run_grc_protocol.py

```
#!/usr/bin/env python3
import argparse, json, math, os, random, re, subprocess, sys
from pathlib import Path

# ---------- tiny helpers ----------
def load_config(p): return json.loads(Path(p).read_text())
def outdir(cfg):
   d = Path(cfg.get("output_dir","grc_outputs")); d.mkdir(parents=True,
exist_ok=True); return d
def run(cmd): return subprocess.run(cmd, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT, text=True).stdout

METRIC_PATTERNS = {
   "rel_min": [r"rel[_ ]?min[:=]\s*([0-9.eE+-]+)", r"minNorm[:=]\s*([0-9.eE+-]+)"],
   "convergence_rate": [r"convergence[_ ]?rate[:=]\s*([0-9.eE+-]+)",
r"conv(?:er)?gence[:=]\s*([0-9.eE+-]+)"],
   "rho_T": [r"rho\(T\)[:=]\s*([0-9.eE+-]+)", r"spectral[_ ]?radius.*?([0-9.eE+-]+)"],
   "lambda_min": [r"lambda[_ ]?min.*?([0-9.eE+-]+)", r"λ[_ ]?min.*?([0-9.eE+-]+)"],
   "lambda_max": [r"lambda[_ ]?max.*?([0-9.eE+-]+)", r"λ[_ ]?max.*?([0-9.eE+-]+)"],
   "cond": [r"cond(?:ition)?(?:ing)?(?:\(G\))?[:=]\s*([0-9.eE+-]+)"]
}
def parse_metrics(txt):
   def fkey(pats):
       for pat in pats:
           m=re.search(pat, txt, re.I)
           if m:
               try: return float(m.group(1))
               except: pass
       return None
   out = {k:fkey(v) for k,v in METRIC_PATTERNS.items()}
   if re.search(r"\bverdict\b.*\bPASS\b", txt, re.I): out["verdict"]="PASS"
   if re.search(r"\bverdict\b.*\bFAIL\b", txt, re.I): out["verdict"]="FAIL"
   return out

def su_pair(N): return (N+2, N+3)  # unused now; suite runs on {su3,su4}
def spearman(xs, ys):
   ranks = lambda a: {v:i+1 for i,v in enumerate(sorted(set(a)))}
   rx, ry = ranks(xs), ranks(ys)
   n=len(xs);
   return None if n<3 else 1 - 6*sum((rx[x]-ry[y])**2 for x,y in
zip(xs,ys))/(n*(n*n-1))
```

```
# ---------- gates ----------
def gate_offset(cfg, od):
   cmd = cfg["scanner_cmd"]; obj = cfg["objective"]
   a = cmd +
["--group","su4","--harmonics","5,6","--objective",obj,"--delta","0.0","--check-dynami
cs"]
   b = cmd +
["--group","su4","--harmonics","5,6","--objective",obj,"--delta",str(cfg.get("delta_of
fset",0.44879895)),"--check-dynamics"]
   A, B = run(a), run(b)
   mA, mB = parse_metrics(A), parse_metrics(B)
   res =
{"group":"su4","harmonics":[5,6],"delta0":0.0,"delta1":cfg.get("delta_offset",0.448798
95),
         "run0":{"stdout":A,"metrics":mA},"run1":{"stdout":B,"metrics":mB},
         "verdict_unchanged": (mA.get("verdict")==mB.get("verdict"))}
   (od/"offset_result.json").write_text(json.dumps(res, indent=2))
   print("Gate 1 (offset) →", od/"offset_result.json")

def gate_delta_scan(cfg, od):  # replaces gamma-scan
   cmd, obj, ds = cfg["scanner_cmd"], cfg["objective"], cfg["delta_values"]
   rows=[]
   for d in ds:
       r = run(cmd +
["--group","su3","--harmonics","5,6","--objective",obj,"--delta",str(d),"--check-dynam
ics"])
       rows.append({"delta":d,"metrics":parse_metrics(r),"stdout":r})
   verdicts = sorted({r["metrics"].get("verdict") for r in rows})
   j =
{"group":"su3","harmonics":[5,6],"rows":rows,"verdicts":verdicts,"invariant":(len(verd
icts)==1)}
   (od/"gamma_scan_su3_5_6.json").write_text(json.dumps(j, indent=2))  # keep filename
for paper
   print("Gate 2 (δ-scan for invariance) →", od/"gamma_scan_su3_5_6.json")

def gate_null(cfg, od):
   cmd, obj = cfg["scanner_cmd"], cfg["objective"]
   scr = int(cfg["null"]["scrambles_per_pair"]); su = cfg["null"]["su_for_null"]
   rank_pairs = [tuple(p) for p in cfg["null"]["rank_pairs"]]
   cox_pairs  = [tuple(p) for p in cfg["null"]["coxeter_pairs"]]

   fam = {"rank":[], "coxeter":[]}
```

```
   for (m,M) in rank_pairs:
       for _ in range(scr):
           d=random.choice(cfg["delta_values"])

r=run(cmd+["--group",su,"--harmonics",f"{m},{M}","--objective",obj,"--delta",str(d),"-
-check-dynamics"])
           met=parse_metrics(r);
           if met.get("rel_min") and met.get("convergence_rate"):
               met.update({"pair":[m,M],"delta":d}); fam["rank"].append(met)
   for (m,M) in cox_pairs:
       for _ in range(scr):
           d=random.choice(cfg["delta_values"])

r=run(cmd+["--group",su,"--harmonics",f"{m},{M}","--objective",obj,"--delta",str(d),"-
-check-dynamics"])
           met=parse_metrics(r);
           if met.get("rel_min") and met.get("convergence_rate"):
               met.update({"pair":[m,M],"delta":d}); fam["coxeter"].append(met)

   alpha = float(cfg.get("alpha",0.01))
   union = fam["rank"] + fam["coxeter"]
   rel = [u["rel_min"] for u in union]
   dyn = [u["convergence_rate"] for u in union]

   def qtail(arr, q, lower=True):
       arr = sorted(arr)
       if not arr: return None
       return arr[max(0,int(math.floor(q*len(arr)))-1)] if lower else
arr[min(len(arr)-1,int(math.ceil((1-q)*len(arr))))]

   rel_thr = qtail(rel, alpha, lower=True)
   dyn_thr = qtail(dyn, alpha, lower=False)

   lrel=[math.log(x) for x in rel] if rel else []
   ldyn=[math.log(x) for x in dyn] if dyn else []
   mean = (lambda a: sum(a)/len(a)) if lrel or ldyn else None
   def stats(a):
       if not a: return {"mu":None,"sigma":None,"n":0}
       mu=sum(a)/len(a); var=sum((x-mu)**2 for x in a)/len(a) if len(a)>1 else None
       return {"mu":mu,"sigma":(var**0.5 if var is not None else None),"n":len(a)}

   out = {"source":"union-null(rank+coxeter)","alpha":alpha,
```

```
          "thresholds":{"rel_min":rel_thr,"convergence_rate":dyn_thr},
          "summary":{"rel_min":stats(lrel),"convergence_rate":stats(ldyn)},

"counts":{"rank":len(fam["rank"]),"coxeter":len(fam["coxeter"]),"union":len(union)}}
   (od/"null_thresholds.json").write_text(json.dumps(out, indent=2))
   print("Gate 3 (null thresholds) →", od/"null_thresholds.json")

def gate_su_suite(cfg, od):
   j = json.loads((od/"null_thresholds.json").read_text())
   t_rel, t_dyn = j["thresholds"]["rel_min"], j["thresholds"]["convergence_rate"]
   mu_rel, sig_rel = j["summary"]["rel_min"]["mu"], j["summary"]["rel_min"]["sigma"]
   mu_dyn, sig_dyn = j["summary"]["convergence_rate"]["mu"],
j["summary"]["convergence_rate"]["sigma"]

   cmd, obj = cfg["scanner_cmd"], cfg["objective"]
   rows=[]; labels=[]
   for G in cfg["su_groups_for_suite"]:
       # use canonical pairs for the paper: su3→(5,6), su4→(6,7)
       pair = (5,6) if G=="su3" else (6,7)

r=run(cmd+["--group",G,"--harmonics",f"{pair[0]},{pair[1]}","--objective",obj,"--delta
","0.0","--check-dynamics"])
       met=parse_metrics(r)
       rel, dyn = met.get("rel_min"), met.get("convergence_rate")
       z_static = ( (mu_rel - math.log(rel))/sig_rel ) if (rel and mu_rel and sig_rel)
else None
       z_dynamic= ( (mu_dyn - math.log(dyn))/sig_dyn ) if (dyn and mu_dyn and sig_dyn)
else None
       access = (z_static - z_dynamic) if (z_static is not None and z_dynamic is not
None) else None
       verdict = None
       if all(v is not None for v in [rel,dyn,t_rel,t_dyn]): verdict = "PASS" if
(rel<=t_rel and dyn<=t_dyn) else "FAIL"

rows.append({"group":G,"harmonics":list(pair),"rel_min":rel,"convergence_rate":dyn,

"z_static":z_static,"z_dynamic":z_dynamic,"accessibility":access,"verdict":verdict,"st
dout":r})
       if z_dynamic is not None: labels.append((G,z_dynamic))

   # With only two groups, monotonicity over N isn't defined → set None.
   outj = {"thresholds":{"rel_min":t_rel,"convergence_rate":t_dyn},
```

```
          "rows": rows, "spearman_dynamic_z_vs_N": None}
   (od/"su_suite_results.json").write_text(json.dumps(outj, indent=2))
   print("Gate 4 (SU3/SU4 suite) →", od/"su_suite_results.json")

def gate_blinded_min(cfg, od):
   thr = json.loads((od/"null_thresholds.json").read_text())
   t_rel, t_dyn = thr["thresholds"]["rel_min"], thr["thresholds"]["convergence_rate"]
   cmd, obj = cfg["scanner_cmd"], cfg["objective"]
   labels=["A","B"]; groups=["su3","su4"]; random.seed(42); random.shuffle(labels)
   mask = dict(zip(labels, groups))
   runs=[]
   for L in labels:
       G = mask[L]; pair = (5,6) if G=="su3" else (6,7)

r=run(cmd+["--group",G,"--harmonics",f"{pair[0]},{pair[1]}","--objective",obj,"--delta
","0.0","--check-dynamics"])
       met=parse_metrics(r); rel, dyn = met.get("rel_min"),
met.get("convergence_rate")
       verdict = None
       if all(v is not None for v in [rel,dyn,t_rel,t_dyn]): verdict = "PASS" if
(rel<=t_rel and dyn<=t_dyn) else "FAIL"
       runs.append({"label":L,"metrics":met,"verdict":verdict,"stdout":r})
   (od/"blinded_min.json").write_text(json.dumps({"masked_runs":runs,"unmask":mask},
indent=2))
   print("Gate 5 (blinded SU3↔SU4) →", od/"blinded_min.json")

# ---------- main ----------
if __name__=="__main__":
   ap=argparse.ArgumentParser()
   ap.add_argument("--config", required=True)
   args=ap.parse_args()
   cfg=load_config(args.config); od=outdir(cfg)

   gate_offset(cfg, od)
   gate_delta_scan(cfg, od)
   gate_null(cfg, od)
   gate_su_suite(cfg, od)
   gate_blinded_min(cfg, od)

   print("\nAll minimal gates complete. See:", od, "\n")
```

