SalsaSpectrum (Python Edition)

An enhanced implementation of the SALSA reduction pipeline with rigorous error propagation and physical modeling

Tiago H. F. Baroni

January 20, 2026

Source code repository:
https://github.com/tiagobaroni/SalsaSpectrum

Contents

1 Introduction

The *SalsaSpectrum (Python Edition)* is a robust scientific software library designed for the reduction, calibration, and kinematic analysis of 21-cm spectral line data. Originally developed for the SALSA (Such a Lovely Small Antenna) radio telescopes at the Onsala Space Observatory, this modern implementation represents a complete architectural overhaul of the legacy MATLAB tools. The transition to a pure Python ecosystem ensures reproducibility, modularity, and adherence to open scientific standards.

1.1 Scientific Context

The SALSA telescopes are 2.3 m antennae optimized for detecting the hyperfine transition of neutral hydrogen (HI) at 1420.4 MHz. Observing this spectral line enables the derivation of galactic rotation curves and the mapping of large-scale kinematic structures. However, raw data acquired from these instruments require rigorous processing to account for instrumental noise, standing waves, and calibration uncertainties.

SalsaSpectrum provides a computational framework that transforms raw FITS data into scientifically calibrated brightness temperature spectra, expressed in terms of the main-beam temperature T_{mb}.

1.2 Core Capabilities and Improvements

Unlike simple visualization or exploratory scripts, SalsaSpectrum operates as a strict and auditable reduction pipeline, structured around four core methodological pillars:

Physical Rigor and Calibration The library implements the complete Radiometer Equation to compute theoretical noise limits (σ_{theo}), explicitly accounting for integration time (t_{int}), number of polarization modes (n_{pol}), and effective noise bandwidth (B_{eff}). In addition, the software enforces strict traceability of physical constants (e.g., R_0, V_0, ν_{HI}) through a dedicated immutable registry, ensuring consistency across all kinematic derivations.

Advanced Baseline Modeling To mitigate instrumental standing waves (spectral ripples) commonly observed in radio data, the software introduces a Hybrid Baseline Model. This approach combines adaptive polynomial fitting with a robust non-linear sinusoidal estimator based on Levenberg–Marquardt optimization. The resulting model enables precise removal of periodic artifacts while preserving the integrity of the underlying astronomical signal.

Statistical Robustness Gaussian spectral fitting is validated using established information criteria, including the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), and reduced chi-squared statistics (χ^2_{red}). Crucially, the error propagation framework corrects for spectral smoothing operations (e.g., Hanning or Boxcar filters) by explicitly computing the Effective Degrees of Freedom (ν_{eff}), preventing systematic underestimation of parameter uncertainties.

Modern Architecture Built upon the `astropy` and `scipy` scientific ecosystems, the library ensures full compliance with the FITS World Coordinate System (WCS) standard, including accurate handling of velocity reference frames (e.g., LSR and TOPO). The software exposes a clean object-oriented API, supporting batch processing workflows, reproducible analyses, and automated testing.

This document describes the installation, usage, and theoretical foundations of the SalsaSpectrum library, serving as a practical guide for both novice observers and advanced researchers requiring audit-grade spectral analysis.

2 Original MATLAB Implementation

2.1 Context and Objectives

The original *SalsaSpectrum* MATLAB class, developed by Daniel Dahlin and Eskil Varenius, represents a foundational milestone in the data reduction pipeline for the SALSA (Such a Lovely Small Antenna) radio telescopes at the Onsala Space Observatory [1]. Prior to its release, the reduction of 21-cm spectral line data relied largely on ad-hoc scripts written by individual students or on the SalsaJ software package. While SalsaJ provided a graphical interface, it was primarily conceived as an image processing tool and proved inefficient and cumbersome for the systematic reduction of large datasets or multiple spectra.

Developed within the framework of the European Hands-On Universe (EU-HOU) project, the MATLAB implementation was designed with a clear pedagogical and scientific objective: to provide a unified, programmable environment enabling high school and university students to derive galactic kinematics in a reproducible and methodologically consistent manner. By standardizing the reduction workflow, Dahlin and Varenius addressed a significant operational gap at the observatory, delivering the first published and reusable codebase explicitly tailored to SALSA data.

The tool was architected as an object-oriented MATLAB class, allowing users to instantiate spectral objects directly from FITS files and apply modular methods for baseline subtraction and Gaussian line fitting. Its success stemmed from a careful balance between functionality and accessibility. The inclusion of a detailed "Cookbook" and interactive fitting modes significantly lowered the barrier to entry for novice astronomers, while preserving sufficient flexibility and rigor for meaningful scientific analysis.

