The Journal of Open Source Software

PDENLPModels.jl: An NLPModel API for Optimization Problems with PDE-Constraints

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 ¶ Corresponding author

DOI: 10.21105/joss.04736

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @jmejia8
- @HaoZeke

Submitted: 31 August 2022
Published: 16 December 2022

Summary

Shape optimization, optimal control, and parameter estimation of systems governed by partial differential equations (PDE) give rise to a class of problems known as PDE-constrained optimization (Hinze et al., 2008). `PDENLPModels.jl` is a Julia (Bezanson et al., 2017) package for modeling and discretizing optimization problems with mixed algebraic and PDE in the constraints. The general form of the problems over some domain $\Omega \subset \mathbb{R}^d$ is

$$\underset{y,u,\theta}{\text{minimize}} \int_\Omega J(y,u,\theta)d\Omega \quad \text{subject to} \quad \begin{array}{ll} e(y,u,\theta) = 0, & \text{(governing PDE on } \Omega) \\ l_{yu} \leq (y,u) \leq u_{yu}, & \text{(functional bound constraints)} \\ l_\theta \leq \theta \leq u_\theta, & \text{(bound constraints)} \end{array}$$

where $y : \Omega \to \mathcal{Y}$ is the state, $u : \Omega \to \mathcal{U}$ is the control, and $\theta \in \mathbb{R}^k$ are algebraic variables. $J : \mathcal{Y} \times \mathcal{U} \times \mathbb{R}^k \to \mathbb{R}$ and $e : \mathcal{Y} \times \mathcal{U} \times \mathbb{R}^k \to \mathcal{C}$ are smooth mappings. $(\mathcal{Y}, \|\cdot\|_\mathcal{Y})$, $(\mathcal{U}, \|\cdot\|_\mathcal{U})$, and $(\mathcal{C}, \|\cdot\|_\mathcal{C})$ are real Banach spaces, $l_\theta, u_\theta \in \mathbb{R}^k$ are bounds on θ, and $l_{yu}, u_{yu} : \Omega \to \mathcal{Y} \times \mathcal{U}$ are functional bounds on the controls and states.

After discretization of the domain Ω, the integral, and the derivatives, the resulting problem is a nonlinear optimization problem of the form

$$\underset{x \in \mathbb{R}^{N_y+N_u+N_\theta}}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad c(x) = 0, \quad l \leq x \leq u,$$

where $l, u \in \mathbb{R}^{N_y+N_u+N_\theta}$, $f : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \times \mathbb{R}^{N_\theta} \to \mathbb{R}$ and $c : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \times \mathbb{R}^{N_\theta} \to \mathbb{R}^{N_y}$.

The two main challenges in modeling such a problem are to be able to (i) discretize the domain and generate corresponding discretizations of the objective and constraints, and (ii) evaluate derivatives of f and c with respect to all variables. Several packages allow the user to define the domain, meshes, function spaces, and finite-element families to approximate unknowns, and to model functionals and sets of PDEs in a weak form using powerful high-level notation. The main ones are `FEniCS.jl`, a wrapper for the FEniCS library (Logg et al., 2012), `Ferrite.jl` (Carlsson et al., 2021), `FinEtools.jl` (Krysl, 2021), `JuliaFEM.jl` (Aho et al., 2018) (Aho et al., 2019), and `Gridap.jl` (Badia & Verdugo, 2020) (Verdugo & Badia, 2022). In `PDENLPModels.jl`, we focus on the latter as it is exclusively written in Julia and supports a variety of discretizations and meshing possibilities. Additionally, `Gridap.jl` has an expressive API allowing one to model complex PDEs with few lines of code, and to write the underlying weak form with a syntax that is almost one-to-one with mathematical notation.

However, the above packages are designed for sets of PDEs and not for optimization, so that only the derivatives of the PDE with respect to y can be evaluated. In addition, inequalities are not supported. `PDENLPModels.jl` extends `Gridap.jl`'s differentiation facilities to also obtain derivatives useful for optimization, i.e., first and second derivatives of the objective and constraint functions with respect to controls and finite-dimensional variables. Because

we aim to solve nonconvex optimization problems with inequality constraints, it would not be appropriate, or even feasible, to solve a system of PDEs representing the optimality conditions with `Gridap.jl`.

