**The Journal of Open Source Software**

Enlsip.jl: A Julia optimization package to solve constrained nonlinear least-squares problems

Pierre Borie [1], **Alain Marcotte** [2], **Fabian Bastin** [1], and **Stéphane Dellacherie** [2,3]

1 Department of Computer Science and Operations Research, University of Montreal, Montreal, QC, Canada **2** Unit of Inflow and Load Forecasting, Hydro-Québec, Montreal, QC, Canada **3** Department of Computer Science, UQÀM, Montreal, QC, Canada

Summary

Enlsip.jl is a Julia (Bezanson et al., 2017) package that implements a solver for nonlinear least-squares problems with nonlinear constraints.

This type of problem is mathematically formulated as:

$$\min_{x \in \mathbb{R}^n} \quad \frac{1}{2} \sum_{i=1}^{m} r_i(x)^2$$
$$\text{s.t.} \quad c_i(x) = 0, \quad i = 1, \dots, q$$
$$\qquad c_i(x) \geq 0, \quad i = q+1, \dots, \ell, \tag{1}$$

where the functions $(r_1, \dots, r_m)$, often denoted as the residuals, and constraints $(c_1, \dots, c_\ell)$ are two-times differentiable.

This package is the Julia version of ENLSIP (Easy Nonlinear Least Squares Inequality Program), an open-source Fortran77 library developed by Lindström & Wedin (1988)[1].

Statement of need

The ENLSIP Fortran77 library has been successfully used since the early 2000s by Hydro-Québec, the main electricity supplier for the province of Quebec in Canada, to calibrate its short-term electricity demand forecast models (Grenier et al., 2006), which are coded in Fortran90. Since Hydro-Québec is transitioning from Fortran77 to Julia and its systems are used in a highly critical context, the primary goal of this transition is to ensure that the replacing Julia version reproduces the results obtained with the original Fortran77 version. The conversion of the above-mentioned ENLSIP library to Julia is a crucial part of this process.

Nonlinear least-squares problems arise in a variety of model calibration scenarios. Formulation (1) is particularly relevant in contexts where additional constraints, such as those related to physical models, need to be satisfied. This is due to the high-risk nature of Hydro-Québec's forecasting operations. Moreover, the specific structure of the objective function in (1) can be exploited to design algorithms more efficient than solvers for general nonlinear programming applied to problem (1).

Comparison of results and performance on operational Hydro-Québec optimization problems have been conducted using a Julia-Fortran interface and they have shown very good concordance results. We additionally compared numerical results on nonlinear programming test problems

---

[1]The source code is available at https://plato.asu.edu/sub/nonlsq.html

(Hock & Schittkowski, 1980; Lukšan & Vlček, 1999) to ascertain whether the two versions could yield significantly disparate outcomes or distinct solutions. On the tested problems, we observed no differences in convergence behavior. Furthermore, the obtained solutions did not differ from a predetermined tolerance, the same one we previously employed to ensure the results of our Julia version were consistent with the requirements of Hydro-Québec.

Method

The `ENLSIP` solver incorporates an iterative Gauss-Newton method designed to find a first-order critical point of problem (1). At each iteration, the algorithm solves an approximation of the original problem obtained after linearizing both residuals and constraints in a small neighborhood of the current point. Then, a subset of constraints, treated as equalities for the ongoing iteration, is formed. It contains all the equality constraints and the inequality constraints that are believed to be active, i.e. satisfied with equality, at the solution. To select the appropriate inequality constraints at each iteration, the authors implemented a strategy that follows the principles outlined in the Chapter 6 of Gill et al. (1981). The resulting subproblem, with a linear least-squares objective and linear equality constraints, is then solved using a null-space type method (Nocedal & Wright, 2006, Chapter 15).

To our knowledge, there is no formal proof of convergence for the method implemented in `ENLSIP`, although local convergence at a linear rate is to be expected from the Gauss-Newton paradigm. In practice, one can observe better performance when the starting point is relatively close to a solution of the problem. However, the algorithm is not suitable for large-scale applications. Indeed, its performance tend to deteriorate on problems with a few hundreds of parameters and constraints ($n + \ell \geq 200$) and more than a thousand residuals ($m \geq 1000$).

From Fortran77 to Julia

Our first motivation to convert `ENLSIP` in Julia was to improve reliability, readability and ease of maintenance of the original code. Also, linear algebra tools in Julia, based on OpenBLAS, benefit from improved implementations than those of the algorithm by Lindström & Wedin (1988), based on MINPACK. Furthermore, this language is highly convenient for optimization, offering various interface tools such as JuMP (Dunning et al., 2017) or NLPModels (Orban et al., 2020), to model optimization problems. Although these libraries are not currently used in our package, they are under consideration for future developments.

Numerical experiments

The performance of the two versions have been compared on problems derived from Hydro-Québec operations and also problems from the literature. We illustrate this comparison on the estimation of CM1 (Carole Mercier-1) model parameters; it is a nonlinear regression model used for the hourly load forecast (Grenier et al., 2006). The calibration process requires the use of weather data collected across the province of Quebec. The configuration of this model can be adapted to the number of parameters to be calibrated and the amount of data to be used. In total, 90 different instances can be thus defined with the following features:

- from 258 to 726 parameters,
- from 468 to 716 total constraints (with 2 to 10 equalities),
- from 4392 to 17,568 residuals.