The current Python-based modernization pays explicit tribute to this legacy. While the internal computational engine has been fully rewritten to comply with contemporary open-source practices, modern numerical libraries, and rigorous physical error modeling, the logical workflow originally established by Dahlin and Varenius remains the guiding blueprint for the user experience and overall pipeline structure.

2.2 Limitations and Rationale for Migration

While the original MATLAB implementation successfully fulfilled its educational objectives within the Onsala Space Observatory ecosystem, the rapid evolution of data science practices and

open astronomy over the past decade has exposed a number of structural constraints inherent to that environment. The decision to port and modernize the library was therefore driven by several complementary factors, aimed at expanding accessibility while simultaneously strengthening scientific rigor and long-term sustainability.

2.2.1 Licensing and Accessibility

The most significant limitation of the legacy version is its dependency on the proprietary MATLAB computing environment. Although MATLAB remains a powerful and widely adopted industry standard, its licensing model introduces a financial barrier that effectively restricts access to affiliated academics and institutions with active subscriptions. By migrating to Python, the SalsaSpectrum library removes these licensing constraints and fully embraces the open-source scientific ecosystem, notably `NumPy`, `SciPy`, and `Astropy`. This transition democratizes access to the pipeline, enabling independent researchers, amateur radio astronomers, and students from resource-limited institutions to deploy and use the software without administrative or financial friction.

2.2.2 Architectural Scalability

The original implementation was encapsulated within a single MATLAB `.m` file containing both class definitions and functional logic. While this monolithic structure facilitated simple distribution through direct file copying, it imposed increasing challenges for maintenance, version control, and systematic testing as the codebase evolved. The modern Python implementation adopts a modular architecture, explicitly separating physical constants, mathematical utilities, and core processing logic. This separation of concerns improves code readability, supports collaborative development workflows, enables continuous integration (CI), and allows complex algorithms to be isolated and validated independently.

2.2.3 Standardization and Interoperability

Since the release of the original MATLAB version, the astronomical community has largely converged on Python, with the `astropy` project emerging as the de facto standard for data handling and coordinate transformations. The legacy implementation relied on internally defined mechanisms for parsing FITS headers and managing spectral coordinates. The Python migration ensures native compliance with modern World Coordinate System (WCS) standards and seamless interoperability with the broader astronomical software ecosystem, guaranteeing that spectra reduced with SalsaSpectrum can be readily ingested by downstream analysis and visualization tools.

2.2.4 Statistical Error Propagation

In the original MATLAB class, noise levels (RMS) and parameter uncertainties were estimated empirically under the assumption that spectral channels represent statistically independent data points. In practice, radio astronomical spectra are frequently subjected to smoothing

operations, such as Hanning or Boxcar filtering, which introduce correlations between adjacent channels. Treating these correlated channels as independent leads to systematic underestimation of parameter uncertainties.

The Python implementation addresses this limitation by explicitly introducing the concept of Effective Degrees of Freedom (ν_{eff}). This approach adjusts the statistical weight of the data during spectral fitting according to the applied smoothing width, resulting in more conservative and physically meaningful uncertainty estimates for the Gaussian line parameters.

2.2.5 Physical Modeling of Baselines

Baseline subtraction in the legacy implementation relied exclusively on polynomial fitting. While effective for removing continuum offsets and large-scale gradients, the original user documentation recommended limiting the polynomial order to values below nine as a practical rule of thumb. This strategy proves insufficient for accurately modeling standing waves—sinusoidal ripples arising from internal signal reflections within the telescope structure—and often forces the use of high-order polynomials that risk overfitting and distorting the underlying spectral lines.

The modern library introduces a Hybrid Baseline Model capable of explicitly fitting and removing sinusoidal components associated with instrumental standing waves. This approach enables the use of lower-order polynomials while preserving the integrity of the astronomical signal and providing a physically motivated treatment of instrumental artifacts.

2.2.6 Radiometric Calibration Standards

In the original MATLAB version, noise estimation was performed purely through observational methods, typically by measuring the RMS in line-free regions of the spectrum. While practical, this approach does not allow for a direct comparison between measured noise levels and the theoretical performance limits of the telescope system.

The updated implementation incorporates the full Radiometer Equation, enabling calculation of the theoretical thermal noise limit (σ_{theo}) based on physical parameters such as integration time, effective bandwidth, and number of polarization modes. This addition allows users to audit observation quality by identifying spectra for which the measured noise deviates significantly from theoretical expectations.

