




PyProximal - scalable convex optimization in Python

Matteo Ravasi ^{1¶}, Marcus Valtonen Örnham ², Nick Luiken ¹, Olivier Leblanc ³, and Eneko Uruñuela ⁴

¹ Earth Science and Engineering, Physical Sciences and Engineering (PSE), King Abdullah University of Science and Technology (KAUST), Thuwal, Kingdom of Saudi Arabia ² Ericsson Research, Lund, Sweden. ³ ISPGROUP, INMA/ICTEAM, UCLouvain, Louvain-la-Neuve, Belgium. ⁴ Basque Center on Cognition, Brain and Language (BCBL), Donostia-San Sebastián, Spain. ¶ Corresponding author

DOI: [10.21105/joss.06326](https://doi.org/10.21105/joss.06326)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Stefan Appelhoff](#) 

Reviewers:

- [@nirum](#)
- [@ewu63](#)

Submitted: 31 December 2023

Published: 12 March 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

A broad class of problems in scientific disciplines ranging from image processing and astrophysics, to geophysics and medical imaging call for the optimization of convex, non-smooth objective functions. Whereas practitioners are usually familiar with gradient-based algorithms, commonly used to solve unconstrained, smooth optimization problems, proximal algorithms can be viewed as analogous tools for non-smooth and possibly constrained versions of such problems. These algorithms sit at a higher level of abstraction than gradient-based algorithms and require a basic operation to be performed at each iteration: the evaluation of the so-called proximal operator of the functional to be optimized. PyProximal is a Python-based library aimed at democratizing the application of convex optimization to scientific problems; it provides the required building blocks (i.e., proximal operators and algorithms) to define and solve complex, convex objective functions in a high-level, abstract fashion, shielding users away from any unneeded mathematical and implementation details.

Statement of need

PyProximal is a Python library for convex optimization, developed as an integral part of the PyLops framework. It provides practitioners with an easy-to-use framework to define and solve composite convex objective functions arising in many modern inverse problems. Its API is designed to offer a class-based and user-friendly interface to proximal operators, coupled with function-based optimizers; because of its modular design, researchers in the field of convex optimization can also benefit from this library in a number of ways when developing new algorithms: first, they can easily include their newly developed proximal operators and solvers; second, they can compare these methods with state-of-the-art algorithms already provided in the library.

Several projects in the Python ecosystem provide implementations of proximal operators and/or algorithms, which present some overlap with those available in PyProximal. A (possibly not exhaustive) list of other projects is composed of *proxalgs* ([Maheswaranathan et al., 2019](#)), *proxmin* ([Moolekamp & Melchior, 2018](#)), *The Proximity Operator Repository* ([Chierchia et al., 2024](#)), *ProxImaL* ([Heide et al., 2016](#)), and *pyxu* ([Simeoni et al., 2024](#)). A key common feature of all of the above mentioned packages is to be self-contained; as such, not only proximal operators and solvers are provided, but also linear operators that are useful for the applications that the package targets. Moreover, to the best of our knowledge, all of these packages provide purely CPU-based implementations (apart from *pyxu*). On the other hand, PyProximal heavily relies on and seamlessly integrates with PyLops ([Ravasi & Vasconcelos, 2020](#)), a Python library for matrix-free linear algebra and optimization. As such, it can easily handle problems with millions of unknowns and inherits the interchangeable CPU/GPU backend of PyLops ([Ravasi,](#)

2021). More specifically, PyLops is leveraged in the implementation of proximal operators that require access to linear operators (e.g., numerical derivatives) and/or least-squares solvers (e.g., conjugate gradient). Whilst libraries with similar capabilities exist in the Python ecosystem, their design usually leads to a tight coupling between linear and proximal operators, and their respective solvers. On the other hand, by following the Separation of Concerns (SoC) design principle, the overlap between PyLops and PyProximal is reduced to a minimum, easing both their development and maintenance, as well as allowing newcomers to learn how to solve inverse problems in a step-by-step fashion. As such, PyProximal can be ultimately described as a light-weight extension of PyLops that users of the latter can easily learn and adopt with minimal additional effort.

Mathematical framework

Convex optimization is routinely used to solve problems of the form (Parikh, 2013):

$$\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{L}\mathbf{x}) \quad (1)$$

where f and g are possibly non-smooth convex functionals and \mathbf{L} is a linear operator. A special case appearing in many scientific applications is represented by $f = 1/2\|\mathbf{y} - \mathcal{A}(\mathbf{x})\|_2^2$. Here, \mathcal{A} is a (possibly non-linear) modeling operator, describing the underlying physical process that links the unknown model vector \mathbf{x} to the vector of observations \mathbf{y} . In this case, we usually refer to g as the regularizer, where one or multiple functions are added to the data misfit term to promote certain features in the sought after solution and/or constraint the solution to fall within a given set of allowed vectors.

