Journal of Machine Learning Research 25 (2024) 1-9 Submitted 11/23; Revised 8/24; Published 9/24

skscope: Fast Sparsity-Constrained Optimization in Python

Zezhi Wang[1], Junxian Zhu[2] HOMURA@MAIL.USTC.EDU.CN, JUNXIAN@NUS.EDU.SG
Xueqin Wang[1], Jin Zhu[3] WANGXQ20@USTC.EDU.CN, J.ZHU69@LSE.AC.UK
Huiyang Peng[1], Peng Chen[1] {KISSTHERAIN, CHENPENG1}@MAIL.USTC.EDU.CN
Anran Wang[1], Xiaoke Zhang[1] {WANGANRAN, ZXK170091}@MAIL.USTC.EDU.CN
[1] *Department of Statistics and Finance/International Institute of Finance, School of Management, University of Science and Technology of China, Hefei, Anhui, China*
[2] *Saw Swee Hock School of Public Health, National University of Singapore, Singapore*
[3] *Department of Statistics, London School of Economics and Political Science, London, UK*

Editor: Sebastian Schelter

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, nonlinear optimization, high-dimensional data, Python

1. Introduction

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

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

where $f : \mathbb{R}^p \to \mathbb{R}$ is a differential objective function, $\|\boldsymbol{\theta}\|_0$ is the number of nonzero entries in $\boldsymbol{\theta}$, and s is a given integer. Such optimization covers a wide range of problems in machine learning, such as compressive sensing, trend filtering, and graphical models. SCO is extremely important for the ML community because it naturally reflects Occam's razor principle of simplicity. 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., 2013; Shen and Li, 2017; Yuan et al., 2020; Zhou et al., 2021; Zhu et al., 2024). In spite of the successful progress, two reasons still hinder the application of SCO in practice. The first reason may

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

be that recruiting these solvers for general objective functions requires tedious mathematics 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 efficiently 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 Conda[1]. We offer a website[2] to present skscope's features and syntax. To demonstrate the versatility of skscope, it has been applied to more than 25 machine learning problems[3], covering linear models (for example, quantile regression and robust regression), survival analysis (for example, 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 implemented solver, once it receives the objective function programmed by users, it will leverage AD and an 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 repository[4].

Solver	Reference
ForwardSolver	Marcano et al. (2010)
OMPSolver	Cai and Wang (2011)
HTPSolver	Foucart (2011)
IHTSolver	Beck and Eldar (2013)
GraspSolver	Bahmani et al. (2013)
PDASSolver	Wen et al. (2020)
FoBaSolver	Liu et al. (2013)
ScopeSolver	Zhu et al. (2024)

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/en/latest/gallery
4. https://github.com/abess-team/skscope-reproducibility

Besides, skscope introduces two generic features to broaden the application range. Specifically, skscope enables the SCO on group-structured parameters and enables pre-determining a part of non-sparse parameters. Moreover, skscope allows using information criteria or cross-validation for selecting the sparsity level, catering to the urgent needs of the data science community. Warm-start initialization is supported to speed up the selection. In terms of computation, skscope can transparently run on the GPU/TPU and 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. Typically, skscope maintains computational scalability as state-of-the-art regression solvers like abess (Zhu et al., 2022) while it possesses capability in solving general SCO problems (see Table A1). Furthermore, skscope enables optimized objective functions implemented with sparse matrices[5] to save memory usage. Finally, as a factory for machine learning methods, the skscope continuously supplies scikit-learn compatible methods (listed in Table A2 in Appendix), which allows practitioners to directly call them to solve practical problems.

3. Usage Examples

An example of compressing sensing with GraspSolver is demonstrated in Figure 1. From the results in lines 16–17, we witness that 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. Even more impressively, for a concrete SCO problem[6] with parameters of order $O(10^4)$ and an objective function involving matrices of size $O(10^8)$, solvers in skscope can tackle the problem in less than two minutes on a personal laptop.

