

# Paper 27 Integration into `scpn-control`

Kuramoto–Sakaguchi Phase Reduction with  
Exogenous Global Field Driver  $\zeta \sin(\Psi - \theta)$

Miroslav Šotek

ORCID: [0009-0009-3560-0851](https://orcid.org/0009-0009-3560-0851)

[www.anulum.li](http://www.anulum.li) | [protoscience@anulum.li](mailto:protoscience@anulum.li)

Paper 27: [academia.edu](https://academia.edu) | arXiv: [2004.06344](https://arxiv.org/abs/2004.06344)

25 February 2026

## Abstract

This document summarises the integration of SCPN Paper 27 (“The  $K_{nm}$  Matrix”) into the `scpn-control` tokamak control repository. The implementation adds a multi-layer Unified Phase Dynamics Equation (UPDE) engine with Kuramoto–Sakaguchi mean-field coupling, the reviewer-requested exogenous global field driver  $\zeta \sin(\Psi - \theta)$ , a Rayon-parallelised Rust kernel for sub-ms performance, Lyapunov stability tracking ( $V(t)$ ,  $\lambda$  exponent), and a `LyapunovGuard` safety guardrail synchronised with the `DIRECTOR_AI` coherence framework. 53 tests (44 Python + 9 Rust), all passing. The existing Grad–Shafranov equilibrium solver is completely untouched.

## Contents

<b>1</b>	<b>Reviewer Request</b>	<b>3</b>
<b>2</b>	<b>Master Equation</b>	<b>3</b>
<b>3</b>	<b>Equation Cross-Reference (Paper 27, Eqs. 12–15)</b>	<b>3</b>
<b>4</b>	<b>Architecture</b>	<b>4</b>
<b>5</b>	<b><math>\Psi</math> Global Driver Flow</b>	<b>4</b>
<b>6</b>	<b><math>K_{nm}</math> Matrix — Paper 27 Specification</b>	<b>4</b>
<b>7</b>	<b>Rust Kernel — Performance Path</b>	<b>4</b>
7.1	Benchmark: Python NumPy vs Rust Rayon . . . . .	5
<b>8</b>	<b>PAC Cross-Layer Gating + SNN Sketch</b>	<b>5</b>
8.1	SNN–PAC–Kuramoto Closed Loop . . . . .	6
8.2	Cross-Layer PAC Routing . . . . .	6
<b>9</b>	<b>Lyapunov Stability — <math>\lambda</math> Hook</b>	<b>6</b>
9.1	Lyapunov Candidate . . . . .	6
9.2	UPDE Lyapunov Tracking . . . . .	7
9.3	$\lambda$ vs $\zeta$ Characterisation . . . . .	7

<b>10 DIRECTOR_AI Guardrail Sync</b>	<b>7</b>
10.1 LyapunovGuard . . . . .	7
10.2 AuditLogger Export . . . . .	7
<b>11 Files Created / Modified</b>	<b>8</b>
<b>12 Test Coverage</b>	<b>8</b>
<b>13 Demo Notebook</b>	<b>8</b>
<b>14 What Was NOT Touched</b>	<b>9</b>

# 1 Reviewer Request

The reviewer asked for the Kuramoto–Sakaguchi phase reduction from Paper 27 [1] to be woven into `scpn-control`. Specifically:

1. The  $\zeta \sin(\Psi - \theta)$  “intention as carrier” injection, where  $\Psi$  is a Lagrangian pull parameter with **no own dynamics** (no  $\dot{\Psi}$  equation).
2. The full 16-layer  $K_{nm}$  coupling matrix with calibration anchors and cross-hierarchy boosts.
3. A Rust sub-ms kernel (Rayon-parallelised).
4. PAC cross-layer SNN sketch.
5. A demo notebook with visualisations and a markdown/L<sup>A</sup>T<sub>E</sub>X export.