phi_gram_ref.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
phi_gram_ref.py — Reference computation for twisted Gram certificates
(fused-map + √Jacobian compensation + spectral radius/PSD diagnostics
+ P_2N stability
+ optional B-doubling + dynamic stability metrics)

Usage examples:
 python phi_gram_ref.py
 python phi_gram_ref.py --group su3 --grid 28 --B 4096 --refine 2 --refine-grid 12
 python phi_gram_ref.py --group su4 --tau 0.03 --r 0.1 --delta 0.0
 python phi_gram_ref.py --no-check-pn --check-b-doubling
 python phi_gram_ref.py --cesaro-start-T    # exclude identity term from Cesàro sum
 python phi_gram_ref.py --check-dynamics --N-dynamics 256
 python phi_gram_ref.py --harmonics 4,5     # NEW: pick adjacent harmonic modes

Outputs:
 - phi_gram_summary.json     (summary per group)
 - scan_<group>.json         (coarse + refined scan points with metrics)

Model sketch:
 Boundary Möbius with a = r·e^{i·delta}, composed with an in-plane twist τ in W =
span{b_m,b_{m+1}}.
 Pullback on L²(dθ) uses the fused angle map θ_src_total and multiplies by
√(dθ_src/dθ) for unitarity.
 Cesàro projector: P_N = (1/N) Σ_{k=0}^{N-1} T^k (or, with --cesaro-start-T, from k=1
to N).

Diagnostics:
 - Spectral radius ρ(T): if ρ(T) > 1 + 1e-6, Cesàro may not converge.
 - PSD: λ_min(G), λ_max(G), cond(G), violation flag.
 - Cesàro stability: ΔP₂N via ‖P_{2N} - P_N‖₂ and max|·|.
 - Optional B-doubling: re-evaluate best point at 2B and report deltas.
 - Dynamics (optional): convergence_rate, max_drift_angle, coherence_ratio from the
evolution of
   P_k = (1/k) Σ_{i=1}^k T^i.
"""
import argparse
import json
from functools import lru_cache
from typing import Optional
```

```
import numpy as np
import sys

# -------- pretty printing (icons + ANSI colors) --------
class _Style:
   CODES = {
       'reset':'\033[0m','bold':'\033[1m','dim':'\033[2m','italic':'\033[3m',
       'red':'\033[31m','green':'\033[32m','yellow':'\033[33m','blue':'\033[34m',
       'magenta':'\033[35m','cyan':'\033[36m','white':'\033[37m'
   }
   def __init__(self, enable=True):
       self.enable = enable
   def color(self, text, *names):
       if not self.enable or not names:
           return str(text)
       seq = ''.join(self.CODES.get(n,'') for n in names)
       return f"{seq}{text}{self.CODES['reset']}"

def _tick(ok):
   return '✅' if ok else '❌'
def _warn():
   return '⚠️'

def _coherence_bar(rel_min: float, style, width: int = 20) -> str:
   """Render a left-filled bar where SMALLER rel_min fills MORE cells (more coherent).
   rel_min=0 → full bar; rel_min≥0.3 → empty bar. Threshold ticks at 0.03 and 0.15.
   """
   x = max(0.0, min(1.0, 1.0 - (rel_min / 0.3)))
   filled = int(round(x * width))
   empty = width - filled
   bar = '▓' * filled + '░' * empty
   color = 'green' if rel_min <= 0.03 else ('yellow' if rel_min <= 0.15 else 'red')
   return style.color(f"[{bar}]", color)

# ---------------- tolerances ----------------
EPS_RHO = 1e-6     # tolerance for spectral radius > 1
EPS_PSD = 1e-12    # tolerance for PSD eigenvalue negativity
EPS_H   = 1e-12    # stability threshold for c in h_min = -b/c

# --------------- periodic interpolation ---------------
def periodic_interp(values: np.ndarray, theta_src: np.ndarray) -> np.ndarray:
   """
```

```
    Linear interpolation for samples on the circle.
    values[j] corresponds to θ_j = 2π j / B, j=0..B-1. theta_src is arbitrary angles.
    """
    B = values.shape[0]
    two_pi = 2.0 * np.pi
    x = np.mod(theta_src, two_pi) * (B / two_pi)
    i0 = np.floor(x).astype(int)
    frac = x - i0
    i0 = i0 % B
    i1 = (i0 + 1) % B
    return (1.0 - frac) * values[i0] + frac * values[i1]

# ---------------- cached theta grid ----------------
@lru_cache(maxsize=None)
def _theta(B: int) -> np.ndarray:
    """Cached uniform grid on [0, 2π)."""
    return np.linspace(0.0, 2.0*np.pi, B, endpoint=False)

# ---------------- generalized boundary basis ----------------
def basis_b(B: int, mode: int) -> np.ndarray:
    """Generates a complex exponential basis vector for a given harmonic mode."""
    theta = _theta(B)
    return np.exp(1j * float(mode) * theta)

