Contents

Paper 27 Integration — Reviewer Summary

Repository: anulum/scpn-control **Branch:** `main` **Commits:** `81704be..HEAD` **Date:** 2026-02-26 **Author:** Miroslav Šotek — ORCID 0009-0009-3560-0851 **Paper 27:** academia.edu | arXiv: 2004.06344 (Kuramoto–Sakaguchi finite-size) **PDF export:** `REVIEWER_PAPER27_INTEGRATION.pdf`

---

1. What Was Requested

The reviewer asked for the **Kuramoto–Sakaguchi + global field driver** from SCPN Paper 27 ("The Knm Matrix") to be woven into the `scpn-control` tokamak control codebase. Specifically:

1. The sin(Ψ −) **"intention as carrier" injection**, where Ψ is a Lagrangian pull parameter with **no own dynamics** (no $\dot{\Psi}$ equation).
2. The full 16-layer **Knm coupling matrix** with calibration anchors and cross-hierarchy boosts.
3. A **Rust sub-ms kernel** for the hot Kuramoto loop (rayon-parallelised).
4. **PAC cross-layer SNN sketch** showing phase-amplitude coupling gating.
5. A **demo notebook** with visualisations and a **markdown export** to `docs/`.

---

2. Master Equation

```
d _{m,i}/dt =  _{m,i}
            + K_{mm} · R_m · sin( _m -  _{m,i} -  _{mm})       [intra-layer]
            + Σ_{n m} K_{nm} · R_n · sin( _n -  _{m,i} -  _{nm})  [inter-layer]
            +  _m · sin(Ψ -  _{m,i})                            [global driver]
```

- **K_{mm}** (diagonal): intra-layer synchronisation strength
- **K_{nm}** (off-diagonal): inter-layer bidirectional causality
- **sin(Ψ —)**: exogenous global field driver — Ψ resolved externally or from mean-field
- **_{nm}**: Sakaguchi phase-lag frustration (optional)
- **R,** : Kuramoto order parameter R · exp(i ·) = exp(i ·)

Reference: arXiv:2004.06344 (generalized Kuramoto–Sakaguchi finite-size)

---

3. Files Created / Modified

3.1 Python — Phase Dynamics Package

File	Lines	Purpose
`src/scpn_control/phase/__init__.py`	36	Package exports
`src/scpn_control/phase/kuramoto.py`	161	Kuramoto–Sakaguchi + sin(Ψ−) + Lyapunov V/ , Rust auto-dispatch
`src/scpn_control/phase/knm.py`	101	Paper 27 Knm matrix builder + OMEGA_N_16
`src/scpn_control/phase/upde.py`	215	Multi-layer UPDE engine + run_lyapunov()
`src/scpn_control/phase/lyapunov_guard.py`	130	Lyapunov stability guardrail (DIRECTOR_AI sync)

3.2 Rust — Sub-ms Kernel

File	Lines	Purpose
`scpn-control-rs/crates/control-math/src/kuramoto.rs`	+195	Rayon-parallelised Kuramoto step + run + 7 unit tests
`scpn-control-rs/crates/control-math/src/lib.rs`	+1	`pub mod kuramoto;`
`scpn-control-rs/crates/control-python/src/lib.rs`	+67	PyO3 bindings: `kuramoto_step()`, `kuramoto_run()`

3.3 FusionKernel Integration

File	Lines	Purpose
`src/scpn_control/core/fusion_kernel.py`	+86	`phase_sync_step()` + `phase_sync_step_lyapunov()`

3.4 Tests

File	Lines	Tests
`tests/test_phase_kuramoto.py`	475	44
`kuramoto.rs` (inline `#[cfg(test)]`)	—	9

3.5 Documentation

File	Purpose
`examples/paper27_phase_dynamics_demo.ipynb`	10-section notebook with plots
`docs/paper27_phase_dynamics.md`	Markdown export of notebook

