

skscope: Fast Sparsity-Constrained Optimization in Python

Zezhi Wang, Peng Chen¹

{HOMURA, CHENPENG1}@MAIL.USTC.EDU.CN

Xueqin Wang¹

WANGXQ20@USTC.EDU.CN

Huiyang Peng, Xiaoke Zhang¹

{KISSTHERAIN, ZXK170091}@MAIL.USTC.EDU.CN

Anran Wang, Yu Zheng¹

{WANGANRAN, BELZHENG}@MAIL.USTC.EDU.CN

Jin Zhu², Junxian Zhu³

J.ZHU69@LSE.AC.UK, JUNXIAN@NUS.EDU.SG

¹ Department of Statistics and Finance/International Institute of Finance, School of Management, University of Science and Technology of China, Hefei, Anhui, China

² Department of Statistics, London School of Economics and Political Science, London, UK

³ Saw Swee Hock School of Public Health, National University of Singapore, Singapore

Editor:

Abstract

Applying iterative solvers on sparsity-constrained optimization (SCO) requires tedious mathematical deduction and careful programming/debugging that hinders these solvers' broad impact. In the paper, the library `skscope` is introduced to overcome such an obstacle. With `skscope`, users can solve the SCO by just programming the objective function. The convenience of `skscope` is demonstrated through two examples in the paper, where sparse linear regression and trend filtering are addressed with just four lines of code. More importantly, `skscope`'s efficient implementation allows state-of-the-art solvers to quickly attain the sparse solution regardless of the high dimensionality of parameter space. Numerical experiments reveal the available solvers in `skscope` can achieve up to 80x speedup on the competing relaxation solutions obtained via the benchmarked convex solver. `skscope` is published on the Python Package Index (PyPI) and Conda, and its source code is available at: <https://github.com/abess-team/skscope>.

Keywords: sparsity-constrained optimization, automatic differentiation

1. Introduction

Sparsity-constrained optimization (SCO) seeks for the solution of

$$\arg \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \text{ s.t. } \|\boldsymbol{\theta}\|_0 \leq s, \quad (1)$$

where $f : \mathbb{R}^p \rightarrow \mathbb{R}$ is a differential objective function, $\|\boldsymbol{\theta}\|_0$ is the number of nonzero entries in $\boldsymbol{\theta}$, and s is a given integer. Such an optimization covers a wide range of problems in machine learning such as compressive sensing, trend filtering and graphical model. Due to the natural reflect of Occam's razor principle of simplicity, SCO is extremely important for ML community. Nowadays, active studies prosper solvers for the SCO (Cai and Wang, 2011; Foucart, 2011; Beck and Eldar, 2013; Bahmani et al., 2013; Liu et al., 2014; Shen and Li, 2017; Yuan et al., 2020; Zhou et al., 2021; Wang et al., 2023). In spite of the successful progress, two reasons still hinder the application of SCO in practice. The first reason may because recruiting these solvers for general objective functions requires tedious mathematics

*. Zezhi Wang and Jin Zhu contributed equally. Xueqin Wang is the corresponding author.

derivations that impose highly non-trivial tasks for board users. Next but even worse, users have to program for the complicated mathematics derivations and algorithmic procedures by themselves, which is another thorny task for general users. Finally and fatally, there is no publicly available software implementing these solvers for general SCO problems.

In this paper, we propose a Python library for the SCO to fill this gap such that users can conduct these solvers with minimal mathematics and programming skills. This library, called `skscope`, implements the prototypical procedures of well-known iterative solvers for general objective functions. More importantly, `skscope` leverages the powerful automatic differentiation (AD) to conduct the algorithmic procedures without deriving and programming the exact form of gradient or hessian matrix (Rall, 1981; Baydin et al., 2018). There is no doubt that AD is the cornerstone of the computational framework of deep learning (Paszke et al., 2017); and now, it is first used for solving SCO problems.

