CR-Sparse: Hardware accelerated functional algorithms for sparse signal processing in Python using JAX

Shailesh Kumar[1]

1 Indian Institute of Technology, Delhi

DOI: 10.21105/joss.03917

Software

- Review
- Repository
- Archive

Editor: Pierre de Buyl

Reviewers:

- @Saran-nns
- @mirca

Submitted: 12 November 2021
Published: 02 December 2021

Summary

We introduce `CR-Sparse`, a Python library that enables to efficiently solve a wide variety of sparse representation based signal processing problems. It is a cohesive collection of sub-libraries working together. Individual sub-libraries provide functionalities for: wavelets, linear operators, greedy and convex optimization based sparse recovery algorithms, subspace clustering, standard signal processing transforms, and linear algebra subroutines for solving sparse linear systems. It has been built using Google JAX (Bradbury et al., 2018), which enables the same high level Python code to get efficiently compiled on CPU, GPU and TPU architectures using XLA (Abadi et al., 2017).

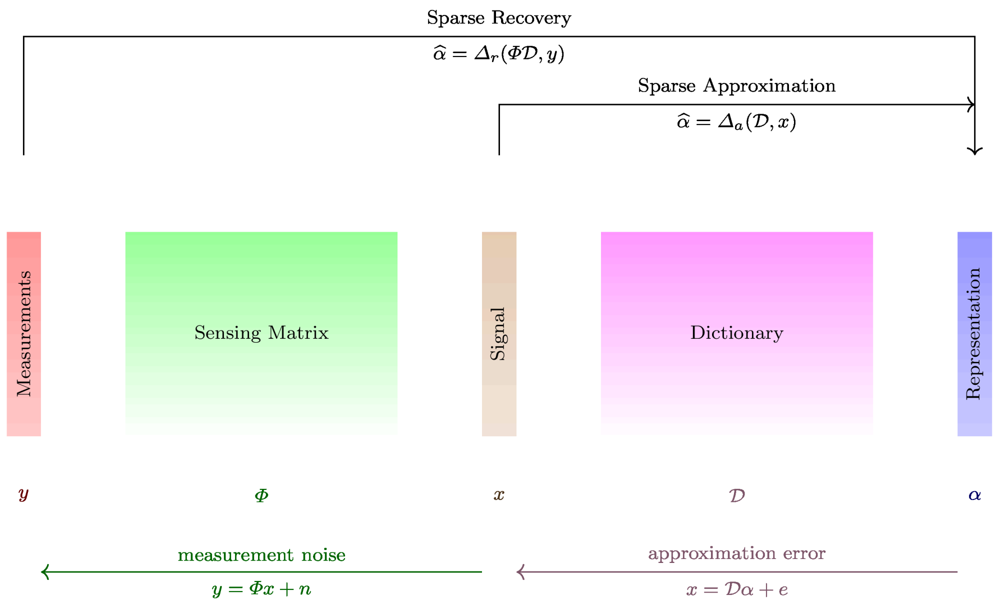

Figure 1: Sparse signal representations and compressive sensing

Traditional signal processing exploits the underlying structure in signals by representing them using Fourier or wavelet orthonormal bases. In these representations, most of the signal energy is concentrated in few coefficients allowing greater flexibility in analysis and processing of signals. More flexibility can be achieved by using overcomplete dictionaries (Mallat, 2009) (e.g. unions of orthonormal bases). However, the construction of sparse representations of signals in these overcomplete dictionaries is no longer straightforward and requires use of specialized sparse coding algorithms like orthogonal matching pursuit (Pati et al., 1993) or basis pursuit (Chen et al., 2001). The key idea behind these algorithms is the fact that underdetermined systems $Ax = b$ can be solved efficiently to provide sparse solutions x if the matrix A satisfies specific conditions on its properties like coherence. Compressive sensing takes the same idea in the other direction and contends that signals having sparse representations in suitable bases can be acquired by very few data-independent random measurements $y = \Phi x$ if the sensing or measurement system Φ satisfies certain conditions like restricted isometry property (Candes, 2008). The same sparse coding algorithms can be tailored for sparse signal recovery from compressed measurements.