---

4. Architecture — How It Fits

4.1 Equation Cross-Reference (Paper 27, Eqs. 12–15)

Paper 27 Eq.	Description	Implementation
(12)	Mean-field Kuramoto order parameter: R · e^{i } = (1/N) Σ e^{i _j}	`order_parameter()` in `kuramoto.py:47` / `kuramoto.rs:15`
(13)	Single-layer Kuramoto–Sakaguchi: d _i/dt = _i + K · R · sin(− _i−)	`kuramoto_sakaguchi_step()` in `kuramoto.py:87` / `kuramoto.rs:53`
(14)	Multi-layer UPDE with Knm inter-layer coupling: d _{m,i}/dt = _{m,i} + K_{mm} · R_m · sin(_m− _{m,i}− _{mm}) + Σ_{n m} K_{nm} · R_n · sin(_n− _{m,i}− _{nm})	`UPDESystem.step()` in `upde.py:45`
(15)	Exogenous global field driver: + _m · sin(Ψ− _{m,i}), Ψ exogenous (no ˙Ψ)	term in `kuramoto.py:126-127`, `upde.py:115-116`; `GlobalPsiDriver` in `kuramoto.py:67`

4.2 Module Architecture

```
                FusionKernel

solve_equilibrium()   ← GS solver (untouched)
compute_stability()   ← MHD stability (untouched)
phase_sync_step()     ← NEW: Paper 27 Eqs. 12-15
```

```
scpn_control.phase

kuramoto_sakaguchi_step() [Eq.13]    Rust
  order_parameter()     [Eq.12]    fast-path
  GlobalPsiDriver       [Eq.15]    (rayon,
  wrap_phase()                      sub-ms
                                      N>1000)
KnmSpec / build_knm_paper27()
UPDESystem.step()       [Eq.14]
```

Non-invasive: the GS equilibrium solver, SNN controllers, and all existing code paths are completely untouched. `phase_sync_step()` is a new method on `FusionKernel` that reads defaults from `cfg["phase_sync"]`.

4.3 Ψ Global Driver Flowchart

```
                Caller / Controller



                 psi_mode?


          "external"         "mean_field"


           Ψ = caller-      Ψ = arg( e^{i } )
           supplied float    from oscillator
           (intention        population
            carrier)         (self-organised)




                Ψ resolved (scalar)
                NO ˙Ψ dynamics



             For each oscillator i:
```

```
d _i +=   · sin(Ψ −  _i)

•   > 0: pull toward Ψ
•   = 0: term vanishes
• gain scales both K and



Euler step:  ' =   + dt·d
wrap to (− ,  ]



Return:  ', d , R,  _r, Ψ
```

---

5. Knm Matrix — Paper 27 Specification

```
# Canonical 16-layer natural frequencies (rad/s)
OMEGA_N_16 = [1.329, 2.610, 0.844, 1.520, 0.710, 3.780, 1.055, 0.625,
              2.210, 1.740, 0.480, 3.210, 0.915, 1.410, 2.830, 0.991]

# Base coupling with exponential distance decay
K[i,j] = K_base · exp(−  · |i − j|)    # K_base=0.45,  =0.3

# Calibration anchors (Paper 27 Table 2)
K[0,1] = K[1,0] = 0.302
K[1,2] = K[2,1] = 0.201
K[2,3] = K[3,2] = 0.252
K[3,4] = K[4,3] = 0.154

# Cross-hierarchy boosts (Paper 27 §4.3)
K[0,15] = K[15,0]   0.05   # L1   L16
K[4,6]  = K[6,4]    0.15   # L5   L7
```

---

6. Rust Kernel — Performance Path

The Python solver auto-dispatches to Rust when: - `scpn_control_rs` is importable (maturin build) - `wrap=True` and `alpha=0.0` (common fast-path)

```
// Hot loop - rayon parallel chunks of 64
theta_out
    .par_chunks_mut(64)
    .enumerate()
    .for_each(|(chunk_idx, chunk)| {
        for (local_i, val) in chunk.iter_mut().enumerate() {
            let i = base + local_i;
            let mut dth = om + kr_sin_base * (psi_r - th - alpha).sin();
            if zeta != 0.0 {
                dth += zeta * (psi_global - th).sin();
            }
            *val = wrap_phase(th + dt * dth);
```

```
        }
    });
```

PyO3 bindings expose `kuramoto_step(theta, omega, dt, k, alpha, zeta, psi_external)` and `kuramoto_run(...)` returning NumPy arrays directly.

