SCOP: An Open-Source JAX-Powered Framework for Generic Photogrammetric Bundle Adjustment

Adrien Gressin[1]

[1]University of Applied Sciences Western Switzerland (HES-SO / HEIG-VD),
Yverdon-les-Bains, Switzerland - adrien.gressin@heig-vd.ch

Keywords: Bundle Adjustment, Photogrammetry, Automatic Differentiation, Sparse Optimization, Python Framework

Abstract

We present SCOP, an open-source Python framework leveraging JAX automatic differentiation for generic photogrammetric bundle adjustment. Unlike traditional implementations relying on manually derived or symbolically precomputed Jacobians, SCOP computes analytically exact derivatives automatically for any camera model, enabling rapid prototyping of new sensor types. The framework implements sparse Jacobian assembly, Schur complement elimination, and Levenberg-Marquardt optimization with step acceptance control, tested on networks with 181k observations and 79k unknowns. We validate SCOP on both synthetic unmanned aerial vehicle (UAV) surveys and the real-world Roma dataset (60 images, 26k tie points), achieving $\sigma_0 = 0.575$ px (identical to DBAT) while providing full statistical quality assessment (w_i, z_i) in less than 1 s per iteration. The educational design enables students to inspect bundle adjustment internals without manual derivative derivation.

1. Introduction

Bundle adjustment is the cornerstone of photogrammetric reconstruction, simultaneously refining camera poses, 3D point positions, and calibration parameters by minimizing reprojection errors (Triggs et al., 2000, Hartley and Zisserman, 2004). Traditional implementations such as COLMAP (Schönberger and Frahm, 2016), MicMac/MMVII (Rupnik et al., 2017, Pierrot-Deseilligny, 2024), or the Ceres Solver (Agarwal et al., 2024) rely on manually derived or symbolically generated Jacobian expressions for each camera model, increasing development effort and the risk of implementation errors. DBAT (Börlin and Grussenmeyer, 2016) provides an open-source educational bundle adjustment (BA) tool in MATLAB, enabling external verification of commercial software results, but relies on manual Jacobian derivation and MATLAB licensing (it also runs on GNU Octave, albeit with reduced performance).

Automatic differentiation (AD) offers an alternative: projection functions are defined as pure mathematical expressions, and exact derivatives are computed mechanically by the AD framework, without finite differences or symbolic manipulation. Recent work has demonstrated the viability of this approach: Zhan et al. (Zhan et al., 2024) implement BA in PyTorch with GPU acceleration, achieving significant speedups on large datasets. This paradigm shift enables camera-model-agnostic bundle adjustment where adding a new sensor requires only defining its projection function.

Beyond computational aspects, bundle adjustment is a central topic in photogrammetry education. Students need to understand the interplay between observation equations, least-squares estimation, and statistical quality control, yet existing tools are either black boxes (commercial software) or written in C++ (difficult to modify for non-experts), and typically lack statistical output beyond residuals. A transparent framework exposing the full adjustment pipeline, from Jacobian assembly to covariance propagation and reliability analysis, would bridge the gap between theory and practice.

We present SCOP (*Sparse Compensation and Optimization for Photogrammetry*), an open-source Python framework built on JAX (Bradbury et al., 2018) that combines:

- Automatic differentiation for exact, model-agnostic Jacobian computation;
- Sparse Jacobian assembly with Schur complement elimination for scalability;
- Robust Levenberg-Marquardt optimization with step acceptance/rejection;
- Full statistical quality assessment (σ_0, covariance, reliability indices).

This paper addresses three questions: (1) how does AD compare to classical Jacobian formulation in terms of correctness and flexibility, (2) can a Python framework compete with established C++ tools on real-world bundle adjustments, and (3) can such a framework serve as an educational platform for teaching bundle adjustment internals and statistical quality control.

2. Automatic Differentiation for Bundle Adjustment

2.1 Classical Approach: Manual Jacobians

In the classical formulation of bundle adjustment, the observation equations relate 2D image measurements $\mathbf{l}$ to 3D scene parameters $\mathbf{x}$ through a nonlinear projection function:

$$\mathbf{l} + \mathbf{v} = f(\mathbf{x}) \tag{1}$$

where $\mathbf{v}$ is the residual vector. Linearization around an approximate solution $\mathbf{x}_0$ yields:

$$\mathbf{v} = \mathbf{A}\,\delta\mathbf{x} - \mathbf{d}\ell, \quad \mathbf{A} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \tag{2}$$

The Jacobian matrix $\mathbf{A}$ must be derived for each camera model (pinhole, fisheye, equirectangular) and each distortion parametrization (Brown-Conrady (Brown, 1971, Conrady, 1919), Agisoft (Agisoft, 2023), OpenCV (Zhang, 2000), etc.), leading to dozens of model-specific derivative expressions. Any error in these derivations can cause subtle convergence failures that are difficult to diagnose.

2.2 Automatic Differentiation with JAX

Automatic differentiation (AD) (Griewank and Walther, 2008, Baydin et al., 2018) computes exact derivatives by mechanically applying the chain rule to elementary operations. JAX (Bradbury et al., 2018) implements AD by tracing Python functions into an intermediate representation. Two modes are available:

- **Forward mode** (`jacfwd`): propagates tangent vectors alongside the primal computation. For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, each tangent direction requires one forward pass, so the full Jacobian costs n passes. Efficient when $n < m$ (few inputs, many outputs).
- **Reverse mode** (`jacrev`): propagates adjoint (cotangent) vectors backward. Each adjoint costs one reverse pass, so the full Jacobian costs m passes. Efficient when $m < n$ (few outputs, many inputs), as in deep learning.