The `skscope` can run on most Linux distributions, macOS, and Windows 32 or 64-bit with Python (version ≥ 3.9), and can be easily installed from PyPI and Conda¹. We offer a website² to present `skscope`'s features and syntax. To demonstrate the versatility of `skscope`, it has been applied to more than 25 machine learning problems³, covering linear models (e.g., quantile regression and robust regression), survival analysis (e.g., Cox proportional hazard model, competitive risk model), graphical models, trend filtering and so on. It relies on GitHub Actions

for continuous integration. The `Black` style guide keeps the source Python code clean without hand-formatting. Code quality is assessed by standard code coverage metrics. The coverages for the Python packages at the time of writing were over 95%. The dependencies of `skscope` are minimal and just include the standard Python library such as `numpy`, `scikit-learn`; additionally, two powerful and well-maintained libraries, `jax` and `nlopt` (Frostig et al., 2018; Johnson, 2014), are used for obtaining AD and solving unconstrained nonlinear optimization, respectively. The source code is distributed under the MIT license.

2. Overview of Software Features

`skscope` provides a comprehensive set of state-of-the-art solvers for SCO listed in Table 1. For each of implemented solver, once it receives the objective function programmed by users, it would leverage AD and unconstrained nonlinear solver to get the ingredients to perform iterations until a certain convergence criterion is met. The implementation of each solver has been rigorously tested and validated through reproducible experiments, ensuring its correctness and reliability. Detailed reproducible results can be found on the public GitHub repository⁴.

Besides, `skscope` introduces two generic features inspired by the work of Zhu et al. (2022) to broaden the application range. Specifically, `skscope` enables the SCO on group-structured

Solver	Reference
OMPSolver	Cai and Wang (2011)
HTPSolver	Foucart (2011)
IHTSolver	Beck and Eldar (2013)
GraspSolver	Bahmani et al. (2013)
FoBaSolver	Liu et al. (2014)
ScopeSolver	Wang et al. (2023)

Table 1: Supported SCO solvers.

1. PyPI: <https://pypi.org/project/skscope>, and Conda: <https://anaconda.org/conda-forge/skscope>

2. <https://skscope.readthedocs.io>

3. <https://skscope.readthedocs.io/userguide/examples>

4. <https://github.com/abess-team/skscope-reproducibility>

parameters and enables pre-determining a part of non-sparse parameters. Moreover, `skscope` offers flexible benchmark methods for selecting the sparsity level, catering to the urgent needs of the data science community. In terms of computation, `skscope` is compatible with the just-in-time compilation provided by the `jax` library. This enables efficient computing of AD, resulting in the acceleration of all solvers. Finally, as a factory for machine learning methods, the `skscope` continuously supplies `scikit-learn` compatible methods.

3. Usage Examples

An example of compressing sensing with `GraspSolver` is demonstrated in Figure 1. From the results in lines 16-17, we witness `GraspSolver` correctly identifies the effective variables and gives an accurate estimation. More impressively, the solution is easily obtained by programming 4 lines of code.

```

1 import numpy as np
2 import jax.numpy as jnp
3 from skscope import GraspSolver  ## the gradient support pursuit solver
4 from sklearn.datasets import make_regression
5 ## generate data
6 x, y, coef = make_regression(n_features=10, n_informative=3, coef=True)
7 print("Effective variables: ", np.nonzero(coef)[0],
8       "coefficients: ", np.around(coef[np.nonzero(coef)[0]], 2))
9 def ols_loss(params):  ## define loss function
10     return jnp.linalg.norm(y - x @ params)
11 ## initialize the solver: ten parameters in total, three of which are non-zero
12 solver = GraspSolver(10, 3)
13 params = solver.solve(ols_loss)
14 print("Estimated variables: ", solver.get_support(),
15       "estimated coefficients:", np.around(params[solver.get_support()], 2))
16 >>> Effective variables: [3 4 7] coefficients: [ 9.71 19.16 13.53]
17 >>> Estimated variables: [3 4 7] estimated coefficients: [ 9.71 19.16 13.53]
```

Figure 1: Using the `skscope` for compressive sensing.

Figure 2 presents for filtering trend via `ScopeSolver`, serving as a non-trivial example because the dimensionality of parameters is hundreds. From 2 shows that the solution of `ScopeSolver` captures the main trend of the observed data. In this case, 6 lines of code help us attain the solution.

```

1 import numpy as np
2 import jax.numpy as jnp
3 import matplotlib.pyplot as plt
4 from skscope import ScopeSolver
5 np.random.seed(2023)
6 # observed data, random walk with normal increment:
7 x = np.cumsum(np.random.randn(500))
8 def tf_objective(params):
9     return jnp.linalg.norm(data - jnp.cumsum(params))
10 solver = ScopeSolver(len(x), 10)
11 params = solver.solve(tf_objective)
12 plt.plot(x, label='observation', linewidth=0.8)
13 plt.plot(jnp.cumsum(params), label='filtering trend')
14 plt.legend(); plt.show()
```