6.1 Benchmark: Python NumPy vs Rust Rayon

Median wall-time for a single `kuramoto_sakaguchi_step()` with =0.5, Ψ=0.3. Python: NumPy vectorised (AMD Ryzen, single-thread). Rust: Rayon `par_chunks_mut(64)` + `criterion` harness.

N	Python (ms)	Rust (ms)	Speedup
64	0.050	0.003	17.3×
256	0.029	0.033	0.9×
1 000	0.087	0.062	1.4×
4 096	0.328	0.180	1.8×
16 384	1.240	0.544	2.3×

N=64: Rust wins on per-element throughput (no NumPy dispatch overhead). N=256: parity — NumPy SIMD matches rayon for this size. N 1000: Rust rayon parallelism scales; **sub-ms for N=16k** (0.544 ms).

Benchmark source: `benches/bench_kuramoto.rs` (criterion, `--quick` mode).

---

7. Global Field Driver — sin(Ψ −)

`GlobalPsiDriver` resolves Ψ before the integration step:

Mode	Ψ source	Use case
`"external"`	Caller supplies float	Intention-as-carrier injection
`"mean_field"`	arg(exp(i))	Self-organised collective phase

There is **no Ψ̇ equation** — Ψ is a Lagrangian pull parameter. When `zeta > 0`, all oscillators are pulled toward Ψ with strength proportional to `sin(Ψ -  _i)`.

---

8. PAC Cross-Layer Gating + SNN Sketch

8.1 PAC Gate Equation

The UPDE engine supports phase-amplitude coupling gating via `pac_gamma`:

```
# Inter-layer term with PAC gate
pac_gate = 1.0 + pac_gamma * (1.0 - R_source)
d  += gain * pac_gate * K[n,m] * R_n * sin( _n -   -  [n,m])
```

When a source layer is incoherent (low R), the gate amplifies its coupling, implementing the PAC hypothesis that desynchronised layers drive downstream amplitude modulation.

8.2 SNN PAC Full Architecture Sketch

The SNN closed-loop couples spiking neural networks with the Kuramoto phase oscillator population through a PAC gating mechanism:

```
                    SNN-PAC-Kuramoto Closed Loop


               spike rate
    LIF Layer A                 Rate Decoder
    (N_a neurons)      _a(t)             → Ψ mapping
    I_syn = f( )                       Ψ =  (2 / _max
                                     - 1)
          synaptic
          input
                                                 Ψ (exogenous)

    LIF Layer B        PAC gate         Kuramoto
    (N_b neurons)              Oscillators
    w_ab modulated     R_source →       N oscillators
    by R_source       gate strength     d /dt =   + ..
                                + sin(Ψ -  )

          spike output
                                                 R,  _r

                     PAC Feedback Controller

    1. Read R_source from each Kuramoto layer
    2. Compute PAC gate: g = 1 +  ·(1 - R_n)
    3. Modulate inter-layer SNN weights: w' = g · w_base
    4. Inject Kuramoto R into LIF synaptic current:
       I_syn = I_base +   · R · cos( _r -  _preferred)
    5. Map spike rate → Ψ for next Kuramoto step



  Data Flow per Timestep (dt = 1 ms):

  t=0:  Kuramoto step → R_m,  _m for each layer m
  t=1:  PAC gate g_m = 1 +  (1 - R_m) → modulate SNN weights
  t=2:  LIF neurons integrate I_syn(R,  ) → spike/no-spike
  t=3:  Decode spike rate   → Ψ_next =  (2 / _max - 1)
  t=4:  Feed Ψ_next back as exogenous driver → next Kuramoto step


  Key Equations:

  LIF:    dV/dt = -(V - V_rest) + R_mem · I_syn
         if V   V_th: spike, V → V_reset

  PAC gate:  g_{n→m} = 1 +  _PAC · (1 - R_n)

  Synaptic current from Kuramoto:
```

```
    I_syn,i = Σ_j w_ij ·  (t − t_j^spike) +   · R_m · cos( _m− _i)

  Rate→Ψ decoder:  Ψ =   · (2· _window/ _max − 1)
     _window = spike_count / T_window  (T_window = 50 ms)

  Closed-loop stability (Lyapunov candidate):
    V(t) = (1/N) Σ_i (1 − cos( _i − Ψ)) +  ·|  −  _target|²
    dV/dt   0 when   > 0 and SNN rate tracks target
```