# ---------------- primitive map ----------------
def mobius_inverse_theta(theta_out: np.ndarray, a: complex, gamma: float) ->
np.ndarray:
    """
    Inverse angle map for the Möbius transform z ↦ e^{iγ} (z - a) / (1 - \bar{a} z).
    Given θ_out, returns θ_in such that z_in maps to z_out on the unit circle.
    """
    e_minus_i_gamma = np.exp(-1j * gamma)
    z_out = np.exp(1j * theta_out)
    w = e_minus_i_gamma * z_out
    z_in = (w + a) / (1.0 + np.conj(a) * w)
    return np.angle(z_in)

# ---------------- projection to W ----------------
def project_to_W(g: np.ndarray, b1: np.ndarray, b2: np.ndarray) -> np.ndarray:
    B = g.shape[0]
    norm = 1.0 / B
    c1 = norm * np.vdot(b1, g)
```

```
    c2 = norm * np.vdot(b2, g)
    return np.array([c1, c2], dtype=np.complex128)

# ---------------- build T^τ(α,β) with fused map + √Jacobian ----------------
def build_T_matrix(alpha: float, beta: float, tau: float, delta: float, r: float,
                   B: int, gamma: float,
                   b1: np.ndarray, b2: np.ndarray,
                   ablate_jacobian: bool = False) -> np.ndarray:
    """
    Construct the 2×2 matrix T in basis {b1,b2} for one cycle:
        rotate(α) → invert → rotate(β) → circumscribe(Möbius(a, γ)) → project to W
    Fused pullback:
        θ_src_total(θ) = -MobiusInverse(θ) + (β - α)
        dθ_src/dθ = (1 - |a|²) / |1 + \bar a · e^{-iγ} e^{iθ}|²
        g(θ) = √(dθ_src/dθ) · f(θ_src_total(θ))
    """
    a = r * np.exp(1j * delta)

    theta = _theta(B)
    # fused source angles
    theta_src_mob = mobius_inverse_theta(theta, a=a, gamma=gamma)
    theta_src_total = -theta_src_mob + (beta - alpha)

    # √Jacobian factor for unitary pullback on L²(dθ)
    z_out = np.exp(1j * theta)
    w = np.exp(-1j * gamma) * z_out
    jac = (1.0 - (r * r)) / (np.abs(1.0 + np.conj(a) * w) ** 2)
    jac = np.maximum(jac, 0.0)
    sqrt_jac = np.sqrt(jac)
    if ablate_jacobian:
        sqrt_jac = np.ones_like(sqrt_jac)

    def apply_cycle_fused(f: np.ndarray) -> np.ndarray:
        return sqrt_jac * periodic_interp(f, theta_src_total)

    g1 = apply_cycle_fused(b1)
    g2 = apply_cycle_fused(b2)
    c1 = project_to_W(g1, b1, b2)
    c2 = project_to_W(g2, b1, b2)
    T = np.column_stack([c1, c2])

    if tau != 0.0:
```

```
        # In-plane rotation by τ in W
        mix = np.array([[np.cos(tau), -np.sin(tau)],
                        [np.sin(tau),  np.cos(tau)]], dtype=np.complex128)
        T = mix @ T
    return T

# ---------------- Cesàro projector ----------------
def cesaro_projector(T: np.ndarray, N: int = 64, start_from_T: bool = False) ->
np.ndarray:
    """
    If start_from_T=False (default): P_N = (1/N) * sum_{k=0}^{N-1} T^k
    If start_from_T=True:          P_N = (1/N) * sum_{k=1}^{N}   T^k   (diagnostic to
avoid the identity term)
    """
    if not start_from_T:
        P = np.eye(2, dtype=np.complex128)
        Tk = np.eye(2, dtype=np.complex128)
        for _ in range(1, N):
            Tk = Tk @ T
            P += Tk
        P /= N
        return P
    else:
        P = np.zeros((2,2), dtype=np.complex128)
        Tk = np.eye(2, dtype=np.complex128)
        for _ in range(N):
            Tk = Tk @ T
            P += Tk
        P /= N
        return P

def projector_with_stability(T: np.ndarray, N: int = 64, start_from_T: bool = False):
    Pn  = cesaro_projector(T, N=N,   start_from_T=start_from_T)
    P2n = cesaro_projector(T, N=2*N, start_from_T=start_from_T)
    D = P2n - Pn
    svals = np.linalg.svd(D, compute_uv=False)
    delta_spec = float(svals[0]) if svals.size else 0.0
    delta_maxabs = float(np.max(np.abs(D)))
    return Pn, {"delta_P_spectral": delta_spec, "delta_P_maxabs": delta_maxabs}

# ---------------- dynamic stability diagnostics ----------------
def analyze_dynamics(T: np.ndarray, N_dynamics: int = 128):
```

```
    N = max(2, int(N_dynamics))
    Pk = np.zeros((2,2), dtype=np.complex128)
    Tk = np.eye(2, dtype=np.complex128)
    deltas = []
    angles = []
    v_prev = None

    for k in range(1, N+1):
        Tk = Tk @ T
        Pk_next = (Pk * (k-1)/k) + (Tk / k)

        if k >= 2:
            D = Pk_next - Pk
            deltas.append(float(np.linalg.norm(D, ord='fro')))

        Hk = 0.5 * (Pk_next + Pk_next.conj().T)
        w, V = np.linalg.eigh(Hk)
        idx = int(np.argmax(w))
        v_k = V[:, idx] / np.linalg.norm(V[:, idx])
        if v_prev is not None:
            dot = abs(np.vdot(v_k, v_prev))
            dot = min(1.0, max(0.0, float(dot.real)))
            angles.append(float(np.arccos(dot)))
        v_prev = v_k

        Pk = Pk_next

    half = len(deltas) // 2
    conv_rate = float(np.mean(deltas[half:])) if deltas else 0.0
    max_drift = float(np.max(angles)) if angles else 0.0

    wN, _ = np.linalg.eigh(0.5 * (Pk + Pk.conj().T))
    lam_min = float(np.min(wN))
    lam_max = float(np.max(wN))
    if lam_min <= 0.0:
        coh_ratio = float('inf')
    else:
        coh_ratio = float(lam_max / lam_min)
    trace = float(lam_min + lam_max)
    floor = max(1e-16 * max(trace, 1.0), 0.0)
    coh_ratio_reg = float(lam_max / max(lam_min, floor)) if lam_max > 0 else
float('nan')
```

```
    return {
        "convergence_rate": conv_rate,
        "max_drift_angle": max_drift,
        "coherence_ratio": coh_ratio,
        "coherence_ratio_reg": coh_ratio_reg,
        "dyn_lambda_min": lam_min,
        "dyn_lambda_max": lam_max,
    }

# ---------------- Gram + diagnostics ----------------
def gram_from_projector(P: np.ndarray):
    e1 = np.array([1.0+0j, 0.0+0j])
    e2 = np.array([0.0+0j, 1.0+0j])
    u = P @ e1
    v = P @ e2
    a = np.vdot(u, u).real
    c = np.vdot(v, v).real

    b = np.vdot(u, v)
    G = np.array([[a, b], [np.conj(b), c]], dtype=np.complex128)
    detG = (a * c - (np.abs(b) ** 2)).real

    if c > 0.0:
        h_min = - b / c
        min_norm = (a - (np.abs(b) ** 2) / c).real
    else:
        h_min = np.nan + 1j*np.nan
        min_norm = np.nan

    lam = np.linalg.eigvalsh(G)
    lam_min = float(lam[0])
    lam_max = float(lam[-1])
    psd_violation = (lam_min < -EPS_PSD)
    cond_G = float(lam_max / lam_min) if lam_min > 0.0 else float('inf')

    diag = {
        "a": float(a), "c": float(c),
        "b_real": float(b.real), "b_imag": float(b.imag),
        "detG": float(detG),
        "h_min_real": float(h_min.real) if np.isfinite(h_min.real) else float('nan'),
        "h_min_imag": float(h_min.imag) if np.isfinite(h_min.imag) else float('nan'),
```

```
        "min_norm": float(min_norm),
        "lambda_min_G": lam_min,
        "lambda_max_G": lam_max,
        "cond_G": cond_G,
        "psd_violation": bool(psd_violation),
        "psd_margin": lam_min,
        "stable_h": bool(c > EPS_H)
    }
    return G, diag

def spectral_radius(T: np.ndarray) -> float:
    vals = np.linalg.eigvals(T)
    return float(np.max(np.abs(vals)))