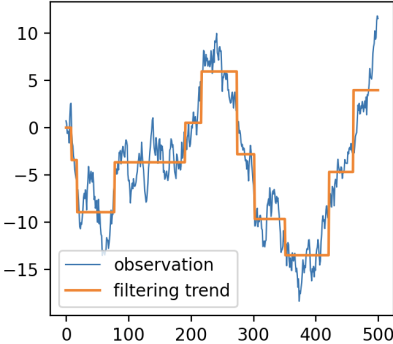


Figure 2: Using the `skscope` for trend filtering.

4. Performance

We conducted a comprehensive comparison among the solvers employed in `skscope` and two alternative approaches. The first approach utilizes the ℓ_1 relaxation of (1), implemented

using the open-source solver, `cvxpy` (Diamond and Boyd, 2016). The second approach solves (1) by recruiting the widely-used mixed-integer optimization solver, `GUROBI`⁵. These comparisons covered a wide range of SCO problems and were performed on a Ubuntu platform with Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz and 64 RAM. Python version is 3.10.9. Table 2 reveals that all solvers in `skscope` consistently outperformed `cvxpy` in terms of support-set-selection accuracy. `GUROBI` may have a higher accuracy, but its runtime far exceeds others. Furthermore, `skscope` still works when $f(x)$ is non-convex or non-linear where `cvxpy` or `GUROBI` may fail. In terms of computation, `skscope` demonstrated significant computational advantages against `cvxpy` and `GUROBI`, exhibiting approximately 1-80x speedups on `cvxpy` and 30-1000x speedups on `GUROBI`. Among the solvers in `skscope`, `ScopeSolver` and `FoBaSolver` have particularly promising results in support set selection, with `ScopeSolver` achieving speedups of around 1.5x-7x compared to `FoBaSolver`.

Method	Linear regression		Logistic regression		Robust feature selection	
	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime
OMPSolver	1.00(0.01)	2.45(0.68)	0.91(0.05)	1.66(0.67)	0.56(0.17)	0.73(0.14)
IHTSolver	0.79(0.04)	3.42(0.88)	0.97(0.03)	1.06(0.67)	0.67(0.07)	0.89(0.22)
HTPSolver	1.00(0.00)	4.14(1.25)	0.84(0.05)	2.37(0.92)	0.91(0.05)	5.00(0.94)
GraspSolver	1.00(0.00)	1.16(0.38)	0.90(0.08)	12.70(8.20)	1.00(0.00)	0.50(0.25)
FoBaSolver	1.00(0.00)	11.70(2.90)	0.92(0.06)	6.31(2.15)	0.98(0.08)	3.37(0.66)
ScopeSolver	1.00(0.00)	2.11(0.70)	0.94(0.04)	3.24(2.67)	0.98(0.09)	1.86(0.55)
cvxpy	0.83(0.17)	14.59(5.60)	0.83(0.05)	69.45(53.47)	X	X
GUROBI	1.00(0.00)	1009.94(0.66)	X	X	X	X

Method	Trend filtering		Ising model		Nonlinear feature selection	
	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime
OMPSolver	0.86(0.03)	1.77(0.57)	0.98(0.03)	2.86(0.86)	0.77(0.09)	11.53(3.61)
IHTSolver	0.08(0.00)	0.76(0.28)	0.96(0.05)	3.24(1.43)	0.78(0.09)	6.37(2.32)
HTPSolver	0.47(0.03)	0.71(0.23)	0.97(0.03)	5.26(2.03)	0.78(0.09)	10.82(7.86)
GraspSolver	0.78(0.12)	0.95(0.38)	0.99(0.01)	1.02(0.44)	0.78(0.08)	7.34(2.75)
FoBaSolver	1.00(0.00)	8.27(1.66)	1.00(0.01)	11.59(3.55)	0.77(0.09)	31.26(8.80)
ScopeSolver	0.98(0.02)	4.73(1.13)	1.00(0.01)	1.69(0.67)	0.77(0.09)	8.60(2.70)
cvxpy	0.21(0.04)	19.75(9.57)	0.94(0.04)	32.26(17.88)	0.74(0.09)	534.49(337.72)
GUROBI	1.00(0.00)	1013.16(0.62)	X	X	0.79(0.08)	1003.88(1.53)

Table 2: The numerical experiment results on six specific SCO problems. Accuracy is equal to $|\text{supp}\{\theta^*\} \cap \text{supp}\{\theta\}| / |\text{supp}\{\theta^*\}|$ and the runtime is measured by seconds. The results are the average of 100 replications, and the parentheses record standard deviation. Robust (or nonlinear) variable selection is based on the work of Wang et al. (2013) (or Yamada et al. (2014)). `GUROBI`: version 10.0.2; `cvxpy`: version 1.3.1; `skscope`: version 0.1.0. **X**: not available.

