```
"""
Figure 2 for FoP paper: Gravitational time dilation from the substrate
======================================================================

A 2D lattice of K_N substrate sites is exposed to a localised mass.
The mass induces a spatial profile of substrate tension

    T(r)/T_0 = 1 - (r_s / r) * exp(-r / lambda)

where r_s plays the role of (twice) the gravitational radius and
lambda is the LST-specific screening length (the typical distance
over which a phase distortion is absorbed by other knots in the
substrate).

The local "tick rate" is given by E1:
    Omega(r) / omega_0 = sqrt( T(r)/T_0 )

We integrate the full noisy substrate dynamics (intra-site K_N
coupling + inter-site nearest-neighbour coupling + Brownian noise)
and measure the macroscopic tick rate at every site, then bin
radially and compare against:
    - GR weak-field prediction:  sqrt(1 - r_s/r)
    - LST prediction:            sqrt(1 - (r_s/r) exp(-r/lambda))

Two-panel figure:
    (a)  2D map of Omega(x,y)/omega_0 -- the tick-rate well
    (b)  radial cut: simulation vs GR vs LST
"""

import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import time

# ---------------------------------------------------------------------
# Spatial grid
# ---------------------------------------------------------------------
L      = 60                                  # lattice size (LxL sites)
xs     = np.arange(L) - L/2 + 0.5
X, Y   = np.meshgrid(xs, xs, indexing="xy")
R      = np.sqrt(X**2 + Y**2)
R      = np.maximum(R, 1.0)                  # avoid r=0 singularity

# ---------------------------------------------------------------------
```

```
# Mass / tension profile
# ---------------------------------------------------------------------
r_s    = 0.6                                  # weak-field "Schwarzschild" scale
lam    = 12.0                                 # LST screening length

T_over_T0 = 1.0 - (r_s / R) * np.exp(-R / lam)
T_over_T0 = np.maximum(T_over_T0, 0.05)      # safety floor

omega_0     = 1.0
omega_local = omega_0 * np.sqrt(T_over_T0)   # E1: f propto sqrt(T/mu)

# ---------------------------------------------------------------------
# Substrate dynamics parameters
# ---------------------------------------------------------------------
N       = 3                                  # layers per site (K_N=K_3)
K_intra = 5.0                                # intra-site Kuramoto coupling
J_inter = 0.4                                # nearest-neighbour coupling (kept s
sigma   = 0.05                               # noise amplitude

T_total = 60.0
dt      = 0.02
n_steps = int(T_total / dt)
n_burn  = n_steps // 5                       # discard initial transient
n_meas  = n_steps - n_burn                   # measurement window

# ---------------------------------------------------------------------
# Integration
# ---------------------------------------------------------------------
rng     = np.random.default_rng(2026)
phi     = rng.uniform(0.0, 2*np.pi, size=(L, L, N))
sqrt_dt = np.sqrt(dt)

phi_at_burn = None
print(f"Integrating {L}x{L} grid, N={N}, n_steps={n_steps} ...", flush=True)
t0 = time.time()

for step in range(n_steps):
    # Intra-site Kuramoto coupling via order parameter (per site)
    e_iphi = np.exp(1j * phi)
    Z      = e_iphi.mean(axis=2, keepdims=True)             # (L, L, 1)
    intra  = K_intra * np.imag(Z * np.conj(e_iphi))         # (L, L, N)

    # Inter-site nearest-neighbour coupling (per layer, open boundaries)
    inter  = np.zeros_like(phi)
    # +x neighbour (skip last column)
    inter[:-1, :, :] += np.sin(phi[1:, :, :] - phi[:-1, :, :])
    # -x neighbour (skip first column)
```

```
    inter[1:,  :, :] += np.sin(phi[:-1, :, :] - phi[1:, :, :])
    # +y neighbour
    inter[:, :-1, :] += np.sin(phi[:, 1:, :] - phi[:, :-1, :])
    # -y neighbour
    inter[:, 1:,  :] += np.sin(phi[:, :-1, :] - phi[:, 1:, :])
    inter *= J_inter / 4.0

    noise = sigma * sqrt_dt * rng.standard_normal(size=phi.shape)