# ---------------- evaluation helper ----------------
def evaluate_point(alpha: float, beta: float, tau: float, delta_eff: float, r: float,
                   B: int, Nproj: int, check_pn: bool, start_from_T: bool,
                   b1: np.ndarray, b2: np.ndarray,
                   check_dynamics: bool = False, N_dynamics: int = 128,
                   ablate_jacobian: bool = False):
    T = build_T_matrix(alpha=alpha, beta=beta, tau=tau, delta=delta_eff, r=r,
                       B=B, gamma=0.0, b1=b1, b2=b2, ablate_jacobian=ablate_jacobian)
    rho = spectral_radius(T)
    if check_pn:
        P, pn_diag = projector_with_stability(T, N=Nproj, start_from_T=start_from_T)
    else:
        P = cesaro_projector(T, N=Nproj, start_from_T=start_from_T)
        pn_diag = {}
    _, gdiag = gram_from_projector(P)
    trG = gdiag['a'] + gdiag['c']
    rel_min = (gdiag['min_norm'] / max(trG, 1e-30)) if (trG == trG) else float('nan')
    rec = {"alpha": float(alpha), "beta": float(beta), "rho_T": rho, "rel_min":
float(rel_min), **pn_diag, **gdiag}
    if check_dynamics:
        dyn = analyze_dynamics(T, N_dynamics=N_dynamics)
        rec.update(dyn)
    return rec

# ---------------- coarse scan ----------------
def coarse_scan(group: str, tau: float, delta: float, r: float,
                B: int, Nproj: int, grid: int, check_pn: bool, start_from_T: bool,
                m1: int, m2: int,
```

```
                check_dynamics: bool = False, N_dynamics: int = 128,
                objective: str = "relmin", ablate_jacobian: bool = False):
    two_pi = 2.0 * np.pi
    # Apply the historical SU(4) parity offset ONLY for the legacy (5,6) case.
    eff_delta = delta
    if m1 == 5 and m2 == 6 and group.lower() == 'su4':
        eff_delta = delta + np.pi/7.0

    # Precompute bases once for speed
    b1 = basis_b(B, m1)
    b2 = basis_b(B, m2)

    records = []
    best = None

    for ia in range(grid):
        alpha = ia * two_pi / grid
        for ib in range(grid):
            beta = ib * two_pi / grid
            rec = evaluate_point(alpha, beta, tau, eff_delta, r, B, Nproj, check_pn,
start_from_T,
                                 b1=b1, b2=b2,
                                 check_dynamics=check_dynamics, N_dynamics=N_dynamics,
                                 ablate_jacobian=ablate_jacobian)
            records.append(rec)
            # Unified scoring across groups (direction still uses group rule)
            if objective == "relmin":
                score = rec.get("rel_min", float("inf"))
            elif objective == "det":
                score = abs(rec["detG"])
            elif objective == "minnorm":
                score = rec["min_norm"]
            else:
                score = rec.get("rel_min", float("inf"))
            if best is None:
                best = (score, rec)
            else:
                if (group.lower() == 'su3' and score < best[0]) or (group.lower() ==
'su4' and score > best[0]):
                    best = (score, rec)
    return eff_delta, records, best[1]
```

```
# ---------------- local refinement ----------------
def local_refine(group: str, tau: float, eff_delta: float, r: float,
                 B: int, Nproj: int,
                 center_ab, m1: int, m2: int,
                 rounds: int = 1, local_grid: int = 12, radius: float = 0.2,
                 check_pn: bool = True, start_from_T: bool = False,
                 check_dynamics: bool = False, N_dynamics: int = 128,
                 objective: str = "relmin", ablate_jacobian: bool = False):
    two_pi = 2.0 * np.pi
    cx, cy = center_ab["alpha"], center_ab["beta"]
    best = None
    refinements = []

    # Precompute bases once for speed
    b1 = basis_b(B, m1)
    b2 = basis_b(B, m2)

    if local_grid < 2:
        rec = evaluate_point(cx, cy, tau, eff_delta, r, B, Nproj, check_pn,
start_from_T,
                             b1=b1, b2=b2,
                             check_dynamics=check_dynamics, N_dynamics=N_dynamics,
                             ablate_jacobian=ablate_jacobian)
        rec["refined"] = True
        return rec, [rec]

    for _ in range(rounds):
        local_records = []
        for ia in range(local_grid):
            alpha = cx + radius * ((ia / (local_grid-1)) - 0.5) * two_pi
            for ib in range(local_grid):
                beta = cy + radius * ((ib / (local_grid-1)) - 0.5) * two_pi
                rec = evaluate_point(alpha, beta, tau, eff_delta, r, B, Nproj,
check_pn, start_from_T,
                                     b1=b1, b2=b2,
                                     check_dynamics=check_dynamics,
N_dynamics=N_dynamics,
                                     ablate_jacobian=ablate_jacobian)
                rec["refined"] = True
                local_records.append(rec)

                if objective == "relmin":
```

```
                    score = rec.get("rel_min", float("inf"))
                elif objective == "det":
                    score = abs(rec["detG"])
                elif objective == "minnorm":
                    score = rec["min_norm"]
                else:
                    score = rec.get("rel_min", float("inf"))
                if best is None:
                    best = (score, rec)
                else:
                    if (group.lower() == 'su3' and score < best[0]) or (group.lower()
== 'su4' and score > best[0]):
                        best = (score, rec)
        refinements.extend(local_records)
        cx, cy = best[1]["alpha"], best[1]["beta"]
        radius *= 0.5
    return best[1], refinements

# ---------------- B-doubling check ----------------
def b_doubling_check(alpha: float, beta: float, tau: float, delta_eff: float, r:
float,
                     B: int, Nproj: int, check_pn: bool, start_from_T: bool,
                     m1: int, m2: int,
                     check_dynamics: bool = False, N_dynamics: int = 128):
    """
    Recompute the same point at 2B and report key metrics and deltas.
    """
    # Baseline at B
    b1_B = basis_b(B, m1); b2_B = basis_b(B, m2)
    rec_B = evaluate_point(alpha, beta, tau, delta_eff, r, B, Nproj, check_pn,
start_from_T,
                           b1=b1_B, b2=b2_B,
                           check_dynamics=check_dynamics, N_dynamics=N_dynamics)

    # Re-evaluate at 2B
    B2 = 2 * B
    b1_B2 = basis_b(B2, m1); b2_B2 = basis_b(B2, m2)
    rec_B2 = evaluate_point(alpha, beta, tau, delta_eff, r, B2, Nproj, check_pn,
start_from_T,
                            b1=b1_B2, b2=b2_B2,
                            check_dynamics=check_dynamics, N_dynamics=N_dynamics)
```

```
    compare = {
        "B2": B2,
        "detG_B2": rec_B2["detG"],
        "min_norm_B2": rec_B2["min_norm"],
        "lambda_min_G_B2": rec_B2["lambda_min_G"],
        "lambda_max_G_B2": rec_B2["lambda_max_G"],
        "cond_G_B2": rec_B2["cond_G"],
        "rho_T_B2": rec_B2["rho_T"],
        "delta_detG_B2_minus_B": rec_B2["detG"] - rec_B["detG"],
        "delta_min_norm_B2_minus_B": rec_B2["min_norm"] - rec_B["min_norm"],
        "delta_lambda_min_G_B2_minus_B": rec_B2["lambda_min_G"] -
rec_B["lambda_min_G"],
        "delta_cond_G_B2_minus_B": rec_B2["cond_G"] - rec_B["cond_G"],
    }
    if check_pn:
        compare.update({
            "delta_P_spectral_B2": rec_B2.get("delta_P_spectral", float("nan")),
            "delta_P_maxabs_B2": rec_B2.get("delta_P_maxabs", float("nan")),
        })
    if check_dynamics:
        compare.update({
            "convergence_rate_B2": rec_B2.get("convergence_rate", float("nan")),
            "max_drift_angle_B2": rec_B2.get("max_drift_angle", float("nan")),
            "coherence_ratio_B2": rec_B2.get("coherence_ratio", float("nan")),
        })
    return compare