2.3 Python Porting Strategy

The migration from MATLAB to Python was executed using a refactoring-based strategy rather than a direct line-by-line translation. The primary objective was to preserve the functional logic and conceptual workflow of the original *SalsaSpectrum* class while restructuring the codebase to comply with modern software engineering practices, including adherence to PEP 8 conventions and established best practices within the Python scientific ecosystem.

The following strategic decisions guided the development process.

2.3.1 Modular Decomposition

The original MATLAB implementation concentrated all class definitions, helper functions, and physical constants within a single `.m` file. In contrast, the Python port decomposes these responsibilities into a cohesive package structure designed to improve maintainability, extensibility, and testability:

- **Core Logic (`spectrum.py`)**: Encapsulates the `SalsaSpectrum` class, responsible for managing internal state, handling input/output operations, and orchestrating high-level reduction workflows.
- **Mathematical Utilities (`utils.py`)**: Isolates pure numerical and signal-processing functions, such as sinusoidal baseline fitting and effective degrees of freedom calculations. This separation allows complex algorithms to be unit-tested independently of telescope-specific data.
- **Physical Constants (`const_physical.py`)**: Establishes a single source of truth for immutable physical parameters (e.g., c, ν_{HI}, Galactic constants), preventing the proliferation of hard-coded values throughout the codebase.

2.3.2 Explicit Type Safety and Immutability

To reduce the likelihood of runtime errors commonly associated with dynamically typed languages, the Python implementation makes extensive use of the `typing` module. Function signatures are explicitly annotated (e.g., `def fit_baseline(..., mixed_model: bool) -> None`), enabling static analysis tools and improving integrated development environment (IDE) support.

In addition, physical constants are declared using `typing.Final`, enforcing immutability at the semantic level and preventing accidental reassignment of critical values during execution.

2.3.3 Memory Optimization

Spectral reduction workflows frequently involve processing large datacubes or batch-processing hundreds of FITS files, making memory efficiency a critical design consideration. To address this, the `SalsaSpectrum` class employs the `__slots__` mechanism. By explicitly declaring permissible data attributes (e.g., `data`, `vel`, `residuals`), the implementation avoids the overhead associated with the per-instance `__dict__` attribute, significantly reducing memory usage when handling large collections of spectral objects.

2.3.4 Integration with the Scientific Ecosystem

Rather than re-implementing well-established numerical algorithms, the Python port deliberately delegates complex operations to the mature and extensively validated scientific Python stack:

- **FITS Handling**: Custom parsing logic is replaced by `astropy.io.fits`, ensuring robust support for standard FITS headers, accurate World Coordinate System (WCS) handling, and reliable coordinate transformations.
- **Optimization**: Both Gaussian spectral decomposition and the hybrid sinusoidal baseline model rely on `scipy.optimize.curve_fit`, which implements the Levenberg–Marquardt algorithm and provides superior convergence properties compared to simple polynomial fitting approaches.
- **Signal Processing**: Automatic line detection and preprocessing tasks leverage `scipy.signal.find_peaks` and `scipy.signal.medfilt`, replacing ad-hoc threshold-based methods with statistically robust signal-processing techniques.

3 Mathematical Formulation of New Features

This section presents the mathematical models and statistical frameworks introduced in the *SalsaSpectrum (Python Edition)*. The focus is strictly on the methodological improvements relative to the legacy implementation, particularly in radiometric calibration, mitigation of standing waves, and rigorous error propagation in the presence of spectral smoothing.

3.1 Radiometric Calibration and Theoretical Noise

Unlike the legacy implementation, which relied exclusively on empirical RMS estimates derived from signal-free spectral channels, the modern library implements the full Radiometer Equation. This approach enables the computation of the theoretical thermal noise limit (σ_{theo}), providing a physically grounded benchmark for assessing receiver performance and observing conditions.

The theoretical noise level is computed as:

$$\sigma_{\text{theo}} = \frac{T_{\text{sys}}}{\sqrt{n_{\text{pol}} \cdot \Delta\nu \cdot \eta_{\text{bw}} \cdot \tau}} \tag{3.1}$$

where:

- T_{sys} is the system temperature in Kelvin (K). The software prioritizes values read directly from the FITS header; if unavailable, a user-provided estimate is used.
- n_{pol} is the number of polarization modes (typically $n_{\text{pol}} = 2$ for unpolarized sources observed with dual-polarization feeds).
- $\Delta\nu$ is the channel width in Hertz (Hz), derived from the frequency axis defined by the WCS.
- η_{bw} is the effective noise bandwidth correction factor, accounting for the spectral windowing function applied by the spectrometer (e.g., Hanning smoothing reduces the number of statistically independent samples).

- τ is the integration time in seconds (s).

