Doc 15 — Alpha-Elasticity CI Suite (Synthetic Verification Protocol)

Synthetic CI, Verification-First (Specification)

Status: CI / Synthetic protocol.

Purpose: Validate auditability, preregistered gates, and failure-mode coverage for an alpha-like estimator.

Non-claim: This document does not constitute physical validation of the fine-structure constant α.

Purpose

This document specifies a synthetic CI validation suite for an "alpha-elasticity" estimator within SFT. The goal is not to claim physical confirmation, but to ensure the estimator pipeline behaves consistently, reproducibly, and fails when it should.

Scope note: This suite validates the verification pipeline (CI / synthetic). It does not constitute end-to-end physical validation. REAL validation requires solver outputs (P) and independent replication.

Definition (for CI scoring only): alpha_reference is computed from the synthetic generator parameters using a locked mapping and is used only for scoring. The estimator must not read alpha_reference (or any target labels) and must operate solely on the synthetic field artifacts.

Core principle: No "success by construction"

Synthetic CI is acceptable only if the estimator is forced to work on structured inputs. Therefore:

- We never generate alpha directly as alpha_target*(1+noise) and then "recover it".
- We never adjust thresholds based on closeness to the target ("adaptive gates").
- We include null baselines and adversarial cases to measure false positives.

Inputs and Outputs

Inputs

The suite consumes one or more synthetic datasets produced by a deterministic generator:

- S_field_* (synthetic S field or equivalent internal representation)

- meta.json (grid, dt, BC tags, seed, generator_version)
- Optional: derived artifacts needed by the estimator (phase/theta maps, currents, etc.)

Outputs (Artifacts)

Each CI run produces:

1. ALPHA_ELASTICITY_REPORT.json
2. ALPHA_ELASTICITY_MANIFEST_SHA256.txt (hash list excluding itself)

MANIFEST_SHA256.txt must include hashes for DECISION_RULE.json and ALPHA_ELASTICITY_REPORT.json.

DECISION_RULE.json (fixed decision rule/thresholds used in discriminative tests; referenced by gate_id)

3. Optional plots (non-authoritative): alpha_hist.png, sensitivity_sweep.png, confusion_matrix.png

All outputs must be reproducible given the same inputs and seed.

Artifact Contract: ALPHA_ELASTICITY_REPORT.json

Required fields (minimum)

- schema_version (e.g., "1.0.0")
- suite_version
- gate_id (immutable identifier for this gate set)

decision_rule_sha256 (SHA-256 of DECISION_RULE.json; required)

status (e.g., CI / REAL-ready / REAL-verified; for this suite expect CI)

- provenance: seed, commit (or release tag), timestamp_utc, inputs_sha256
- tests: dictionary keyed by test name, each containing metrics, pass (boolean), notes (optional)
- GLOBAL_PASS (boolean)
- normalization (required): explicit definition of alpha_est (estimator formula/summary), masking/windows and any R≈0 excision rule, units or dimensionless scaling conventions, and any internal label mappings used for reporting

Note: normalization is mandatory. Gates and comparisons are meaningful only if the estimator definition, windows/masks, and scaling conventions are stated unambiguously and remain stable across runs.

Strong recommendation

- generator_config (parameters used to generate synthetic fields)

Gate policy (pre-registered)

- Gates are fixed per gate_id and must not change after results are observed.
- Any gate change requires a new gate_id and an explicit gate_change_reason.
- Gates may differ for CI vs REAL runs, but must be labeled explicitly (status: CI / REAL-ready / REAL-verified).

Tests (CI synthetic)

The suite contains five tests. Each test is designed to be falsifiable and to catch specific failure modes.

Test 1 — Recovery on structured synthetic fields (non-tautological)

Goal: Verify that the estimator returns stable alpha estimates for synthetic fields generated from a known parameterization without directly injecting alpha into the output.

Method:

- Generate structured S fields from a parametric generator G(params, seed) where params control field stiffness/coupling indirectly (not "alpha itself"). Generator parameters must not directly encode the same features used by the estimator; any unavoidable overlap must be documented and shown non-invertible/degenerate.
- Use a locked mapping params -> alpha_reference only for scoring (not for generating the estimator output).
- Run the estimator on N samples.

Metrics:

- mean_rel_error = mean(|alpha_est - alpha_ref| / alpha_ref)
- median_rel_error
- cv = std(alpha_est) / mean(alpha_est)
- n = sample_count

Gate (example CI):

- median_rel_error <= 0.05
- cv <= 0.05
- n >= 20

Notes: This test is valid only if the generator does not bake alpha directly into the estimator output.