Unlike finite differences, AD computes *exact* derivatives to machine precision, without the numerical instability of choosing a perturbation step ϵ. The computational cost for one block is n_{local} evaluations of the residual function, comparable to central finite differences ($2n_{\text{local}}$ evaluations) but without the $O(\epsilon^2)$ truncation error.

In photogrammetric bundle adjustment, each residual block depends on a small number of local variables ($n_{\text{local}} \approx 9$–$14$ for a tie point: 6 exterior orientation (EO) + 3 point coordinates + interior orientation (IO) parameters) and produces 2 observations ($m = 2$). With $m = 2$ outputs and $n_{\text{local}} \approx 14$ inputs, reverse mode requires only 2 passes versus 14 for forward mode. On the Roma dataset (2000+ points per image), `jacrev` is 2.9× faster than `jacfwd`, qualitatively consistent with the expected advantage of reverse mode (the gap with the theoretical 7× ratio reflects the higher per-pass overhead of reverse-mode intermediate storage). SCOP therefore uses reverse mode (`jacrev`) by default.

A concrete example illustrates the composability advantage. In SCOP, the collinearity equation for a tie point composes:

1. A rotation $\mathbf{R}(\omega, \varphi, \kappa)$ (Euler angles or Lie algebra),
2. A 3D-to-camera projection $\mathbf{x}_c = \mathbf{R}(\mathbf{X} - \mathbf{X}_0)$,
3. A perspective division and focal scaling,
4. A radial/decentering distortion model.

Each step is a standard Python function. JAX traces the full chain and differentiates through all four steps automatically. Changing the rotation convention, adding a new distortion model, or introducing a fisheye projection requires only modifying the corresponding Python function, with no re-derivation of Jacobian expressions.

For computational efficiency, SCOP batches all tie points of one image into a single vectorized call using `jax.vmap`. This dispatches one JIT (Just-In-Time)-compiled kernel per image rather than one per point, reducing Python overhead and enabling SIMD parallelism. Since images may have different numbers of tie points, input arrays are padded to a uniform size per camera model, ensuring a single JIT-compiled kernel can be reused across all images through a camera-level cache (only one compilation per camera model).

Key advantages of this approach:

1. **Correctness**: derivatives are exact to machine precision, eliminating manual derivation errors;
2. **Extensibility**: a new observation type (GNSS position, distance, angle, IMU) requires only its prediction function $f(\mathbf{x})$;
3. **Hardware portability**: JAX compiles to XLA (Accelerated Linear Algebra) kernels for CPU, GPU, or TPU without code changes;
4. **Composability**: models can be freely combined without re-deriving coupled Jacobians.

2.3 Comparison of Jacobian Computation Strategies

Four main strategies exist for computing the Jacobian matrix in bundle adjustment (Table 1):

1. **Manual derivation**: closed-form expressions handcrafted for each camera model. Used by DBAT (Börlin and Grussenmeyer, 2016) and internally by COLMAP via Ceres (Agarwal et al., 2024). This yields optimal runtime performance but is error-prone and requires re-derivation for every new model or distortion parametrization.
2. **Finite differences**: approximating $\partial f/\partial x_k \approx (f(x + \epsilon_k) - f(x - \epsilon_k))/2\epsilon$. Simple to implement but numerically unstable (sensitive to the choice of ϵ) and slow ($2n$ function evaluations for the full Jacobian).
3. **Automatic differentiation (AD)**: propagating exact derivatives through operator overloading or source transformation. SCOP uses JAX reverse-mode AD (`jacrev`); Ceres also supports AD internally. AD is exact to machine precision and model-agnostic, with a runtime overhead from JIT compilation and dispatch.
4. **Symbolic derivation**: building a directed acyclic graph (DAG) of the formula, applying differentiation rules ($\partial(F_1 \cdot F_2)/\partial x = F_2 \cdot \partial F_1/\partial x + F_1 \cdot \partial F_2/\partial x$), simplifying ($0 \cdot F \to 0$, $1 \cdot F \to F$, etc.), then generating compiled code. Used by MMVII (Pierrot-Deseilligny, 2024), this approach combines the correctness of AD with the performance of manual derivation, at the cost of a more complex build-time code generation step.

SCOP's choice of runtime AD via JAX prioritizes accessibility and rapid prototyping: adding a new observation type requires only a Python function, with no compilation step or C++ code generation. The trade-off is a per-iteration overhead from JIT tracing and dispatch. Interestingly, COLMAP 4 recently re-introduced hand-coded analytical Jacobians for its simplest

	Manual	Finite diff.	AD	Symbolic
Exact derivatives	Yes	No	Yes	Yes
Model-agnostic	No	Yes	Yes	Yes
Runtime overhead	None	High	Medium	None
New model effort	High	Low	Low	Low
Used by	DBAT	(fallback)	**SCOP**	MMVII

Table 1. Comparison of Jacobian computation strategies.

camera model, gaining ∼15% over Ceres' automatic Jets[1], illustrating that the speed/flexibility trade-off remains an active design choice even in mature C++ pipelines. In practice, `vmap` batching reduces this overhead to one dispatch per image (not per point), and JAX's persistent compilation cache eliminates JIT cost on subsequent runs.