This formulation allows users to systematically flag observations for which the measured RMS significantly exceeds σ_{theo}, indicating potential radio frequency interference (RFI) or instrumental gain instability.

3.2 Hybrid Baseline Modeling (Standing Wave Removal)

Standing waves, often referred to as spectral "ripples", arise from signal reflections between the feed horn and the dish surface and represent a common artifact in single-dish radio observations. Standard polynomial fitting techniques frequently fail to remove these sinusoidal structures without resorting to high polynomial orders ($n > 9$), which risks overfitting and attenuation of the astronomical signal.

To address this limitation, SalsaSpectrum introduces a Hybrid Baseline Model, denoted M_{hybrid}, which decomposes the baseline into a smooth polynomial continuum and an explicit sinusoidal component:

$$M_{\text{hybrid}}(x) = \sum_{i=0}^{k} c_i x^i + A \cdot \sin(2\pi f x + \phi) \tag{3.2}$$

The fitting procedure is executed in three sequential and well-defined stages:

Polynomial Detrending A low-order polynomial fit (typically first or second order) is applied to predefined baseline windows to remove large-scale continuum slopes.

Frequency Estimation (FFT) The residuals from the polynomial detrending step are analyzed using a Fast Fourier Transform (FFT) over the largest contiguous baseline segment. The dominant peak in the resulting power spectrum provides an initial estimate of the ripple frequency, denoted f_{guess}.

Non-Linear Optimization Using f_{guess} as a prior, a non-linear least-squares optimization problem is solved via the Levenberg–Marquardt algorithm. This step simultaneously refines the amplitude (A), frequency (f), and phase (ϕ) of the standing wave. The optimized sinusoidal component is then combined with the polynomial baseline.

This approach mathematically decouples instrumental ripple structures from the galactic HI emission, preserving line flux that would otherwise be absorbed by high-order polynomial fits.

3.3 Statistical Error Propagation

Spectral data are frequently subjected to smoothing operations, such as Boxcar or Hanning filtering, in order to improve the signal-to-noise ratio (SNR). While effective, these operations introduce correlations between adjacent spectral channels, violating the assumption of statistical independence underlying standard χ^2 minimization.

To correct for this effect, SalsaSpectrum explicitly implements the concept of Effective Degrees of Freedom (ν_{eff}). The effective number of independent samples (N_{eff}) is approximated by scaling

the total number of spectral channels (N) by the smoothing kernel width (W_{smooth}):

$$N_{\text{eff}} \approx \frac{N}{W_{\text{smooth}}} \tag{3.3}$$

The effective degrees of freedom used for statistical inference are then given by:

$$\nu_{\text{eff}} = N_{\text{eff}} - k \tag{3.4}$$

where k denotes the number of free parameters in the model (e.g., amplitude, centroid, and width for each Gaussian component). This correction is essential for obtaining unbiased estimates of parameter uncertainties and fit quality metrics.

3.4 Fit Quality Metrics

To provide an objective and statistically consistent assessment of the Gaussian decomposition, the library computes standardized goodness-of-fit and model-selection metrics, explicitly corrected for the effective degrees of freedom defined above.

3.4.1 Reduced Chi-Squared (χ^2_{red})

The reduced chi-squared statistic quantifies the goodness of fit relative to the noise variance σ^2_{rms}:

$$\chi^2_{\text{red}} = \frac{1}{\nu_{\text{eff}}} \sum_{i=1}^{N} \frac{(y_i - M(y_i))^2}{\sigma^2_{\text{rms}}} \tag{3.5}$$

A value of $\chi^2_{\text{red}} \approx 1$ indicates that the residuals are consistent with thermal noise. Values significantly greater than unity suggest an under-fitted model (e.g., missing spectral components), whereas values below unity may indicate overfitting or overestimated noise.

3.4.2 Akaike Information Criterion (AIC)

To support objective model selection—for example, determining whether the inclusion of an additional Gaussian component is statistically justified—the library computes the Akaike Information Criterion (AIC). The implementation is based on the residual sum of squares (RSS) and explicitly corrects for the effective sample size:

$$\text{AIC} = N_{\text{eff}} \cdot \ln\left(\frac{RSS}{N_{\text{eff}}}\right) + 2k \tag{3.6}$$

By penalizing model complexity through the $2k$ term, the AIC discourages the inclusion of spurious components that fit noise rather than physically meaningful HI structures.

4 Tests and Validation

To validate the functional integrity of the ported library and the robustness of the newly introduced mathematical models, a comprehensive automated test suite was developed

(`tests/test_workflow_generator.py`). Unlike the manual, step-by-step verification process required by the original MATLAB implementation, this suite performs a full end-to-end integration test, processing raw FITS files through the complete reduction pipeline without user intervention.

