DCISolver.jl: A Julia Solver for Nonlinear Optimization using Dynamic Control of Infeasibility

Tangi Migot[*1], **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, NL

Summary

`DCISolver.jl` is a new Julia implementation of the Dynamic Control of Infeasibility method (DCI), introduced by Bielschowsky & Gomes (2008), for solving the equality-constrained nonlinear optimization problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad h(x) = 0, \tag{1}$$

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 $\|h(x)\| \le \rho$, 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 ρ 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 (Bielschowsky & Gomes, 2008).

`DCISolver.jl` is built upon the JuliaSmoothOptimizers (JSO) tools (Migot et al., 2021b). 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` (Orban et al., 2020a), 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., 2003), CUTEst (Gould et al., 2015), or JuMP (Dunning et al., 2017).

Internally, `DCISolver.jl` combines cutting-edge numerical linear algebra solvers. The normal step relies heavily on iterative methods for linear algebra from `Krylov.jl` (Montoison et al., 2020), which provides more than 25 implementations of standard and novel Krylov methods, and they all can be used with Nvidia GPU via CUDA.jl (Besard et al., 2018). The tangential step is computed using the sparse factorization of a symmetric and quasi-definite matrix via `LDLFactorizations.jl` (Orban & contributors, 2020), or the well-known Fortran code MA57 (Duff, 2004) from the HSL (2007), via `HSL.jl` (Orban & contributors, 2021).

---

*corresponding author

40 One of the significant advantages of our implementation is that the normal step is
41 factorization-free, i.e., it uses second-order information via Hessian-vector products but does
42 not need access to the Hessian as an explicit matrix. This makes `DCISolver.jl` a valuable
43 asset for large-scale problems, for instance to solve PDE-constrained optimization problems
44 (Migot et al., 2021a). In the current implementation, the tangential step requires the explicit
45 hessian, but removing that restriction is the subject of ongoing research, as is the treatment
46 of inequality constraints.

Statement of need

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

51 There already exist ways to solve (1) in Julia. If (1) is amenable to being modeled in JuMP
52 (Dunning et al., 2017), the model may be passed to state-of-the-art solvers, implemented
53 in low-level compiled languages, via wrappers thanks to Julia's native interoperability with
54 such languages. However, interfaces to low-level languages have limitations that pure Julia
55 implementations do not have, including the ability to apply solvers with various arithmetic
56 types. `Optim.jl` (Mogensen & Riseth, 2018) implements a factorization-based pure Julia
57 primal-dual interior-point method for problems with both equality and inequality constraints
58 modeled after Artelys Knitro (Byrd et al., 2006) and Ipopt (Wächter & Biegler, 2006).
59 `Percival.jl` (Santos & Siqueira, 2020) is a factorization-free pure Julia implementation of
60 an augmented Lagrangian method for problems with both equality and inequality constraints
61 based on bound-constrained subproblems.

62 To the best of our knowledge, there is no available maintained open-source implementation
63 of DCI in existence. The original authors did not make their implementation public, and the
64 other known implementation is `dcicpp` (Siqueira, 2016), extending the original method to
65 inequalities in the Ph.D. thesis by Siqueira (2013), and it has had no updates in the last 5
66 years. Hence, we offer an interesting alternative to augmented Lagrangian and interior-point
67 methods in the form of an evolving, research level yet stable solver.