3. Framework Architecture

3.1 Modular Observation Model

SCOP organizes the adjustment problem as a collection of typed observation objects registered in a dictionary structure:

- **Camera**: intrinsic parameters (focal length, principal point, distortion). Supported models include frame/pinhole, fisheye, and equirectangular. Six distortion parametrizations are available: Agisoft (Agisoft, 2023) (5 radial + 4 decentering + affinity), Brown-Conrady (Brown, 1971, Conrady, 1919) (5 radial + affinity), OpenCV standard (Zhang, 2000) (2 radial + 2 decentering) and full (rational model, 6 radial + 2 decentering), SimpleRadial (f, c_x, c_y, k_1), and Radial (f, c_x, c_y, k_1, k_2).
- **Image**: extrinsic parameters (position X_s, Y_s, Z_s, orientation as $\omega\varphi\kappa$ (OPK) Euler angles or SO(3) Lie algebra).
- **Point3D**: object coordinates, typed as ground control points (GCP, with known coordinates introduced as weighted observations in the adjustment), check points (known coordinates excluded from the adjustment, used for independent accuracy assessment), tie points (unknown coordinates estimated as free parameters), or fixed points (coordinates held constant, not estimated).
- **Point2D**: image measurements linking an Image to a Point3D, generating two observation equations each (column, row).

Each observation type defines a residual function $f(\mathbf{x}_{\text{local}})$ mapping local variables to predicted observations. The `ResidualBlock` architecture encapsulates the residual function together with references to the associated objects and their variable indices in the global state vector.

3.2 Optimizer Pipeline

The `Optimizer` class orchestrates the complete bundle adjustment workflow in four phases: problem assembly, Jacobian computation, iterative optimization, and statistical analysis.

Problem assembly. Each registered object (Camera, Image, Point3D) receives a global offset in the state vector $\mathbf{x}$ and observation vector $\mathbf{l}$. The sparsity pattern of $\mathbf{A}$ is pre-computed once and reused across iterations; only the numerical values are updated at each step.

[1] `https://github.com/colmap/colmap/pull/4017`

Batched Jacobian and residual computation. Rather than dispatching one JAX call per residual block (∼90k calls for Roma), SCOP groups all Point2D blocks sharing the same Image and applies `vmap(jacrev(f))` in a single dispatch per image. The resulting batched Jacobians, of shape $(N_{\text{pts}}, 2, n_{\text{local}})$, are scattered into the pre-allocated sparse arrays via NumPy advanced indexing. A Camera-level cache ensures that all images with the same variable layout share one compiled XLA kernel, reducing JIT compilation from $O(N_{\text{images}})$ to $O(N_{\text{layouts}})$ (typically 1).

Identity observation optimization. Scalar constraints of the form $r_i = l_i - x_j$ (e.g. GNSS position priors, control point coordinates) have a constant Jacobian entry of -1. SCOP detects these automatically during assembly and stores them as compact index arrays, bypassing both JAX tracing and `ResidualBlock` instantiation. On the Roma dataset, this eliminates 180 identity blocks (GNSS constraints), with their residuals and Jacobian entries computed via pure NumPy vectorization.

3.3 Sparse Assembly and Schur Complement

For large networks, the Jacobian $\mathbf{A} \in \mathbb{R}^{m\times n}$ is extremely sparse: each 2D observation depends only on the camera, image, and point parameters (∼20 variables out of tens of thousands). Exploiting this sparsity is essential for both memory efficiency and computational scalability. SCOP assembles the Jacobian in Compressed Sparse Row (CSR) format through the following steps:

1. Pre-computing the sparsity pattern (row/column indices) once at setup;
2. Evaluating local Jacobians per block via JIT-compiled AD;
3. Scattering values into pre-allocated CSR arrays.

The normal equations $(\mathbf{A}^T\mathbf{P}\mathbf{A})\,\delta\mathbf{x} = \mathbf{A}^T\mathbf{P}\,\mathbf{d}\ell$ are solved via Schur complement elimination (Triggs et al., 2000). Partitioning variables into camera/image parameters ($\mathbf{x}_c$) and point parameters ($\mathbf{x}_p$):

$$\begin{bmatrix}\mathbf{N}_{cc} & \mathbf{N}_{cp}\\ \mathbf{N}_{pc} & \mathbf{N}_{pp}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_c\\ \delta\mathbf{x}_p\end{bmatrix} = \begin{bmatrix}\mathbf{b}_c\\ \mathbf{b}_p\end{bmatrix} \tag{3}$$

where $\mathbf{N}_{pp}$ is block-diagonal (3×3 per point). The reduced system $\mathbf{S}\,\delta\mathbf{x}_c = \mathbf{b}_c - \mathbf{N}_{cp}\mathbf{N}_{pp}^{-1}\mathbf{b}_p$ has dimension equal to the number of camera/image parameters only, dramatically reducing computational cost.

3.4 Levenberg-Marquardt with Step Control

The Levenberg-Marquardt method adds diagonal damping to the normal equations:

$$(\mathbf{A}^T\mathbf{P}\mathbf{A} + \lambda\,\mathrm{diag}(\mathbf{A}^T\mathbf{P}\mathbf{A}))\,\delta\mathbf{x} = \mathbf{A}^T\mathbf{P}\,\mathbf{d}\ell \tag{4}$$

SCOP implements step acceptance/rejection: after solving for $\delta\mathbf{x}$, the trial cost is evaluated. If the cost decreases, the step is accepted and λ is reduced ($\times 0.5$). Otherwise, the step is rejected, λ is increased ($\times 10$), and the solve is retried with higher damping (up to 10 inner iterations). This prevents divergence when the initial linearization point is far from the optimum,

which is a common situation in large-scale photogrammetric networks initialized with approximate values.