4.1 Automated Workflow Methodology

The validation workflow employs a lightweight graphical interface based on `Tkinter`, allowing batch selection of input directories and ensuring that multiple FITS files can be processed iteratively in an unattended manner. For each detected file, the pipeline executes the following sequence:

- **Ingestion and WCS Parsing**: Validation of FITS header integrity and inference of the appropriate velocity reference frame (LSR or TOPO).
- **Radiometric Calibration**: Computation of the theoretical noise level σ_{theo} using the Radiometer Equation, based on the integration time ($\tau = 60\,\text{s}$) and system temperature ($T_{\text{sys}} \approx 100\,\text{K}$) extracted directly from the FITS headers.
- **Comparative Baseline Fitting**: Application of both a standard polynomial baseline model (second order) and the new Hybrid Mixed Model (Polynomial + Sine) to identical baseline windows.
- **Blind Gaussian Fitting**: Automatic peak detection followed by Gaussian decomposition, with fit quality metrics (AIC and χ^2_{red}) logged for each spectrum.

4.2 Baseline Model Comparison

A critical improvement introduced in the Python edition concerns the handling of instrumental standing waves. The automated test suite generates comparative diagnostic plots to visualize the performance difference between the legacy polynomial-only approach and the new hybrid baseline model.

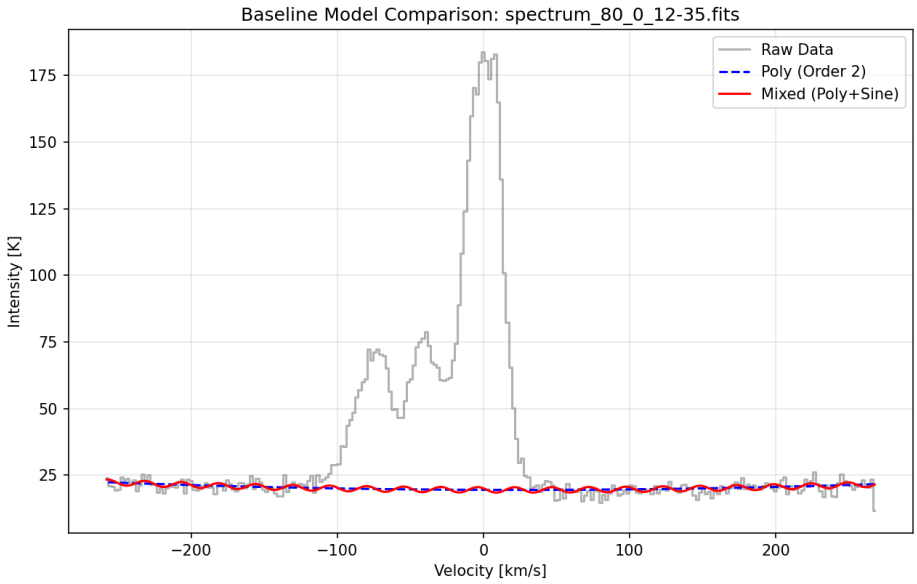

Figure 1: Comparison of baseline models. The dashed blue curve represents the standard second-order polynomial baseline, while the solid red curve shows the Hybrid Mixed Model (Polynomial + Sine). The hybrid model captures subtle instrumental ripples without requiring high-order polynomials that risk overfitting the astronomical signal.

In the example shown in Figure 1, the software automatically identifies baseline windows (shaded green regions). The RMS values associated with each baseline model are computed and logged, enabling the user to quantitatively assess whether inclusion of the sinusoidal component reduces the residual noise floor.

4.3 Statistical Fit and Residual Analysis

The final validation step involves Gaussian decomposition of the calibrated spectral line. The Python implementation introduces a rigorous statistical assessment of the fit, summarized in the final analysis artifacts produced by the pipeline.