A short mathematical introduction to compressive sensing and sparse representation problems is provided in docs. For comprehensive introduction to sparse representations and compressive sensing, please refer to excellent books (Elad, 2010; Foucart & Rauhut, 2013; Mallat, 2009), papers (Donoho, 2006; Marques et al., 2018; Qaisar et al., 2013), Rice Compressive Sensing Resources and references therein.

Package Overview

The `cr.sparse.pursuit` package includes greedy and thresholding based solvers for sparse recovery. It includes: `OMP`, `CoSaMP`, `HTP`, `IHT`, `SP` algorithms. (provided in `cr.sparse.lop` package). The `cr.sparse.cvx` package includes efficient solvers for l1-minimization problems using convex optimization methods. The `cr.sparse.sls` package provides JAX versions of `LSQR`, `ISTA`, `FISTA` algorithms for solving sparse linear systems. These algorithms can work with unstructured random and dense sensing matrices as well as structured sensing matrices represented as linear operators The `cr.sparse.lop` package includes a collection of linear operators influenced by `PyLops` (Ravasi & Vasconcelos, 2019). `cr.sparse.wt` package includes a JAX version of major functionality from `PyWavelets` (Lee et al., 2019) making it a first major pure Python wavelets implementation which can work across CPUs, GPUs and TPUs.

Statement of need

Currently, there is no single Package which provides a comprehensive set of tools for solving sparse recovery problems in one place. Individual researchers provide their codes along with their research paper only for the algorithms they have developed. Most of this work is available in the form of MATLAB (MATLAB, 2018) libraries. E.g.: `YALL1` is the original MATLAB implementation of the ADMM based sparse recovery algorithms. `L1-LS` is the original MATLAB implementation of the Truncated Newton Interior Points Method for solving the l1-minimization problem. `Sparsify` provides the MATLAB implementations of IHT, NIHT, AIHT algorithms. `aaren/wavelets` is a CWT implementation following (Torrence & Compo, 1998). `HTP` provides implementation of Hard Thresholding Pursuit in MATLAB. `WaveLab` is the reference open source wavelet implementation in MATLAB. However, its API has largely been superseded by later libraries. `Sparse and Redundant Representations book code` (Elad, 2010) provides basic implementations of a number of sparse recovery and related algorithms. Several of these libraries contain key performance critical sub-routines in the form of C/C++ extensions making portability to GPUs harder.

There are some Python libraries which focus on specific areas however they are generally CPU based. E.g., `pyCSalgos` is a Python implementation of various Compressed Sensing algorithms. `spgl1` is a `NumPy` based implementation of spectral projected gradient for L1 minimization. `c-lasso` (Simpson et al., 2021) is a Python package for constrained sparse regression and classification. This is also CPU only. `PyWavelets` is an excellent CPU only wavelets implementation in Python closely following the API of Wavelet toolbox in MATLAB. The performance critical parts have been written entirely in C. There are several attempts to port it on GPU using PyTorch (PyTorch-Wavelet-Toolbox) or Tensorflow (tf-wavelets) backends. PyLops includes GPU support. They have built a backend.py layer to switch explicitly between NumPy and CuPy for GPU support. In contrast, our use of JAX enables us to perform jit compilation with abstracted out end-to-end XLA optimization to multiple backend.

The algorithms in this package have a wide variety of applications. We list a few: image denoising, deblurring, compression, inpainting, impulse noise removal, super-resolution, subspace clustering, dictionary learning, compressive imaging, medical imaging, compressive radar, wireless sensor networks, astrophysical signals, cognitive radio, sparse channel estimation, analog to information conversion, speech recognition, seismology, direction of arrival.

Sparse signal processing problems and available solvers

We provide JAX based implementations for the following algorithms:

- `cr.sparse.pursuit.omp`: Orthogonal Matching Pursuit (OMP) (Davenport & Wakin, 2010; Pati et al., 1993; Tropp, 2004)
- `cr.sparse.pursuit.cosamp`: Compressive Sampling Matching Pursuit (CoSaMP) (Needell & Tropp, 2009)
- `cr.sparse.pursuit.sp`: Subspace Pursuit (SP) (Dai & Milenkovic, 2009)
- `cr.sparse.pursuit.iht`: Iterative Hard Thresholding and its normalized version (IHT, NIHT) (Blumensath & Davies, 2009; Blumensath & Davies, 2010)
- `cr.sparse.pursuit.htp`: Hard Thresholding Pursuit and its normalized version (HTP, NHTP) (Foucart, 2011)
- `cr.sparse.cvx.l1ls`: *Truncated Newton Interior Points Method* for solving the l1-minimization problem (Kim et al., 2007)
- `cr.sparse.cvx.admm`: Solvers for basis pursuit (BP), basis pursuit denoising (BPDN), basis pursuit with inequality constraints (BPIC), and their nonnegative variants based on ADMM (Yang & Zhang, 2011; Zhang et al., 2010)
- `cr.sparse.sls.lsqr`: LSQR algorithm for sparse linear equations (Paige & Saunders, 1982)
- `cr.sparse.sls.ista`: Iterative Shrinkage and Thresholding Algorithm (ISTA) (Daubechies et al., 2004)
- `cr.sparse.sls.fista`: Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