Four solver methods are available:

- **GD** (Gradient Descent): first-order steepest descent, using only the gradient $\mathbf{A}^T\mathbf{P}\,\mathbf{d}\ell$. Very slow convergence but included for pedagogical comparison with second-order methods (Section 4).
- **GN** (Gauss-Newton): undamped normal equations, quadratic convergence near the optimum. Can diverge far from the minimum.
- **GNA** (Gauss-Newton-Armijo): GN step with Armijo backtracking line search. Combines quadratic convergence with guaranteed cost reduction, matching the convergence strategy used by DBAT (Börlin and Grussenmeyer, 2016). Convergence is detected via the relative cost change $|\Delta C/C| < \tau$ (default $\tau = 10^{-6}$).
- **LM** (Levenberg-Marquardt): damped normal equations with step acceptance. Robust for poor initialization but converges linearly near the optimum.

Hybrid computation pipeline. Performance-critical operations are delegated to compiled backends while maintaining Python accessibility for modeling. Residuals and Jacobians are computed by JAX (JIT-compiled, vectorized via `vmap`). The Schur complement solve is implemented entirely in Rust (parallel via `rayon`): a single pass over the CSR Jacobian builds $\mathbf{N}_{cc}$, $\mathbf{N}_{pp}$ blocks, and a sparse representation of $\mathbf{N}_{cp}$ (storing only non-zero camera–point entries). The Schur complement $\mathbf{S} = \mathbf{N}_{cc} - \sum_p \mathbf{v}_p \mathbf{N}_{pp,p}^{-1} \mathbf{v}_p^T$ is then accumulated point-by-point using only the sparse entries, avoiding any dense matrix–matrix multiplication. This hybrid approach achieves ∼0.5 s per iteration on the Roma dataset (79k unknowns, 181k observations) on a standard desktop CPU.

3.5 Educational Design

SCOP's architecture separates *problem formulation* from *numerical optimization*, enabling students to focus on photogrammetric concepts:

- **Declarative API**: cameras, images, and points are defined through intuitive constructors (Figure 3), hiding index management and Jacobian mechanics.
- **Transparent internals**: all intermediate quantities (Jacobian matrix, normal equations, covariance) are accessible as standard NumPy/JAX arrays for inspection.
- **Multiple optimizers**: implementing GN, GNA, LM, and GD in the same framework enables pedagogical comparison of convergence behavior, both per iteration and in wall-clock time (Figure 5).
- **Jupyter integration**: rich HTML display, inline plots, and interactive 3D visualization support exploratory learning.

The `ResidualBlock` architecture enables students to define new observation types by writing only the residual function $f(\mathbf{x})$. For example, adding a distance observation between two points requires a single function returning $\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2} - d_{\text{obs}}$; the Jacobian is computed automatically.

4. Experiments and Validation

4.1 Synthetic UAV Survey

A simulated survey (100×100 m area, 60 m altitude, 70–80% overlap, 12 images) validates correctness against known ground truth. SCOP converges within 10 iterations for Gauss-Newton, Gauss-Newton-Armijo, and Levenberg-Marquardt, all achieving $\sigma_0 \approx 0.29$ pixels (Figure 5). GN and GNA exhibit nearly identical convergence, while LM converges slightly slower due to its linear rate near the optimum. Gradient Descent requires >2,000 iterations, illustrating the well-known convergence advantage of second-order methods in a pedagogical setting.

4.2 Real-World Dataset: Roma (DBAT)

To validate on real data, we use the Roma dataset distributed with DBAT (Börlin and Grussenmeyer, 2016): 60 images, 1 camera (Agisoft/Brown distortion model), ∼26,000 tie points, and ∼90,000 Point2D observations. With GNSS position constraints (Cledat et al., 2020) ($\sigma = 3$ m) and 5 self-calibration parameters, the problem comprises 181,302 scalar observations and 79,328 unknowns (Table 2).

Property	Value
Images	60
3D Points	26,321
2D Observations	90,561
GNSS constraints	180 (X_s, Y_s, Z_s per image)
Scalar observations	181,302
Unknowns	79,328 (incl. 5 IO)
Degrees of freedom	101,974
Method	GNA + Schur
Iterations	4 (effective)
Final σ_0 (px)	0.575
Time per iteration	∼0.5 s (GNA)
Qxx diagonal	8 s
$\lvert w_i \rvert > 3$	75 / 181,302 (0.04%)
$\bar{z}_i$ (mean reliability)	0.56

Table 2. Roma dataset: problem size and optimization results (self-calibration scenario: self-calibration with GNSS constraints, $\sigma_{\text{GNSS}} = 3$ m).

This dataset demonstrates SCOP's ability to handle real-world photogrammetric problems with sparse assembly and Schur complement. A dense Jacobian would require ∼114 GB (181k × 79k × 8 bytes); the sparse CSR representation stores only ∼2.5M non-zeros (∼20 MB, 0.02% fill ratio).

4.3 Comparison with Existing Software

Table 3 positions SCOP relative to established bundle adjustment tools. SCOP is the only framework combining automatic differentiation with a complete stochastic model ($\hat{\sigma}_0$, $\mathbf{Q}_{xx}$, w_i, z_i), whereas C++ tools such as COLMAP and Ceres focus on computational efficiency but omit posterior quality indicators. Conversely, DBAT offers comparable statistical output but lacks GNSS integration and requires a MATLAB/Octave environment. MMVII provides the richest observation model among C++ tools (GNSS, GCP, angles) but does not expose covariance or reliability diagnostics, limiting post-adjustment quality control.