# ---------------- main ----------------
def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--group", type=str, default="both", choices=["su3", "su4",
"both"])
    ap.add_argument("--B", type=int, default=1024, help="boundary samples")
    ap.add_argument("--Nproj", type=int, default=48, help="Cesàro steps")
    ap.add_argument("--grid", type=int, default=20, help="coarse alpha/beta grid")
    ap.add_argument("--tau", type=float, default=0.02, help="twist mixing in W")
    ap.add_argument("--r", type=float, default=0.08, help="Möbius magnitude (|a|)")
    ap.add_argument("--delta", type=float, default=0.0, help="Möbius phase (arg a)
baseline")
    ap.add_argument("--refine", type=int, default=1, help="number of local refinement
rounds")
```

```
    ap.add_argument("--objective", type=str, default="relmin",
choices=["relmin","det","minnorm"],
                    help="scan objective (unified for both groups): relmin = minimize
min‖u+h v‖²/trace, det = minimize |det G|, minnorm = minimize min‖u+h v‖²")
    ap.add_argument("--report-topk", type=int, default=5, help="report top-K points per
group in summary JSON")
    ap.add_argument("--refine-grid", type=int, default=12, help="local grid per round")
    ap.add_argument("--refine-radius", type=float, default=0.25, help="initial local
radius (fraction of 2π)")
    # Stability checks
    ap.add_argument("--check-pn", dest="check_pn", action="store_true", help="enable
ΔP₂N stability metric")
    ap.add_argument("--no-check-pn", dest="check_pn", action="store_false",
help="disable ΔP₂N stability metric")
    ap.set_defaults(check_pn=True)
    ap.add_argument("--check-b-doubling", action="store_true", help="re-evaluate best
point at 2B")
    # Bridge (Sections 5-6): calibrate Δα̂ on SU(3) and predict SU(4)
    ap.add_argument("--bridge", action="store_true", help="compute one-loop bridge
prediction using convergence rates")
    ap.add_argument("--R0_SU3", type=float, default=1.152, help="baseline R0(SU3;
D_eff) from paper")
    ap.add_argument("--R0_SU4", type=float, default=1.200, help="baseline R0(SU4;
D_eff) from paper")
    ap.add_argument("--bridge-agg", type=str, default="best",
choices=["best","median_topk"], help="use best point or median over topK")
    # Cesàro variant: optionally exclude identity term
    ap.add_argument("--cesaro-start-T", action="store_true", help="start Cesàro sum at
T¹ instead of I")
    # Dynamic analysis (optional)
    ap.add_argument("--check-dynamics", action="store_true", help="analyze iterative
convergence of P_k")
    ap.add_argument("--N-dynamics", type=int, default=128, help="iterations for dynamic
analysis (P_k)")
    ap.add_argument("--no-color", action="store_true", help="disable ANSI colors and
emoji icons")
    # Ablations
    ap.add_argument("--ablate-jacobian", action="store_true", help="disable √Jacobian
weight in pullback")
    ap.add_argument("--compare-ablation", action="store_true", help="run baseline and
ablation side-by-side and report deltas")
    # NEW: Harmonic modes
```

```
    ap.add_argument("--harmonics", type=str, default="5,6",
                    help="Comma-separated pair of harmonic modes to test (e.g.,
'4,5').")

    args = ap.parse_args()
    style = _Style(enable=(not args.no_color) and sys.stdout.isatty())

    # Parse and validate harmonic modes
    try:
        m1_str, m2_str = args.harmonics.split(',')
        m1 = int(m1_str)
        m2 = int(m2_str)
        if m2 != m1 + 1:
            raise ValueError("Harmonic modes must be adjacent integers (e.g., '4,5').")
    except Exception as e:
        print(f"Error parsing --harmonics argument: {e}")
        sys.exit(1)

    # Basic validation
    if args.B < 2:
        raise ValueError("B must be ≥ 2")
    if args.Nproj < 1:
        raise ValueError("Nproj must be ≥ 1")
    if args.grid < 1:
        raise ValueError("grid must be ≥ 1")
    if args.refine < 0:
        raise ValueError("refine must be ≥ 0")
    if args.refine_grid < 1:
        print("[warn] refine-grid < 2 will evaluate only the center point per round.")

    groups = ["su3", "su4"] if args.group == "both" else [args.group]
    summary = {}
    for g in groups:
        eff_delta, records, best = coarse_scan(
            group=g, tau=args.tau, delta=args.delta, r=args.r,
            B=args.B, Nproj=args.Nproj, grid=args.grid,
            check_pn=args.check_pn, start_from_T=args.cesaro_start_T,
            m1=m1, m2=m2,
            check_dynamics=args.check_dynamics, N_dynamics=args.N_dynamics,
            objective=args.objective, ablate_jacobian=False
        )
```

```
        rho_flag = " !rho>1!" if best["rho_T"] > 1.0 + EPS_RHO else ""
        psd_flag = " !PSD-viol!" if best["psd_violation"] else ""
        pn_str = ""
        if args.check_pn:
            pn_str = f"  ΔP₂N(‖·‖₂)={best['delta_P_spectral']:.3e}
ΔP₂N(max)={best['delta_P_maxabs']:.3e}"
        print(f"[{g}] coarse best at (α,β)=({best['alpha']:.6f},{best['beta']:.6f})  "
              f"detG={best['detG']:.3e}  minNorm={best['min_norm']:.3e}  "
              f"rho(T)={best['rho_T']:.6f}{rho_flag}
λ_min(G)={best['lambda_min_G']:.3e}{psd_flag}{pn_str}")

        best_refined, ref_records = local_refine(
            group=g, tau=args.tau, eff_delta=eff_delta, r=args.r,
            B=args.B, Nproj=args.Nproj,
            center_ab=best, m1=m1, m2=m2,
            rounds=max(0, args.refine),
            local_grid=args.refine_grid, radius=args.refine_radius,
            check_pn=args.check_pn, start_from_T=args.cesaro_start_T,
            check_dynamics=args.check_dynamics, N_dynamics=args.N_dynamics,
            objective=args.objective, ablate_jacobian=False
        )

        rho_flag = " !rho>1!" if best_refined["rho_T"] > 1.0 + EPS_RHO else ""
        psd_flag = " !PSD-viol!" if best_refined["psd_violation"] else ""
        pn_str = ""
        if args.check_pn:
            pn_str = f"  ΔP₂N(‖·‖₂)={best_refined['delta_P_spectral']:.3e}
ΔP₂N(max)={best_refined['delta_P_maxabs']:.3e}"
        print(f"[{g}] final best at
(α,β)=({best_refined['alpha']:.6f},{best_refined['beta']:.6f})  "
              f"detG={best_refined['detG']:.3e}  minNorm={best_refined['min_norm']:.3e}
"
              f"rho(T)={best_refined['rho_T']:.6f}{rho_flag}
λ_min(G)={best_refined['lambda_min_G']:.3e}{psd_flag}{pn_str}")

        # Optional: run ablation branch
        ablation_out = None
        if args.compare_ablation:
            eff_delta_ab, rec_ab, best_ab = coarse_scan(
                group=g, tau=args.tau, delta=args.delta, r=args.r,
                B=args.B, Nproj=args.Nproj, grid=args.grid,
                check_pn=args.check_pn, start_from_T=args.cesaro_start_T,
```

```
                m1=m1, m2=m2,
                check_dynamics=args.check_dynamics, N_dynamics=args.N_dynamics,
                objective=args.objective, ablate_jacobian=True
            )
            best_refined_ab, ref_records_ab = local_refine(
                group=g, tau=args.tau, eff_delta=eff_delta_ab, r=args.r,
                B=args.B, Nproj=args.Nproj,
                center_ab=best_ab, m1=m1, m2=m2,
                rounds=max(0, args.refine),
                local_grid=args.refine_grid, radius=args.refine_radius,
                check_pn=args.check_pn, start_from_T=args.cesaro_start_T,
                check_dynamics=args.check_dynamics, N_dynamics=args.N_dynamics,
                objective=args.objective, ablate_jacobian=True
            )
            print(style.color(f"[{g}] Ablation (no √Jacobian) best
detG={best_refined_ab['detG']:.3e} minNorm={best_refined_ab['min_norm']:.3e}
rel_min={best_refined_ab.get('rel_min', float('nan')):.3e}", 'yellow'))
            ablation_out = {"best": best_refined_ab, "records": rec_ab, "refined":
ref_records_ab}