A common feature of all proximal algorithms is represented by the fact that one must be able to repeatedly evaluate the proximal operator of f and/or g . The proximal operator of a function f is defined as

$$prox_{\tau f}(\mathbf{x}) = \min_{\mathbf{y}} f(\mathbf{y}) + \frac{1}{2\tau}\|\mathbf{y} - \mathbf{x}\|_2^2 \quad (2)$$

Whilst evaluating a proximal operator does itself require solving an optimization problem, these problems often admit closed form solutions or can be solved very efficiently with ad-hoc specialized methods. Several of such proximal operators are efficiently implemented in the PyProximal library.

Finally, there exists three main families of proximal algorithms that can be used to solve various flavors of Equation 1, namely:

- Proximal Gradient (Combettes & Pesquet, 2011): this method, also commonly referred to as the Forward-Backward Splitting (FBS) algorithm, is usually the preferred choice when $\mathbf{L} = \mathbf{I}$ (i.e. identity operator). Accelerated versions such as the FISTA and TwIST algorithms exist and are usually preferred to the vanilla FBS method;
- Alternating Direction Method of Multipliers (Boyd et al., 2011): this method is based on a splitting strategy and can be used for a broader class of problem than FBS and its accelerated versions.
- Primal-Dual (Chambolle & Pock, 2011): another popular algorithm able to tackle problems in the form of Equation 1 with any choice of \mathbf{L} . It reformulates the original problem into its primal-dual version and solves a saddle optimization problem.

PyProximal provides implementations for these three families of algorithms; moreover, all solvers include additional features such as back-tracking for automatic selection of step-sizes, logging of the objective function evolution through iterations, and possibility to inject custom callbacks.

Code structure

PyProximal's modular and easy-to-use Application Programming Interface (API) allows scientists to define and solve convex objective functions by means of proximal algorithms. The API is composed of two main parts as shown in Fig. 1.

The first part contains the entire suite of proximal operators, which are class-based objects subclassing the `pylops.ProxOperator` parent class. For each of these operators, the solution to the proximal optimization problem in Equation 2 (and/or the dual proximal problem) is implemented in the `prox` (and/or `dualprox`) method. As in most cases a closed-form solution exists for such a problem, our implementation provides users with the most efficient way to evaluate a proximal operator. The second part comprises of so-called proximal solvers, optimization algorithms that are suited to solve problems of the form in Equation 1. Finally, some specialized solvers that rely on one or more of the previously described optimizers are also provided.

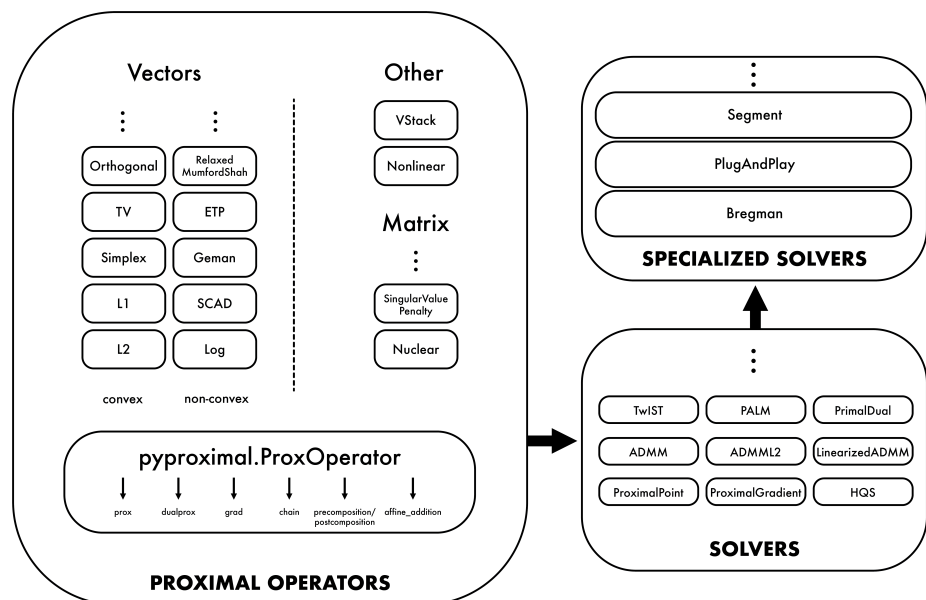


Figure 1: Schematic representation of the PyProximal API.

