PyProximal - scalable convex optimization in Python

Matteo Ravasi [1][¶], **Marcus Valtonen Ornhag** [2], **Nick Luiken** [1], **Olivier Leblanc** [3], **and Eneko Urunuela** [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, 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

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.

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 seamlessy handle problems with millions of unknowns and inherits the interchangle 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

43 learn and adopt with minimal additional effort.

Mathematical framework

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

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

46 where f and g are possibly non-smooth convex functionals and $\mathbf{L}$ is a linear operator. A special
47 case appearing in many scientific applications is represented by $f = 1/2\|\mathbf{y} - \mathcal{A}(\mathbf{x})\|_2^2$. Here,
48 $\mathcal{A}$ is a (possibly non-linear) modeling operator, describing the underlying physical process that
49 links the unknown model vector $\mathbf{x}$ to the vector of observations $\mathbf{y}$. In this case, we usually
50 refer to g as the regularizer, where one or multiple functions are added to the data misfit term
51 to promote certain features in the sought after solution and/or constraint the solution to fall
52 within a given set of allowed vectors.

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

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

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

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

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

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

Code structure

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

80 The first part contains the entire suite of proximal operators, which are class-based objects
81 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.
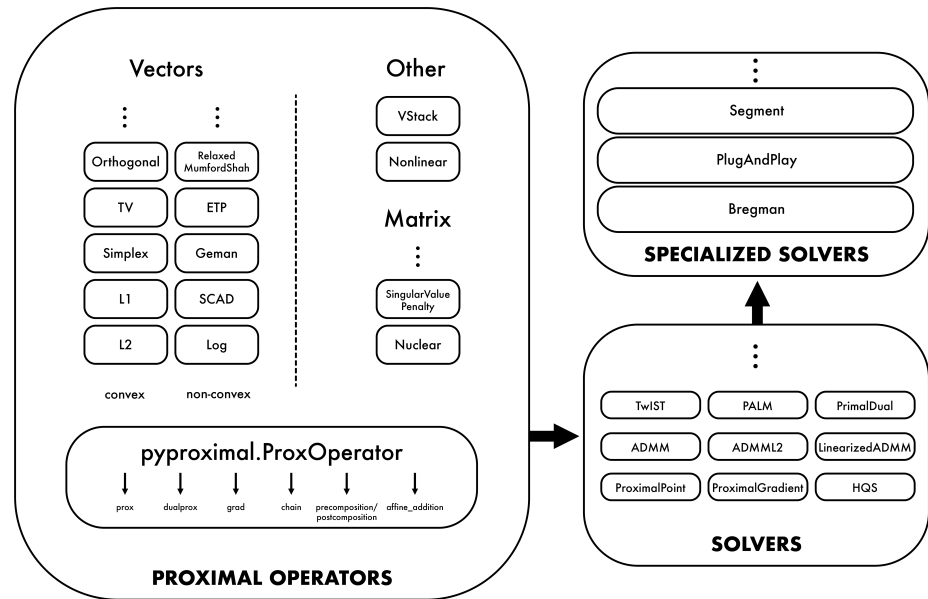

Figure 1: Schematic representation of the `PyProximal` API.

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/10.1561/2200000016

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

Combettes, P., & Pesquet, J.-C. (2011). *Proximal splitting methods in signal processing*. Springer Optimization; Its Applications. https://link.springer.com/chapter/10.1007/978-1-4419-9569-8_10

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://arxiv.org/abs/2306.12698

Parikh, N. (2013). Foundations; Trends in Optimization. https://web.stanford.edu/~boyd/papers/pdf/prox_algs.pdf

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

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