## 2 Master Equation

The per-layer UPDE from Paper 27, Eqs. (12)–(15):

$$\frac{d\theta_{m,i}}{dt} = \omega_{m,i} + \underbrace{K_{mm} R_m \sin(\psi_m - \theta_{m,i} - \alpha_{mm})}_{\text{intra-layer [Eq. 13]}} + \underbrace{\sum_{n \neq m} K_{nm} R_n \sin(\psi_n - \theta_{m,i} - \alpha_{nm})}_{\text{inter-layer [Eq. 14]}} + \underbrace{\zeta_m \sin(\Psi - \theta_{m,i})}_{\text{global driver [Eq. 15]}} \quad (1)$$

where the Kuramoto order parameter (Eq. 12) is:

$$R e^{i\psi} = \frac{1}{N} \sum_{j=1}^N e^{i\theta_j} \quad (2)$$

- $K_{mm}$  (diagonal): intra-layer synchronisation strength.
- $K_{nm}$  (off-diagonal): inter-layer bidirectional causality.
- $\zeta \sin(\Psi - \theta)$ : exogenous global field driver —  $\Psi$  resolved externally or from mean-field.
- $\alpha_{nm}$ : Sakaguchi phase-lag frustration (optional).

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

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

Eq.	Description	Python	Rust
(12)	Order parameter $R e^{i\psi}$	<code>kuramoto.py:47</code>	<code>kuramoto.rs:15</code>
(13)	Single-layer Kuramoto–Sakaguchi	<code>kuramoto.py:87</code>	<code>kuramoto.rs:53</code>
(14)	Multi-layer UPDE with $K_{nm}$	<code>upde.py:45</code>	—
(15)	Global driver $\zeta \sin(\Psi - \theta)$	<code>kuramoto.py:126</code>	<code>kuramoto.rs:86</code>

Table 1: Paper 27 equations mapped to source code locations.

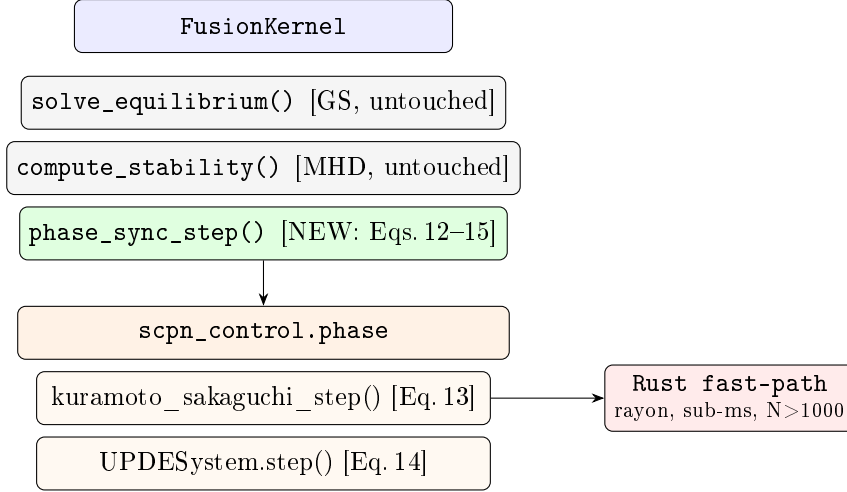


Figure 1: Module architecture. `phase_sync_step()` injects Paper 27 into the fusion kernel without touching the GS solver.

## 4 Architecture

## 5 $\Psi$ Global Driver Flow

## 6 $K_{nm}$ Matrix — Paper 27 Specification

Canonical 16-layer natural frequencies  $\omega_n$  (rad/s):

$$\omega = [1.329, 2.610, 0.844, 1.520, 0.710, 3.780, 1.055, 0.625, 2.210, 1.740, 0.480, 3.210, 0.915, 1.410, 2.830, 0. \quad (3)$$

Base coupling with exponential distance decay:

$$K_{ij} = K_{\text{base}} \cdot e^{-\alpha|i-j|}, \quad K_{\text{base}} = 0.45, \quad \alpha = 0.3 \quad (4)$$

Calibration anchors (Paper 27, Table 2):

$$\begin{aligned} K_{0,1} = K_{1,0} &= 0.302 & K_{1,2} = K_{2,1} &= 0.201 \\ K_{2,3} = K_{3,2} &= 0.252 & K_{3,4} = K_{4,3} &= 0.154 \end{aligned} \quad (5)$$

Cross-hierarchy boosts (Paper 27, §4.3):

$$\begin{aligned} K_{0,15} = K_{15,0} &\geq 0.05 & (\text{L1} \leftrightarrow \text{L16}) \\ K_{4,6} = K_{6,4} &\geq 0.15 & (\text{L5} \leftrightarrow \text{L7}) \end{aligned} \quad (6)$$