5. Sparse matrices are recommended primarily for memory-intensive scenarios.
6. https://skscope.readthedocs.io/en/latest/gallery/GeneralizedLinearModels/poisson-identity-link.html

```python
import numpy as np
import jax.numpy as jnp
import matplotlib.pyplot as plt
from skscope import ScopeSolver
np.random.seed(2023)
# observed data, random walk with normal increment:
x = np.cumsum(np.random.randn(500))
def tf_objective(params):
    return jnp.linalg.norm(data - jnp.cumsum(params))
solver = ScopeSolver(len(x), 10)
params = solver.solve(tf_objective)
plt.plot(x, label='observation', linewidth=0.8)
plt.plot(jnp.cumsum(params), label='filtering trend')
plt.legend(); plt.show()
```

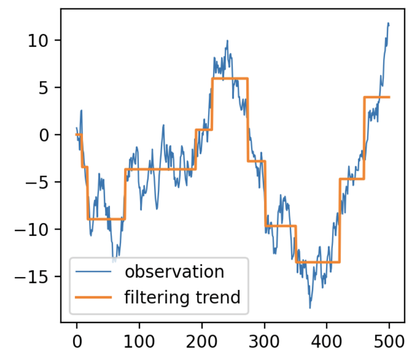

Figure 2: Using the skscope for trend filtering.

4. Performance

We conducted a comprehensive comparison among the sparse-learning solvers employed in skscope and two alternative approaches. The first competing approach solves (1) by recruiting the widely-used mixed-integer optimization solver, GUROBI[7]. We compare this approach assuming the optimal s of (1) is known and present the results in Table A3. The second approach utilizes the ℓ_1 relaxation of (1), implemented using the open-source solver, cvxpy (Diamond and Boyd, 2016). The comparison with cvxpy assumes the optimal s of (1) is unknown and searches with information criteria. The corresponding results are reported in Table A4. These comparisons covered a wide range of concrete 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 A3 shows that skscope not only achieves highly competitive support-set selection accuracy but also has a significantly lower runtime than GUROBI. Table A4 reveals that, in terms of support-set selection, all solvers in skscope generally have a higher precision score while maintaining a competitive recall score, leading to a higher F1-score. The results indicate that skscope outperforms cvxpy in overall selection quality. Furthermore, as shown in Tables A3 and A4, skscope has a desirable support set selection quality when $f(x)$ is non-convex or non-linear where cvxpy or GUROBI may fail. In terms of computation, skscope generally 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.

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 (e.g., Zhou et al., 2021) so as to establish a benchmark toolbox/platform for the SCO.

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

Acknowledgments

We would like to thank the editor and the two referees for their constructive comments and valuable suggestions, which have substantially improved both the library and the paper. We also thank Kangkang Jiang, Junhao Huang, and Yu Zheng for their insightful discussions.

Wang's research is partially supported by National Natural Science Foundation of China grants No. 72171216, 12231017, 71921001, and 71991474.

Appendix A. Additional Tables.

	Method	$n = 500, p = 1000, s = 10$		$n = 5000, p = 10000, s = 100$	
Linear		Accuracy	Runtime	Accuracy	Runtime
	abess	1.00 (0.00)	0.01 (0.00)	1.00 (0.00)	2.19 (0.22)
	ScopeSolver	1.00 (0.00)	0.23 (0.03)	1.00 (0.00)	14.09 (1.28)
Logistic	Method	$n = 500, p = 1000, s = 10$		$n = 5000, p = 10000, s = 100$	
		Accuracy	Runtime	Accuracy	Runtime
	abess	0.99 (0.03)	0.02 (0.00)	1.00 (0.00)	6.81 (3.32)
	ScopeSolver	0.99 (0.03)	0.26 (0.03)	1.00 (0.00)	24.12 (5.83)
NNLS	Method	$n = 500, p = 1000, s = 10$		$n = 5000, p = 10000, s = 100$	
		Accuracy	Runtime	Accuracy	Runtime
	abess	1.00 (0.00)	0.01 (0.00)	1.00 (0.00)	2.26 (0.10)
	ScopeSolver	1.00 (0.00)	0.22 (0.02)	1.00 (0.00)	14.80 (1.00)

Table A1: Comparison between abess and skscope on linear regression, logistic regression models, and non-negative least square (NNLS) estimation. The datasets are generated using the make_glm_data function implemented in the abess package. For all tasks, the non-zero coefficients are randomly chosen from $\{1, \ldots, p\}$. The mean metrics are computed over 10 replications with standard deviation in parentheses. For large problems, skscope runs 3x-7x slower than abess. However, skscope can handle the same problem scale as abess. For instance, if abess can process a dataset of size $n \times p$, skscope can handle a dataset of size $(n/3) \times (p/3)$. For smaller problems, although skscope is much slower than abess, it solves problems in less than 0.3 seconds, providing immediate results for users.

skmodel	Description
PortfolioSelection	Construct sparse Markowitz portfolio
NonlinearSelection	Select relevant features with nonlinear effect
RobustRegression	A robust regression dealing with outliers
MultivariateFailure	Multivariate failure time model in survival analysis
IsotonicRegression	Fit the data with an non-decreasing curve

Table A2: Some application-oriented interfaces implemented in the module skmodel in skscope.