Representative PyProximal Use Cases

Examples of PyProximal applications in different scientific fields include:

- *Joint inversion and segmentation of subsurface models*: when inverting geophysical data for subsurface priorities, prior information can be provided to inversion process in the form of discrete number of rock units; this can be parametrized in terms of their expected mean (or most likely value). Ravasi & Birnie (2022) and Romero et al. (2023) frame such a problem as a joint inversion and segmentation task, where the underlying objective function is optimized in alternating fashion using the Primal-Dual algorithm.
- *Plug-and-Play (PnP) priors*: introduced in 2013 by Venkatakrishnan et al. (2013), the PnP framework lays its foundation on the interpretation of the proximal operator as a denoising problem; as such, powerful statistical or deep learning based denoisers are used to evaluate the proximal operator of implicit regularizers. Romero et al. (2022) applies this concept in the context of seismic inversion, achieving results of superior quality in comparison to traditional model-based regularization techniques.

- *Multi-Core Fiber Lensless Imaging* (MCFLI) is a computational imaging technique to reconstruct biological samples at cellular scale. Leveraging the rank-one projected interferometric sensing of the MCFLI has been shown to improve the efficiency of the acquisition process (Leblanc et al., 2023); this entails solving a regularized inverse problem with the proximal gradient method. Depending on the image to be reconstructed, the regularization term may for instance be L_1 or TV.

References

- Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3. <https://doi.org/10.1561/22000000016>
- Chambolle, A., & Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40. <https://doi.org/10.1007/s10851-010-0251-1>
- Chierchia, G., Chouzenoux, E., Combettes, P. L., & Pesquet, J.-C. (2024). *The proximity operator repository*. <https://proximity-operator.net/>
- Combettes, P., & Pesquet, J.-C. (2011). *Proximal splitting methods in signal processing*. Springer Optimization; Its Applications. https://doi.org/10.1007/978-1-4419-9569-8_10
- Heide, F., Diamond, S., Nießner, M., Ragan-Kelley, J., Heidrich, W., & Wetzstein, G. (2016). *ProxImaL: Efficient image optimization using proximal algorithms*. 35(4). <https://doi.org/10.1145/2897824.2925875>
- Leblanc, O., Hofer, M., Sivankutty, S., Rigneault, H., & Jacques, L. (2023). Interferometric Lensless Imaging: Rank-one Projections of Image Frequencies with Speckle Illuminations. *ArXiv e-Prints*. <https://doi.org/10.1109/tci.2024.3359178>
- Maheswaranathan, N., Zapp, S., & Poole, B. (2019). *Proxalgs*. <https://github.com/ganguli-lab/proxalgs/>
- Moolekamp, F., & Melchior, P. (2018). Block-simultaneous direction method of multipliers: a proximal primal-dual splitting algorithm for nonconvex problems with multiple constraints. *Optimization and Engineering*, 19. <https://doi.org/10.1007/s11081-018-9380-y>
- Parikh, N. (2013). *Foundations; Trends in Optimization*. <https://doi.org/10.1561/24000000003>
- Ravasi, M. (2021). Leveraging GPUs for matrix-free optimization with PyLops. *Fifth EAGE Workshop on High Performance Computing for Upstream*, 1. <https://doi.org/10.3997/2214-4609.2021612003>
- Ravasi, M., & Birnie, C. (2022). A joint inversion-segmentation approach to assisted seismic interpretation. *Geophysical Journal International*, 228. <https://doi.org/10.1093/gji/ggab388>
- Ravasi, M., & Vasconcelos, I. (2020). PyLops - A linear-operator Python library for scalable algebra and optimization. *SoftwareX*, 11. <https://doi.org/10.1016/j.softx.2019.100361>
- Romero, J., Luiken, M. C. N., & Ravasi, M. (2022). Plug and Play Post-Stack Seismic Inversion with CNN-Based Denoisers. *Second EAGE Subsurface Intelligence Workshop*, 1. <https://doi.org/10.3997/2214-4609.2022616015>
- Romero, J., Luiken, N., & Ravasi, M. (2023). Seeing through the CO2 plume: Joint inversion-segmentation of the Sleipner 4D seismic data set. *The Leading Edge*, 42. <https://doi.org/10.1190/tle42070457.1>
- Simeoni, M., Kashani, S., Rué-Queralt, J., & Developers, P. (2024). *Pyxu-org/pyxu: pyxu*. Zenodo. <https://doi.org/10.5281/zenodo.4486431>

Venkatakrishnan, S. V., Bouman, C. A., & Wohlberg, B. (2013). Plug-and-Play Priors for Model Based Reconstruction. *2013 IEEE Global Conference on Signal and Information Processing*. <https://doi.org/10.1109/GlobalSIP.2013.6737048>