`namaster-proof`: Exact pseudo-C_ℓ window inference and content-bound validation for reproducible spin-2 analyses

Houston Golden
Independent researcher

July 16, 2026 — v2B.0.11

Abstract

`namaster-proof` is a focused Python verification layer for two error-prone steps in cut-sky spin-2 analyses. First, it evaluates a uniformly rotated EE, EB, BE, BB spectrum through the complete NaMaster bandpower-window operator, avoiding replacement of the operator by bin-centre or effective-multipole templates. Second, it writes JSON results and content-bound sidecar receipts with atomic per-file replacement and fails closed when result bytes or caller-asserted execution metadata change. The package also supplies explicit multipole-support contracts, fixed-grid rotation-angle recovery, command-line receipt verification, and compatibility tests against the production helpers from which it was extracted. The software is intended for method validation and reproducibility checks; it is not a sky-analysis pipeline, foreground model, or cosmological inference engine.

1 Overview

Keywords. Python; cosmology; pseudo-C_ℓ; NaMaster; reproducibility; provenance.

2 Introduction

Pseudo-C_ℓ estimators make masked spin-2 analyses computationally practical by relating full-sky spectra to coupled and decoupled bandpowers [1, 2]. Reproducible use of these estimators requires more than recording a best-fit number: the same window operator must be used in simulation and inference, harmonic limits must agree across fields and bins, and intermediate results must remain bound to their execution metadata.

`namaster-proof` packages these checks behind a small public API. Its numerical module constructs exact windowed responses for uniform polarization rotation, evaluates them at one or many angles, recovers an angle on a declared grid, and compares the window contraction with NaMaster's couple–decouple operator. Its provenance module publishes a JSON result and a SHA-256-bound receipt using atomic replacement for each file, then verifies the pair and caller-asserted metadata. A third module defines deterministic multipole and binning contracts. NumPy is the only required dependency; a NaMaster workspace is supplied by the user when exercising the physical window operator.

3 Statement of Need

For two spin-2 fields, NaMaster exposes a bandpower-window tensor with spectrum ordering $[EE, EB, BE, BB]$. A common shortcut is to evaluate a theory spectrum at a representative multipole for each bin. That shortcut is not generally identical to applying the complete mode-coupling and binning operator, and a mismatch between the simulation and fitting operators can appear as a parameter-recovery bias.

A separate reproducibility risk arises in long or sharded calculations. Ordinary filenames do not prove that a summary corresponds to the intended seeds, realization count, configuration, or result bytes. A process can also terminate between writing a result and its metadata, leaving a plausible but incomplete artifact. These problems are usually addressed with analysis-local helpers. `namaster-proof` provides a reusable, tested implementation with explicit failure behavior, while remaining narrow enough to audit.

4 Implementation and Architecture

The package is divided into three modules:

`windows` Rotation of EE, EB, BE, BB spectra, exact contraction with a `get_bandpower_windows()` tensor, fixed-grid angle recovery, and an equivalence check against `couple_cell()` followed by `decouple_cell()`.

Multipoles Validation of a common harmonic cutoff and construction of integer bandpower edges whose final exclusive edge is $\ell_{\max} + 1$.

`receipts` Atomic JSON publication, streaming SHA-256 calculation, sidecar-receipt construction, and fail-closed content and metadata validation.

The functions accept array-like inputs and return NumPy arrays or plain Python dictionaries. NaMaster is deliberately not an installation dependency: window functions require a workspace exposing `get_bandpower_windows()`, `couple_cell(cls)`, and `decouple_cell(coupled)`. The first must return a finite $[4, n_b, 4, n_\ell]$ tensor and the couple–decouple route must return a finite $[4, n_b]$ array. This small protocol makes the algebra testable with synthetic linear operators and permits users to pin PyMaster independently.

