juliacon

JSOSuite.jl: Solving continuous optimization problems with JuliaSmoothOptimizers

Tangi Migot[1], Dominique Orban[1], and Abel S. Siqueira[2]

[1]GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Canada
[2]Netherlands eScience Center, Amsterdam, Netherlands

ABSTRACT

`JSOSuite.jl` is a new Julia package offering a user-friendly interface for continuous nonlinear optimization. The solvers available cover unconstrained to generally-constrained, and least-squares problems. This new package caters to practitioners as it does not require an understanding of the inner mechanism of solvers, but instead performs a cursory analysis of the problem to match it with an appropriate solver.

Keywords

Julia, Optimization, Nonlinear optimization, Nonlinear programming, Automatic differentiation

1. Introduction

`JSOSuite.jl` is a Julia [2] package to find a local solution of continuous nonlinear optimization problems of the form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x) \text{ subject to } c_L \le c(x) \le c_U,\ \ell \le x \le u, \qquad (1)$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are continuously differentiable, with $c_L \in (\mathbb{R} \cup \{-\infty\})^m$, $c_U \in (\mathbb{R} \cup \{+\infty\})^m$, $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, and $u \in (\mathbb{R} \cup \{+\infty\})^n$. Bounds on the variables appear separately from other types of constraints because numerical methods often treat them differently.

2. Statement of need

`JSOSuite.jl` is part of the JuliaSmoothOptimizers (JSO) ecosystem, an academic organization that offers a collection of Julia packages for nonlinear optimization software development, testing, and benchmarking.
JSO provides a general API for solvers to interact with models by providing flexible data types to represent the objective and constraint functions, evaluate their derivatives, and provide essentially any information that a solver might request from a model. `NLPModels.jl` [15] is the core package that defines the type `AbstractNLPModel` and introduces the API, including in-place and out-of-place evaluation of the objective gradient, sparse Jacobian and Hessian matrices as well as operators for matrix-free implementations. The user can hand-code derivatives, use automatic differentiation [8], or JSO interfaces to optimization modeling languages such as AMPL [6] or JuMP [4]. Hence, solvers can be designed to rely on the API's independently of the problem's origin.
JSO defines a minimal set of rules for a solver to be compliant: (i) The input is an instance of `AbstractNLPModel`; (ii) the output is a `GenericExecutionStats` [12], that contains the solution, optimal value, primal and dual feasibility, elapsed time, etc. There are a growing number of JSO-compliant solvers accessible from registered Julia packages for unconstrained and bound-constrained problems [5, 11], equality-constrained problems [9, 10, 14], and problems of the form (1) in [1, 17]. Most of those solvers also have variants for nonlinear least squares. A solver for convex quadratic programs is in `RipQP.jl` [13]. The solvers mentioned are pure Julia implementations, but there also exist thin wrappers to well-established solvers such as `Artelys Knitro` [3] and `Ipopt` [18].
There exist other packages in Julia that bring together multiple solvers. If (1) can be modeled using `JuMP`, the model may be passed to solvers compatible with `MathOptInterface.jl` [7]. `Optimization.jl` aims to unify local and global optimization packages into a single interface. It adds high-level features, such as integration with automatic differentiation. The aim of `JSOSuite.jl` is to provide an interface for navigating among the JSO-compliant solvers and selecting the best one without needing to know all of them. There is a strong emphasis on justifying the choice of solvers, which helps the overall engineering process from application to modeling and solving a problem. Connecting modeling tools and solvers into a unique entry point simplifies prototyping and choosing the right solver. Moreover, it helps connect up-to-date solvers to other optimization wrappers and applications.

3. Functionalities

The main function exported by `JSOSuite.jl` is `minimize`, which, given an instance of `AbstractNLPModel`, selects an appropriate solver and locally minimizes the problem, e.g. Code 1.

Code 1: `minimize` takes any NLPModels compatible input.

```
using OptimizationProblems, JSOSuite
nlp = OptimizationProblems.PureJuMP.kirby2()
stats = minimize(nlp)
```

Internally, a `DataFrame`, `JSOSuite.optimizers`, contains all the useful information regarding each solver such as its name, package of origin, highest derivative used, type of problems handled, whether it accepts arbitrary arithmetic types, etc. The function `JSOSuite.select_optimizers` analyzes an instance of `AbstractNLPModel` and returns a `DataFrame` with the available solvers, along with the information that guided their selection.
The solvers are then selected by identifying the arithmetic type, type of constraints (unconstrained, bound-constrained, equality-

constrained, inequalities, linear/nonlinear), objective function (linear, quadratic, linear/nonlinear least squares, nonlinear), and highest accessible derivative (1st order, 2nd order, or 2nd order matrix-free). Finally, the user can either choose to call `minimize` with the appropriate solver or use the corresponding package directly. The current strategy is to select the most specialized solver. Further heuristics are being researched to improve the selection process when multiple solvers are available for a given problem.
An important feature common to most JSO-compliant solvers is the possibility to run in-place solve, i.e., it is possible to pre-allocate the output and the storage used during the iterations to run the optimizer allocation free; see Code 2. This is of great interest as it shows memory efficiency and allows re-solving problems without storage overhead.