`PDENLPModels.jl` exports the `GridapPDENLPModel` type, which uses `Gridap.jl` for the discretization of the functional spaces by finite-elements. The resulting model is an instance of an `AbstractNLPModel`, as defined in `NLPModels.jl` (Orban et al., 2020d), that provides access to objective and constraint function values, to their first and second derivatives, and to any information that a solver might request from a model.

The role of `NLPModels.jl` is to define an API that users and solvers can rely on. It is the role of other packages to implement facilities that create models compliant with the NLPModels API. There are several examples of this in JuliaSmoothOptimizers organization: `AmplNLReader.jl` (Orban et al., 2020a) allows users to connect the AMPL modeling language with NLPModels, `CUTEst.jl` (Orban et al., 2020b) does the same for the SIF language, `NLPModelsJuMP.jl` (Montoison et al., 2020) does the same with JuMP models, etc. In those three examples, there exist underlying modeling tools (in Julia or not). `PDENLPModels.jl` is different in that there is no existing generic interface for optimization with PDEs. Instead, `PDENLPModels.jl` interacts with `Gridap.jl` to evaluate functionals and differential operators based on a discretization. `Gridap.jl` in itself does not let users model optimization problems; only systems of PDEs. `PDENLPModels.jl` provides all the extra facilities for users and solvers to interact with a PDE-constrained optimization problem as they would with a JuMP model, an AMPL model, or any other model that complies with the NLPModels API. As such, `PDENLPModels.jl` offers an interface between generic PDE-constrained optimization problems and cutting-edge optimization solvers such as `Artelys Knitro` (Byrd et al., 2006) via `NLPModelsKnitro.jl` (Orban et al., 2020e), `Ipopt` (Wächter & Biegler, 2006) via `NLPModelsIpopt.jl` (Orban et al., 2020c), `DCISolver.jl` (Migot et al., 2022), `Percival.jl` (Santos & Siqueira, 2020), and any solver accepting an `AbstractNLPModel` as input, see JuliaSmoothOptimizers (JSO) (Migot et al., 2021).

The following example shows how to solve a Poisson control problem with Dirichlet boundary conditions using `DCISolver.jl`:

$$\underset{y,u}{\text{minimize}} \int_{(-1,1)^2} \frac{1}{2}\|y_d - y\|^2 + \frac{\alpha}{2}\|u\|^2 d\Omega \quad \text{subject to} \quad \begin{aligned} &\Delta y - u - h = 0, && \text{on } \Omega. \\ &y = 0, && \text{on } \partial\Omega, \end{aligned}$$

for some given functions $y_d, h : (-1,1)^2 \to \mathbb{R}$, and $\alpha > 0$.

```julia
using DCISolver, Gridap, PDENLPModels
Ω = (-1, 1, -1, 1) # Cartesian discretization of Ω=(-1,1)² in 100² squares.
model = CartesianDiscreteModel(Ω, (100, 100))
fe_y = ReferenceFE(lagrangian, Float64, 2) # Finite-elements for the state
Xpde = TestFESpace(model, fe_y; dirichlet_tags = "boundary")
Ypde = TrialFESpace(Xpde, x -> 0.0) # y is 0 over ∂Ω
fe_u = ReferenceFE(lagrangian, Float64, 1) # Finite-elements for the control
Xcon = TestFESpace(model, fe_u)
Ycon = TrialFESpace(Xcon)
dΩ = Measure(Triangulation(model), 1) # Gridap's integration machinery
# Define the objective function f
yd(x) = -x[1]^2
f(y, u) = ∫(0.5 * (yd - y) * (yd - y) + 0.5 * 1e-2 * u * u) * dΩ
# Define the constraint operator in weak form
h(x) = -sin(7π / 8 * x[1]) * sin(7π / 8 * x[2])
c(y, u, v) = ∫(∇(v) ⊙ ∇(y) - v * u - v * h) * dΩ
# Define an initial guess for the discretized problem
x0 = zeros(num_free_dofs(Ypde) + num_free_dofs(Ycon))
# Build a GridapPDENLPModel, which implements the NLPModel API.
name = "Control elastic membrane"
nlp = GridapPDENLPModel(x0, f, dΩ, Ypde, Ycon, Xpde, Xcon, c, name = name)
dci(nlp, verbose = 1) # solve the problem with DCI
```

Statement of need

For PDEs, there are five main ways to discretize functions and their derivatives:

- Finite-difference methods: functions are represented on a grid, e.g., `DiffEqOperators.jl` (Rackauckas & Nie, 2017) or Trixi.jl (Schlottke-Lakemper et al., 2020);
- Finite-volume methods: functions are represented by a discretization of their integral;
- Spectral methods: functions are expanded in a global basis, e.g., `FFTW.jl` (Frigo & Johnson, 2005) and `ApproxFun.jl` (Olver & Townsend, 2014);
- Physics-informed neural networks: functions are represented by neural networks, e.g., `NeuralPDE.jl` (Zubov et al., 2021);
- Finite-element methods: functions are expanded in a local basis.