5 Exact-Window Inference

For an initially vanishing EB spectrum and a uniform rotation angle β, the rotated spectra can be decomposed into angle-independent, $\cos(4\beta)$, and $\sin(4\beta)$ components. In particular,

$$C_\ell^{EE}(\beta) = \tfrac{1}{2}(C_\ell^{EE} + C_\ell^{BB}) + \tfrac{1}{2}(C_\ell^{EE} - C_\ell^{BB})\cos(4\beta), \tag{1}$$

$$C_\ell^{BB}(\beta) = \tfrac{1}{2}(C_\ell^{EE} + C_\ell^{BB}) - \tfrac{1}{2}(C_\ell^{EE} - C_\ell^{BB})\cos(4\beta), \tag{2}$$

$$C_\ell^{EB}(\beta) = C_\ell^{BE}(\beta) = \tfrac{1}{2}(C_\ell^{EE} - C_\ell^{BB})\sin(4\beta). \tag{3}$$

Let $W_{b\ell}^{ij}$ be the workspace tensor mapping input spectrum j to output spectrum i in band b. The package precontracts each of the three components,

$$R_b^{i,k} = \sum_{j,\ell} W_{b\ell}^{ij} C_\ell^{j,k}, \tag{4}$$

and subsequently evaluates

$$C_b^i(\beta) = R_b^{i,0} + \cos(4\beta) R_b^{i,c} + \sin(4\beta) R_b^{i,s}. \tag{5}$$

This retains the full window tensor while making repeated grid evaluation inexpensive. The recovery function minimizes a declared weighted or unweighted squared residual over a user-provided grid; its default is 2001 points spanning -1° to $+1^\circ$.

6 Content Validation

`publish_json` serializes a mapping with sorted keys, derives the receipt digest and byte count from that immutable serialized snapshot, writes and flushes a process-specific temporary file, atomically replaces the result, and synchronizes the containing directory. It then separately publishes a sidecar receipt containing schema version, result filename, byte count, SHA-256 digest, and caller-supplied execution metadata. The content-binding fields are protected from metadata override.

`verify_json_receipt` requires both files, reads each through one file handle, parses those immutable byte snapshots as JSON objects, and recomputes every protected field from the same result bytes it returns. `validate_json_receipt` additionally checks caller-asserted metadata such as a suite identifier, seed range, or realization count using recursive JSON-type-strict equality, so booleans, integers, and floating-point numbers are not interchangeable. Missing files, malformed objects, changed bytes, changes to protected receipt fields, or asserted-contract mismatches raise an exception. The two files are not one filesystem transaction, and coordinated replacement of both requires an external receipt anchor or trusted expected metadata to detect. The `namaster-proof verify` and `namaster-proof validate` commands expose the same behavior to shell workflows and return a nonzero status on failure.

7 Quality Control

Version 0.1.7 contains 41 automated tests covering numerical behavior, invalid inputs, the CLI, receipt mutation, protected fields, and compatibility with the original production helpers. All 41 run in a monorepo checkout; in a standalone package install 39 run and the 2 replay-equivalence tests skip cleanly, because those two tests replay against production helper modules that reside elsewhere in the monorepo rather than inside the package. The window suite uses an exactly represented synthetic linear workspace to compare the precontracted response with the couple–decouple route and to recover an injected grid angle. Rotation tests check conservation of total $EE + BB$ auto-power. Receipt tests mutate result bytes and receipt digests independently and require rejection.

