The Journal of Open Source Software

Minion: A C++ and Python Library for Single-Objective Optimization Algorithms

Khoirul Faiq Muzakka [1], **Sören Möller** [1], and **Martin Finsterbusch** [1]

1 Institute of Energy Materials and Devices (IMD-2), Forschungszentrum Jülich GmbH, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 05 October 2025
Published: unpublished

Summary

Minion is a derivative-free optimization library for single-objective problems in which gradient-based methods are impractical. It provides a C++ backend with a Python interface (MinionPy), supporting applications in engineering, machine learning, and scientific computing.

The library offers a centralized implementation of state-of-the-art Differential Evolution (DE) algorithms that have performed strongly in IEEE CEC competitions. Alongside these research-grade solvers, Minion ships widely used optimizers such as Nelder–Mead, Dual Annealing (generalized simulated annealing), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), and several Particle Swarm Optimization (PSO) variants, allowing practitioners to combine established baselines with advanced heuristics within a single API. Many existing optimization libraries include only elementary DE variants and lack standardized benchmark problems. Minion addresses this by integrating multiple CEC benchmark suites (2011, 2014, 2017, 2019, 2020, and 2022) to facilitate algorithm evaluation and comparison.

Compared with widely adopted toolkits such as SciPy, NLopt, and pagmo2/pygmo, Minion emphasises a unified interface for batch-evaluated objective functions, provides native support for modern CEC-winning DE variants, and bundles curated benchmark suites for reproducible experimentation. These design choices serve researchers developing bespoke algorithms as well as practitioners seeking robust defaults for black-box optimisation.

Review of existing optimization libraries

SciPy (Virtanen et al., 2020) underpins a large fraction of scientific computing in Python and offers a stable interface to classical optimisation routines, including gradient-based methods and a handful of derivative-free heuristics such as Nelder–Mead and Powell's method. Its design prioritises broad accessibility and numerical reliability; consequently, coverage of recent population-based metaheuristics or fully vectorised objective evaluations is therefore limited.

NLopt (Johnson, 2007) collects an extensive set of local and global optimisers behind a C API with bindings to multiple languages. The library provides deterministic algorithms (e.g. COBYLA, BOBYQA) and stochastic search methods (e.g. CRS, ISRES, ESCH), yet relies on single-sample objective calls. Users who require Differential Evolution or particle-swarm heuristics typically integrate third-party implementations alongside NLopt's core offerings.

pagmo2/pygmo (Biscani & Izzo, 2020) is geared towards island-based, massively parallel search. It excels at composing heterogeneous portfolios of solvers and supports sophisticated multi-objective workflows. For practitioners focused on single-objective, derivative-free problems, realising a streamlined setup—particularly when benchmarking CEC-style test suites or coupling to noisy quasi-Newton updates—can involve additional configuration effort.

DEAP (Fortin et al., 2012) provides a highly extensible Python framework for constructing evolutionary algorithms from modular operators. This flexibility is valuable for exploratory research, but achieving high-throughput optimisation requires users to supply their own performance-oriented backends, batched evaluation loops, and curated algorithm configurations.

Minion aims to complement these ecosystems by concentrating on single-objective optimisation with built-in support for batch evaluation, vectorised quasi-Newton updates, and implementations of recent Differential Evolution and swarm variants that have performed well on modern CEC benchmarks. The goal is not to replace these libraries, but to offer an option tailored to scenarios where such capabilities are central requirements.

Statement of need

Minion was created to address several limitations in existing optimization libraries:

1. Centralized library for state-of-the-art Differential Evolution algorithms. Many optimization libraries lack a unified framework that combines advanced Differential Evolution (DE) variants with the latest CEC benchmark problems through a simple interface in both C++ and Python. While basic DE algorithms are common, modern variants such as L-SHADE and jSO—which have demonstrated superior performance in CEC competitions—are often absent. Minion fills this gap by providing these algorithms alongside a platform for researchers to create and test new optimizers, streamlining benchmarking and comparison with existing methods.
2. Limited support for straightforward batch evaluation. Several widely used libraries offer population-based optimisers, but their APIs typically accept one sample at a time, making it cumbersome to exploit highly parallel objective evaluations. Minion accepts batches natively so that vectorised or distributed functions can be used without additional wrappers.
3. Lack of a robust L-BFGS-B implementation that performs well under noise and supports batch objective evaluations. Traditional L-BFGS-B implementations struggle with noisy function calls, which are common in real-world applications such as experimental data fitting. Minion provides an improved L-BFGS-B implementation that mitigates these issues.

Algorithms

Minion currently implements the following optimization algorithms:

- Basic Differential Evolution (DE) (Storn & Price, 1997)
- JADE (Zhang & Sanderson, 2009)
- LSHADE (Tanabe & Fukunaga, 2014)
- LSHADE-cnEpSin (**?**)
- jSO (Brest et al., 2017)
- j2020 (Brest et al., 2020)
- NL-SHADE-RSP (Stanovov et al., 2021)
- LSRTDE (Stanovov & Semenkin, 2024)
- Adaptive Restart-Refine Differential Evolution (ARRDE)
- Artificial Bee Colony (ABC) (Karaboga, 2005)
- Canonical PSO (Kennedy & Eberhart, 1995)
- SPSO-2011 (Zambrano-Bigiarini et al., 2013)
- Dynamic Multi-Swarm PSO (DMS-PSO) (Liang & Suganthan, 2005)
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (N. Hansen & Ostermeier, 1996)
- BI-population CMA-ES (BIPOP-ACMAES) (Nikolaus Hansen, 2009)
- Generalized Simulated Annealing (Dual Annealing) (Xiang et al., 1997)

- Nelder–Mead (Nelder & Mead, 1965)
- L-BFGS-B (Byrd et al., 1995)
- L-BFGS (Liu & Nocedal, 1989)

Additional algorithms are planned for future releases. Minion also ships benchmark suites from IEEE CEC competitions spanning 2011, 2014, 2017, 2019, 2020, and 2022. The library further bundles classic analytic test functions—such as sphere, Rosenbrock, and Rastrigin—for quick experimentation and unit testing.