5. Conclusion and Discussion

`skscope` is a fast Python library for solving general SCO problems. It offers well-designed and user-friendly interfaces such that users can tackle SCO with minimal knowledge of mathematics and programming. Therefore, `skscope` must have a broad application in diverse domains. Future versions of `skscope` plan to support more iterative solvers for the SCO like Zhou et al. (2021) so as to establish a benchmark toolbox/platform for the SCO.

5. TimeLimit is set to 1000. Note that optimization may not stop immediately upon hitting TimeLimit.

Acknowledgments

Wang’s research is partially supported by NSFC (72171216, 71921001, 71991474), The Key Research and Development Program of Guangdong, China (2019B020228001), and Science and Technology Program of Guangzhou, China (202002030129). We thank to Kangkang Jiang and Junhao Huang for their insightful discussions.

References

- Sohail Bahmani, Bhiksha Raj, and Petros T Boufounos. Greedy sparsity-constrained optimization. *Journal of Machine Learning Research*, 14(25):807–841, 2013.
- Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:1–43, 2018.
- Amir Beck and Yonina C Eldar. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms. *SIAM Journal on Optimization*, 23(3):1480–1509, 2013.
- T Tony Cai and Lie Wang. Orthogonal matching pursuit for sparse signal recovery with noise. *IEEE Transactions on Information theory*, 57(7):4680–4688, 2011.
- Steven Diamond and Stephen Boyd. CVXPY: a python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
- Simon Foucart. Hard thresholding pursuit: an algorithm for compressive sensing. *SIAM Journal on numerical analysis*, 49(6):2543–2563, 2011.
- Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 4(9), 2018.
- Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2022. URL <https://www.gurobi.com>.
- Steven G Johnson. The nlopt nonlinear-optimization package, 2014.
- Ji Liu, Jieping Ye, and Ryohei Fujimaki. Forward-backward greedy algorithms for general convex smooth functions over a cardinality constraint. In *International Conference on Machine Learning*, pages 503–511. PMLR, 2014.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- Louis B Rall. *Automatic differentiation: Techniques and applications*. Springer, 1981.
- Jie Shen and Ping Li. A tight bound of hard thresholding. *Journal of Machine Learning Research*, 18(1):7650–7691, 2017. ISSN 1532-4435.

- Xueqin Wang, Yunlu Jiang, Mian Huang, and Heping Zhang. Robust variable selection with exponential squared loss. *Journal of the American Statistical Association*, 108(502): 632–643, 2013. doi: 10.1080/01621459.2013.766613. PMID: 23913996.
- ZeZhi Wang, Borui Tang, Xueqin Wang, Jin Zhu, Junxian Zhu, and Hongmei Lin. Sparsity-constrained optimization via splicing iteration. *Submitted*, 2023.
- Makoto Yamada, Wittawat Jitkrittum, Leonid Sigal, Eric P. Xing, and Masashi Sugiyama. High-dimensional feature selection by feature-wise kernelized lasso. *Neural Computation*, 26(1):185–207, 2014.
- Xiaotong Yuan, Bo Liu, Lezi Wang, Qingshan Liu, and Dimitris N. Metaxas. Dual iterative hard thresholding. *Journal of Machine Learning Research*, 21(1), jan 2020. ISSN 1532-4435.
- Shenglong Zhou, Naihua Xiu, and Hou-Duo Qi. Global and quadratic convergence of newton hard-thresholding pursuit. *Journal of Machine Learning Research*, 22(12):1–45, 2021.
- Jin Zhu, Xueqin Wang, Liyuan Hu, Junhao Huang, Kangkang Jiang, Yanhang Zhang, Shiyun Lin, and Junxian Zhu. abess: a fast best-subset selection library in Python and R. *Journal of Machine Learning Research*, 23(1):9206–9212, 2022.