Test 2 — Cross-configuration consistency (repeatability)

Goal: alpha estimates remain consistent across different synthetic configurations that share the same underlying reference setting.

Method:

- Generate K families of synthetic inputs (different spatial layouts, seeds, perturbations) holding the "reference alpha condition" constant.

Metrics:

- family_means[]
- between_family_cv
- worst_family_rel_drift = max(|mean_k - global_mean| / global_mean)

Gate:

- between_family_cv <= 0.03
- worst_family_rel_drift <= 0.05

Test 3 — Sensitivity (monotone response to controlled parameter change)

Goal: alpha estimate changes in the correct direction when a single generator parameter is varied.

Method:

- Sweep one parameter p across M values while holding everything else fixed.
- For each p, generate multiple seeds and estimate alpha.

Metrics:

- spearman_rho(alpha_est, p) (directional monotonicity)
- slope_sign_correct (based on preregistered expected direction)
- slope_stability (bootstrap CI excludes 0)

Gate:

- |spearman_rho| >= 0.8
- slope_sign_correct = true
- CI95(slope) excludes 0

This replaces arbitrary random "mean error < X%" checks with a real sensitivity test.

Test 4 — Discriminative power vs null / adversarial (no adaptive thresholds)

Goal: The pipeline must distinguish "structure-consistent" cases from null or adversarial cases, without tuning thresholds post-hoc.

Method:

Null/adversarial generation must not access alpha_reference or target labels; it must operate only on unlabeled synthetic artifacts.

- Construct a labeled dataset:
- Positive class: structured fields from G(params).
- Null class: random fields, phase-scrambled fields, or fields with injected continuity violations.
- Adversarial: near-structured fields with subtle violations that should break alpha extraction.
- Run a fixed decision rule: "valid alpha extraction" vs "invalid/reject". The decision rule must be declared in DECISION_RULE.json, hashed in the manifest, and referenced by gate_id (no post-hoc edits).

Metrics:

- Confusion matrix: TP, FP, TN, FN
- precision, recall, F1
- FP_rate

Gate:

- F1 >= 0.85
- FP_rate <= 0.05

No thresholds may depend on observed deviation from target. All thresholds come from gate_id. The operative thresholds must be those in DECISION_RULE.json for the declared gate_id.

Test 5 — Robustness to grid/BC perturbations (controlled)

Goal: Within a preregistered perturbation envelope (grid, BC tags, mild noise), alpha remains stable or the run is correctly flagged as invalid.

Method:

- Generate the same scenario under nuisance variants:
- Resolution levels (e.g., N=64, 96, 128)
- Minor BC variants (tagged)
- Small injected noise in initial conditions

Metrics:

- robust_mean_rel_drift
- robust_max_rel_drift
- valid_fraction (how often the estimator returns a valid result)

Gate:

- If validity is expected: robust_max_rel_drift <= 0.05 and valid_fraction >= 0.9
- If invalidity is expected (stress cases): valid_fraction <= 0.1 (correctly rejects)

Report example (minimal)


```
{
  "schema_version": "1.0.0",
  "suite_version": "alpha_elasticity_ci_v1",
  "gate_id": "AE-CI-0003",
  "status": "CI",
  "decision_rule_ref": "DECISION_RULE.json",
  "decision_rule_sha256": "...",
  "provenance": {
    "seed": 12345,
    "commit": "abc1234",
    "timestamp_utc": "2025-12-28T12:00:00Z",
    "inputs_sha256": {
      "synthetic_set.tar.gz": "...",
      "meta.json": "..."
    }
  },
  "tests": {
    "T1_recovery_structured": {
      "metrics": { "median_rel_error": 0.031, "cv": 0.021, "n": 40
},
      "pass": true
    },
    "T4_null_adversarial": {
      "metrics": { "F1": 0.91, "FP_rate": 0.03 },
      "pass": true
    }
  },
  "GLOBAL_PASS": true
}
```

Failure modes (explicit)

The suite must fail (or mark invalid) under these conditions:

- Any gate is changed without updating gate_id.
- Missing provenance (seed, commit, inputs_sha256).
- Adaptive gates based on closeness to target.
- Target leakage (features derived from reference alpha or target labels).
- Synthetic generator injects alpha directly into estimator outputs.
- Non-determinism with fixed seed.

What this suite does not claim

- It does not validate SFT physics end-to-end.
- It does not prove alpha is physically derived from SFT.
- It does not replace REAL solver campaigns (P) and convergence/GCI.

