DCISolver.jl: A Julia solver for nonlinear optimization using dynamic control of infeasibility

T. Migot, D. Orban, A.S. Siqueira

G–2021–67
December 2021
DCISolver.jl: A Julia solver for nonlinear optimization using dynamic control of infeasibility

Tangi Migot \textsuperscript{a}
Dominique Orban \textsuperscript{a}
Abel S. Siqueira \textsuperscript{b}

\textsuperscript{a} GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 2A7

\textsuperscript{b} Netherlands eScience Center, Amsterdam, Netherlands

tangi.migot@polymtl.ca
dominique.orban@gerad.ca
abel.s.siqueira@gmail.com

December 2021
Les Cahiers du GERAD
G–2021–67
Copyright © 2021 GERAD, Migot, Orban, Siqueira

The authors are exclusively responsible for the content of their research papers published in the series Les Cahiers du GERAD. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

— May download and print one copy of any publication from the public portal for the purpose of private study or research;
— May not further distribute the material or use it for any profit-making activity or commercial gain;
— May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Abstract: This paper presents `DCISolver.jl` a new Julia package implementing the Dynamic Control of Infeasibility method (DCI), introduced by Bielschowsky & Gomes (2008), for solving equality-constrained nonlinear optimization problems.

Keywords: Julia, numerical optimization, nonlinear optimization, nonlinear programming

Résumé: Cet article présente le package Julia `DCISolver.jl` qui implémente la méthode Dynamic Control of Infeasibility, introduite par Bielschowsky & Gomes (2008), pour les problèmes d’optimisation non-linéaire avec contraintes d’égalités.

Mots clés: Julia, optimisation numérique, optimization non-linéaire, programmation non-linéaire

Acknowledgements: Tangi Migot is supported by IVADO and the Canada First Research Excellence Fund / Apogée, and Dominique Orban is partially supported by an NSERC Discovery Grant.
1 Summary

DCISolver.jl is a new Julia implementation of the Dynamic Control of Infeasibility method (DCI), introduced by Bielschowsky & Gomes [2], for solving the equality-constrained nonlinear optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } h(x) = 0,$$

where $f: \mathbb{R}^n \to \mathbb{R}$ and $h: \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable. DCI is an iterative method that aims to compute a local minimum of (1) using first and second-order derivatives. Our initial motivation for developing DCISolver.jl is to solve PDE-constrained optimization problems, many of which have equality constraints only.

Each DCI iteration is a two-step process. A tangential step first approximately minimizes a quadratic model subject to linearized constraints within a trust region. A normal step then recenters feasibility by way of a trust cylinder, which is the set of points such that $\|h(x)\| \leq \rho$, where $\rho > 0$. The idea of trust cylinders is to control infeasibility, contrary to penalty methods, which encourage feasibility by penalizing infeasibility. Each time the trust cylinder is violated during the tangential step, the normal step brings infeasibility back within prescribed limits. The radius $\rho$ of the trust cylinder decreases with the iterations, so a feasible and optimal point results in the limit. For details and theoretical convergence, we refer the reader to the original paper [2].

DCISolver.jl is built upon the JuliaSmoothOptimizers (JSO) tools [11]. 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 problems repositories, and solving subproblems. DCISolver.jl takes as input an AbstractNLPModel, JSO’s general model API defined in NLPModels.jl [17], 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 [8], CUTEst [9], or JuMP [7].

Internally, DCISolver.jl combines cutting-edge numerical linear algebra solvers. The normal step relies heavily on iterative methods for linear algebra from Krylov.jl [14], which provides more than 25 implementations of standard and novel Krylov methods, and they all can be used with Nvidia GPU via CUDA.jl [1]. The tangential step is computed using the sparse factorization of a symmetric and quasi-definite matrix via LDLFactorizations.jl [15], or the well-known Fortran code MA57 [6] from the HSL (2007), via HSL.jl [16].

One of the significant advantages of our implementation is that the normal step is factorization-free, i.e., it uses second-order information via Hessian-vector products but does not need access to the Hessian as an explicit matrix. This makes DCISolver.jl a valuable asset for large-scale problems, for instance to solve PDE-constrained optimization problems [12]. In the current implementation, the tangential step requires the explicit hessian, but removing that restriction is the subject of ongoing research, as is the treatment of inequality constraints.

2 Statement of need

Julia’s JIT compiler is attractive for the design of efficient scientific computing software, and, in particular, mathematical optimization [10], and has become a natural choice for developing new solvers.

There already exist ways to solve (1) in Julia. If (1) is amenable to being modeled in JuMP [7], the model may be passed to state-of-the-art solvers, implemented in low-level compiled languages, via wrappers thanks to Julia’s native interoperability with such languages. However, interfaces to low-level languages have limitations that pure Julia implementations do not have, including the ability to apply solvers with various arithmetic types. Optim.jl [13] implements a factorization-based pure

To the best of our knowledge, there is no available maintained open-source implementation of DCI in existence. The original authors did not make their implementation public, and the other known implementation is dcicpp [20], extending the original method to inequalities in the Ph.D. thesis by Siqueira [19], and it has had no updates in the last 5 years. Hence, we offer an interesting alternative to augmented Lagrangian and interior-point methods in the form of an evolving, research level yet stable solver.

DCISolver.jl can solve large-scale problems and can be benchmarked easily against other JSO-compliant solvers using SolverBenchmark.jl [18]. We include below performance profiles [4] of DCISolver.jl against Ipopt on 82 problems from CUTEst [9] with up to 10 000 variables and 10 000 constraints. Ipopt solved 72 problems (88%) successfully, which is one more than DCI. Without explaining performance profiles in full detail, the plot on the left shows that Ipopt is the fastest on 20 of the problems (28%), while DCI is the fastest on 51 of the problems (72%) among the 71 problems solved by both solvers. The plot on the right shows that Ipopt used fewer evaluations of objective and constraint functions on 50 of the problems (70%), DCI used fewer evaluations on 17 of the problems (24%), and there was a tie in the number of evaluations on 4 problems (6%). Overall, this performance profile is very encouraging for such a young implementation. The package’s documentation includes more extensive benchmarks on classical test sets showing that DCISolver.jl is also competitive with Artelys Knitro.

Références


