The Journal of Open Source Software

JSOSolvers.jl: Unconstrained and bound-constrained optimization solvers

Tangi Migot[1¶], Dominique Monnet[3], Dominique Orban[1], and Abel Soares Siqueira[2]

1 GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, QC, Canada. **2** Netherlands eScience Center, Amsterdam, Netherlands **3** Univ Rennes, INSA Rennes, CNRS, IRMAR - UMR 6625, Rennes, France ¶ Corresponding author

DOI: 10.21105/joss.09467

Software

- Review
- Repository
- Archive

Editor: Neea Rusch

Reviewers:

- @giovannifereoli
- @dourouc05
- @raphaelchenouard

Submitted: 10 November 2025
Published: 14 January 2026

Summary

`JSOSolvers.jl` is a collection of Julia (Bezanson et al., 2017) optimization solvers for nonlinear, potentially nonconvex, continuous optimization problems that are unconstrained or bound-constrained:

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x) \quad \text{subject to} \quad \ell \le x \le u, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function, with $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, and $u \in (\mathbb{R} \cup \{+\infty\})^n$. The algorithms implemented here are iterative methods that aim to compute a stationary point of (1) using first and, if possible, second-order derivatives.

Our initial motivation for considering (1) and developing `JSOSolvers.jl` is to solve large-scale unconstrained and bound-constrained problems such as parameter estimation in inverse problems, design optimization in engineering, and regularized machine learning models, and use these solvers to solve subproblems of penalty algorithms, such as `Percival.jl` (Antunes dos Santos et al., 2026) or `FletcherPenaltySolver.jl` (Migot et al., 2025), for constrained nonlinear continuous optimization problems. In many of these problems, explicitly storing Hessian matrices is either computationally prohibitive or impractical. The solvers in `JSOSolvers.jl` adopt a matrix-free approach, where standard optimization methods are implemented without forming derivative matrices explicitly. This strategy enables the solution of large-scale problems even when function and gradient evaluations are expensive.

The library includes TRON, a trust-region Newton method for bound-constrained problems following the classical formulation of Lin & Moré (1999), TRUNK, a factorization-free trust-region Newton method based on the truncated conjugate gradient method, as described by Conn et al. (2000), an implementation of L-BFGS, a limited-memory quasi-Newton method using a line search globalization strategy, and FOMO, a first-order method based on quadratic regularization designed for unconstrained optimization. FOMO is an extension of a quadratic regularization method described by Aravkin et al. (2022), and called R2 in `JSOSolvers.jl`. Unlike textbook implementations, our solvers introduce several design differences. TRON operates in a factorization-free mode, while the original Fortran TRON requires an explicitly formed Hessian. TRUNK departs from the Conn–Gould–Toint formulation by supporting a non-monotone mode and multiple subproblem solvers (CG, CR, MINRES, etc.) Our L-BFGS implementation uses a simplified line-search strategy that avoids the standard Wolfe conditions while maintaining robust convergence in practice.

A nonlinear least-squares problem is a special case of (1), where $f(x) = \frac{1}{2}\|F(x)\|_2^2$ and the residual $F : \mathbb{R}^n \to \mathbb{R}^m$ is continuously differentiable, which appears in many applications, including inverse problems in imaging, geophysics, and machine learning. While it is possible to solve the problem using only the objective, knowing F independently allows the development

of more efficient methods. Specialized variants of TRON and TRUNK, called TRON-NLS and TRUNK-NLS, leverage the structure of residual models to improve performance and scalability.

Key strengths of `JSOSolvers.jl` are its efficiency and flexibility. The solvers support fully in-place execution, allowing repeated solves without additional memory allocation, which is particularly beneficial in high-performance and GPU computing environments where memory management is critical. The solvers support any floating-point type, including extended and multi-precision types such as BigFloat, DoubleFloats, and QuadMath. Moreover, TRUNK, TRUNK-NLS, and FOMO support GPU arrays, broadening the range of hardware where the package can be effectively deployed, for instance when used together with `ExaModels.jl` (Shin et al., 2024). The package documentation and https://jso.dev/tutorials provide examples illustrating the use of different floating-point systems. Furthermore, the solvers expose in-place function variants, allowing multiple optimization problems with identical dimensions and data types to be solved efficiently without reallocations.

`JSOSolvers.jl` is built upon the JuliaSmoothOptimizers (JSO) tools[1]. JSO is an academic organization containing a collection of Julia packages for nonlinear optimization software development, testing, and benchmarking. It provides tools for building models, accessing problem repositories, and solving subproblems. Solvers in `JSOSolvers.jl` take as input an `AbstractNLPModel`, JSO's general model API defined in `NLPModels.jl` (Orban et al., 2026a), a flexible data type to evaluate objective and constraints, their derivatives, and to provide any information that a solver might request from a model. The user can hand-code derivatives, use automatic differentiation, or use JSO-interfaces to classical mathematical optimization modeling languages such as AMPL (Fourer et al., 1990), CUTEst (Gould et al., 2015), and JuMP (Dunning et al., 2017). The solvers heavily rely on iterative linear algebra methods from `Krylov.jl` (Montoison & Orban, 2023).