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)
GUROBI	1.00(0.00)	1009.94(0.66)	✗	✗	✗	✗
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)
GUROBI	1.00(0.00)	1013.16(0.62)	✗	✗	0.79(0.08)	1003.88(1.53)

Table A3: The numerical experiment results on six specific SCO problems. Accuracy is equal to $|\mathrm{supp}\{\boldsymbol{\theta}^*\} \cap \mathrm{supp}\{\boldsymbol{\theta}\}|/|\mathrm{supp}\{\boldsymbol{\theta}^*\}|$ 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 0.1.8. ✗: not available.

Method	Linear regression				Logistic regression			
	Recall	Precision	F1-score	Runtime	Recall	Precision	F1-score	Runtime
OMPSolver	0.90 (0.28)	1.00 (0.01)	0.92 (0.24)	69.29 (3.74)	0.91 (0.06)	0.91 (0.07)	0.91 (0.06)	64.78 (3.63)
IHTSolver	0.90 (0.28)	1.00 (0.01)	0.92 (0.24)	14.27 (1.59)	0.92 (0.06)	0.93 (0.07)	0.93 (0.06)	39.38 (1.87)
HTPSolver	0.93 (0.25)	1.00 (0.01)	0.94 (0.21)	60.76 (20.04)	0.91 (0.06)	0.91 (0.07)	0.91 (0.06)	68.33 (31.86)
GraspSolver	0.66 (0.45)	1.00 (0.00)	0.69 (0.42)	15.10 (1.94)	0.98 (0.04)	0.94 (0.07)	0.96 (0.05)	86.07 (9.21)
FoBaSolver	0.92 (0.26)	1.00 (0.00)	0.93 (0.23)	234.09 (9.39)	0.91 (0.06)	0.93 (0.07)	0.92 (0.07)	206.86 (13.51)
ScopeSolver	0.92 (0.26)	1.00 (0.00)	0.93 (0.23)	9.38 (0.46)	0.93 (0.05)	0.96 (0.06)	0.95 (0.05)	7.29 (1.03)
cvxpy	1.00 (0.00)	0.43 (0.03)	0.60 (0.03)	55.44 (10.9)	1.00 (0.01)	0.42 (0.03)	0.59 (0.03)	867.76 (203.71)

Method	Robust feature selection				Trend filtering			
	Recall	Precision	F1-score	Runtime	Recall	Precision	F1-score	Runtime
OMPSolver	0.98 (0.09)	0.66 (0.12)	0.66 (0.12)	69.01 (4.49)	0.53 (0.04)	0.79 (0.05)	0.63 (0.04)	143.94 (8.72)
IHTSolver	1.00 (0.02)	0.78 (0.09)	0.78 (0.09)	65.99 (4.43)	0.53 (0.04)	0.79 (0.05)	0.63 (0.04)	6.59 (0.28)
HTPSolver	1.00 (0.00)	0.80 (0.11)	0.80 (0.11)	667.02 (36.75)	0.53 (0.04)	0.80 (0.05)	0.64 (0.04)	33.53 (1.56)
GraspSolver	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	23.91 (3.54)	0.63 (0.17)	0.71 (0.21)	0.66 (0.18)	21.46 (13.19)
FoBaSolver	0.99 (0.06)	0.98 (0.09)	0.98 (0.09)	246.68 (19.46)	0.84 (0.09)	1.00 (0.00)	0.91 (0.06)	322.80 (17.12)
ScopeSolver	1.00 (0.00)	0.99 (0.06)	0.99 (0.06)	9.28 (0.77)	0.66 (0.09)	0.90 (0.05)	0.76 (0.07)	17.81 (1.25)
cvxpy	✗	✗	✗	✗	0.76 (0.11)	0.31 (0.05)	0.44 (0.07)	67.69 (10.28)