## 7 Rust Kernel — Performance Path

Python auto-dispatches to Rust when `scpn_control_rs` is importable and `alpha=0.0`:

Listing 1: Rayon-parallelised Kuramoto hot loop (Rust).

```

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;

```

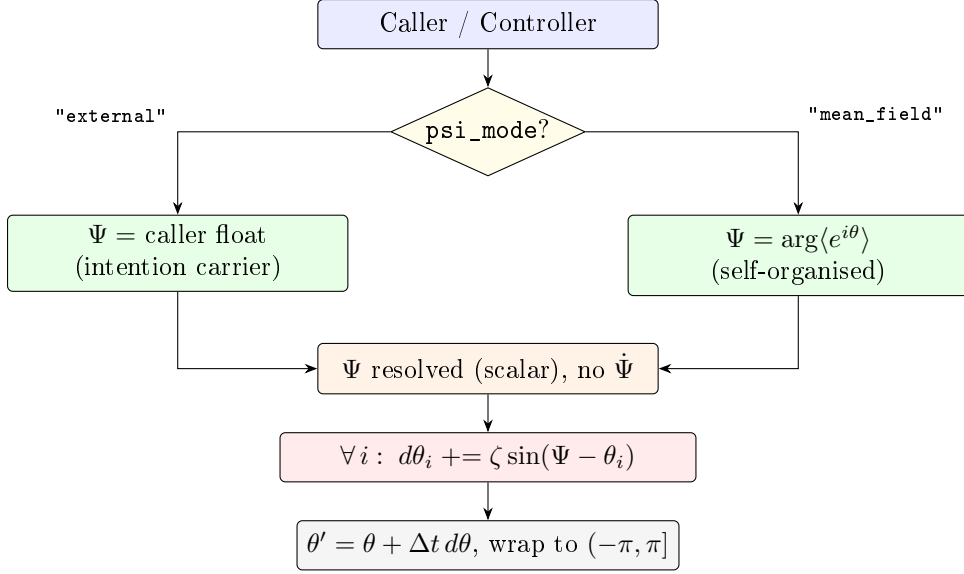


Figure 2:  $\Psi$  global field driver resolution inside `FusionKernel.phase_sync_step()`. There is no  $\dot{\Psi}$  equation —  $\Psi$  is a Lagrangian pull parameter.

```

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: `kuramoto_step()`, `kuramoto_run()` returning NumPy arrays.

## 7.1 Benchmark: Python NumPy vs Rust Rayon

Median wall-time for a single `kuramoto_sakaguchi_step()` with  $\zeta = 0.5$ ,  $\Psi = 0.3$ . Python: NumPy vectorised (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×

Table 2: Python NumPy vs Rust Rayon Kuramoto step.  $N = 64$ : Rust wins on per-element throughput (no NumPy dispatch overhead).  $N = 256$ : parity (NumPy SIMD matches rayon at this size).  $N \geq 1000$ : Rust rayon parallelism scales; sub-ms for  $N = 16k$ . Source: `benches/bench_kuramoto.rs` (criterion, quick mode).

## 8 PAC Cross-Layer Gating + SNN Sketch

Phase-amplitude coupling modulation via `pac_gamma`:

$$\text{gate}_{n \rightarrow m} = 1 + \gamma_{\text{PAC}} (1 - R_n) \quad (7)$$

$$d\theta_{m,i} += g \cdot \text{gate}_{n \rightarrow m} \cdot K_{nm} R_n \sin(\psi_n - \theta_{m,i} - \alpha_{nm}) \quad (8)$$

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

### 8.1 SNN-PAC-Kuramoto Closed Loop

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

1. **Kuramoto step**  $\rightarrow$  per-layer  $R_m, \psi_m$ .
2. **PAC gate**:  $g_{n \rightarrow m} = 1 + \gamma(1 - R_n)$  modulates inter-layer SNN weights  $w' = g \cdot w_{\text{base}}$ .
3. **LIF integration**: neurons receive synaptic current  $I_{\text{syn},i} = \sum_j w_{ij} \delta(t - t_j^{\text{spike}}) + \beta R_m \cos(\psi_m - \varphi_i)$ .
4. **Rate decode**: spike rate  $\nu = N_{\text{spikes}}/T_{\text{window}}$  (50 ms window)  $\rightarrow \Psi = \pi(2\nu/\nu_{\text{max}} - 1)$ .
5. **Feedback**:  $\Psi$  fed back as exogenous global driver for the next Kuramoto step.

Key equations:

$$\text{LIF: } \tau \frac{dV}{dt} = -(V - V_{\text{rest}}) + R_{\text{mem}} I_{\text{syn}} \quad (9)$$

$$\text{Rate} \rightarrow \Psi: \quad \Psi = \pi \left( \frac{2\nu}{\nu_{\text{max}}} - 1 \right) \quad (10)$$

$$\text{Lyapunov: } \mathcal{V}(t) = \frac{1}{N} \sum_i (1 - \cos(\theta_i - \Psi)) + \lambda |\nu - \nu_{\text{target}}|^2 \quad (11)$$