8.3 Cross-Layer PAC Routing (Multi-Layer SNN)

```
Layer L1 (Quantum)    Layer L7 (Symbolic)    Layer L16 (Director)

 LIF  (64 neu)         LIF  (64 neu)         LIF (64 neu)
   = 1.329               = 1.055               = 0.991


        K[0,6]=0.15           K[6,15]                K[15,0]=0.05
        PAC g               PAC g                  PAC g


                 Kuramoto Phase Bus (16 layers)

  Per-layer R_m,  _m → PAC gates g_{n→m} → SNN weight mod
  Spike rates  _m → Ψ decoder → exogenous driver feedback

  Cross-hierarchy fast channels:
    L1   L16: K=0.05 (quantum   director)
    L5   L7:  K=0.15 (bio   symbolic)
```

Demo: notebook §9 (SNN closed-loop) and §10 (PAC cross-layer SNN).

---

9. Lyapunov Stability —   Hook

9.1 Python Functions

```python
# Lyapunov candidate V(t) = (1/N) Σ (1 − cos( _i − Ψ))
from scpn_control.phase import lyapunov_v, lyapunov_exponent

V = lyapunov_v(theta, psi)                # scalar   [0, 2]
lam = lyapunov_exponent(v_hist, dt=1e-3)  #   = (1/T)·ln(V_f/V_0)
#   < 0   stable convergence toward Ψ
```

Matches `control-math/kuramoto.rs::lyapunov_v` and `kuramoto_run_lyapunov`.

9.2 UPDE Lyapunov Tracking

`UPDESystem.step()` now returns `V_layer` (per-layer) and `V_global`. `UPDESystem.run_lyapunov()` returns full V histories and per-layer + global :

```python
out = sys.run_lyapunov(200, theta_layers, omega_layers, psi_driver=0.5, pac_gamma=1.0)
# out["V_layer_hist"]  - (n_steps, L)
```

```
# out["lambda_layer"]  - (L,) per-layer Lyapunov exponents
# out["lambda_global"] - scalar global
```

9.3 FusionKernel.phase_sync_step_lyapunov()

Multi-step Kuramoto with Lyapunov tracking:

```
out = kernel.phase_sync_step_lyapunov(
    theta, omega, n_steps=100, dt=0.01,
    zeta=3.0, psi_driver=0.5,
)
# out["lambda"]  - Lyapunov exponent
# out["stable"]  - True if   < 0
# out["V_hist"]  - (100,) trajectory
# out["R_hist"]  - (100,) coherence trajectory
```

9.4 Lyapunov Exponent vs Strength

See `docs/bench_lyapunov_vs_zeta.vl.json` — Vega-Lite plot showing:

- $= 0$: 0 (no convergence, drift)
- $= 0.5$: -0.23 (moderate pull)
- $= 3.0$: -1.83 (strong convergence)
- $= 5.0$: -3.35 (rapid sync)

K=2.0 (Kuramoto coupling) amplifies the effect due to cooperative self-organisation.

---

10. DIRECTOR_AI Guardrail Sync

10.1 LyapunovGuard

`scpn_control.phase.lyapunov_guard.LyapunovGuard` monitors V(t) over a sliding window and flags instability when > 0 for K consecutive windows. Interface mirrors DIRECTOR_AI's `CoherenceScorer` → `CoherenceScore` pattern:

```
from scpn_control.phase import LyapunovGuard

guard = LyapunovGuard(window=50, dt=1e-3, max_violations=3)

# Per-timestep check (online monitoring)
verdict = guard.check(theta, psi)
verdict.approved         # True if stable
verdict.lambda_exp       # current
verdict.score            # stability score   [0, 1]
verdict.consecutive_violations

# Batch check (post-hoc)
verdict = guard.check_trajectory(v_hist)
```