It only validates that the alpha-elasticity pipeline is reproducible, falsifiable, and resistant to trivial self-confirmation.

Companion Toy Example (CI Visualization) — How to Use This Suite in Practice

Status and scope (read this first)

This companion example is a **toy visualization module** meant to demonstrate how the **SFT Alpha-Elasticity Validation Suite** works operationally (inputs → metrics → gates → PASS/FAIL → hashed artifacts).
It is **not** a physical derivation of the fine-structure constant and must **not** be used as evidence of physical validation. Its purpose is to validate **auditability**, **anti-leakage discipline**, and **failure-mode coverage** under controlled synthetic inputs.

Core idea of the toy example

The toy pipeline produces a dimensionless **alpha_like** scalar from synthetic field artifacts using fixed, preregistered operators and gates. A recommended default is:

- **alpha_like** := `median( ||H(S)||_F ) / ( median(|∇S|) + eps )`

Note: alpha_like is a demonstration metric for CI validation; it is not asserted to equal the physical fine-structure constant α.

All operators (∇, H) are computed using the preregistered discrete stencil and boundary handling specified in DECISION_RULE.json.

where:

- ∇S is the discrete gradient of the scalar field `S`,
- `H(S)` is the discrete Hessian,
- $||\cdot||_F$ is Frobenius norm,
- `eps` is a fixed numerical stabilizer declared in the decision rule.

This definition is chosen because it is:

- fully internal (no use of 1/137 or any α target),
- dimensionless (ratio of invariants),
- sensitive to controlled changes in synthetic stiffness/roughness,
- and easy to visualize.

Required artifacts produced by the toy runner

This companion demo produces the same core artifacts defined in the suite Outputs section (report, preregistered decision rule, and hash manifest). In addition, it may emit the following non-authoritative plots for visualization:

- alpha_hist.png
- sensitivity_sweep.png
- confusion_matrix.png
- robustness_vs_resolution.png

How this toy example maps to the suite tests (T1–T5)

T1 — Recovery on structured synthetic fields (non-tautological)

- Generate structured synthetic `S` fields from a parameterized generator `G(params, seed)`.
- Compute `alpha_like` for N samples.
- Score against a **locked reference mapping** only for evaluation (not accessible to the estimator).
- Expected output: stable alpha_like distribution with bounded median error and low CV.

T2 — Cross-configuration consistency (repeatability)

- Create K families of synthetic configurations that share the same underlying condition (same generator regime) but vary layout/seed/perturbations.
- Verify that between-family drift remains within gates.

T3 — Sensitivity to controlled parameter change (monotonic response)

- Sweep a single generator parameter `p` (e.g., synthetic stiffness/roughness knob).
- Check monotonicity via Spearman ρ and slope significance (bootstrap CI).

T4 — Discriminative power vs null/adversarial (no adaptive thresholds)

- Build labeled sets:
  - Positive: structured fields.
  - Null: random or phase-scrambled fields.
  - Adversarial: near-structured fields with injected violations.
- **Critical constraint:** Null/adversarial generation must not access `alpha_reference` or target labels; it must operate only on unlabeled synthetic artifacts.
- Evaluate with confusion matrix + F1 + FP_rate under a **fixed** decision rule from `DECISION_RULE.json`.

T5 — Robustness to grid / BC perturbations

- Re-run the same scenario across preregistered resolution levels and mild BC variants.
- Verify stability (or correct rejection) according to the preregistered robustness policy.

PASS/FAIL semantics

- A run is **PASS** only if:
  1. all required provenance fields are present,
  2. `decision_rule_sha256` matches the provided `DECISION_RULE.json`,
  3. input and output hashes are recorded,
  4. all test gates pass, and
  5. `GLOBAL_PASS = true`.

Any missing provenance, missing rule hash, or post-hoc gate change is an automatic **FAIL**.

Why this "toy visualization" matters

- Third-party runnable demonstration of the suite's PASS/FAIL workflow.
- Auditable integrity: preregistered gates (gate_id) and hash-tracked artifacts.

Relationship to other SFT packs

- alpha_like (this CI suite) is a synthetic stability/consistency metric; it is not asserted to equal the physical fine-structure constant.
- alpha_out (A-OUT / Doc 10) is the physical-constant composition pipeline and should be treated as a separate, REAL-ready route.
- Capstone exploratory demos may reference α only for post-hoc comparison; thresholds and decision rules must remain preregistered (no target-driven tuning).