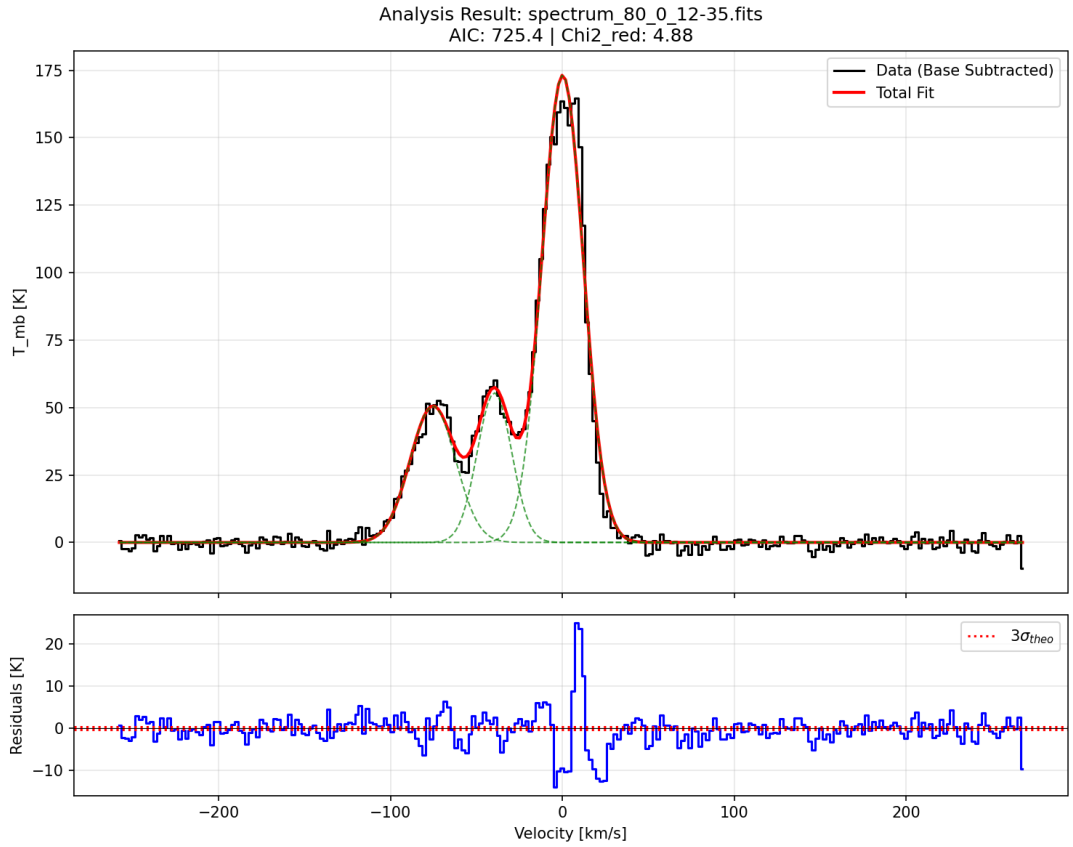

Figure 2: Final reduction analysis. **Top panel**: Baseline-subtracted data (black), overlaid with the total Gaussian fit (red) and individual Gaussian components (green dashed). **Bottom panel**: Spectral residuals (blue) compared against the theoretical $\pm 3\sigma_{\text{theo}}$ noise envelope (red dotted) derived from the Radiometer Equation.

The results highlight two validation capabilities absent in the legacy implementation:

Fit Quality Metrics The plot header reports the Akaike Information Criterion (AIC) and the reduced chi-squared statistic (χ^2_{red}). In the example shown in Figure 2, a value of $\chi^2_{\text{red}} > 1$ indicates that the residuals retain statistically significant structure not fully captured by the Gaussian model, such as the high-velocity feature visible in the residual spectrum.

Physical Residual Auditing The lower panel includes a reference line corresponding to the theoretical $\pm 3\sigma_{\text{theo}}$ threshold. This diagnostic allows immediate verification of whether the observation reached the expected thermal sensitivity limit of the telescope. In Figure 2, residual excursions beyond this limit suggest the presence of radio frequency interference (RFI) or unresolved kinematic substructure.

4.4 Comparison with Legacy Implementation

A concise comparison between the original MATLAB implementation and the modern Python edition is summarized in Table 1.

Table 1: Comparison between the legacy MATLAB implementation and the SalsaSpectrum Python edition.

Feature	**Original MATLAB Implementation**	**SalsaSpectrum (Python Edition)**
Workflow	Manual step-by-step execution	Automated batch processing via object-oriented API
Baseline Modeling	Polynomial only (visual inspection)	Hybrid (Polynomial + Sine) with algorithmic ripple detection
Noise Estimation	Empirical RMS (signal-free regions)	Radiometer Equation (σ_{theo}) combined with robust empirical RMS
Error Propagation	Assumes statistically independent channels	Corrects for spectral smoothing via ν_{eff}
Validation Criteria	Visual assessment of fit quality	Quantitative AIC and χ^2_{red} metrics

5 Conclusion and Outlook

The modernization of the *SalsaSpectrum* library represents more than a linguistic translation from MATLAB to Python; it constitutes a fundamental re-engineering of the data reduction pipeline to meet the rigor expected in contemporary radio astronomy. By transitioning to an open-source ecosystem, the project not only removes licensing barriers but also integrates the SALSA telescopes into the broader `astropy`-based scientific Python community.