With finite-elements discretization, it is easy to increase the order of the elements or locally refine the mesh so that the physical fields can be approximated accurately. Another advantage is that you can straightforwardly combine different kinds of approximation functions, leading to mixed formulations. Finally, curved or irregular geometries of the domain are handled in a natural way.

Outside of Julia, there exist libraries handling finite-elements methods such as `deal.II` (Bangerth et al., 2007), `FEniCS` (Logg et al., 2012), `PETSc` (Balay et al., 2021), and `FreeFEM++` (Hecht, 2012). There exists a Julia wrapper to `FEniCS` (Rackauckas & Nie, 2017) and `PETSc` (Crean et al., 2021). However, interfaces to low-level libraries have limitations that pure Julia implementations do not have, including the ability to generate models with various arithmetic types.

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 modeling tools. There are other packages available in Julia for optimization problems with PDE in the constraints. `jInv.jl` (Ruthotto et al., 2017) and `ADCME.jl` (Xu & Darve, 2020) focus on inverse problems. `DifferentialEquations.jl` (Rackauckas & Nie, 2017) is a suite for numerically solving differential equations written in Julia, which includes features for parameter estimation and Bayesian analysis. `InfiniteOpt.jl` (Pulsipher et al., 2022) provides a general mathematical abstraction to express and solve infinite-dimensional optimization problems including with PDEs in the constraints handled by finite-differences. `TopOpt.jl` (Huang & Tarek, 2021) is a package for topology optimization. However, to the best of our knowledge, there are no packages with the generality of `PDENLPModels.jl`.

Optimization problems with PDEs in the constraints have been in the spotlight in recent years as challenging and highly structured. The great divide between optimization libraries and PDE libraries makes it difficult for optimization research to benefit from testing on a large base of PDE-constrained problems and PDE libraries to benefit from the latest advances in optimization. `PDENLPModels.jl` fills this gap by providing generic discretized models that can be solved by any solver from JuliaSmoothOptimizers.

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.

References

Aho, J., Frondelius, T., Ovainola, Arilaakk, Kelman, T., Stoian, V., Badger, T. G., & Rapo, M. (2018). *JuliaFEM/JuliaFEM.jl: Julia V1 compatible release*. Zenodo. https://doi.org/10.5281/zenodo.1410189

Aho, J., Vuotikka, A.-J., & Frondelius, T. (2019). Introduction to JuliaFEM an open-source FEM solver. *Rakenteiden Mekaniikka*, *52*, 148–159. https://doi.org/10.23998/rm.75103

Badia, S., & Verdugo, F. (2020). Gridap: An extensible finite element toolbox in Julia. *Journal of Open Source Software*, *5*(52), 2520. https://doi.org/10.21105/joss.02520

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Karpeyev, D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., … Zhang, H. (2021). *PETSc users manual* (ANL-95/11 - Revision 3.15). Argonne National Laboratory. https://www.mcs.anl.gov/petsc

Bangerth, W., Hartmann, R., & Kanschat, G. (2007). Deal. II—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, *33*(4), 24–es. https://doi.org/10.1145/1268776.1268779

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 G. Di Pillo & M. Roma (Eds.), *Large-scale nonlinear optimization* (pp. 35–59). Springer US. https://doi.org/10.1007/0-387-30065-1_4

Carlsson, K., Ekre, F., & Contributors. (2021). *Ferrite.jl* (Version 0.3.0) [Computer software]. https://github.com/Ferrite-FEM/Ferrite.jl

Crean, J., Kozdon, J. E., Hyatt, K., & Contributors. (2021). *PETSc.jl* (Version 0.1.3) [Computer software]. https://github.com/JuliaParallel/PETSc.jl

Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, *93*(2), 216–231. https://doi.org/10.1109/JPROC.2004.840301

Hecht, F. (2012). New development in FreeFem++. *Journal of Numerical Mathematics*, *20*(3-4), 251–266. https://doi.org/10.1515/jnum-2012-0013

Hinze, M., Pinnau, R., Ulbrich, M., & Ulbrich, S. (2008). *Optimization with PDE constraints* (Vol. 23). Springer Science & Business Media.