        # Optional B-doubling check at the final best
        b2 = None
        if args.check_b_doubling:
            b2 = b_doubling_check(
                alpha=best_refined["alpha"], beta=best_refined["beta"],
                tau=args.tau, delta_eff=eff_delta, r=args.r,
                B=args.B, Nproj=args.Nproj, check_pn=args.check_pn,
start_from_T=args.cesaro_start_T,
                m1=m1, m2=m2,
                check_dynamics=args.check_dynamics, N_dynamics=args.N_dynamics
            )
            print(f"[{g}] B-doubling: B→{b2['B2']}  "
                  f"ΔdetG={b2['delta_detG_B2_minus_B']:.3e}  "
                  f"ΔminNorm={b2['delta_min_norm_B2_minus_B']:.3e}  "
                  f"Δλ_min(G)={b2['delta_lambda_min_G_B2_minus_B']:.3e}  "
                  f"Δcond(G)={b2['delta_cond_G_B2_minus_B']:.3e}")

        # SU(3) certificate (scale-aware): near-rank-1 if λ_min ≪ trace
        near_rank1 = None
        h_star_real = None
        if g == "su3":
            trG = best_refined["a"] + best_refined["c"]
```

```
            near_rank1 = (best_refined["lambda_min_G"] <= 1e-6 * trG)
            h_star_real = - best_refined["b_real"] / best_refined["c"] if
best_refined["c"] != 0.0 else float('nan')

        out = {
            "group": g,
            "harmonics": [m1, m2],
            "B_samples": args.B,
            "Nproj": args.Nproj,
            "grid": args.grid,
            "tau": args.tau,
            "r": args.r,
            "delta": float(eff_delta),

            "best_alpha": best_refined["alpha"],
            "best_beta": best_refined["beta"],

            "detG_at_best": best_refined["detG"],
            "min_norm_at_best": best_refined["min_norm"],
            "h_min_real_at_best": best_refined["h_min_real"],
            "h_min_imag_at_best": best_refined["h_min_imag"],

            "a_entry": best_refined["a"],
            "c_entry": best_refined["c"],
            "b_real": best_refined["b_real"],
            "b_imag": best_refined["b_imag"],

            "rho_T_at_best": best_refined["rho_T"],
            "lambda_min_G_at_best": best_refined["lambda_min_G"],
            "lambda_max_G_at_best": best_refined["lambda_max_G"],
            "cond_G_at_best": best_refined["cond_G"],
            "psd_violation_at_best": best_refined["psd_violation"],
            "psd_margin_at_best": best_refined["psd_margin"],
            "stable_h_at_best": best_refined["stable_h"],
        }
        if args.check_pn:
            out["delta_P_spectral_at_best"] = best_refined["delta_P_spectral"]
            out["delta_P_maxabs_at_best"] = best_refined["delta_P_maxabs"]
        if args.check_dynamics:
            out["convergence_rate_at_best"] = best_refined.get("convergence_rate",
float('nan'))
```

```
            out["max_drift_angle_at_best"] = best_refined.get("max_drift_angle",
float('nan'))
            out["coherence_ratio_at_best"] = best_refined.get("coherence_ratio",
float('nan'))
            out["coherence_ratio_reg_at_best"] =
best_refined.get("coherence_ratio_reg", float('nan'))
            if "dyn_lambda_min" in best_refined:
                out["dyn_lambda_min_at_best"] = best_refined["dyn_lambda_min"]
                out["dyn_lambda_max_at_best"] = best_refined["dyn_lambda_max"]
        if g == "su3":
            out["near_rank1"] = near_rank1
            out["h_star_estimate_real"] = h_star_real
        if args.check_b_doubling and b2 is not None:
            out["B_doubling"] = b2

        out['rel_min_norm'] = float(out['min_norm_at_best'] / max(out['a_entry'] +
out['c_entry'], 1e-30))
        out['verdict'] = None  # placeholder, console prints the verdict
        # Top-K points by the same objective for transparency
        def _score(rec):
            if args.objective == 'relmin':
                return rec.get('rel_min', float('inf'))
            if args.objective == 'det':
                return abs(rec['detG'])
            if args.objective == 'minnorm':
                return rec['min_norm']
            return rec.get('rel_min', float('inf'))
        all_records = sorted(records + ref_records, key=_score)[:max(1,
args.report_topk)]
        out['topK'] = [{k: r.get(k) for k in
['alpha','beta','rel_min','detG','min_norm','lambda_min_G','rho_T','convergence_rate']
} for r in all_records]
        if ablation_out is not None:
            out['ablation_no_sqrt_jacobian'] = {
                'detG_at_best': ablation_out['best']['detG'],
                'min_norm_at_best': ablation_out['best']['min_norm'],
                'rel_min_at_best': ablation_out['best'].get('rel_min'),
                'lambda_min_G_at_best': ablation_out['best']['lambda_min_G'],
                'rho_T_at_best': ablation_out['best']['rho_T']
            }
        summary[g] = out
```

```
        # Save scans (coarse + refined)
        with open(f"scan_{g}.json", "w") as f:
            json.dump(records + ref_records, f, indent=2)

    with open("phi_gram_summary.json", "w") as f:
        json.dump(summary, f, indent=2)

    # ---------------- Bridge computation (Appendix C) ----------------
    if args.bridge and all(k in summary for k in ("su3","su4")):
        s3 = summary["su3"]
        s4 = summary["su4"]
        def _pick_conv(sx):
            if args.bridge_agg == "median_topk" and "topK" in sx and sx["topK"]:
                vals = [r.get("convergence_rate", float("nan")) for r in sx["topK"] if
not (r.get("convergence_rate") is None)]
                vals = [v for v in vals if isinstance(v, (int,float)) and v==v and v>0]
                if vals:
                    vals.sort()
                    mid = vals[len(vals)//2] if len(vals)%2==1 else
0.5*(vals[len(vals)//2-1]+vals[len(vals)//2])
                    return mid
            return sx.get("convergence_rate_at_best", float("nan"))
        conv3 = _pick_conv(s3)
        conv4 = _pick_conv(s4)

        if not (isinstance(conv3,(int,float)) and conv3>0 and
isinstance(conv4,(int,float)) and conv4>0):
            print(style.color("[bridge] Missing/invalid convergence rates; cannot
compute bridge.", "red"))
        else:
            R0_3 = float(args.R0_SU3)
            R0_4 = float(args.R0_SU4)
            try:
                delta_alpha_hat = (np.log(R0_4 / R0_3)) / (np.log(1.0 / conv3))
            except Exception:
                delta_alpha_hat = float("nan")

            if np.isfinite(delta_alpha_hat):
                phi4 = (1.0 / conv4) ** delta_alpha_hat
                R4_pred = R0_4 * phi4
                discr = (conv4 / conv3) ** delta_alpha_hat
            else:
```

```
                phi4 = float("nan"); R4_pred = float("nan"); discr = float("nan")

            print("")
            print(style.color("=== One-Loop Bridge (Appendix C) ===", "bold",
"magenta"))
            print(f"Baseline: R0(SU3)={R0_3:.6f}, R0(SU4)={R0_4:.6f}")
            print(f"Convergence rates: SU3={conv3:.6e}, SU4={conv4:.6e}")
            print(f"Calibrated Δα̂ = ln(R0_4/R0_3)/ln(1/conv_SU3) ≈
{delta_alpha_hat:.6e}")
            print(f"Dynamic discriminant (conv4/conv3)^Δα̂ ≈ {discr:.6e}")
            print(f"Predicted R(SU4; D_eff) = R0(SU4) · (1/conv4)^Δα̂ ≈ {R4_pred:.6e}")
            dev = R4_pred - R0_4
            print(f"Deviation from baseline R0(SU4): {dev:+.6e}
({(dev/R0_4)*100:+.2f}%)")

            summary["bridge"] = {
                "R0_SU3": R0_3, "R0_SU4": R0_4,
                "conv_SU3": conv3, "conv_SU4": conv4,
                "delta_alpha_hat": float(delta_alpha_hat),
                "dynamic_discriminant": float(discr),
                "R_SU4_pred": float(R4_pred),
                "R_SU4_dev": float(dev),
                "R_SU4_dev_pct": float((dev/R0_4)*100.0)
            }