Table 4 compares the four tools on the Roma dataset (self-calibration scenario: self-calibration with GNSS constraints

Feature	SCOP	COLMAP 4	MMVII	DBAT	Ceres
Software					
Language	Python/JAX/Rust	C++	C++	MATLAB	C++
Open source	Yes (MIT)	Yes (BSD)	Yes (CeCILL-B)	Yes (GPL)	Yes (BSD)
Educational design	Yes	No	No	Yes	No
Differentiation					
Jacobians	Auto (JAX AD)	Analytical	Symbolic (codegen)	Analytical	Analytical/Auto
New model effort	Write $f(x)$ only	Derive + code	Write formula	Derive + code	Derive + code
Solver					
Method	GNA / LM	LM (Ceres)	GN	GNA	LM / Dogleg
Schur complement	Yes (Rust)	Yes	Yes (Eigen)	Yes (CHOLMOD)	Yes
Robust kernel	No	Cauchy	No	No	Huber/Cauchy/...
GPU support	Yes (JAX XLA)	No	No	No	CUDA (partial)
Stochastic model					
Observation weighting	Per-obs σ	Global	Per-type σ	Per-obs σ	Per-residual
$\hat{\sigma}_0$ (post. var. factor)	Yes	No	Yes	Yes	No
$\mathbf{Q}_{xx}$ covariance	Full / diagonal	No	No	Full	No
w_i (normalized res.)	Yes	No	No	Yes	No
z_i (reliability)	Yes	No	No	No	No
Observation types					
Tie points (2D)	Yes	Yes	Yes	Yes	Yes
GNSS / position	Yes (σ)	SfM only*	Yes (RefOri)	No	User-defined
Angles / distances	Yes	No	Yes	No	User-defined
Extensible obs.	Any (via AD)	Fixed	Fixed	Fixed	User-defined
GCP in BA	Yes	No	Yes	No	User-defined

*COLMAP 4 `pose_prior_mapper` only; not available in standalone `bundle_adjuster`.

Table 3. Comparison of bundle adjustment frameworks. MMVII refers to the MicMac v2 framework.

where available). All runs were performed on the same machine (Intel Core Ultra 7 155H, 32 GB RAM, Ubuntu 24.04). Direct comparison is complicated by significant differences in problem formulation: the number of self-calibration parameters ranges from 3 (COLMAP) to 10 (MMVII), and only SCOP and MMVII incorporate GNSS constraints. These modelling choices affect both the σ_0 values and convergence behaviour, so cross-tool comparisons should be interpreted as indicative rather than as strict rankings.

Tool	σ_0 (px)	Iter	Time	IO	GNSS
MMVII	0.415	4	16 s	10	Yes
DBAT	0.575	5	85 s[a]	8	No
SCOP	**0.575**	**4**	**19 s**[b]	5	Yes
COLMAP (L2)	0.749	201	39 s	3	No
COLMAP (Cauchy)	0.306[†]	201	39 s	3	No

[a] Incl. Octave startup; per-iter. cost ∼3 s.
[b] Incl. ∼14 s JIT (first run); steady-state ∼0.5 s/iter.
[†] Cauchy-weighted mean reproj. error, not σ_0.

Table 4. Benchmark comparison on Roma self-calibration.

SCOP and DBAT converge to identical σ_0 values (0.575 px), validating the solver implementation. The lower σ_0 of MMVII (0.415) is partly explained by its richer distortion model (Fraser with 10 parameters vs. 5 for SCOP), though differences in iteration strategy (retriangulation, outlier handling) may also contribute. COLMAP's higher σ_0 (0.749) reflects a combination of factors: a simpler Radial model (3 parameters: f, k_1, k_2) and the use of a Cauchy robust loss function, which reweights residuals and makes σ_0 not directly comparable to L2-based solvers. These differences highlight that cross-tool σ_0 comparisons are only meaningful when the observation model and cost function are identical, as is the case for SCOP and DBAT.

Computation times should be compared with caution, as they depend on implementation language, solver backend, and number of estimated parameters. With GNA (Table 4), SCOP converges in 19 s (4 iterations + JIT + $\mathbf{Q}_{xx}$); excluding the one-time JIT overhead (∼14 s), the solve completes in ∼5 s at ∼0.5 s per steady-state iteration. MMVII converges in 16 s (4 iterations, ∼4 s/iter). COLMAP completes 201 iterations in 39 s (∼0.2 s/iter in C++) but does not reach convergence. DBAT's 85 s total for 5 iterations includes Octave startup overhead; its per-iteration cost is ∼3 s, partly explained by its internal duplication of IO parameters per image (480 unknowns for a single shared camera), a design inherited from the PhotoModeler export format.

As an educational tool, SCOP prioritizes *transparency and accessibility*: the combination of automatic differentiation and Python enables students and researchers to prototype new camera models or observation types without manual Jacobian derivation. Compared to DBAT, which shares this educational motivation, SCOP adds sparse assembly with Schur complement, statistical quality assessment (z_i reliability indices), and avoids MATLAB licensing constraints. Like MMVII, SCOP natively integrates GNSS position constraints and ground control points (GCP) into the bundle adjustment. While COLMAP 4 now supports GNSS pose priors, it does not integrate GCP as weighted observations in the bundle adjustment; DBAT supports neither.

4.4 Computational Performance

Table 5 provides a breakdown of computational cost for the Roma dataset. Jacobian computation dominates the per-iteration budget (178 ms, 40%), followed by the Schur complement solve (120 ms) and the LM acceptance test (108 ms), which requires a full residual re-evaluation. The hybrid architecture proves effective: JAX handles the embarrassingly parallel differentiation, while the Rust backend exploits the block-sparse structure of the normal equations for the linear solve.

The first iteration includes JIT compilation overhead (∼14 s) as JAX traces and compiles the residual and Jacobian functions.