Huang, Y., & Tarek, M. (2021). TopOpt.jl: Truss and continuum topology optimization, interactive visualization, automatic differentiation and more. *Proceedings of the 14th World Congress of Structural and Multidisciplinary Optimization*. https://github.com/JuliaTopOpt/TopOpt.jl

Krysl, P. (2021). *FinEtools.jl* (Version 5.3.1) [Computer software]. https://doi.org/10.5281/zenodo.7187507

Logg, A., Mardal, K.-A., & Wells, G. (2012). *Automated solution of differential equations by the finite element method: The FEniCS book* (Vol. 84). Springer Science & Business Media. https://doi.org/10.1007/978-3-642-23099-8

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., & Siqueira, A. S. (2021). *The JuliaSmoothOptimizers ecosystem for linear and nonlinear optimization*. https://doi.org/10.5281/zenodo.2655082

Migot, T., Orban, D., & Siqueira, A. S. (2022). DCISolver.jl: A Julia solver for nonlinear optimization using dynamic control of infeasibility. *Journal of Open Source Software*, *7*(70), 3991. https://doi.org/10.21105/joss.03991

Montoison, A., Orban, D., & Siqueira, A. S. (2020). *NLPModelsJuMP.jl: Conversion from JuMP models to NLPModels*. https://doi.org/10.5281/zenodo.2574162

Olver, S., & Townsend, A. (2014). A practical framework for infinite-dimensional linear algebra. *Proceedings of the 1st Workshop for High Performance Technical Computing in Dynamic Languages – HPTCDL '14*. https://doi.org/10.1109/HPTCDL.2014.10

Orban, D., Siqueira, A. S., & contributors. (2020a). *AmplNLReader.jl: A Julia interface to AMPL*. https://doi.org/10.5281/zenodo.3700941

Orban, D., Siqueira, A. S., & contributors. (2020b). *CUTEst.jl: Julia's CUTEst interface*. https://doi.org/10.5281/zenodo.1188851

Orban, D., Siqueira, A. S., & contributors. (2020c). *NLPModelsIpopt.jl: A thin IPOPT wrapper for NLPModels*. https://doi.org/10.5281/zenodo.2629034

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

Orban, D., Siqueira, A. S., & contributors. (2020e). *NLPModelsKnitro.jl: A thin KNITRO wrapper for NLPModels*. https://doi.org/10.5281/zenodo.3994983

Pulsipher, J. L., Zhang, W., Hongisto, T. J., & Zavala, V. M. (2022). A unifying modeling abstraction for infinite-dimensional optimization. *Computers & Chemical Engineering*, *156*. https://doi.org/10.1016/j.compchemeng.2021.107567

Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.151

Ruthotto, L., Treister, E., & Haber, E. (2017). jInv – a flexible Julia package for PDE parameter estimation. *SIAM Journal on Scientific Computing*, *39*(5), S702–S722. https://doi.org/10.1137/16M1081063

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

Schlottke-Lakemper, M., Gassner, G. J., Ranocha, H., & Winters, A. R. (2020). *Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*. https://doi.org/10.5281/zenodo.3996439

Verdugo, F., & Badia, S. (2022). The software design of Gridap: A finite element package based on the Julia JIT compiler. *Computer Physics Communications*, 108341. https://doi.org/10.1016/j.cpc.2022.108341

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

Xu, K., & Darve, E. (2020). ADCME: Learning spatially-varying physical fields using deep neural networks. *arXiv Preprint arXiv:2011.11955*. https://github.com/kailaix/ADCME.jl

Zubov, K., McCarthy, Z., Ma, Y., Calisto, F., Pagliarino, V., Azeglio, S., Bottero, L., Luján, E., Sulzer, V., Bharambe, A., & others. (2021). NeuralPDE: Automating physics-informed neural networks (PINNs) with error approximations. *arXiv Preprint arXiv:2107.09443*. https://github.com/SciML/NeuralPDE.jl