Compatibility tests load the pre-package production modules and compare windowed bandpowers at negative, zero, and positive angles with zero requested absolute or relative tolerance. They also compare algebraic spectrum rotation, bandpower edges, and harmonic keyword contracts. In the associated physical production artifact, a workspace of shape $[4, 20, 4, 1025]$ gave a maximum absolute difference of 1.41×10^{-18} between direct window contraction and the couple–decouple operator. This recorded execution check used IEEE-754 double-precision arrays and the production driver's 10^{-10} acceptance threshold; the package's separate synthetic legacy-helper comparison uses zero requested absolute and relative tolerance. Because the original workspace tensor was not retained, the scalar is not a self-contained reproducibility claim or a universal error bound. The tensor is nonetheless deterministically regenerable from committed, RNG-free code—the mask is a pure function of N_{side} and fixed latitude cuts and the 10^{-10}-gated equivalence check is committed—so the `examples/rebuild_workspace_check.py` script rebuilds the $[4, 20, 4, 1025]$ workspace and re-verifies $\max|\Delta| < 10^{-10}$ whenever PyMaster is installed (only the last digits of the $\sim 10^{-18}$ value are platform-dependent). The full suite runs with `python -m pytest packages/namaster-proof/tests` from the repository root (or `pytest` from within the installed package's test extra).

8 Worked Examples

Minimal synthetic operator. The documentation constructs an identity-like workspace, builds a response from synthetic EE and BB arrays, evaluates a known rotation, and checks operator equivalence. This example has no dependency on a paper-specific dataset and is suitable for installation tests. The minimal call sequence is `response = build_rotation_response(workspace, ee,`

`bb)` followed by `windowed_bandpowers(response, beta_rad=np.deg2rad(0.25))`; the complete executable example also calls `validate_window_equivalence`.

Real PyMaster integration. The optional `examples/pymaster_integration.py` example constructs a real PyMaster workspace, verifies the exact operator equivalence, recovers a declared grid injection, and records the result beside an effective-multipole shortcut comparison. It requires separately installed PyMaster and healpy.

Synthetic CMB recovery campaign. The production use case employed CAMB 1.6.6 raw EE/BB spectra, PyMaster 2.6, $N_{\mathrm{side}} = 512$, $\ell_{\mathrm{max}} = 1024$, and deterministic seeds. For each of two nonzero injected angles, a 500-realization, foreground-free synthetic run recovered the injection from the mean bandpowers on the declared 0.001° grid: $0.270^\circ \mapsto 0.270^\circ$ and $0.342^\circ \mapsto 0.342^\circ$. The null injection recovered 0.000°. These are software-recovery checks under the stated simulation contract, not measurements, detection significances, or evidence for a physical birefringence model.

Sharded result validation. Production shards record fields such as suite, realization count, and seed range alongside the content digest. Aggregation can therefore reject a shard whose bytes were altered or whose declared range does not match the requested campaign, rather than silently combining it with valid results.

9 Author Contributions

Houston Golden: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing—original draft, writing—review and editing, visualization, project administration, and funding acquisition. Correspondence metadata remain author-supplied submission metadata and are not inferred by the software release process.

10 Limitations

`namaster-proof` does not create masks or maps, estimate covariance matrices, separate cosmic rotation from instrumental polarization calibration, model foregrounds, choose a likelihood, or infer cosmological parameters. Uniform rotation and initially vanishing EB are assumptions of the optimized three-component response; the general algebraic rotation helper accepts all four input spectra, but spatially varying rotation requires another model. Fixed-grid recovery inherits the range and resolution selected by the caller. Receipt validation establishes file integrity and agreement with caller-asserted metadata, not the scientific correctness of that metadata. SHA-256 sidecars are content bindings, not digital signatures, an authorization system, or protection against coordinated replacement without an external anchor. Schema version 1 has no migration requirement yet; a future incompatible schema will require an explicit reader or conversion path.

11 Availability

Operating system. The package supports Linux and Windows where Python and NumPy are available; CI covers Linux with Python 3.10–3.13 and Windows with Python 3.12. Directory metadata are synchronized on POSIX systems after atomic replacement. macOS is expected to work as a POSIX platform wherever Python and NumPy are available, but is not currently exercised in continuous integration and is therefore listed as untested.