Operation	Time	Backend
Residuals	44 ms	JAX
Jacobian (sparse, `jacrev`)	178 ms	JAX
Schur solve (sparse $\mathbf{N}_{cp}$)	120 ms	Rust
LM acceptance test	108 ms	JAX
Total / iteration	**∼0.5 s**	
JIT (first iter.)	∼14 s	JAX XLA
$\mathbf{Q}_{xx}$ diagonal + w_i/z_i	8 s	Rust

Table 5. Cost breakdown per iteration (Roma, CPU).

Subsequent iterations benefit from cached compiled kernels, reducing per-iteration cost to ∼0.5 s.

4.5 Statistical Quality Assessment

Following classical photogrammetric theory (Luhmann et al., 2020, Baarda, 1968), SCOP computes the full posterior covariance matrix and derived quality indicators:

- **Posterior variance factor**: $\sigma_0^2 = \mathbf{v}^T\mathbf{P}\mathbf{v}/(m-n)$, where m is the number of observations and n the number of unknowns.
- **Covariance of unknowns**: $\mathbf{Q}_{xx} = (\mathbf{A}^T\mathbf{P}\mathbf{A})^{-1}$, providing standard deviations and correlations between all estimated parameters.
- **Normalized residuals**: $w_i = v_i/(\sigma_0\sqrt{q_{vv,i}})$, enabling outlier detection ($|w_i| > 3$).
- **Reliability indices**: $z_i = q_{vv,i}/q_{ll,i}$, quantifying the local redundancy of each observation in the network.

These indicators are computed in both full ($\mathbf{Q}_{xx} \in \mathbb{R}^{n\times n}$) and diagonal-only modes. For large problems, the diagonal mode exploits the Schur complement structure: $\mathbf{Q}_{cc} = \mathbf{S}^{-1}$ (dense, $n_c \times n_c$), and each point block $\mathbf{Q}_{pp}^{(i)}$ is computed independently using the block inversion formula. The $\mathbf{Q}_{vv}$ diagonal is then obtained per observation from the partitioned blocks, avoiding the $O(n^3)$ cost of a full matrix inversion. This Schur-based approach computes the full $\mathbf{Q}_{xx}$ diagonal and exact $\mathbf{Q}_{vv}$ for all 181k observations in 8 s (vs. >30 min for a direct LU-based approach on the 79k×79k normal matrix).

Results on Roma. Table 6 summarizes the quality indicators for the Roma dataset (self-calibration scenario). The w_i distribution is approximately Gaussian with $\sigma = 0.60$; the standard deviation below unity is expected because the a posteriori $\hat{\sigma}_0$ is used in the denominator, absorbing part of the variance. Only 75 observations (0.04%) exceed the $|w_i| > 3$ threshold, suggesting very few outliers among the 90k tie point measurements. The reliability indices z_i indicate good local redundancy: the median $z_i = 0.66$ means that most observations contribute meaningfully to the adjustment while retaining sufficient redundancy for outlier detection.

The lowest z_i values, close to zero, correspond to observations with very low redundancy, typically points seen in only two images, where the observation is nearly fully determined by the unknowns. These observations have limited self-checking capability and would benefit from additional image coverage.

Regarding outlier detection, the highest w_i values ($|w_i| \approx 5.7$) identify potential mismatches in the tie point extraction that could be removed in a subsequent robust adjustment iteration.

Statistic	$\mathbf{w_i}$	$\mathbf{z_i}$
Mean	0.002	0.563
Std	0.597	—
P01 / P99	−1.68 / 1.65	0.00 / 0.91
P05 / P95	−0.97 / 0.97	0.00 / 0.88
P25 / P75	−0.30 / 0.31	0.43 / 0.79
Median	0.000	0.660
$\lvert w_i\rvert > 3$	75 (0.04%)	

Table 6. Quality indicators on Roma (self-calibration scenario, 181,302 observations).

Figure 1 shows the w_i distribution, which is approximately Gaussian and symmetric, as expected for a well-adjusted network. The 75 observations exceeding $|w_i| > 3$ (dashed lines) are scattered spatially without systematic patterns, confirming random measurement noise rather than model bias. Figure 2 shows z_i as a function of the number of views per point: reliability increases monotonically with multiplicity, confirming that additional image coverage improves local controllability. Observations at block edges (fewer views) have lower redundancy and thus limited self-checking capability.

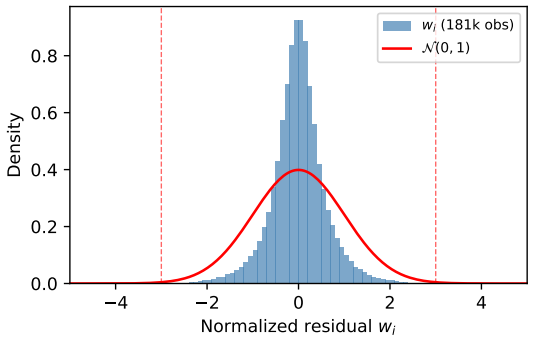

Figure 1. Distribution of normalized residuals w_i on Roma (181k observations). Dashed lines indicate the $|w_i| > 3$ outlier threshold; only 75 observations (0.04%) exceed it.

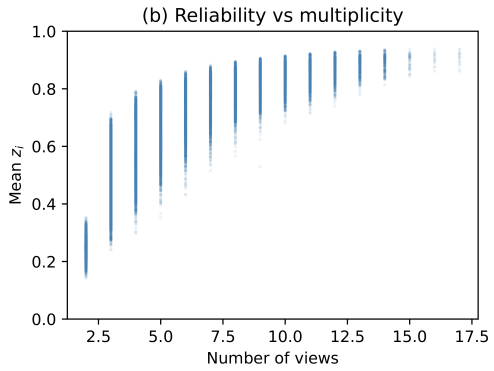

Figure 2. Reliability index z_i vs. number of views per point (Roma, 90k observations). Higher multiplicity yields higher redundancy.

