The Journal of Open Source Software

PICOS: A Python interface to conic optimization solvers

Guillaume Sagnol[1] **and Maximilian Stahlberg**[1]

1 Technische Universität Berlin

DOI: 10.21105/joss.03915

Software

- Review
- Repository
- Archive

Editor: Melissa Weber Mendonça

Reviewers:

- @sfuxy
- @marwahaha

Submitted: 18 October 2021
Published: 13 November 2021

Summary

Many convex and mixed integer constrained optimization problems can be compiled into a canonical form and handed to some off-the-shelf optimization solver for numeric solution. However, the necessary reformulations can be technically challenging and a program written to use one solver cannot easily be made to use another. PICOS is a well-established Python meta-interface to many proprietary and open source optimization solvers that allows problems to be input in a natural algebraic form and that handles solver selection and the required transformations transparently. This enables users to focus more on the high level optimization model and its application and less on technicalities.

Statement of need

Python for many is the programming language of choice for fast prototyping and data analysis while convex optimization problems are omnipresent in virtually all fields of science and industry. If no specialized algorithm is known to solve a problem at hand, then its numeric solution is typically left to one of many tremendously optimized low-level optimization solvers. Clearly, one would like to connect the ease of Python with the uncompromising speed that these solvers offer. Unfortunately though, most solvers require their input to be posed in a canonical form, usually such that all constraints are amalgamated into a single constraint matrix. For many real world problems, the necessary transformations are nontrivial and time-consuming to write down. PICOS resolves this issue by offering the user a powerful object-oriented modeling language to state their problem in a natural way. When a solution is requested, a suitable solver is selected and the problem is transformed to an efficient low-level representation understood by that solver. Finally, the solution is converted back to refer to the original problem as defined by the user. Subsequent solution attempts recycle as much work as possible.

PICOS was designed to be used by both application developers and researchers as well as instructors teaching courses on mixed integer, convex, conic, or robust optimization. Compared to similar software, it has outstanding support for complex semidefinite programming and for matrix operations used in quantum information theory, such as partial trace, partial transpose, and matrix realignment. It further stands out for supporting a number of robust and distributionally robust optimization models to deal with uncertainty in the data (Stahlberg, 2020), some of which appear to be unique in the Python ecosystem.

We note that PICOS is not the only Python library to provide an optimization modeling language and the automatic rewriting of problems to match the requirements of a solver. Most notably, CVXPY (Agrawal et al., 2018; Diamond & Boyd, 2016) and Pyomo (Bynum et al., 2021; Hart et al., 2011) offer comparable functionality and depth as both have evolved in parallel with PICOS for almost a decade. The reader is invited to compare features and performance to make an informed choice as to which platform fits their application best.

Optimization modeling language

An application using PICOS would define an instance of an optimization problem and a number of decision variables, then use these variables in concert with external data to assign the objective function and any constraints to the problem. The data can be a NumPy array (Harris et al., 2020) or a SciPy (Virtanen et al., 2020) or CVXOPT (Andersen et al., n.d.) (sparse) matrix or some other type understood by PICOS, such as a nested list visually representing a matrix or the string `"I"` denoting the identity. For example, the problem

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \|Ax - b\| \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = 1, \\ & x \succeq 0, \end{aligned}$$

which asks for the projection Ax of the point $b \in \mathbb{R}^m$ onto the convex hull of the columns of $A \in \mathbb{R}^{m \times n}$, can be solved in PICOS as follows:

```
import picos as pc                  # Import PICOS…
import numpy as np                  # …and NumPy.

m, n = 2, 20                        # Define the data…
A = np.random.rand(m, n)            # …using NumPy…
b = [1, 0]                          # and a Python list.

P = pc.Problem()                    # Create a problem.
x = pc.RealVariable("x", n)         # Define a variable x.
P += pc.sum(x) == 1, x >= 0         # Add two constraints.
P.minimize = abs(A*x - b)           # Assign the objective.

P.solve(solver="cvxopt")            # Solve the problem.
projection = (A*x).value            # Get the value of Ax.
```

The explicit choice of a backend solver is optional: PICOS currently supports ten low-level solver interfaces and will automatically select a well-suited one among those that are available at runtime, making models written in PICOS extremely portable. Note that PICOS comes with only two dependencies, NumPy and CVXOPT, the latter acting as a baseline solver.

Modular reformulation pipeline

When PICOS is asked to solve the above problem, it will attempt to rewrite it in a canonical form that the selected solver understands. To this end it first creates a `Footprint` object containing an abstract representation, but no data, of the problem. PICOS then searches the space of available problem reformulations by predicting their outcome in terms of the footprint. This approach has two advantages over simple rule-based canonicalization: First, even problems featuring a variety of constraint and variable types and an uncommon objective function are handled through a suitable sequence of reformulations. Second, working with footprints is much faster than actually reformulating the problem, which is only done when a promising solution strategy is found. Then, the reformulations are assembled into a pipeline, which aims to minimize reformulation work if the problem is changed and solved again.

Extensible design

PICOS is highly modular and made to be extended with emerging results in mathematical optimization. This is achieved through a massively object-oriented architecture: Every mathematical expression and constraint type is represented by a class that implements the requirements of an abstract base class, with a hierarchy of classes representing multidimensional, (complex, bi-)affine expressions serving as a building block for more powerful types. Constraint types may further define any number of inner classes that describe how the constraint can be rewritten as an equivalent set of different constraints and auxiliary variables; such recipes are collected for use in the reformulation pipeline. Additionally, the PICOS repository defines a framework for production tests, so that support for new problem types can be validated quickly and new solvers can be tested against a large body of existing problems.

Acknowledgements

Early work on PICOS was supported by the Zuse Institute Berlin (ZIB).

We dedicate this paper to the memory of Peter Wittek, whose sustained comments and contributions were vital in establishing PICOS within the field of quantum information.

References

Agrawal, A., Verschueren, R., Diamond, S., & Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, *5*(1), 42–60. https://doi.org/10.1080/23307706.2017.1397554

Andersen, M. S., Dahl, J., & Vandenberghe, L. (n.d.). *CVXOPT: Python software for convex optimization*. https://cvxopt.org/index.html.

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J.-P., & Woodruff, D. L. (2021). *Pyomo – optimization modeling in Python* (Third, Vol. 67). Springer Science & Business Media.

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hart, W. E., Watson, J.-P., & Woodruff, D. L. (2011). Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, *3*(3), 219–260. https://doi.org/10.1007/s12532-011-0026-8

Stahlberg, M. (2020). *Robust conic optimization in Python* [Master's Thesis]. https://www.static.tu.berlin/fileadmin/www/10005693/Publications/Stahlberg20.pdf

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2