Additional system requirements. The package has no special hardware requirements: it is pure Python with a single NumPy runtime dependency, requires no GPU, and its full test suite completes in under one second with a negligible memory footprint on a commodity CPU. The optional PyMaster integration example ($N_{\rm side} = 16$) completes in a few seconds, whereas the retained physical validation campaign ($N_{\rm side} = 512$, $\ell_{\rm max} = 1024$; 500 realizations at each of three injected angles) recorded a wall time of order 7×10^2 s with eight parallel realization workers on a commodity multi-core CPU (correspondingly of order a couple of hours single-worker). These campaign timings are environment-dependent; both workloads depend on the user-supplied PyMaster and healpy stacks and correspondingly more memory, but lie outside the self-contained packaged code path.

Programming language and dependencies. Version 0.1.7 requires Python 3.10 or later and NumPy 1.24 or later. PyMaster is optional and is supplied by the user for physical NaMaster workspaces; the retained physical validation used PyMaster 2.6 and healpy 1.19.0.

Code repository. The source, issue tracker, installation instructions, API documentation, tests, and examples are available at `https://github.com/Hubify-Projects/bigbounce/tree/main/packages/namaster-proof`. From a checkout of that repository the package installs locally with `python -m pip install ./packages/namaster-proof` (append `-e '.[test]'` from within the package directory for the test extra). `namaster-proof` is an independent downstream verification layer and is not an official release of, or affiliated with, the NaMaster project. Citation metadata are supplied in `CITATION.cff` and machine-readable metadata in `codemeta.json`. The package directory was first published to the repository on 2026-07-16.

License. The software is released under the MIT License.

Archive. A persistent archival identifier is not yet available. This is an explicit submission blocker, not a completed release claim. The submission candidate must bind version 0.1.7 to an immutable archive before journal submission.

Validation artifacts. The corrected physical example is bound to the repository summary artifact (SHA-256 `745b0a2f060773ce69c005ea84b74b305ec26a85f6aaafe58f0b3244b7f39914`) and the repository bandpower artifact (SHA-256 `b00f850e338007caea6af76f4e9305ab6b54a68e6799efd450bc76f1c325f331`).

12 Reuse Potential

Researchers can reuse the window layer to test uniform spin-2 rotation models against any workspace exposing the small NaMaster protocol, reuse the multipole contracts to prevent field/bin support drift, or use the receipt layer for sharded JSON calculations outside cosmology. The receipt primitive remains in the same package because it authenticates the exact validation artifacts produced by the window workflows; it is a small reusable module rather than a claim that the package is a general provenance framework. Extensions can add other angle-dependent spectrum bases, signed receipt anchors, or schema migrations. Questions, bug reports, and contributions are handled through the public repository issue tracker and pull-request workflow.

13 AI Usage Disclosure

AI coding and review agents assisted with implementation review, test generation, documentation, and manuscript editing. Their outputs were checked against committed source and retained numerical

artifacts; executable claims were accepted only after local tests or direct recomputation.

Acknowledgements

The exact-window operations build on NaMaster [2] and the MASTER pseudo-C_ℓ method [1]. The production example used CAMB [3] and HEALPix [4].

Funding Statement. No external funding supported this software release.

Competing Interests. The author declares no competing interests.

References

[1] E. Hivon, K. M. Górski, C. B. Netterfield, B. P. Crill, S. Prunet, and F. Hansen, "MASTER of the CMB anisotropy power spectrum: A fast method for statistical analysis of large and complex cosmic microwave background data sets," *Astrophysical Journal* 567, 2 (2002), doi:10.1086/338126.

[2] D. Alonso, J. Sanchez, and A. Slosar, "A unified pseudo-C_ℓ framework," *Monthly Notices of the Royal Astronomical Society* 484, 4127–4151 (2019), doi:10.1093/mnras/stz093.

[3] A. Lewis, A. Challinor, and A. Lasenby, "Efficient computation of cosmic microwave background anisotropies in closed Friedmann–Robertson–Walker models," *Astrophysical Journal* 538, 473–476 (2000), doi:10.1086/309179.

[4] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann, "HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere," *Astrophysical Journal* 622, 759–771 (2005), doi:10.1086/427976.