68 `DCISolver.jl` can solve large-scale problems and can be benchmarked easily against other
69 JSO-compliant solvers using `SolverBenchmark.jl` (Orban et al., 2020b). We include below
70 performance profiles (Dolan & Moré, 2002) of `DCISolver.jl` against Ipopt on 82 problems
71 from CUTEst (Gould et al., 2015) with up to 10 000 variables and 10 000 constraints. Ipopt
72 solved 72 problems (88%) successfully, which is one more than DCI. Without explaining
73 performance profiles in full detail, the plot on the left shows that Ipopt is the fastest on 20 of the
74 problems (28%), while DCI is the fastest on 51 of the problems (72%) among the 71 problems
75 solved by both solvers. The plot on the right shows that Ipopt used fewer evaluations of
76 objective and constraint functions on 50 of the problems (70%), DCI used fewer evaluations on
77 17 of the problems (24%), and there was a tie in the number of evaluations on 4 problems (6%).
78 Overall, this performance profile is very encouraging for such a young implementation. The
79 package's documentation includes more extensive benchmarks on classical test sets showing
80 that `DCISolver.jl` is also competitive with Artelys Knitro.
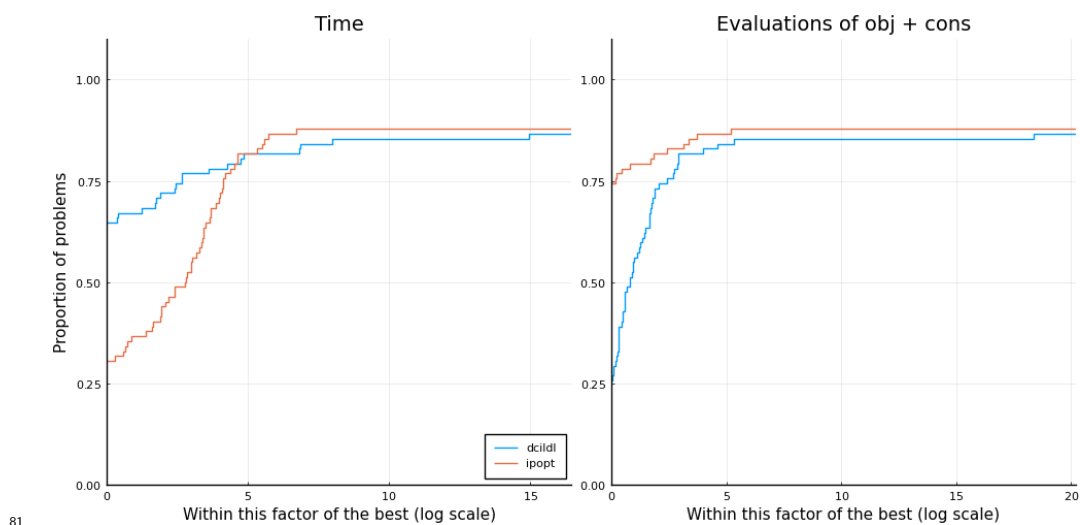
Time — Evaluations of obj + cons

References

Besard, T., Foket, C., & De Sutter, B. (2018). Effective extensible programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*. https://doi.org/10.1109/TPDS.2018.2872064

Bielschowsky, R. H., & Gomes, F. A. M. (2008). Dynamic control of infeasibility in equality constrained optimization. *SIAM Journal on Optimization*, *19*(3), 1299–1325. https://doi.org/10.1137/070679557

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

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

Duff, I. S. (2004). MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, *30*(2), 118–144. https://doi.org/10.1145/992200.992202

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

Fourer, R., Gay, D. M., & Kernighan, B. W. (2003). *AMPL. A modeling language for mathematical programming.* 2nd ed., Brooks/Cole, Pacific Grove, CA. https://doi.org/10.1287/mnsc.36.5.519

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

109 HSL. (2007). *The HSL mathematical software library*. STFC Rutherford Appleton Laboratory.
110 http://www.hsl.rl.ac.uk

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

113 Migot, T., Orban, D., & Siqueira, A. S. (2021a). *PDENLPModels.jl: A NLPModel API for*
114 *optimization problems with PDE-constraints*. https://doi.org/10.5281/zenodo.5056629

115 Migot, T., Orban, D., & Siqueira, A. S. (2021b). *The JuliaSmoothOptimizers ecosystem for*
116 *linear and nonlinear optimization*. https://doi.org/10.5281/zenodo.2655082

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

119 Montoison, A., Orban, D., & contributors. (2020). *Krylov.jl: A Julia basket of hand-picked*
120 *Krylov methods*. https://doi.org/10.5281/zenodo.822073

121 Orban, D., & contributors. (2020). *LDLFactorizations.jl: Factorization of symmetric matrices*.
122 https://doi.org/10.5281/zenodo.3900668

123 Orban, D., & contributors. (2021). *HSL.jl: A julia interface to the HSL mathematical software*
124 *library*. https://doi.org/10.5281/zenodo.2658672

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

127 Orban, D., Siqueira, A. S., & contributors. (2020b). *SolverBenchmark.jl: Benchmark tools*
128 *for solvers*. https://doi.org/10.5281/zenodo.3948381

129 Santos, E. A. dos, & Siqueira, A. S. (2020). *Percival.jl: An augmented lagrangian method*.
130 https://doi.org/10.5281/zenodo.3969045

131 Siqueira, A. S. (2013). *Controle dinâmico de infactibilidade para programação não linear* [PhD
132 thesis, Universidade Estadual de Campinas]. https://doi.org/10.47749/T/UNICAMP.
133 2013.918998

134 Siqueira, A. S. (2016). Dcicpp: Dynamic control of infeasibility. In *GitHub repository*. GitHub.
135 https://github.com/abelsiqueira/dcicpp

136 Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter
137 line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*,
138 *106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y