Code 2: `JSOSuite.solve!` can be used to re-solve without additional memory allocation.

```
using NLPModelsTest, Percival, SolverCore
nlp = NLPModelsTest.HS6()
solver = PercivalSolver(nlp)
stats = GenericExecutionStats(nlp)
solve!(solver, nlp, stats)
SolverCore.reset!(solver)
@allocated solve!(solver, nlp, stats) # = 0
```

A list of parameters common to all JSO-compliant solvers is maintained, and can be passed to `minimize`. Combined with `SolverBenchmark.jl` [16], it can seamlessly compare algorithms and generate data and performance profiles on collection of test problems. Additionally, the package implements strategies that are classical for continuous optimization solvers such as methods to find a feasible initial guess or run a multi-start strategy. The documentation of the package `https://jso.dev/JSOSuite.jl` contains further examples of these functionalities.

4. Concluding remarks

Julia's JIT compiler is ideal for efficient scientific computing and optimization software, making it a natural choice for developing new solvers. The JuliaSmoothOptimizers organization provides a comprehensive set of tools for large-scale continuous optimization, designed for ease of use by practitioners, researchers, and developers. Actively maintained and constantly evolving, these tools include links to cutting-edge external solvers and implementations of promising new ones. `JSOSuite.jl` builds on these features to offer a user-friendly interface that simplifies solver selection and problem-solving in research-level continuous optimization.

Acknowledgement

Tangi Migot is supported by an NSERC Alliance grant 544900-19 in collaboration with Huawei-Canada and NSERC PGS-D, and Dominique Orban is partially supported by an NSERC Grant.

5. References

[1] Egmara Antunes dos Santos, Tangi Migot, Dominique Orban, Abel Soares Siqueira, and contributors. Percival.jl: an augmented Lagrangian method, 2023. doi:10.5281/zenodo.3969045.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi:10.1137/141000671.

[3] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. *Knitro: An integrated package for nonlinear optimization*, pages 35–59. Springer US, Boston, MA, 2006. doi:10.1007/0-387-30065-1.

[4] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi:10.1137/15M1020575.

[5] Jean-Pierre Dussault, Samuel Goyette, Tangi Migot, Dominique Orban, and contributors. AdaptiveRegularization.jl: A unified efficient implementation of trust-region type algorithms for unconstrained optimization, 2023. doi:10.5281/zenodo.10434673.

[6] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL. A modeling language for mathematical programming*. 2nd ed., Brooks/Cole, Pacific Grove, CA, 2003. doi:10.1287/mnsc.36.5.519.

[7] Benoît Legat, Oscar Dowson, Joaquim Dias Garcia, and Miles Lubin. MathOptInterface: a data structure for mathematical optimization problems. *INFORMS Journal on Computing*, 34(2):672–689, 2021. doi:10.1287/ijoc.2021.1067.

[8] Tangi Migot, Alexis Montoison, Dominique Orban, Abel Soares Siqueira, and contributors. ADNLPModels.jl: Automatic differentiation models implementing the NLPModels API, 2023. doi:10.5281/zenodo.4605982.

[9] Tangi Migot, Dominique Orban, and Abel Soares Siqueira. DCISolver.jl: A Julia solver for nonlinear optimization using dynamic control of infeasibility. *Journal of Open Source Software*, 7(70):3991, 2022. doi:10.21105/joss.03991.

[10] Tangi Migot, Dominique Orban, Abel Soares Siqueira, and contributors. FletcherPenaltySolver.jl: Fletcher's penalty method for nonlinear optimization models, 2023. doi:10.5281/zenodo.7153564.

[11] Tangi Migot, Dominique Orban, Abel Soares Siqueira, and contributors. JSOSolvers.jl: JuliaSmoothOptimizers optimization solvers, 2023. doi:10.5281/zenodo.3991143.

[12] Tangi Migot, Dominique Orban, Abel Soares Siqueira, and contributors. SolverCore.jl: Core package to build novel solvers in Julia, 2023. doi:10.5281/zenodo.4758376.

[13] Dominique Orban, Geoffroy Leconte, and contributors. RipQP.jl: Regularized interior point solver for quadratic problems, 2020. doi:10.5281/zenodo.4309783.

[14] Dominique Orban and Abel Soares Siqueira. A regularization method for constrained nonlinear least squares. *Computational Optimization and Applications*, 76(3):961, 2020. doi:10.1007/s10589-020-00201-2.

[15] Dominique Orban, Abel Soares Siqueira, and contributors. NLPModels.jl: Data Structures for Optimization Models, 2023. doi:10.5281/zenodo.2558627.

[16] Dominique Orban, Abel Soares Siqueira, and contributors. SolverBenchmark.jl: Benchmark Tools for Solvers, 2023. doi:10.5281/zenodo.3948381.

[17] Sungho Shin, François Pacaud, and Mihai Anitescu. Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods. *arXiv preprint arXiv:2307.16830*, 2023. doi:10.2139/ssrn.4601442.

[18] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. doi:10.1007/s10107-004-0559-y.