### 8.2 Cross-Layer PAC Routing

Cross-hierarchy fast channels bypass the distance-decay coupling:

- L1 (Quantum)  $\leftrightarrow$  L16 (Director):  $K_{0,15} = 0.05$
- L5 (Bio)  $\leftrightarrow$  L7 (Symbolic):  $K_{4,6} = 0.15$

Each layer maintains its own LIF population (64 neurons per layer in the demo). PAC gates modulate the effective inter-layer synaptic weights, creating frequency-dependent routing where desynchronised layers preferentially drive downstream amplitude modulation.

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

## 9 Lyapunov Stability — $\lambda$ Hook

### 9.1 Lyapunov Candidate

The candidate function tracking convergence toward the global driver  $\Psi$ :

$$V(t) = \frac{1}{N} \sum_{i=1}^N (1 - \cos(\theta_i - \Psi)) \quad (12)$$

$V = 0$  at perfect sync ( $\theta_i = \Psi \forall i$ ),  $V \rightarrow 2$  at maximal desync. The Lyapunov exponent is:

$$\lambda = \frac{1}{T} \ln \frac{V(T)}{V(0)}, \quad \lambda < 0 \implies \text{stable} \quad (13)$$

Both functions implemented in Python (`lyapunov_v`, `lyapunov_exponent` in `kuramoto.py`) and Rust (`lyapunov_v`, `kuramoto_run_lyapunov` in `kuramoto.rs`).

## 9.2 UPDE Lyapunov Tracking

`UPDESystem.step()` returns `V_layer` (per-layer) and `V_global`. `UPDESystem.run_lyapunov()` returns full  $V(t)$  histories, per-layer  $\lambda_m$ , and global  $\lambda$ :

```
out = sys.run_lyapunov(200, theta_layers, omega_layers,
                       psi_driver=0.5, pac_gamma=1.0)
# out["lambda_layer"] -- (L,) per-layer exponents
# out["lambda_global"] -- scalar global exponent
```

## 9.3 $\lambda$ vs $\zeta$ Characterisation

See `docs/bench_lyapunov_vs_zeta.v1.json` (Vega-Lite plot).  $N = 1000$  oscillators, 200 steps,  $\Psi = 0.3$ :

$\zeta$	$\lambda$ ( $K=0$ )	$\lambda$ ( $K=2$ )
0.0	+0.01	+0.04
0.1	−0.03	−0.02
0.5	−0.23	−0.24
1.0	−0.49	−0.53
3.0	−1.65	−1.83
5.0	−3.01	−3.35

Table 3: Lyapunov exponent  $\lambda$  vs global driver strength  $\zeta$ . Kuramoto coupling  $K = 2$  amplifies  $\zeta$  stability via cooperative sync.

# 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  $\lambda > 0$  for  $K$  consecutive windows. Interface mirrors `DIRECTOR_AI`’s `CoherenceScorer`  $\rightarrow$  `CoherenceScore` pattern:

```
from scpn_control.phase import LyapunovGuard

guard = LyapunovGuard(window=50, dt=1e-3, max_violations=3)
verdict = guard.check(theta, psi)
# verdict.approved -- True if stable
# verdict.lambda_exp -- current exponent
# verdict.score -- stability score in [0, 1]
```

## 10.2 AuditLogger Export