    # Print summary
    for g in groups:
        s = summary[g]
        print("")
        print(f"=== {g.upper()} CERTIFICATE === (harmonics {m1},{m2})")
        print(f"B={s['B_samples']}  Nproj={s['Nproj']}  grid={s['grid']}  "
              f"tau={s['tau']}  r={s['r']}  delta={s['delta']:.4f}  "
              f"Cesàro start={'T¹' if args.cesaro_start_T else 'I'}")
        print(f"best (alpha, beta) = ({s['best_alpha']:.6f}, {s['best_beta']:.6f})")
        print(f"det G = {s['detG_at_best']:.6e}")
        print(f"min ||u + h v||^2 (complex-h) = {s['min_norm_at_best']:.6e}")
        line = (f"ρ(T) = {s['rho_T_at_best']:.6f}  "
                f"λ_min(G) = {s['lambda_min_G_at_best']:.6e}  "
                f"λ_max(G) = {s['lambda_max_G_at_best']:.6e}  "
                f"cond(G) = {s['cond_G_at_best']:.3e}  "
                f"PSD violation? {s['psd_violation_at_best']}")
        print(line)
```

```
        if "delta_P_spectral_at_best" in s:
            print(f"ΔP₂N(‖·‖₂) = {s['delta_P_spectral_at_best']:.6e}   "
                  f"ΔP₂N(max) = {s['delta_P_maxabs_at_best']:.6e}")
        if g == "su3":
            print(f"near rank-1? {s['near_rank1']}   "
                  f"h_* (real estimate) ≈ {s['h_star_estimate_real']:.6f}   "
                  f"stable_h? {s['stable_h_at_best']}")
        if "convergence_rate_at_best" in s:
            print(f"Dynamic Analysis (N={args.N_dynamics}):")
            print(f"  Convergence Rate: {s.get('convergence_rate_at_best',
float('nan')):.6e}")
            print(f"  Max Drift Angle:  {s.get('max_drift_angle_at_best',
float('nan')):.6e} rad")
            print(f"  Coherence Ratio:  {s.get('coherence_ratio_at_best',
float('nan')):.6e}")
            print(f"  Coherence Ratio (reg.):  {s.get('coherence_ratio_reg_at_best',
float('nan')):.6e}")
        trG = s['a_entry'] + s['c_entry']
        rel_min = s['min_norm_at_best'] / max(trG, 1e-30)

        dp_ok = True
        conv_ok = True
        if "delta_P_spectral_at_best" in s:
            dp_ok = (s["delta_P_spectral_at_best"] <= 1e-7)
        if "convergence_rate_at_best" in s:
            conv_ok = (s["convergence_rate_at_best"] <= 3e-10)

        if rel_min <= 0.03 and dp_ok and conv_ok:
            verdict = "Coherent resonance (projector nearly rank-1; stable Cesàro)"
        elif rel_min <= 0.15 and dp_ok:
            verdict = "Weak resonance (partial collapse with stable Cesàro)"
        else:
            verdict = "Geometric obstruction (directions stay distinct; no collapse)"

        banner_color = 'green' if 'Coherent' in verdict else ('yellow' if 'Weak' in
verdict else 'red')
        print(style.color(f"Verdict: {verdict}", 'bold', banner_color))
        ok_rel = (rel_min <= 0.03)
        ok_dp = True
        ok_conv = True
        if 'delta_P_spectral_at_best' in s:
```

```
            ok_dp = (s['delta_P_spectral_at_best'] <= 1e-7)
        if 'convergence_rate_at_best' in s:
            ok_conv = (s['convergence_rate_at_best'] <= 3e-10)
        print(f"  {_tick(ok_rel)} rel_min = {rel_min:.3e}  (smaller is more coherent)")
        print(f"    {_coherence_bar(rel_min, style)}  thresholds: 0.03 (coherent), 0.15
(weak)")
        if 'delta_P_spectral_at_best' in s:
            print(f"  {_tick(ok_dp)} Cesàro stability ΔP₂N(‖·‖₂) =
{s['delta_P_spectral_at_best']:.3e}")
        if 'convergence_rate_at_best' in s:
            print(f"  {_tick(ok_conv)} Dynamic convergence rate =
{s['convergence_rate_at_best']:.3e}")
        if "B_doubling" in s:
            b2 = s["B_doubling"]
            print(f"B-doubling check (B→{b2['B2']}): "
                  f"detG_B2={b2['detG_B2']:.6e}, minNorm_B2={b2['min_norm_B2']:.6e}, "
                  f"λ_min(G)_B2={b2['lambda_min_G_B2']:.6e},
cond(G)_B2={b2['cond_G_B2']:.3e}")
            print(f"Deltas (B2 - B): ΔdetG={b2['delta_detG_B2_minus_B']:.6e}, "
                  f"ΔminNorm={b2['delta_min_norm_B2_minus_B']:.6e}, "
                  f"Δλ_min(G)={b2['delta_lambda_min_G_B2_minus_B']:.6e}, "
                  f"Δcond(G)={b2['delta_cond_G_B2_minus_B']:.6e}")

    print("\nLegend: ✅ pass, ❌ fail; rel_min bar fills more on stronger coherence."
          " Thresholds: rel_min ≤ 0.03 → Coherent, 0.03–0.15 → Weak, > 0.15 →
Obstruction.")

if __name__ == "__main__":
    main()
```

config.grc.json

```
{
 "scanner_cmd": ["python", "phi_gram_ref.py"],
 "objective": "relmin",

 "delta_values": [0.0, 0.785398, 1.570796, 2.356194, 3.141593],
 "delta_offset": 0.44879895,

 "su_groups_for_suite": ["su3", "su4"],

 "null": {
   "families": ["rank", "coxeter"],
   "scrambles_per_pair": 10,
   "su_for_null": "su3",
   "rank_pairs": [[4,5], [5,6], [6,7], [7,8], [8,9], [9,10]],
   "coxeter_pairs": [[4,5], [5,6], [6,7], [7,8], [8,9]]
 },

 "alpha": 0.01,
 "output_dir": "grc_outputs"
}
```

grc_outputs

blinded_min.json
Gamma_scan_su3_5_6.json
Null_thresholds.json
Offset_result.json
Su_suite_results.json

—


```
{
 "masked_runs": [
   {
     "label": "B",
     "metrics": {
       "rel_min": 0.000434,
       "convergence_rate": 6.773082e-10,
       "rho_T": 4e-06,
       "lambda_min": 0.000434,
       "lambda_max": 0.0004340316,
       "cond": null
     },
     "verdict": "PASS",
     "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,0.000000)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,-0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.0000  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, -0.785398)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773082e-10\n  Max Drift Angle:
0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.430508e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
```

```
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
   },
   {
     "label": "A",
     "metrics": {
       "rel_min": 0.000434,
       "convergence_rate": 8.901403e-10,
       "rho_T": 6e-06,
       "lambda_min": 0.000434,
       "lambda_max": 0.0004340316,
       "cond": null
     },
     "verdict": "PASS",
     "stdout": "[su4] coarse best at (\u03b1,\u03b2)=(0.314159,0.314159)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000006  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su4] final best at
(\u03b1,\u03b2)=(-0.471239,-0.471239)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000006  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU4 CERTIFICATE === (harmonics 6,7)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.0000  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.471239, -0.471239)\ndet G = 1.883796e-07\nmin ||u + h v||^2 (complex-h) =
4.340316e-04\n\u03c1(T) = 0.000006  \u03bb_min(G) = 4.340228e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nDynamic Analysis (N=128):\n  Convergence Rate: 8.901403e-10\n  Max Drift
Angle:  0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.441906e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 8.901e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
   }
 ],
 "unmask": {
   "B": "su3",
   "A": "su4"
```

```
  }
}
```

```
{
 "group": "su3",
 "harmonics": [
   5,
   6
 ],
 "rows": [
   {
     "delta": 0.0,
     "metrics": {
       "rel_min": 0.000434,
       "convergence_rate": 6.773082e-10,
       "rho_T": 4e-06,
       "lambda_min": 0.000434,
       "lambda_max": 0.0004340316,
       "cond": null
     },
     "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,0.000000)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,-0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.0000  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, -0.785398)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773082e-10\n  Max Drift Angle:
0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.430508e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
   },
```