Minion offers a unified `Minimizer` interface available in both C++ and Python, providing a consistent way to access all implemented algorithms. Solvers are selected by concise identifiers (e.g., `"ARRDE"` or `"L_BFGS_B"`), and option names are standardized across methods for clarity and interoperability. The result structure, `MinionResult`, adopts the general layout of `scipy.optimize.OptimizeResult` — a familiar convention in the scientific computing community. This design choice enhances usability and readability, allowing users to work with Minion intuitively while remaining fully independent of SciPy.

In Minion, the optional `x0` argument may contain multiple initial guesses—an uncommon capability in optimisation libraries. This is practical when prior knowledge suggests several promising starting points or when restart strategies are desired. Population methods treat the entries as explicit seeds for their initial populations, while single-trajectory solvers (e.g. CMA-ES, Nelder–Mead, L-BFGS variants) evaluate each candidate and proceed from the best-performing one. The behaviour streamlines the reuse of domain heuristics when launching new optimisation runs.

Minion's L-BFGS and L-BFGS-B implementations build on LBFGSpp (Qiu, 2013) but introduce several features tailored to noisy, vectorised workloads. Gradient estimates are generated from batched finite differences, while noise-aware step sizes and a Lanczos-style smoothing filter reduce variance in the resulting updates. This design is motivated by the robustness that has kept Minuit/Migrad(James, 1994) a standard tool in high-energy physics for decades. The accompanying benchmark notebook shows that, under these modifications, Minion's quasi-Newton solvers compete favourably with Minuit/Migrad on noisy CEC test suites while preserving a fully vectorised evaluation pipeline.

Availability and documentation

Minion is primarily implemented in C++. However, recognizing the popularity of Python and its ease of use, a Python wrapper (MinionPy) is also available. It can be installed via PIP, allowing for seamless integration into Python-based workflows. Documentation on how to use both Minion and MinionPy is available at: https://minion-py.readthedocs.io/ .

References

Biscani, F., & Izzo, D. (2020). A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, *5*(53), 2338. https://doi.org/10.21105/joss.02338

Brest, J., Maučec, M. S., & Bošković, B. (2020). Differential evolution algorithm for single objective bound-constrained optimization: Algorithm j2020. *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. https://doi.org/10.1109/CEC48606.2020.9185551

Brest, J., Maučec, M. S., & Bošković, B. (2017). Single objective real-parameter optimization: Algorithm jSO. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 1311–1318. https://doi.org/10.1109/CEC.2017.7969456

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, *16*(5), 1190–1208.

https://doi.org/10.1137/0916069

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Hansen, Nikolaus. (2009). Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2389–2396. https://doi.org/10.1145/1570256.1570333

Hansen, N., & Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. *Proceedings of IEEE International Conference on Evolutionary Computation*, 312–317. https://doi.org/10.1109/ICEC.1996.542381

James, F. (1994). *MINUIT Function Minimization and Error Analysis: Reference Manual Version 94.1*.

Johnson, S. G. (2007). *The NLopt nonlinear-optimization package*. https://github.com/stevengj/nlopt.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization, technical report - TR06. *Technical Report, Erciyes University*.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, *4*, 1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

Liang, J. J., & Suganthan, P. N. (2005). Dynamic multi-swarm particle swarm optimizer. *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 124–129. https://doi.org/10.1109/SIS.2005.1501611

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1), 503–528. https://doi.org/10.1007/BF01589116

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313. https://doi.org/10.1093/comjnl/7.4.308

Qiu, Y. (2013). *LBFGSpp: C++ library for l-BFGS and l-BFGS-b*. https://github.com/yixuan/LBFGSpp.

Stanovov, V., Akhmedova, S., & Semenkin, E. (2021). NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 809–816. https://doi.org/10.1109/CEC45853.2021.9504959

Stanovov, V., & Semenkin, E. (2024). Success rate-based adaptive differential evolution l-SRTDE for CEC 2024 competition. *2024 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. https://doi.org/10.1109/CEC60901.2024.10611907

Storn, R., & Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328

Tanabe, R., & Fukunaga, A. S. (2014). Improving the search performance of SHADE using linear population size reduction. *2014 IEEE Congress on Evolutionary Computation (CEC)*, 1658–1665. https://doi.org/10.1109/CEC.2014.6900380

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S. J. van der, Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Contributors, S. 1. 0. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Xiang, Y., Sun, D. Y., Fan, W., & Gong, X. G. (1997). Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, *233*(3), 216–220. https://doi.org/10.1016/S0375-9601(97)00474-X

Zambrano-Bigiarini, M., Clerc, M., & Rojas, R. (2013). Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. *2013 IEEE Congress on Evolutionary Computation*, 2337–2344. https://doi.org/10.1109/CEC.2013.6557848

Zhang, J., & Sanderson, A. C. (2009). JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, *13*(5), 945–958. https://doi.org/10.1109/TEVC.2009.2014613