10.2 DIRECTOR_AI Integration

Export to DIRECTOR_AI `AuditLogger` format:

```
d = guard.to_director_ai_dict(verdict)
# {"query": "lyapunov_stability_check",
#  "response": "V=0.42,  =-1.23",
```

```
#   "approved": True,
#   "score": 0.99,
#   "h_factual": 0.0,
#   "halt_reason": ""}
```

This enables the DIRECTOR_AI `CoherenceAgent` to incorporate Lyapunov stability into its dual-entropy coherence score. When > 0 for 3 consecutive windows, the guard issues a refusal — analogous to DIRECTOR_AI's `SafetyKernel` emergency stop when coherence drops below the hard limit.

10.3 Data Flow

```
Kuramoto oscillators →  (t) per timestep


LyapunovGuard.check( , Ψ) → LyapunovVerdict

           approved=True  → continue control loop
           approved=False → HALT / parameter clamp

           to_director_ai_dict() → DIRECTOR_AI AuditLogger
                                    h_factual = max(0,  )
                                    halt_reason logged
```

---

11. Real-Time Dashboard Hook

11.1 RealtimeMonitor

`scpn_control.phase.realtime_monitor.RealtimeMonitor` wraps UPDESystem + LyapunovGuard into a tick-by-tick interface for live control dashboards:

```python
from scpn_control.phase import RealtimeMonitor

monitor = RealtimeMonitor.from_paper27(psi_driver=0.0)
for sample in sensor_stream:
    snap = monitor.tick()
    if not snap["guard_approved"]:
        trigger_safety_halt()
    dashboard.push(snap)
```

Each `tick()` returns: `R_global`, `R_layer`, `Psi_global`, `V_global`, `V_layer`, `lambda_exp`, `guard_approved`, `guard_score`, `latency_us`, and a `director_ai` dict ready for AuditLogger.

11.2 Interactive Benchmark Visualisation

docs/bench_interactive.vl.json — single Vega-Lite chart with 3 vertically concatenated panels:

1. **Python vs Rust Speedup** (log-log, N=64..65k, legend-click filtering)
2. **vs** (K=0 / K=2 configs, stability boundary annotation)
3. **PAC vs No-PAC Latency** (grouped bars with 95% CI error bars)

11.3 CI Benchmark — DIII-D Scale

CI job `python-benchmark` runs Kuramoto steps at DIII-D PCS scale: - N=1000, N=4096 single-step P50 $<$ 5 ms gate - RealtimeMonitor tick (16×50 oscillators) P50 $<$ 50 ms gate

WebSocket Phase Sync Monitor

Figure 1: WebSocket Phase Sync Monitor

11.4 Streamlit Dashboard

`dashboard/control_dashboard.py` — 6 tabs:

1. **Trajectory Viewer** — closed-loop PID/SNN trajectory
2. **Phase Sync Monitor** — live RealtimeMonitor with R/V/ plots + DIRECTOR_AI export
3. **Benchmark Plots** — `bench_interactive.vl.json` embedded as `st.vega_lite_chart`
4. **RMSE Dashboard** — validation summary
5. **Timing Benchmark** — PID vs SNN latency
6. **Shot Replay** — disruption shot viewer

11.5 Rust PyO3 UPDE Tick

`PyRealtimeMonitor` in `control-python/src/lib.rs` wraps the Rust `upde_tick()` multi-layer kernel:

```python
import scpn_control_rs as rs
mon = rs.PyRealtimeMonitor(knm_flat, zeta, theta_flat, omega_flat, L, N_per)
snap = mon.tick()  # returns {R_global, R_layer, V_global, V_layer, Psi_global, tick}
```

Rust `upde_tick` in `control-math/src/kuramoto.rs`: per-layer Kuramoto + inter-layer Knm coupling + PAC gate + Lyapunov V tracking. 11 Rust tests (9 existing + 2 new `test_upde_tick_*`).