Due to the differences in how computations are organized between Fortran77 and Julia, especially in linear algebra, minor numerical discrepancies were expected to emerge and accumulate, leading to slightly different outcomes or total number of iterations performed.

Out of 90 instances, 26 either failed to converge or stopped because of a numerical error occurring during the execution of both `ENLSIP` versions. These results were due to inconsistencies in the problem formulation of a given family of instances and thus, they do not reflect

the robustness of the method implemented in ENLSIP. For the 64 remaining instances, both versions reached similar solutions that met Hydro-Québec specifications. Although specific values cannot be disclosed in this paper, the performance profile (Dolan & Moré, 2002) in Figure 1 suggests that our Julia version has similar, if not better, performance than the Fortran77 version. However, these results must be weighted by the number of iterations performed by the two algorithms. Indeed, our Julia version often required less iterations. We argue that the better efficiency of the linear algebra routines from OpenBLAS as compared to those from MINPACK may contribute to shorter computation times.
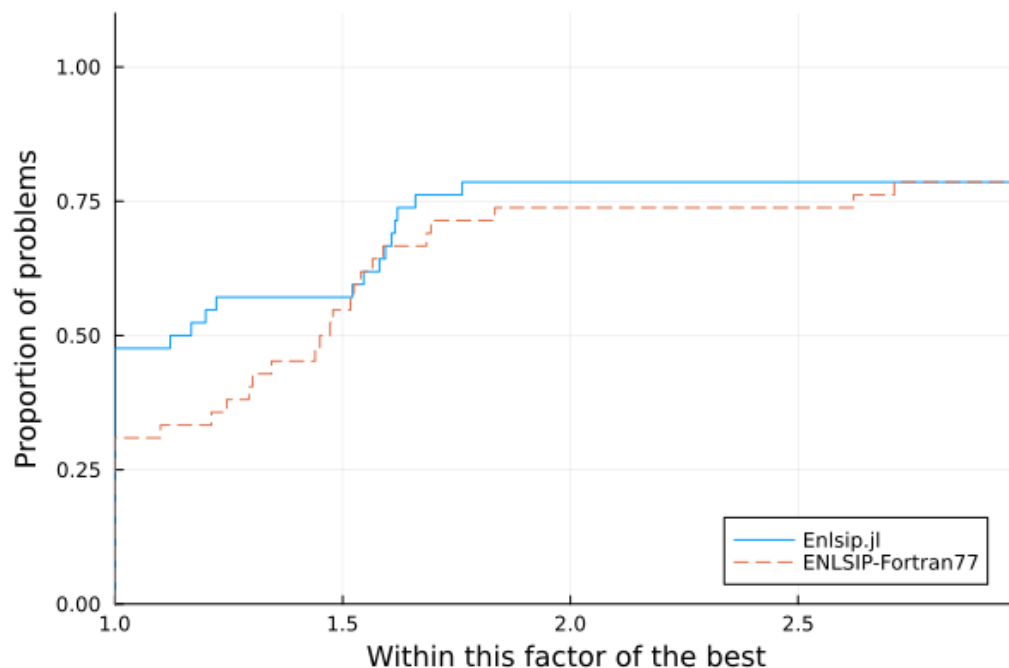

Figure 1: Computation time performance profile of ENLSIP-Fortran77 (dashed curve) and Enlsip.jl (continuous curve) on 90 instances of the CM1 model calibration.

Other nonlinear least-squares packages

Several existing Julia packages can be used to solve nonlinear least-squares problems. For unconstrained and bound-constrained problems, these include NL2sol.jl (Dennis Jr et al., 1981), LsqFit.jl and LeastSquaresOptim.jl. Also available are least-squares variants of the TRUNK and TRON (Lin & Moré, 1999) solvers from JSOSolvers.jl (Migot et al., 2023). The CaNNOLeS.jl (Orban & Siqueira, 2020) package handles the equality-constrained case. However, these libraries do not entirely apply to the formulation stated in (1), for which more general solvers, such as Ipopt.jl (Wächter & Biegler, 2006), are typically used.

Although our package may not incorporate the latest advancements in nonlinear optimization, especially for large-scale problems, its use remains relevant. Indeed, the method employed in ENLSIP manages to exploit the least-squares structure while also remaining very general, covering nonlinearity and non-convexity of the residuals and constraints. This capability renders it an effective tool for solving problems of reasonable dimensions ($n \leq 500$ and $m \leq 1000$). Moreover, in comparison to other categories, like the unconstrained case (as discussed in Dennis Jr & Schnabel, 1996, Chapter 10), this specific class of least-squares problems with general constraints is, to the best of our knowledge, rarely addressed in the literature.