    drift = omega_local[..., None] + intra + inter

    phi = phi + drift * dt + noise

    if step == n_burn:
        phi_at_burn = phi.mean(axis=2).copy()

phi_final = phi.mean(axis=2)
print(f"   done in {time.time()-t0:.1f}s", flush=True)

# Macroscopic tick rate at each site
Omega = (phi_final - phi_at_burn) / (n_meas * dt)
Omega_norm = Omega / omega_0

# ---------------------------------------------------------------------
# Radial binning
# ---------------------------------------------------------------------
r_flat = R.ravel()
o_flat = Omega_norm.ravel()
r_max  = L / 2 - 4                          # avoid edge artefacts
r_bins = np.linspace(1.5, r_max, 24)
r_cen  = 0.5 * (r_bins[1:] + r_bins[:-1])

binned_mean = np.empty(len(r_cen))
binned_err  = np.empty(len(r_cen))
for i in range(len(r_cen)):
    mask = (r_flat >= r_bins[i]) & (r_flat < r_bins[i+1])
    if mask.any():
        binned_mean[i] = o_flat[mask].mean()
        binned_err[i]  = o_flat[mask].std() / np.sqrt(mask.sum())
    else:
        binned_mean[i] = np.nan
        binned_err[i]  = np.nan

# Analytical predictions on r_cen
T_lst = 1.0 - (r_s/r_cen) * np.exp(-r_cen/lam)
T_gr  = 1.0 - (r_s/r_cen)
omega_lst = np.sqrt(np.maximum(T_lst, 0.01))
omega_gr  = np.sqrt(np.maximum(T_gr,  0.01))
```

```
# ----------------------------------------------------------------------
# Plot
# ----------------------------------------------------------------------
plt.rcParams.update({
    "font.family":   "serif",
    "font.size":     11,
    "axes.spines.top":   False,
    "axes.spines.right": False,
})
fig, axes = plt.subplots(1, 2, figsize=(13, 5.0))

# --- panel (a): 2D map of macroscopic tick rate ---
ax = axes[0]
vmin, vmax = 0.96, 1.001
im = ax.imshow(
    Omega_norm,
    extent=[xs[0]-0.5, xs[-1]+0.5, xs[0]-0.5, xs[-1]+0.5],
    origin="lower", cmap="viridis",
    vmin=vmin, vmax=vmax,
)
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"$y$")
ax.set_title(r"(a) macroscopic tick-rate field $\Omega(\mathbf{x})/\omega_0$")
cbar = plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
cbar.set_label(r"$\Omega/\omega_0$")

# Mark the centre
ax.plot(0, 0, marker="*", color="white", markersize=14, mec="black", mew=0.6)

# --- panel (b): radial profile ---
ax = axes[1]
ax.errorbar(r_cen, binned_mean, yerr=binned_err,
            fmt="o", color="#1f3b8b", markersize=6,
            elinewidth=0.8, capsize=2,
            label="simulation (radial bins)")
ax.plot(r_cen, omega_lst, "-",  color="black",  lw=1.6,
        label=r"LST: $\sqrt{1 - (r_s/r)\,e^{-r/\lambda}}$")
ax.plot(r_cen, omega_gr,  "--", color="#c0392b", lw=1.6,
        label=r"GR weak field: $\sqrt{1 - r_s/r}$")
ax.axvline(lam, color="0.5", lw=0.8, ls=":", alpha=0.7)
ax.text(lam, 0.962, r"  $\lambda$", color="0.4", fontsize=10, va="bottom")

ax.set_xlabel(r"radial distance $r$")
ax.set_ylabel(r"$\Omega(r)/\omega_0$")
ax.set_title("(b) radial profile: simulation vs. analytic predictions")
ax.legend(frameon=False, fontsize=10, loc="lower right")
```

```
ax.set_ylim(vmin, 1.005)
ax.grid(alpha=0.25)

fig.suptitle(
    r"Gravitational time dilation as a macroscopic property "
    r"of the substrate ($r_s={:.2f}$, $\lambda={:.1f}$)".format(r_s, lam),
    fontsize=12, y=1.02
)
fig.tight_layout()

out_dir = Path("/mnt/user-data/outputs")
out_dir.mkdir(parents=True, exist_ok=True)
out_png = out_dir / "fig2_gravity.png"
out_pdf = out_dir / "fig2_gravity.pdf"
fig.savefig(out_png, dpi=170, bbox_inches="tight")
fig.savefig(out_pdf,           bbox_inches="tight")

# Quantitative summary
chi2_lst = np.nansum(((binned_mean - omega_lst) / binned_err) ** 2)
chi2_gr  = np.nansum(((binned_mean - omega_gr ) / binned_err) ** 2)
n_pts    = np.sum(~np.isnan(binned_mean))
print(f"\nNumber of radial bins: {n_pts}")
print(f"Reduced chi^2 vs LST:  {chi2_lst/n_pts:.2f}")
print(f"Reduced chi^2 vs GR :  {chi2_gr /n_pts:.2f}")
print(f"Saved: {out_png}")
print(f"Saved: {out_pdf}")
```