Visualization includes 3D uncertainty ellipsoids (Figure 4) computed from the 3×3 sub-blocks of $\mathbf{Q}_{xx}$ corresponding to each 3D point, rendered as 95% confidence regions in the interactive viewer. The ellipsoid shape and orientation reveal the anisotropy of the position uncertainty: elon-

gated ellipsoids along the viewing direction reflect the well-known depth-precision limitation of photogrammetric intersection, while compact ellipsoids indicate well-constrained points with high multiplicity. This visualization provides an immediate, intuitive diagnostic of network geometry quality that complements the scalar z_i indicators.

5. Conclusions

We have presented SCOP, a Python/JAX framework for generic photogrammetric bundle adjustment based on automatic differentiation. Three main findings emerge from this work:

1. **Automatic differentiation is viable for real-world bundle adjustment**: on the Roma dataset (79k unknowns, 181k observations), SCOP converges to $\sigma_0 = 0.575$ px (matching the DBAT reference) in 4 Gauss-Newton-Armijo iterations, validating correctness against an established tool. Adding a new camera model or observation type requires only a Python function, with no manual Jacobian derivation.

2. **Acceptable performance in Python**: the hybrid pipeline (JAX for automatic differentiation, Rust for sparse Schur complement) achieves $\sim$0.5 s per iteration, demonstrating that an interpreted language can handle real-scale photogrammetric problems when combined with compiled backends.

3. **An educational platform for bundle adjustment and statistical quality control**: SCOP computes normalized residuals w_i and reliability indices z_i for all 181k observations, enabling students to explore outlier detection and network reliability on real data, concepts often taught theoretically but rarely practiced hands-on. The transparent Python code allows inspecting every step of the bundle adjustment process.

The main limitation is the Just-In-Time compilation overhead on first execution ($\sim$14 s), which is amortized by persistent caching. The framework currently lacks robust cost functions; outlier rejection relies on post-adjustment w_i analysis rather than iterative re-weighting.

Several directions are planned for future development. First, integrating robust cost functions (Huber, Cauchy) through iteratively re-weighted least squares would enable automatic outlier rejection during the adjustment, replacing the current post-adjustment w_i-based detection. Second, the automatic differentiation architecture makes it straightforward to extend SCOP to additional observation types (terrestrial laser scanning targets, total station angles and distances, inertial measurement unit pre-integration constraints) and to rigid-body constraints such as lever arms and boresight angles for multi-sensor calibration, each requiring only the definition of its residual function. Finally, the emergence of end-to-end learned reconstruction methods such as VGGT (Wang et al., 2025) raises the question of how classical bundle adjustment can complement these approaches; SCOP's statistical quality control (w_i, z_i, $\mathbf{Q}_{xx}$) could provide the rigorous uncertainty quantification that purely learned pipelines currently lack.

SCOP is available under MIT license at `https://gitlab.com/heig-vd-geo/scop`.

```
# 1. Camera with Agisoft distortion
cam = Camera.frame(h=3000, w=4000,
        focal_px=2200, variables=['f'])

# 2. Two images with GNSS constraints
img1 = Image.from_opk(cam,
        position=(0, 0, 100),
        opk_deg=(0, 0, 0),
        variables=["Xs", "Ys", "Zs"],
        observations=["Xs", "Ys", "Zs"])
img2 = Image.from_opk(cam,
        position=(50, 0, 100),
        opk_deg=(0, 0, 0),
        variables=["Xs", "Ys", "Zs"],
        observations=["Xs", "Ys", "Zs"])

# 3. 3D points (typed)
gcp = Point3D.control(x=10, y=20, z=5,
                      sigma=0.01)
tie = Point3D.tie(x=25, y=10, z=3)

# 4. 2D observations
p1 = Point2D(x=1500, y=1200, _m_pt3=gcp)
img1.add_pt2(p1)
p2 = Point2D(x=2100, y=900, _m_pt3=tie)
img1.add_pt2(p2)
p3 = Point2D(x=800, y=950, _m_pt3=tie)
img2.add_pt2(p3)

# 5. Build and inspect problem
opt = Optimizer()
opt.add_objects([cam, img1, img2,
                 gcp, tie])
opt.set_sigma("Point2D", 0.5)   # 0.5 px
opt.set_sigma_pos("Image", 3.0) # GNSS 3 m
opt.update_nb_obs_var()
opt.print_variables()  # problem summary

# 6. Optimize
opt.fit(method="LM", iter_max=20,
        schur=True, compute_Qxx=True)

# 7. Quality assessment
opt.print_report()

# 8. Access internals
print(opt.s0)          # a posteriori sigma_0
print(opt.Qxx)         # covariance matrix
print(opt.get_sigma(gcp))    # std of GCP

p1.get_wi_value("x")        # normalized res.
p1.get_zi_value("x")        # reliability index

# 9. Interactive 3D visualization
from scop.utils.visualization \
    .Scene3DViewer import Scene3DViewer
Scene3DViewer(opt).show()
```

Figure 3. Complete SCOP API example: two-image bundle adjustment with GCP, tie point, GNSS constraints, quality assessment, and 3D visualization.