11.6 WebSocket Phase Stream

`scpn_control.phase.ws_phase_stream.PhaseStreamServer` — async WebSocket server streaming tick snapshots as JSON frames:

```bash
python -m scpn_control.phase.ws_phase_stream --port 8765 --layers 16 --zeta 0.5
```

Clients receive `{"tick": N, "R_global": ..., "V_global": ..., "lambda_exp": ...}` every tick. Control commands: `{"action": "set_psi", "value": 1.0}`, `{"action": "reset"}`, `{"action": "stop"}`.

11.7 HDF5 / NPZ Trajectory Export

```python
monitor = RealtimeMonitor.from_paper27()
for _ in range(1000):
    monitor.tick()

monitor.save_hdf5("trajectory.h5")   # requires h5py
monitor.save_npz("trajectory.npz")   # numpy only
```

Datasets: R_global, R_layer, V_global, V_layer, lambda_exp, guard_approved, latency_us, Psi_global. HDF5 attributes: L, N_per, psi_driver, pac_gamma, n_ticks.

11.8 Mock DIII-D Shot Loader (CI)

`tests/mock_diiid.py` generates synthetic shots matching real DIII-D npz format (14 fields: time_s, Ip_MA, BT_T, beta_N, q95, ne_1e19, MHD modes, etc.). CI job `e2e-diiid` runs end-to-end tests: - Mock shot generation and round-trip load - Shot-driven RealtimeMonitor (Ψ = f(beta_N)) - NPZ and HDF5 trajectory export verification

Phase Sync Convergence

Figure 2: Phase Sync Convergence

11.9 Streamlit WebSocket Client

`examples/streamlit_ws_client.py` — live Streamlit dashboard consuming WS ticks:

```
# Two-terminal mode
python -m scpn_control.phase.ws_phase_stream --port 8765 --zeta 0.5  # terminal 1
streamlit run examples/streamlit_ws_client.py                          # terminal 2

# Embedded mode (server + client in one process)
streamlit run examples/streamlit_ws_client.py -- --embedded
```

Features: auto-reconnect, R/V/ time-series plots, per-layer bar chart, guard status, Ψ control slider, raw JSON expander, auto-refresh at 3 Hz.

11.10 Phase Sync Live Video

Real data from RealtimeMonitor (500 ticks, 16×50 oscillators, =0.5):

- **MP4**: docs/phase_sync_live.mp4 (418 KB, H.264)
- **GIF**: docs/phase_sync_live.gif (1.1 MB)
- **Generator**: `tools/generate_phase_video.py --ticks 500 --fps 20`

Observed convergence: R=0.92, V→0, =−0.47 (stable), 38 μs/tick.

11.11 PyPI Publish Script

`tools/publish.py` — local build + publish pipeline:

```
python tools/publish.py --dry-run                        # build + twine check
python tools/publish.py --target testpypi                # upload to TestPyPI
python tools/publish.py --bump minor --target pypi --confirm  # version bump + PyPI
```

CI workflow `.github/workflows/publish-pypi.yml` handles tag-triggered trusted publishing (no tokens needed).

---

12. FusionKernel.phase_sync_step() — Single Step

```
kernel = FusionKernel("tokamak_config.json")

# Config-driven defaults from cfg["phase_sync"]
out = kernel.phase_sync_step(
    theta=oscillator_phases,
    omega=natural_frequencies,
    dt=1e-3,
    psi_driver=0.0,          # exogenous Ψ
)

# Returns: theta1, dtheta, R, Psi_r, Psi
```

All parameters fall through to `cfg["phase_sync"]` when not explicitly provided. The `actuation_gain` parameter scales both K and uniformly.

