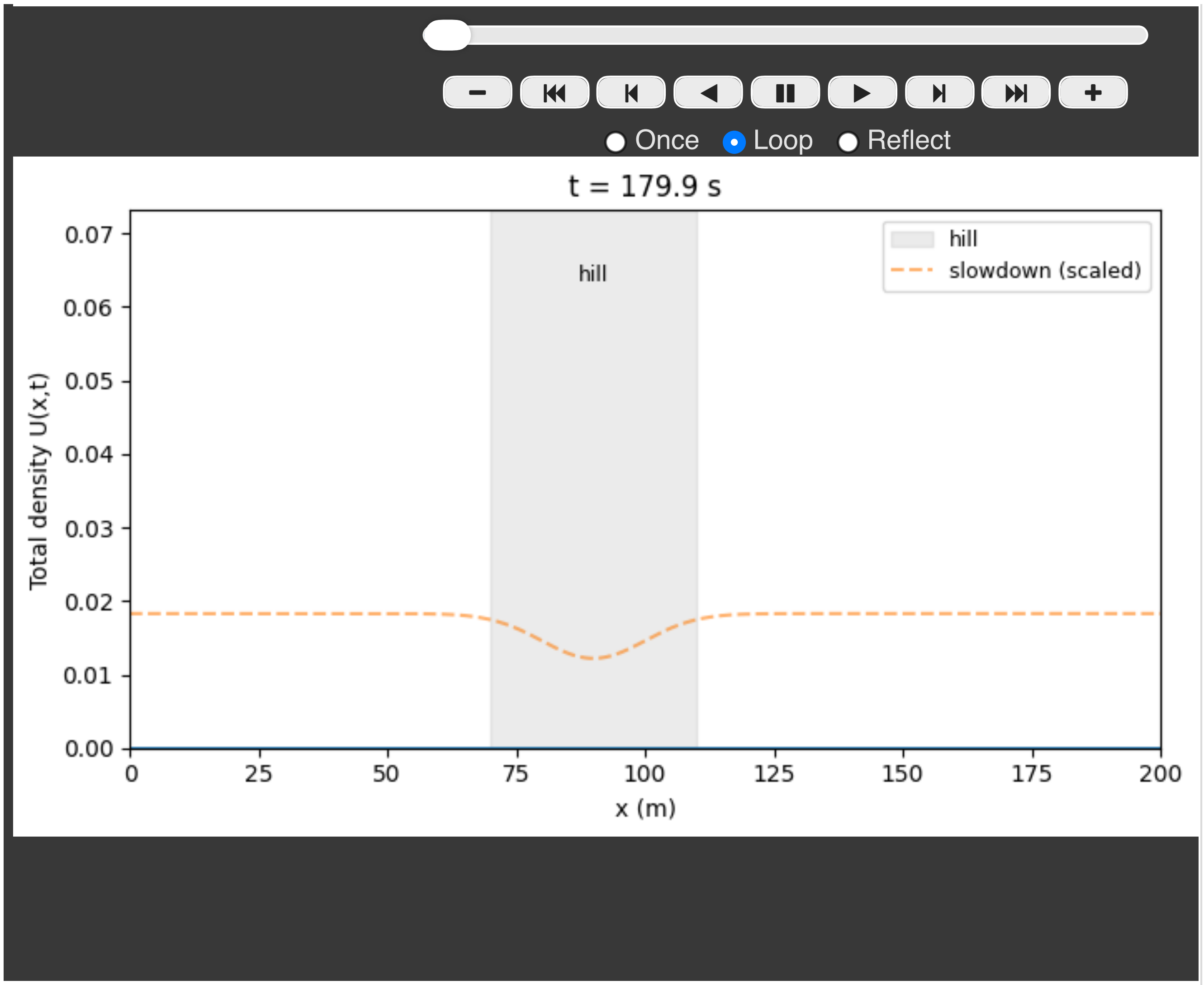
```
/tmp/ipython-input-774049857.  0: DeprecationWarning: `trapz` is deprec
```

```
  base = np.exp(-0.5*((x-x0)/sigma0)**2); base /= np.trapz(base, x)
```

```
fig, ax = plt.subplots(figsize=(8,4.2))
(line,) = ax.plot([],[],lw=2)
ax.set_xlim(0,L); ax.set_ylim(0,1.1*frames.max())
ax.set_xlabel("x (m)"); ax.set_ylabel("Total density U(x,t)")

# HILL BAND + label
ax.axvspan(h-w,h+w,alpha=0.15,color="tab:gray",label="hill",zorder=0)
ax.annotate("hill",xy=(h,0.9*ax.get_ylim()[1]),ha="center",va="top",font
phi_scaled=0.25*phi_vals*ax.get_ylim()[1]
ax.plot(x,phi_scaled,'--',alpha=0.6,label='slowdown (scaled)')
ax.legend(loc="upper right",fontsize=9)
title=ax.set_title("")

def init(): line.set_data([],[]); title.set_text(""); return line,title
def update(i): line.set_data(x,frames[i]); title.set_text(f"t = {times[i

anim=FuncAnimation(fig,update,frames=len(frames),init_func=init,blit=Tru
HTML(anim.to_jshtml())     # Inline HTML animation
```

```
WARNING:matplotlib.animation:Animation size has reached 20991528 bytes, e
```





```
# --- Convert normalized density to actual runner counts ---
TOTAL_RUNNERS = 1500   # starting number of participants

U = U0 * TOTAL_RUNNERS
cum_finish = 0.0

t_arr, oncourse_arr, finished_arr = [0.0], [np.trapz(U.sum(0), dx=dx)], [(

for n in range(1, nsteps + 1):
    # Flux of runners leaving at finish line (x = L)
    F_right_rate = np.sum(V[:, -1] * U[:, -1])     # runners per second
    cum_finish  += F_right_rate * dt               # cumulative finishers

    # Advance one time step (same scheme)
    U_next = np.zeros_like(U)
    for k in range(K):
        adv   = step_adv(U[k], V[k])
        diff  = step_diff(U[k]) if D > 0 else 0.0
        U_next[k] = U[k] + dt * (adv + D * diff)
        U_next[k] = np.maximum(U_next[k], 0.0)
    U = U_next

    # Record
    t = n * dt
    t_arr.append(t)
    oncourse_arr.append(np.trapz(U.sum(0), dx=dx))
    finished_arr.append(cum_finish)

# --- Plot runners still on course vs. finishers ---
plt.figure(figsize=(8,4.4))
plt.plot(t_arr, oncourse_arr, lw=2, label="Runners on course")
plt.plot(t_arr, finished_arr, lw=2, label="Cumulative finishers")

plt.xlabel("Time (s)")
plt.ylabel("Number of runners")
plt.title(f"Race Progress Over Time (Total = {TOTAL_RUNNERS:.0f})")
plt.legend()
plt.grid(alpha=0.25)

# Optional mass-balance check
gap = TOTAL_RUNNERS - (oncourse_arr[-1] + finished_arr[-1])
plt.annotate(f"Check: start - (on+fin) ≈ {gap:.2e}",
             xy=(0.02, 0.92), xycoords="axes fraction", fontsize=9)
plt.show()
```