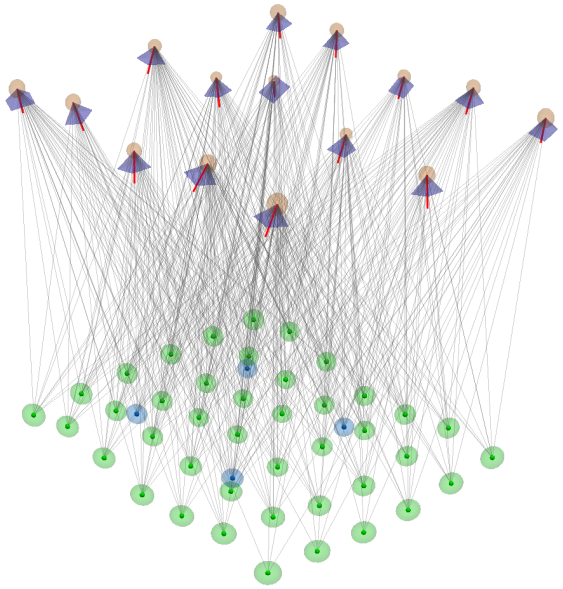

Figure 4. Interactive 3D visualization: camera frustums, 3D points with color-coded residuals, and 95% confidence ellipsoids from $\mathbf{Q}_{xx}$ (scaled 5×).

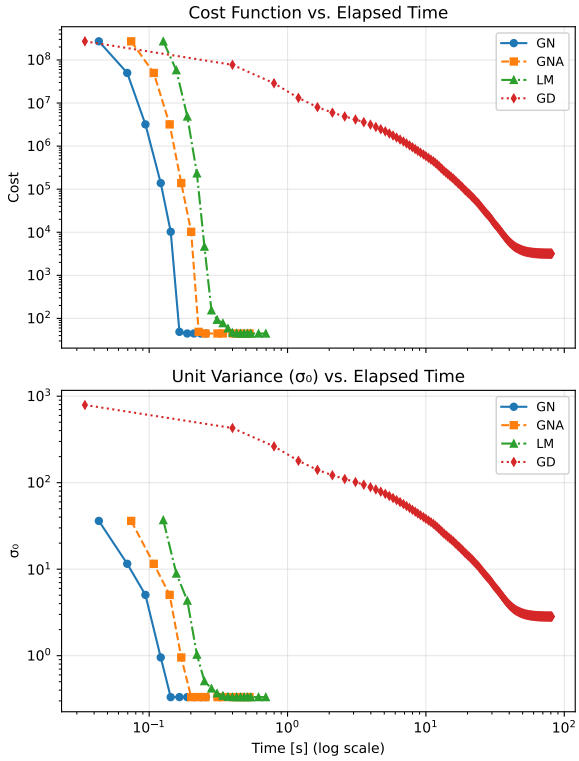

Figure 5. Convergence vs. elapsed time (log scale): GN converges fastest (∼0.2 s) thanks to quadratic convergence, followed by LM (∼0.4 s) and GNA (∼0.5 s). GD remains far from convergence after ∼100 s, illustrating the cost of first-order methods on bundle adjustment problems.

References

Agarwal, S., Mierle, K. et al., 2024. Ceres Solver. `http://ceres-solver.org`.

Agisoft, 2023. Metashape User Manual: Professional Edition, version 2.1. `https://www.agisoft.com/pdf/metashape-pro_2_1_en.pdf`. Accessed: 2026-03-25.

Baarda, W., 1968. A Testing Procedure for Use in Geodetic Networks. *Netherlands Geodetic Commission, Publications on Geodesy, New Series*, 2(5).

Baydin, A., Pearlmutter, B., Radul, A., Siskind, J., 2018. Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, 18(153), 1–43.

Börlin, N., Grussenmeyer, P., 2016. External Verification of the Bundle Adjustment in Photogrammetric Software Using the Damped Bundle Adjustment Toolbox. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLI-B5, 7–14.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q., 2018. JAX: composable transformations of Python+NumPy programs.

Brown, D., 1971. Close-Range Camera Calibration. *Photogrammetric Engineering*, 37(8), 855–866.

Cledat, E., Jospin, L., Cucci, D., Skaloud, J., 2020. Mapping quality prediction for RTK/PPK-equipped micro-drones. *ISPRS J. Photogramm. Remote Sens.*, 167, 24–38.

Conrady, A., 1919. Decentred Lens-Systems. *Monthly Notices of the Royal Astronomical Society*, 79(5), 384–390.

Griewank, A., Walther, A., 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd edn, SIAM.

Hartley, R., Zisserman, A., 2004. *Multiple View Geometry in Computer Vision*. 2nd edn, Cambridge University Press.

Luhmann, T., Robson, S., Kyle, S., Boehm, J., 2020. *Close-Range Photogrammetry and 3D Imaging*. 3rd edn, De Gruyter, Berlin.

Pierrot-Deseilligny, M., 2024. MMVII – MicMac v2: Documentation and source code. `https://github.com/micmacIGN/micmac/wiki/MMVII`. Accessed: 2026-03-25.

Rupnik, E., Daakir, M., Pierrot-Deseilligny, M., 2017. MicMac – a free, open-source solution for photogrammetry. *Open Geospatial Data, Software and Standards*, 2(14), 1–9.

Schönberger, J., Frahm, J.-M., 2016. Structure-from-Motion Revisited. *Proc. IEEE CVPR*, 4104–4113.

Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A., 2000. Bundle Adjustment – A Modern Synthesis. *Vision Algorithms: Theory and Practice*, LNCS, 1883, 298–372.

Wang, J., Chen, M., Karaev, N., Vedaldi, A., Rupprecht, C., Novotny, D., 2025. VGGT: Visual geometry grounded transformer. *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 5294–5306.

Zhan, Z., Xu, H., Fang, Z., Wei, X., Hu, Y., Wang, C., 2024. Bundle Adjustment in the Eager Mode. *arXiv preprint*. arXiv:2409.12190.

Zhang, Z., 2000. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 1330–1334.