The dictionaries and sensing matrices can be efficiently implemented using a pair of functions for the forward Ax and adjoint $A^H x$ operations. `cr.sparse.lop` provides a collection of linear operators (similar to PyLops (Ravasi & Vasconcelos, 2019)) which provide the forward and adjoint operation functions. These operators can be JIT compiled and used efficiently with the algorithms above. Our 2D and ND operators accept 2D/ND arrays as input and return 2D/ND arrays as output. The operators `+`, `-`, `@`, `**` etc. are overridden to provide operator calculus, i.e. ways to combine operators to generate new operators.

As an application area, the library includes an implementation of sparse subspace clustering (SSC) by orthogonal matching pursuit (You et al., 2016) in the `cr.sparse.cluster.ssc` package. The `cr.sparse.cluster.spectral` package provides a custom implementation of spectral clustering step of SSC.

Experimental Results

We conducted a number of experiments to benchmark the runtime of CR-Sparse implementations viz. existing reference software in Python or MATLAB. Jupyter notebooks to reproduce these micro-benchmarks are available on the cr-sparse-companion (Kumar, 2021) repository.

All Python based benchmarks have been run on the machine configuration: Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, 16 Cores, 64 GB RAM, NVIDIA GeForce GTX 1060 6GB GPU, Ubuntu 18.04 64-Bit, Python 3.8.8, NVidia driver version 495.29.05, CUDA version 11.5.

MATLAB based benchmarks were run on the machine configuration: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz, 32 GB RAM, Windows 10 Pro, MATLAB R2020b.

The following table provides comparison of CR-Sparse against reference implementations on a set of representative problems:

Problem	Size	Ref tool	Ref time	Our time	Gain
Hard Thresholding Pursuit	M=2560, N=10240, K=200	HTP (MATLAB)	3.5687 s	160 ms	22x
Orthogonal Matching Pursuit	M=2000, N=10000, K=100	sckit-learn	379 ms	120 ms	3.15x
ADMM, BP	M=2000, N=20000, K=200	YALL1 (MATLAB)	1.542 sec	445 ms	3.46x
ADMM, BPDN	M=2000, N=20000, K=200	YALL1 (MATLAB)	1.572.81 sec	273 ms	5.75x
Image blurring	Image: 500x480, Kernel: 15x25	Pylops	6.63 ms	1.64 ms	4x
Image deblurring using LSQR	Image: 500x480, Kernel: 15x25	Pylops	237 ms	39.3 ms	6x
Image DWT2	Image: 512x512	PyWavelets	4.48 ms	656 µs	6.83x
Image IDWT2	Image: 512x512	PyWavelets	3.4 ms	614 µs	5.54x
OMP for SSC	5 subspaces 50K points	SSCOMP_Code (MATLAB)	52.5 s	10.2 s	4.6x

We see significant though variable gains achieved by CR-Sparse on GPU. We have observed that gain tends to increase for larger problem sizes. GPUs tend to perform better when problem size increases as the matrix/vector products become bigger. vmap and pmap tools provided by JAX can be used to easily parallelize the CR-Sparse algorithms over multiple data and processors.

Following table compares the runtime of linear operators in CR-Sparse on GPU vs PyLops on CPU for large size problems. Timings are measured for both forward and adjoint operati'ons.