Statement of need

Julia's JIT compiler is attractive for the design of efficient scientific computing software, and, in particular, mathematical optimization (Lubin & Dunning, 2015), and has become a natural choice for developing new solvers.

While several options exist to solve (1) in Julia, many rely on wrappers to solvers implemented in low-level compiled languages. For example, if (1) is modeled using JuMP (Dunning et al., 2017), it can be passed to solvers like IPOPT (Wächter & Biegler, 2006) and Artelys Knitro (Byrd et al., 2006) via Julia's native C and Fortran interoperability. However, these interfaces often lack flexibility with respect to data types and numerical precision. In contrast, solvers written in pure Julia can seamlessly operate with a variety of arithmetic types or even GPU array types. This capability is increasingly important as extended-precision arithmetic becomes more accessible through packages such as GNU MPFR, shipped with Julia. Such flexibility enables high-precision computing when numerical accuracy is paramount.

Several alternatives to `JSOSolvers.jl` are available within and outside the Julia ecosystem. `Optim.jl` (Mogensen & Riseth, 2018) is a general-purpose optimization library in pure Julia, suitable for small to medium-scale problems, but it lacks in-place execution and GPU support. `NLopt.jl` (Johnson, 2007) provides access to a broad collection of optimization algorithms via a C library but does not support matrix-free methods or extended precision. `AdaptiveRegularization.jl` (Dussault et al., 2024) offers a matrix-free, multi-precision solver for unconstrained problems and is closely aligned with the design philosophy of `JSOSolvers.jl`. Ipopt (Wächter & Biegler, 2006), via `Ipopt.jl`, is a widely used and efficient solver, but requires explicit derivatives and is limited to CPU execution. GALAHAD (Fowkes & Gould, 2023), a Fortran-based suite for large-scale problems, is accessible through experimental Julia wrappers, yet lacks native composability. Commercial solvers such as Artelys Knitro (Byrd

[1]JuliaSmoothOptimizers https://jso.dev/

et al., 2006) provide robust algorithms but remain constrained by licensing and limited Julia interoperability. `Optimization.jl` is a wrapper to existing optimization packages.

Benchmarking

`JSOSolvers.jl` can solve large-scale problems and can be benchmarked easily against other JSO-compliant solvers using `SolverBenchmark.jl` (Orban et al., 2026b). Below, we include performance profiles (Dolan & Moré, 2002) with respect to elapsed time of `JSOSolvers.jl` solvers against Ipopt on all the 291 unconstrained problems from the CUTEst collection (Gould et al., 2015), whose dimensions range from 2 up to 192,627 variables.[2] LBFGS uses only first-order information, while TRON and TRUNK use Hessian-vector products and Ipopt uses the Hessian as a matrix. Without explaining performance profiles in full detail, the plot shows that Ipopt is fastest on 42 problems (15%), TRON on 9 (3%), TRUNK on 64 (21%), and L-BFGS on 176 (60%). Nearly all problems were solved within the 20-minute limit: TRON solved 272 (93%), Ipopt 270, TRUNK 269, and L-BFGS 267.

Overall, these results are encouraging. Although Ipopt is a mature and highly optimized solver, TRUNK and L-BFGS achieve comparable problem coverage while being significantly faster on many instances. This suggests that the algorithms implemented here are competitive for large-scale problems. We also expect further gains as we continue refining algorithmic hyperparameters, which is one of the project's short-term development goals. A complementary benchmark for bound-constrained problems is available in the package documentation.