```
  {
    "delta": 0.785398,
    "metrics": {
      "rel_min": 0.000434,
      "convergence_rate": 6.773079e-10,
      "rho_T": 4e-06,
      "lambda_min": 0.000434,
      "lambda_max": 0.0004340316,
      "cond": null
    },
    "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,1.570796)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.7854  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, 0.785398)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02  \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False  h_* (real estimate) \u2248 0.000000  stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773079e-10\n  Max Drift Angle:
1.490116e-08 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.431905e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
  },
  {
    "delta": 1.570796,
    "metrics": {
      "rel_min": 0.000434,
      "convergence_rate": 6.773077e-10,
      "rho_T": 4e-06,
      "lambda_min": 0.000434,
```

```
      "lambda_max": 0.0004340316,
      "cond": null
    },
    "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,3.141593)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,2.356194)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=1.5708  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, 2.356194)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773077e-10\n  Max Drift Angle:
1.490116e-08 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.435285e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
  },
  {
    "delta": 2.356194,
    "metrics": {
      "rel_min": 0.000434,
      "convergence_rate": 6.773074e-10,
      "rho_T": 4e-06,
      "lambda_min": 0.000434,
      "lambda_max": 0.0004340316,
      "cond": null
    },
    "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,4.712389)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
```

```
(\u03b1,\u03b2)=(-0.785398,3.926991)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=2.3562  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, 3.926991)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773074e-10\n  Max Drift Angle:
1.490116e-08 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.438665e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
   },
   {
     "delta": 3.141593,
     "metrics": {
       "rel_min": 0.000434,
       "convergence_rate": 6.773077e-10,
       "rho_T": 4e-06,
       "lambda_min": 0.000434,
       "lambda_max": 0.0004340316,
       "cond": null
     },
     "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,0.000000)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,-0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=3.1416  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, -0.785398)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
```

```
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773077e-10\n  Max Drift Angle:
0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.440071e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
    }
  ],
  "verdicts": [
    null
  ],
  "invariant": true
}
```

```
{
 "source": "union-null(rank+coxeter)",
 "alpha": 0.01,
 "thresholds": {
   "rel_min": 0.000434,
   "convergence_rate": 1.182754e-09
 },
 "summary": {
   "rel_min": {
     "mu": -7.742466023863891,
     "sigma": 2.1316282072803006e-14,
     "n": 110
   },
   "convergence_rate": {
     "mu": -20.98371972095056,
     "sigma": 0.3871031451004631,
     "n": 110
   }
 },
 "counts": {
   "rank": 60,
   "coxeter": 50,
   "union": 110
 }
}
```

```
{
 "group": "su4",
 "harmonics": [
   5,
   6
 ],
 "delta0": 0.0,
 "delta1": 0.44879895,
 "run0": {
   "stdout": "[su4] coarse best at (\u03b1,\u03b2)=(0.000000,1.570796)  detG=1.884e-07
minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su4] final best at
(\u03b1,\u03b2)=(-0.785398,0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU4 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.4488  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, 0.785398)\ndet G = 1.883803e-07\nmin ||u + h v||^2 (complex-h) =
4.340306e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340254e-04  \u03bb_max(G) =
4.340306e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041670e-02   \u0394P\u2082N(max) =
1.041670e-02\nDynamic Analysis (N=128):\n  Convergence Rate: 6.786913e-10\n  Max Drift
Angle:  1.781920e-07 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
2.558781e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.787e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n",
   "metrics": {
     "rel_min": 0.000434,
     "convergence_rate": 6.786913e-10,
     "rho_T": 4e-06,
     "lambda_min": 0.000434,
     "lambda_max": 0.0004340306,
     "cond": null
   }
 },
 "run1": {
```

```
  "stdout": "[su4] coarse best at (\u03b1,\u03b2)=(0.000000,0.628319)  detG=1.884e-07
minNorm=4.340e-04  rho(T)=0.000003  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su4] final best at
(\u03b1,\u03b2)=(-0.785398,0.985318)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000003  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU4 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.8976  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, 0.985318)\ndet G = 1.883797e-07\nmin ||u + h v||^2 (complex-h) =
4.340298e-04\n\u03c1(T) = 0.000003  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340298e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041669e-02  \u0394P\u2082N(max) =
1.041669e-02\nDynamic Analysis (N=128):\n  Convergence Rate: 4.957304e-10\n  Max Drift
Angle:  0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
1.793940e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 4.957e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n",
  "metrics": {
    "rel_min": 0.000434,
    "convergence_rate": 4.957304e-10,
    "rho_T": 3e-06,
    "lambda_min": 0.000434,
    "lambda_max": 0.0004340298,
    "cond": null
  }
 },
 "verdict_unchanged": true
}
```

```
{
 "thresholds": {
   "rel_min": 0.000434,
   "convergence_rate": 1.182754e-09
 },
 "rows": [
   {
     "group": "su3",
     "harmonics": [
       5,
       6
     ],
     "rel_min": 0.000434,
     "convergence_rate": 6.773082e-10,
     "z_static": -1.0,
     "z_dynamic": 0.33369654462990284,
     "accessibility": -1.3336965446299027,
     "verdict": "PASS",
     "stdout": "[su3] coarse best at (\u03b1,\u03b2)=(0.000000,0.000000)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su3] final best at
(\u03b1,\u03b2)=(-0.785398,-0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000004  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU3 CERTIFICATE === (harmonics 5,6)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.0000  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, -0.785398)\ndet G = 1.883805e-07\nmin ||u + h v||^2 (complex-h) =
4.340249e-04\n\u03c1(T) = 0.000004  \u03bb_min(G) = 4.340249e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nnear rank-1? False   h_* (real estimate) \u2248 0.000000   stable_h?
True\nDynamic Analysis (N=128):\n  Convergence Rate: 6.773082e-10\n  Max Drift Angle:
0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.430508e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 6.773e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
```

```
  },
  {
    "group": "su4",
    "harmonics": [
      6,
      7
    ],
    "rel_min": 0.000434,
    "convergence_rate": 8.901403e-10,
    "z_static": -1.0,
    "z_dynamic": -0.37219458843790404,
    "accessibility": -0.6278054115620959,
    "verdict": "PASS",
    "stdout": "[su4] coarse best at (\u03b1,\u03b2)=(0.000000,0.000000)
detG=1.884e-07  minNorm=4.340e-04  rho(T)=0.000006  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n[su4] final best at
(\u03b1,\u03b2)=(-0.785398,-0.785398)  detG=1.884e-07  minNorm=4.340e-04
rho(T)=0.000006  \u03bb_min(G)=4.340e-04
\u0394P\u2082N(\u2016\u00b7\u2016\u2082)=1.042e-02
\u0394P\u2082N(max)=1.042e-02\n\n=== SU4 CERTIFICATE === (harmonics 6,7)\nB=1024
Nproj=48  grid=20  tau=0.02  r=0.08  delta=0.0000  Ces\u00e0ro start=I\nbest (alpha,
beta) = (-0.785398, -0.785398)\ndet G = 1.883796e-07\nmin ||u + h v||^2 (complex-h) =
4.340316e-04\n\u03c1(T) = 0.000006  \u03bb_min(G) = 4.340228e-04  \u03bb_max(G) =
4.340316e-04  cond(G) = 1.000e+00  PSD violation?
False\n\u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.041671e-02   \u0394P\u2082N(max) =
1.041671e-02\nDynamic Analysis (N=128):\n  Convergence Rate: 8.901403e-10\n  Max Drift
Angle:  0.000000e+00 rad\n  Coherence Ratio:  inf\n  Coherence Ratio (reg.):
3.441906e+08\nVerdict: Geometric obstruction (directions stay distinct; no collapse)\n
\u274c rel_min = 5.000e-01  (smaller is more coherent)\n
[\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\u2591\
u2591\u2591\u2591\u2591\u2591\u2591\u2591]  thresholds: 0.03 (coherent), 0.15 (weak)\n
\u274c Ces\u00e0ro stability \u0394P\u2082N(\u2016\u00b7\u2016\u2082) = 1.042e-02\n
\u274c Dynamic convergence rate = 8.901e-10\n\nLegend: \u2705 pass, \u274c fail;
rel_min bar fills more on stronger coherence. Thresholds: rel_min \u2264 0.03 \u2192
Coherent, 0.03\u20130.15 \u2192 Weak, > 0.15 \u2192 Obstruction.\n"
  }
 ],
 "spearman_dynamic_z_vs_N": null
}
```