```
"""
Figure 1 for FoP paper: Emergence of Time and √N Stabilization
================================================================

Vectorised over realisations; coupling computed via the Kuramoto
order parameter so cost is O(R*N) per step instead of O(R*N^2).

For all-to-all coupling on K_N:
   Z[r] = (1/N) Σ_j exp(i φ_j[r])
   (K/N) Σ_j sin(φ_j - φ_i) = K · Im( Z · exp(-i φ_i) )
"""

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

omega_0 = 1.0
K       = 2.0
sigma   = 0.6
T_total = 50.0
dt      = 0.02
n_steps = int(T_total / dt)


def step_coupling(phi):
    """Vectorised Kuramoto coupling using order parameter.
    phi: (R, N) array. Returns coupling array of same shape."""
    e_iphi = np.exp(1j * phi)                # (R, N)
    Z      = e_iphi.mean(axis=1, keepdims=True)  # (R, 1)
    # K * Im( Z * exp(-i phi) )
    return K * np.imag(Z * np.conj(e_iphi))


def simulate_kN_batch(N, R, dt, n_steps, seed=0, store_traj=False):
    rng = np.random.default_rng(seed)
    phi = rng.uniform(0.0, 2*np.pi, size=(R, N))
    sqrt_dt = np.sqrt(dt)
    if store_traj:
        traj = np.empty((n_steps, R, N), dtype=np.float64)
    for step in range(n_steps):
        if store_traj:
            traj[step] = phi
        if N > 1:
```

```
            coupling = step_coupling(phi)
        else:
            coupling = 0.0
        noise = sigma * sqrt_dt * rng.standard_normal(size=(R, N))
        phi   = phi + (omega_0 + coupling) * dt + noise
    if store_traj:
        return phi, traj
    return phi


# ---------- (A) Trajectory demo ----------
print("(A) trajectory demo...", flush=True)
t0 = time.time()
N_demo = 16
_, traj = simulate_kN_batch(N_demo, R=1, dt=dt, n_steps=n_steps,
                            seed=2026, store_traj=True)
traj = traj[:, 0, :]
times    = np.arange(n_steps) * dt
phi_dev  = traj - omega_0 * times[:, None]
phi_avg  = phi_dev.mean(axis=1)
print(f"   done in {time.time()-t0:.2f}s", flush=True)

# ---------- (B) √N scaling ----------
print("(B) sqrt(N) scaling sweep...", flush=True)
N_values       = np.array([1, 2, 4, 8, 16, 32, 64, 128])
n_realisations = 200
spreads        = []

for N in N_values:
    t0 = time.time()
    phi_final = simulate_kN_batch(N, n_realisations, dt, n_steps,
                                  seed=10_000 + N)
    macros = phi_final.mean(axis=1) - omega_0 * T_total
    spreads.append(np.std(macros))
    print(f"   N={N:4d}  std={spreads[-1]:.4f}  ({time.time()-t0:.2f}s)",
          flush=True)

spreads = np.asarray(spreads)
ref     = spreads[0] / np.sqrt(N_values)
log_N   = np.log(N_values.astype(float))
log_S   = np.log(spreads)
slope, intercept = np.polyfit(log_N, log_S, 1)
print(f"\nFitted slope: {slope:+.4f}  (theoretical: -0.5)", flush=True)

# ---------- 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=(12, 4.6))

ax = axes[0]
for n in range(min(8, N_demo)):
    ax.plot(times, phi_dev[:, n], color="0.55", lw=0.6, alpha=0.7)
ax.plot(times, phi_avg, color="#c0392b", lw=1.8,
        label=rf"ensemble mean $\langle\phi\rangle$ ($N={N_demo}$)")
ax.plot([], [], color="0.55", lw=0.6,
        label=r"individual layers $\phi_n$")
ax.set_xlabel(r"substrate parameter $\tau$")
ax.set_ylabel(r"phase deviation $\phi - \omega_0 \tau$")
ax.set_title("(a) individual phases vs. macroscopic mean")
ax.legend(loc="upper left", frameon=False, fontsize=10)
ax.grid(alpha=0.25)

ax = axes[1]
ax.loglog(N_values, spreads, "o", color="#1f3b8b", markersize=7,
          label=r"simulation: std of $\langle\phi\rangle(\tau_f)$")
ax.loglog(N_values, ref, "--", color="#c0392b", lw=1.5,
          label=r"$\propto N^{-1/2}$")
ax.set_xlabel(r"number of layers $N$")
ax.set_ylabel(r"std. dev. of macroscopic phase at $\tau_f$")
ax.set_title(r"(b) $\sqrt{N}$ stabilisation of the macroscopic clock")
ax.legend(frameon=False, fontsize=10, loc="upper right")
ax.grid(alpha=0.25, which="both")
ax.text(0.04, 0.06,
        f"fitted slope = {slope:+.3f}\n(theory = $-0.500$)",
        transform=ax.transAxes, fontsize=9.5,
        bbox=dict(facecolor="white", edgecolor="0.7",
                  boxstyle="round,pad=0.3"))

fig.suptitle(
    r"Emergence of macroscopic time from $K_N$-coupled phases",
    fontsize=13, 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 / "fig1_time_emergence.png"
out_pdf = out_dir / "fig1_time_emergence.pdf"
fig.savefig(out_png, dpi=170, bbox_inches="tight")
fig.savefig(out_pdf,           bbox_inches="tight")
```

```
print(f"Saved: {out_png}")
print(f"Saved: {out_pdf}")
```