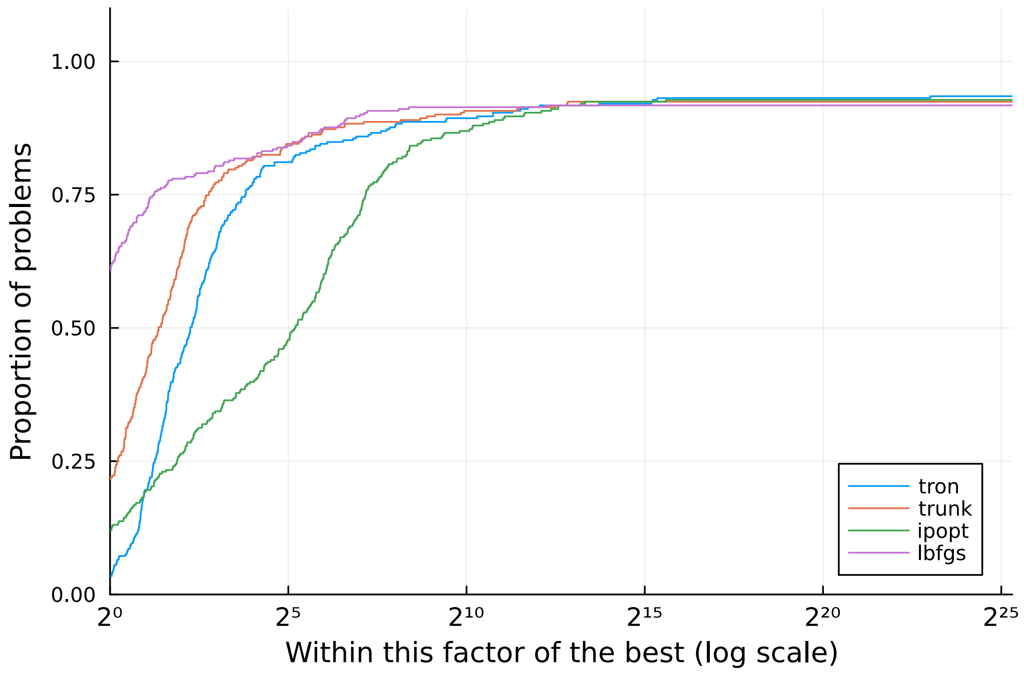

Figure 1: Unconstrained solvers on CUTEst with respect to the elapsed time.

Acknowledgements

Dominique Orban is partially supported by an NSERC Discovery Grant.

[2]Benchmarks were run sequentially on a CPU-only machine. The hardware configuration was an Intel Core i7-class processor with approximately 16 GB of RAM running Linux. Timings are intended for relative comparison only.

References

Antunes dos Santos, E., Migot, T., Orban, D., Soares Siqueira, A., & contributors. (2026). *Percival.jl: An augmented Lagrangian method* (Version 0.7.6). https://doi.org/10.5281/ZENODO.3969045

Aravkin, A. Y., Baraldi, R., & Orban, D. (2022). A proximal quasi-Newton trust-region method for nonsmooth regularized optimization. *SIAM Journal on Optimization*, *32*(2), 900–929. https://doi.org/10.1137/21m1409536

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Byrd, R. H., Nocedal, J., & Waltz, R. A. (2006). KNITRO: An integrated package for nonlinear optimization. In *Large-Scale Nonlinear Optimization* (pp. 35–59). Springer. https://doi.org/10.1007/0-387-30065-1_4

Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). *Trust region methods*. SIAM. https://doi.org/10.1137/1.9780898719857

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*(2), 201–213. https://doi.org/10.1007/s101070100263

Dunning, I., Huchette, J., & Lubin, M. (2017). JuMP: A modeling language for mathematical optimization. *SIAM Review*, *59*(2), 295–320. https://doi.org/10.1137/15M1020575

Dussault, J.-P., Goyette, S., Migot, T., Orban, D., & contributors. (2024). *AdaptiveRegularization.jl: A unified efficient implementation of trust-region type algorithms for unconstrained optimization*. https://doi.org/10.5281/zenodo.10434673

Fourer, R., Gay, D. M., & Kernighan, B. W. (1990). A modeling language for mathematical programming. *Management Science*, *36*(5), 519–554. https://doi.org/10.1287/mnsc.36.5.519

Fowkes, J. M., & Gould, N. I. M. (2023). GALAHAD 4.0: An open source library of Fortran packages with C and Matlab interfaces for continuous optimization. *Journal of Open Source Software*, *8*(87), 4882. https://doi.org/10.21105/joss.04882

Gould, N. I., Orban, D., & Toint, P. L. (2015). CUTEst: A Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, *60*(3), 545–557. https://doi.org/10.1007/s10589-014-9687-3

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

Lin, C.-J., & Moré, J. J. (1999). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, *9*(4), 1100–1127. https://doi.org/10.1137/s1052623498345075

Lubin, M., & Dunning, I. (2015). Computing in operations research using Julia. *INFORMS Journal on Computing*, *27*(2), 238–248. https://doi.org/10.1287/ijoc.2014.0623

Migot, T., Orban, D., Soares Siqueira, A., & contributors. (2025). *FletcherPenaltySolver.jl: Fletcher's penalty method for nonlinear optimization models* (Version 0.3.0). https://doi.org/10.5281/ZENODO.7153563

Mogensen, P. K., & Riseth, A. N. (2018). Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, *3*(24), 615. https://doi.org/10.21105/joss.00615

Montoison, A., & Orban, D. (2023). Krylov.jl: A Julia basket of hand-picked Krylov methods. *Journal of Open Source Software*, *8*(89), 5187. https://doi.org/10.21105/joss.05187

Orban, D., Soares Siqueira, A., & contributors. (2026a). *NLPModels.jl: Data structures for optimization models* (Version 0.21.7). https://doi.org/10.5281/ZENODO.2558626

Orban, D., Soares Siqueira, A., & contributors. (2026b). *SolverBenchmark.jl: Benchmark tools for solvers* (Version 0.6.4). https://doi.org/10.5281/ZENODO.2581661

Shin, S., Anitescu, M., & Pacaud, F. (2024). Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods. *Electric Power Systems Research*, *236*, 110651. https://doi.org/10.1016/j.epsr.2024.110651

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y