5.1 Summary of Technical Enhancements

The development of the Python edition was guided by the need to move from predominantly qualitative inspection toward quantitatively robust, physically grounded analysis. The principal technical milestones achieved include:

- **Traceable Radiometric Calibration**: The implementation of the Radiometer Equation enables computation of the theoretical noise limit (σ_{theo}), allowing researchers to directly assess observational sensitivity against physical constraints such as integration time and effective bandwidth. This capability was absent in the purely empirical noise estimation approach of the legacy implementation.
- **Robust Artifact Mitigation**: The introduction of the Hybrid Baseline Model (Polynomial + Sine) addresses the persistent issue of instrumental standing waves. Validated through non-linear least-squares optimization, this method preserves astronomical signal integrity in cases where high-order polynomial fitting would otherwise fail or lead to overfitting.

- **Statistical Validity**: Explicit correction for the Effective Degrees of Freedom (ν_{eff}) ensures that parameter uncertainties and fit quality metrics, including the Akaike Information Criterion (AIC) and reduced chi-squared (χ^2_{red}), properly account for channel correlations introduced by spectral smoothing (e.g., Hanning windows). This prevents the over-interpretation of noise fluctuations as physical signal.

5.2 Future Directions

With its core architecture now modular, explicitly typed, and object-oriented, SalsaSpectrum is well positioned for future expansion without the architectural constraints imposed by the original monolithic MATLAB script. Promising avenues for continued development include:

- **Batch Pipeline Integration**: The memory-optimized class design based on the `__slots__` mechanism supports seamless integration into large-scale, automated survey pipelines for galactic mapping and statistical studies.
- **Virtual Observatory (VO) Support**: Future releases may incorporate Virtual Observatory–compliant interfaces for direct querying, exchange, and publication of reduced spectra, leveraging the strict World Coordinate System (WCS) compliance already implemented in the library.

In conclusion, while the logical workflow originally established by Dahlin and Varenius remains the pedagogical and conceptual reference standard, the Python implementation presented here delivers the computational engine required for audit-grade radio astronomy. It empowers students and researchers alike to perform data reduction that is not only accessible and educational, but also physically rigorous and statistically defensible.

A API Reference

This appendix provides a comprehensive reference for the public methods exposed by the `SalsaSpectrum` class. Each entry documents the method purpose, input parameters, data types, and functional origin, distinguishing between features ported from the legacy MATLAB implementation and new capabilities introduced in the Python edition.

A.1 Initialization and Data Loading

A.1.1 `__init__(filename)`

Description: Constructor method. Initializes the object, loads the specified FITS file, parses the header metadata, and establishes the World Coordinate System (WCS) for frequency and velocity axes.

Origin: Legacy Ported (refactored using `astropy`).

Parameters:

- `filename` (`str`) [Mandatory]: Path to the FITS file to be loaded.

A.1.2 `get_keyword(key, default)`

Description: Safely retrieves a metadata value from the FITS header. If the requested keyword is not present, a user-defined default value is returned.

Origin: Legacy Ported.

Parameters:

- `key` (`str`) [Mandatory]: FITS header keyword to retrieve (e.g., `TSYS`, `NAXIS1`).
- `default` (`Any`) [Optional, default=`None`]: Value returned if the keyword is not found.

A.2 Calibration and Physics

A.2.1 `calc_theoretical_noise(t_sys, integ_time, n_pol, enbw_factor)`

Description: Computes the theoretical thermal noise limit (σ_{theo}) using the Radiometer Equation. This value provides a physical benchmark for auditing observational sensitivity and receiver performance.

Origin: New Implementation.

Parameters:

- `t_sys` (`float`) [Mandatory]: System temperature in Kelvin (K).
- `integ_time` (`float`) [Mandatory]: Integration time in seconds (s).
- `n_pol` (`int`) [Optional, default=1]: Number of polarization modes (typically 1 or 2).
- `enbw_factor` (`float`) [Optional, default=1.0]: Effective Noise Bandwidth correction factor accounting for spectral windowing.

A.2.2 `apply_beam_efficiency(eta_mb)`

Description: Converts the intensity scale from antenna temperature (T_A) to main-beam brightness temperature (T_{mb}) and rescales all associated quantities, including RMS, baseline models, and σ_{theo}.

Origin: New Implementation.

Parameters:

- `eta_mb` (`float`) [Mandatory]: Main-beam efficiency factor, with $0 < \eta_{\text{mb}} \leq 1$.