Method	Ising model				Nonlinear feature selection			
	Recall	Precision	F1-score	Runtime	Recall	Precision	F1-score	Runtime
OMPSolver	0.99 (0.02)	0.92 (0.06)	0.95(0.04)	132.79 (9.75)	0.79 (0.09)	0.78 (0.12)	0.78 (0.08)	227.62 (59.59)
IHTSolver	0.99 (0.02)	0.78 (0.19)	0.86(0.13)	97.03 (5.35)	0.42 (0.08)	0.77 (0.16)	0.54 (0.09)	158.25 (35.93)
HTPSolver	0.99 (0.02)	0.80 (0.19)	0.87(0.13)	98.18 (15.63)	0.42 (0.08)	0.77 (0.16)	0.54 (0.09)	290.92 (65.97)
GraspSolver	1.00 (0.01)	0.93 (0.06)	0.96(0.04)	32.80 (15.14)	0.79 (0.09)	0.79 (0.12)	0.78 (0.08)	51.72 (11.44)
FoBaSolver	1.00 (0.01)	0.93 (0.04)	0.96(0.02)	432.28 (23.38)	0.79 (0.09)	0.78 (0.12)	0.78 (0.08)	874.08 (195.91)
ScopeSolver	1.00 (0.02)	0.93 (0.05)	0.96(0.03)	36.14 (1.93)	0.79 (0.09)	0.78 (0.12)	0.78 (0.09)	198.37 (43.24)
cvxpy	✗	✗	✗	✗	0.76 (0.09)	0.83 (0.11)	0.79 (0.08)	525.64 (122.28)

Table A4: The numerical experiment when the optimal sparsity s in (1) is unknown and information criteria are used for selecting the optimal one. Specifically, for linear regression, special information criterion (Zhu et al., 2020) is used; as for logistic regression, Ising model, and non-linear feature selection, generalized information criterion (Zhu et al., 2023) is employed; and for trend filtering and robust feature selection, Bayesian information criterion (Wen et al., 2023) is used. Recall: the recall score that is computed by $|\operatorname{supp}\{\boldsymbol{\theta}^*\} \cap \operatorname{supp}\{\boldsymbol{\theta}\}|/|\operatorname{supp}\{\boldsymbol{\theta}^*\}|$; Precision: the precision score computed by $|(\operatorname{supp}\{\boldsymbol{\theta}^*\})^c \cap \operatorname{supp}\{\boldsymbol{\theta}\}|/|(\operatorname{supp}\{\boldsymbol{\theta}^*\})^c|$; F1-score is computed as the harmonic mean of precision and recall.

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.

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*, 2013.

Cedeño Alexis Marcano, J Quintanilla-Domínguez, MG Cortina-Januchs, and Diego Andina. Feature selection using sequential forward selection and classification applying artificial metaplasticity neural network. In *IEEE Industrial Electronics Society*, pages 2845–2850, 2010.

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. In *Advances in Neural Information Processing Systems Workshop on Autodiff*, 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.

Canhong Wen, Aijun Zhang, Shijie Quan, and Xueqin Wang. BeSS: an R package for best subset selection in linear, logistic and cox proportional hazards models. *Journal of Statistical Software*, 94:1–24, 2020.

Canhong Wen, Xueqin Wang, and Aijun Zhang. ℓ_0 trend filtering. *INFORMS Journal on Computing*, 35(6):1491–1510, 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), 2020.

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.

Jin Zhu, Junxian Zhu, Zezhi Wang, Borui Tang, Xueqin Wang, and Hongmei Lin. Sparsity-constrained optimization via splicing iteration. *arXiv preprint arXiv:2406.12017*, 2024.

Junxian Zhu, Canhong Wen, Jin Zhu, Heping Zhang, and Xueqin Wang. A polynomial algorithm for best-subset selection problem. *Proceedings of the National Academy of Sciences*, 117(52):33117–33123, 2020.

Junxian Zhu, Jin Zhu, Borui Tang, Xuanyu Chen, Hongmei Lin, and Xueqin Wang. Best-subset selection in generalized linear models: A fast and consistent algorithm via splicing technique. *arXiv preprint arXiv:2308.00251*, 2023.