---

13. Test Coverage

61 phase-specific tests + 3 Rust parity (675 total in suite, 14 CI jobs, all green):

Class	Tests	What is verified
`TestOrderParameter`	4	R=1 sync, R 0 uniform, R [0,1], weighted
`TestWrapPhase`	2	Identity in range, large angle wrapping
`TestGlobalPsiDriver`	3	External requires value, returns value, mean-field
`TestKuramotoSakaguchiStep`	4	Sync stability, R increase, pull, frustration
`TestKnmSpec`	7	Shape, anchors, boosts, symmetry, zeta, validation
`TestUPDESystem`	6	Step shape, intra-sync, pull, trajectory, PAC, error
`TestFusionKernelPhaseSync`	3	Integration smoke, config-driven , Lyapunov multi-step
`TestLyapunovV`	4	V=0 sync, V=2 anti-sync, empty, range
`TestLyapunovExponent`	3	<0 decreasing, >0 increasing, single sample
`TestUPDELyapunov`	3	step V output, run_lyapunov , PAC effect
`TestLyapunovGuard`	5	Stable approved, unstable refused, batch, DIRECTOR_AI dict, reset
`TestRealtimeMonitor`	6	from_paper27 defaults, tick snapshot, multi-tick, convergence, reset, DIRECTOR_AI export
`TestMockDIIID`	4	Shot generation, shapes, save/reload, safe shot
`TestE2EPhaseSyncWithShot`	2	Shot-driven monitor, disruption guard
`TestTrajectoryExport`	4	NPZ export, HDF5 export, recorder clear, record=False
`TestWebSocketServer`	1	Server construction

11 Rust tests (inline, all passing):

Test	What is verified
`test_order_parameter_synced`	R=1 for identical phases
`test_order_parameter_range`	R [0,1]
`test_wrap_phase_identity`	No-op in range
`test_wrap_phase_large`	7 wraps to (− ,]
`test_step_preserves_count`	Output length matches input
`test_zeta_pulls_toward_psi`	500 steps, spread < 0.1
`test_run_returns_trajectory`	Correct trajectory length
`test_lyapunov_v_synced_is_zero`	V=0 at perfect sync
`test_lyapunov_exponent_negative_with_zeta`	<0 with =3 driver
`test_upde_tick_shape`	Multi-layer tick output dimensions
`test_upde_tick_zeta_convergence`	4-layer =3 convergence to Ψ

Full suite regression: 582 passed, 94 skipped, 0 failures. Total collected: **675 tests** across 41 test files.

---

14. Demo Notebook Sections

`examples/paper27_phase_dynamics_demo.ipynb` (10 sections + summary):

1. **Knm Heatmap** — 16×16 coupling matrix visualisation
2. **Comparison** — with/without global driver, R convergence
3. **Frustration** — Sakaguchi phase-lag effect on synchronisation
4. **16-Layer UPDE** — full multi-layer evolution with R trajectories
5. **PAC Gating** — phase-amplitude coupling modulation demo
6. **FusionKernel Plasma Sync** — integration with tokamak config

7. **Gain Sweep** — actuation_gain parameter exploration
8. **Lyapunov Stability** — V(t) = (1/N)Σ(1−cos(_i−Ψ)) monotone decrease
9. **SNN Closed-Loop** — spike-rate → Ψ feedback via LIF layer
10. **PAC Cross-Layer SNN** — multi-layer SNN with phase-coupled spike routing

Markdown export: `docs/paper27_phase_dynamics.md`

---

15. Commit History

```
4af1c5f fix: silence clippy too_many_arguments / type_complexity on Kuramoto bindings
7453019 style: cargo fmt on kuramoto bindings
ad09c0e feat: Rust Kuramoto kernel, PAC cross-layer SNN, docs export
b11228b docs: add Paper 27 phase dynamics demo notebook
81704be feat: add Paper 27 Knm/UPDE engine +  sin(Ψ- ) global driver
```

---

16. What Was NOT Touched

- GS equilibrium solver (`solve_equilibrium`, `gs_step`, `SOR/multigrid`)
- SNN controllers (`LIFNeuron`, `SNNController`, spike-rate feedback')
- Chebyshev/IGA spectral methods
- Rust control-math crates (SOR, tridiag, FFT, etc.) — only added `kuramoto` module
- All existing tests remain green

---

17. Paper 27 Reference

M. Šotek, "The Knm Matrix: A Simulation Framework for Modelling Multi-Scale Bidirectional Causality in the Self-Consistent Phenomenological Network," SCPN Paper 27, 2026. Available: academia.edu | ORCID 0009-0009-3560-0851