Usage

Enlsip.jl can be dowloaded from the Julia package manager by running the following command into the REPL:

```julia
julia> using Pkg
julia> Pkg.add("Enlsip")
```

The package provides a basic interface for modeling optimization problems of the form (1), by passing the residuals, constraints functions and dimensions of the problem. This is accomplished by creating an instance of our `CnlsModel` structure. Users can also provide functions to compute Jacobian matrices of residuals and constraints, or they can let the algorithm compute them numerically using automatic differentiation[2] (Griewank, 2003).

As a short tutorial, we consider the following problem (Hock & Schittkowski, 1980, problem 65):

$$
\begin{aligned}
\min \quad & (x_1 - x_2)^2 + \frac{1}{9}(x_1 + x_2 - 10)^2 + (x_3 - 5)^2 \\
\text{s.t.} \quad & 48 - x_1^2 - x_2^2 - x_3^2 \geq 0 \\
& -4.5 \leq x_i \leq 4.5, \quad i = 1, 2 \\
& -5 \leq x_3 \leq 5,
\end{aligned}
\tag{2}
$$

and show how to use the package to model and solve problem (2):

```julia
using Enlsip


# Dimensions of the problem

n = 3 # number of parameters
m = 3 # number of residuals
l = 1 # number of nonlinear inequality constraints

# Residuals and Jacobian matrix associated
r(x::Vector) = [x[1] - x[2]; (x[1]+x[2]-10.0) / 3.0; x[3]-5.0]

jac_r(x::Vector) = [1. -1. 0; 1/3 1/3 0.; 0. 0. 1.]

# Constraints (one nonlinear inequality and box constraints)
c(x::Vector) = [48.0 - x[1]^2-x[2]^2-x[3]^2]
jac_c(x::Vector) = [ -2x[1] -2x[2] -2x[3]]
x_l = [-4.5, -4.5, -5.0]
x_u = [4.5, 4.5, 5.0]

# Starting point
x0 = [-5.0, 5.0, 0.0]

# Instantiate the  model associated with the problem
model = Enlsip.CnlsModel(r, n, m; jacobian_residuals=jac_r, starting_point=x0,
        ineq_constraints = c, jacobian_ineqcons=jac_c, nb_ineqcons = l,
        x_low=x_l, x_upp=x_u)
```

Once a model has been instantiated, the solver function can be called.

---

[2]Backend used by default is ForwardDiff.jl

```
# Call of the `solve!` function
Enlsip.solve!(model)
```

After solving, there are available methods to get details about the conduct of the algorithm. For instance, on the example presented above, calling `Enlsip.solution(model)` will return the solution found.

The [online documentation](https://uncertainlab.github.io/Enlsip.jl)[3] provides additional information on how to use the package and examples with test problems from the literature.

Acknowledgements

References

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

Dennis Jr, J. E., Gay, D. M., & Walsh, R. E. (1981). An adaptive nonlinear least-squares algorithm. *ACM Transactions on Mathematical Software*, *7*(3), 348–368. https://doi.org/10.1145/355958.355965

Dennis Jr, J. E., & Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM. https://doi.org/10.1137/1.9781611971200

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, Series A 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

Gill, P. E., Murray, W., & Wright, M. H. (1981). *Practical optimization*. Academic Press. https://doi.org/10.1007/978-0-387-40065-5

Grenier, M., Gagnon, J., Mercier, C., & Richard, J. (2006). Short-term load forecasting at Hydro-Québec TransÉnergie. *2006 IEEE Power Engineering Society General Meeting*. https://doi.org/10.1109/PES.2006.1709029

Griewank, A. (2003). A mathematical view of automatic differentiation. *Acta Numerica*, *12*, 321–398. https://doi.org/10.1017/S0962492902000132

Hock, W., & Schittkowski, K. (1980). *Test examples for nonlinear programming codes* (Second edition, Vol. 187). Springer Berlin. https://doi.org/10.1007/978-3-642-48320-2

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

Lindström, P., & Wedin, P. Å. (1988). Gauss-Newton based algorithms for constrained nonlinear least squares problems. *Technical Report S-901 87, Institute of Information Processing, University of Umeå, Sweden*.

---

[3]https://uncertainlab.github.io/Enlsip.jl

Lukšan, L., & Vlček, J. (1999). Sparse and partially separable test problems for unconstrained and equality constrained optimization. *Technical Report 767, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague.* http://hdl.handle.net/11104/0123965

Migot, T., Orban, D., Siqueira, A. S., & contributors. (2023). *JSOSolvers.jl: JuliaSmoothOptimizers optimization solvers* (Version 0.11.0). https://doi.org/10.5281/zenodo.3991143

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (Second edition). Springer. https://doi.org/10.1007/978-0-387-40065-5

Orban, D., & Siqueira, A. S. (2020). A regularization method for constrained nonlinear least squares. *Computational Optimization and Applications*, *76*(3), 961–989. https://doi.org/10.1007/s10589-020-00201-2

Orban, D., Siqueira, A. S., & contributors. (2020). *NLPModels.jl: Data structures for optimization models*. https://github.com/JuliaSmoothOptimizers/NLPModels.jl. https://doi.org/10.5281/zenodo.2558627

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