A.3 Visualization

A.3.1 `plot(type_, show_individual)`

Description: Generates a visualization of the current spectrum, with support for multiple coordinate systems and optional overlay of fitted Gaussian components.

Origin: Legacy Ported.

Parameters:

- `type_` (`str`) [Optional, default=`'vel'`]: X-axis coordinate system. Supported values are `'vel'` (velocity), `'freq'` (frequency), and `'pix'` (channel index).
- `show_individual` (`bool`) [Optional, default=`False`]: If `True`, plots individual Gaussian components alongside the total fit.

A.3.2 `show_baseline()`

Description: Displays the raw spectrum together with the fitted baseline model (polynomial or hybrid) and the active baseline windows.

Origin: Legacy Ported.

Parameters: None.

A.3.3 `show_lab()`

Description: Overlays comparison data from the Leiden/Argentine/Bonn (LAB) HI survey onto the current spectrum.

Origin: Legacy Ported.

Parameters: None.

A.4 Baseline Operations

A.4.1 `fit_baseline(windows, coord, order, criterion, mixed_model, interactive)`

Description: Fits a baseline model to signal-free regions of the spectrum. Supports standard polynomial baselines and the hybrid polynomial-plus-sinusoid model for standing wave removal.

Origin: Legacy Ported (enhanced with `mixed_model` and information-criterion support).

Parameters:

- `windows` (`List[float]`) [Optional]: Flat list of start/end points defining baseline regions (e.g., `[-200, -100, 100, 200]`). Mandatory if `interactive=False`.
- `coord` (`str`) [Optional, default=`'vel'`]: Coordinate system for window definitions (`'vel'`, `'freq'`, `'pix'`).
- `order` (`int` | `str`) [Optional, default=1]: Polynomial order or `'auto'` for automatic selection using AIC or BIC.
- `criterion` (`str`) [Optional, default=`'BIC'`]: Information criterion used for automatic order selection.
- `mixed_model` (`bool`) [Optional, default=`False`]: Enables the hybrid polynomial-plus-sine baseline model.
- `interactive` (`bool`) [Optional, default=`False`]: Enables interactive window selection via mouse input.

A.4.2 `subtract_baseline()`

Description: Subtracts the currently fitted baseline model from the spectral data.

Origin: Legacy Ported.

Parameters: None.

A.4.3 `reset_baseline()`

Description: Restores the original spectrum by adding the previously subtracted baseline back to the data.

Origin: Legacy Ported.

Parameters: None.

A.5 Analysis and Fitting

A.5.1 `fit_gaussians(guesses, smooth_width)`

Description: Performs non-linear least-squares fitting of Gaussian components. Computes statistical quality metrics, including the Akaike Information Criterion (AIC) and reduced chi-squared (χ^2_{red}), with correction for correlated spectral channels.

Origin: Legacy Ported (enhanced with statistical metrics and effective degrees of freedom correction).

Parameters:

- `guesses` (`List[float]`) [Optional, default=`None`]: Initial parameter guesses provided as a flat list (`[Amp`$_1$`, Cen`$_1$`, Wid`$_1$`, Amp`$_2$`, ...]`). If `None`, a blind peak-detection procedure is applied.
- `smooth_width` (`int`) [Optional, default=1]: Width of the smoothing kernel (in pixels), used to compute effective degrees of freedom.

A.5.2 `smooth_spectrum(factor)`

Description: Applies boxcar smoothing to the spectral data to improve the signal-to-noise ratio.

Origin: Legacy Ported.

Parameters:

- `factor` (`int`) [Optional, default=2]: Number of adjacent channels to average.

A.5.3 `read_lab(resolution)`

Description: Downloads comparison data from the Leiden/Argentine/Bonn (LAB) HI survey based on the sky coordinates contained in the FITS header.

Origin: Legacy Ported.

Parameters:

- `resolution` (`float`) [Optional, default=`None`]: Desired angular resolution for convolution (currently unused in the Python implementation).

A.6 Utilities

A.6.1 `get_indices(val, coord)`

Description: Converts physical values expressed in velocity or frequency units into the nearest spectral channel indices (pixels).

Origin: Legacy Ported.

Parameters:

- `val` (`List[float]` | `np.ndarray`) [Mandatory]: Values to convert.
- `coord` (`str`) [Mandatory]: Source coordinate system (`'vel'`, `'freq'`, `'pix'`).

References

[1] Daniel Dahlin and Eskil Varenius. Salsaspectrum: Reducing data from salsa in matlab. Technical report, Onsala Space Observatory, 2015. URL https://github.com/varenius/salsa/tree/main/SalsaSpectrum/Documentation.