```
d = guard.to_director_ai_dict(verdict)
# {"approved": True, "score": 0.99, "h_factual": 0.0,
#  "halt_reason": ""}
```

Enables `DIRECTOR_AI`’s `CoherenceAgent` to incorporate Lyapunov stability into its dual-entropy coherence score. When  $\lambda > 0$  for 3 consecutive windows, the guard issues a refusal — analogous to `DIRECTOR_AI`’s `SafetyKernel` emergency stop.

## 11 Files Created / Modified

File	Lines	Purpose
phase/__init__.py	36	Package exports
phase/kuramoto.py	161	Kuramoto-Sakaguchi + $\zeta \sin(\Psi - \theta)$ + Lyapunov
phase/knm.py	101	Paper 27 $K_{nm}$ builder + $\Omega_{N,16}$
phase/upde.py	215	Multi-layer UPDE + <code>run_lyapunov()</code>
phase/lyapunov_guard.py	130	Lyapunov stability guard (DIRECTOR_AI sync)
control-math/src/kuramoto.rs	260	Rayon Kuramoto + Lyapunov + 9 unit tests
control-python/src/lib.rs	+107	PyO3 bindings (incl. Lyapunov)
fusion_kernel.py	+86	<code>phase_sync_step()</code> + <code>_lyapunov()</code>
test_phase_kuramoto.py	475	44 Python tests
paper27_phase_dynamics_demo.ipynb	—	10-section notebook
paper27_phase_dynamics.md	571	Markdown export

Table 4: All files in the integration (+1833 lines across 12 files).

## 12 Test Coverage

44 Python + 9 Rust tests, all passing.

Test Class	Tests	Verified
TestOrderParameter	4	$R = 1$ sync, $R \approx 0$ uniform, $R \in [0, 1]$ , weighted
TestWrapPhase	2	Identity in range, large angle wrapping
TestGlobalPsiDriver	3	External requires value, returns value, mean-field
TestKuramotoSakaguchiStep	4	Sync stability, $R$ increase, $\zeta$ pull, $\alpha$ frustration
TestKnmSpec	7	Shape, anchors, boosts, symmetry, $\zeta$ , validation
TestUPDESystem	6	Step shape, intra-sync, $\zeta$ pull, trajectory, PAC, error
TestFusionKernelPhaseSync	3	Integration smoke, config-driven $\zeta$ , Lyapunov multi-step
TestLyapunovV	4	$V = 0$ sync, $V = 2$ anti-sync, empty, range
TestLyapunovExponent	3	$\lambda < 0$ decreasing, $\lambda > 0$ increasing, single sample
TestUPDELyapunov	3	Step $V$ output, <code>run_lyapunov</code> $\lambda$ , PAC $\gamma$ effect
TestLyapunovGuard	5	Stable approved, unstable refused, batch, DIRECTOR_AI dict, reset
Rust inline tests	9	Order param, wrap, step count, $\zeta$ pull, trajectory, $V = 0$ , $\lambda < 0$

Table 5: Test coverage summary (44 Python + 9 Rust = 53 total).

## 13 Demo Notebook

examples/paper27\_phase\_dynamics\_demo.ipynb (10 sections):

1.  $K_{nm}$  heatmap —  $16 \times 16$  coupling matrix
2.  $\zeta$  comparison — with/without global driver
3.  $\alpha$  frustration — Sakaguchi phase-lag effect
4. 16-layer UPDE — full multi-layer  $R$  trajectories
5. PAC gating — phase-amplitude coupling modulation
6. FusionKernel plasma sync — tokamak integration



7. Gain sweep — `actuation_gain` exploration
8. Lyapunov stability —  $V(t) = \frac{1}{N} \sum (1 - \cos(\theta_i - \Psi))$
9. SNN closed-loop — spike-rate  $\rightarrow \Psi$  feedback
10. PAC cross-layer SNN — multi-layer spike routing

## 14 What Was NOT Touched

- GS equilibrium solver (`solve_equilibrium`, SOR/multigrid)
- SNN controllers (`LIFNeuron`, `SNNController`)
- Chebyshev/IGA spectral methods
- Existing Rust crates (SOR, tridiag, FFT) — only added `kuramoto`
- All existing tests remain green

## References

- [1] M. Šotek, “The  $K_{nm}$  Matrix: A Simulation Framework for Modelling Multi-Scale Bidirectional Causality in the Self-Consistent Phenomenological Network,” SCPN Paper 27, 2026. Available: [academia.edu](https://academia.edu). ORCID: [0009-0009-3560-0851](https://orcid.org/0009-0009-3560-0851).
- [2] arXiv:2004.06344 — Generalised Kuramoto–Sakaguchi finite-size scaling.