Operator	Size	Fwd ref	Fwd our	Gain	Adj ref	Adj our	Gain
Diagonal matrix mult	n=1M	966 µs	95.7 µs	10x	992 µs	96.3 µs	10x
Matrix mult	(m,n)=(10K,10K)	11 ms	2.51 ms	4.37x	11.6 ms	2.51 ms	4.63x
First derivative	n=1M	2.15 ms	71.1 µs	30.2x	2.97 ms	186 µs	15.97x
HAAR DWT2, level=8	in=(4K,4K)	981 ms	34.4 ms	28.5x	713 ms	60.8 ms	11.7x

Limitations

Some of the limitations in the library come from the underlying JAX library. JAX is relatively new and still hasn't reached 1.0 level maturity. The programming model chosen by JAX places several restrictions on expressing the program logic. For example, JAX does not have support for dynamic or data dependent shapes in their JIT compiler. Thus, any algorithm parameter which determines the size/shape of individual arrays in an algorithm must be statically provided. E.g. for the greedy algorithms like OMP, the sparsity level K must be known in advance and provided as a static parameter to the API as the size of output array depends on K.

The control flow primitives like `lax.while_loop`, `lax.fori_loop` etc. in JAX require that the algorithm state flowing between iterations must not change shape and size. This makes coding of algorithms like OMP or SVT (singular value thresholding) very difficult. An incremental QR or Cholesky decomposition based implementation of OMP requires growing algorithm state. We ended up using a standard Python `for` loop for now but the JIT compiler simply unrolls it and doesn't allow for tolerance based early termination in them.

1D convolutions are slow in JAX on CPU #7961. This affects the performance of DWT/IDWT in `cr.sparse.dwt`. We are working on exploring ways of making it more efficient while keeping the API intact.

These restrictions necessitate good amount of creativity and a very disciplined coding style so that efficient JIT friendly solvers can be developed.

Future Work

Currently, work is underway to provide a JAX based implementation of `TFOCS` (Becker et al., 2011) in the dev branch. This will help us increase the coverage to a wider set of problems (like total variation minimization, Dantzig selector, l1-analysis, nuclear norm minimization, etc.). As part of this effort, we are expanding our collection of linear operators and building a set of indicator and projector functions on to convex sets and proximal operators (Parikh & Boyd, 2014). This will enable us to cover other applications such as SSC-L1 (Pourkamali-Anaraki et al., 2020). In future, we intend to increase the coverage in following areas: More recovery algorithms (OLS, Split Bergmann, SPGL1, etc.) and specialized cases (partial known support,); Bayesian Compressive Sensing; Dictionary learning (K-SVD, MOD, etc.); Subspace clustering; Image denoising, compression, etc. problems using sparse representation principles; Matrix completion problems; Matrix factorization problems; Model based / Structured compressive sensing problems; Joint recovery problems from multiple measurement vectors.

Acknowledgements

Shailesh would like to thank his Ph.D. supervisors Prof. Surendra Prasad and Prof. Brejesh Lall to inculcate his interest in this area and support him over the years in his exploration. He would also like to thank his employers, Interra Systems Inc., for allowing him to pursue his research interests along with his day job.

References

Abadi, M., Isard, M., & Murray, D. G. (2017). A computational model for TensorFlow: An introduction. *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 1–7. https://doi.org/10.1145/3088525.3088527

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, *2*(1), 183–202. https://doi.org/10.1137/080716542

Becker, S. R., Candès, E. J., & Grant, M. C. (2011). Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation*, *3*(3), 165. https://doi.org/10.1007/s12532-011-0029-5

Blumensath, T., & Davies, M. E. (2009). Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, *27*(3), 265–274. https://doi.org/10.1016/j.acha.2009.04.002

Blumensath, T., & Davies, M. E. (2010). Normalized iterative hard thresholding: Guaranteed stability and performance. *Selected Topics in Signal Processing, IEEE Journal of*, *4*(2), 298–309. https://doi.org/10.1109/jstsp.2010.2042411

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.2.5) [Computer software]. http://github.com/google/jax

Candes, E. J. (2008). The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, *346*(9-10), 589–592. https://doi.org/10.1016/j.crma.2008.03.014

Chen, S. S., Donoho, D. L., & Saunders, M. A. (2001). Atomic decomposition by basis pursuit. *SIAM Review*, *43*(1), 129–159. https://doi.org/10.1137/S003614450037906X

Dai, W., & Milenkovic, O. (2009). Subspace pursuit for compressive sensing signal reconstruction. *Information Theory, IEEE Transactions on*, *55*(5), 2230–2249. https://doi.org/10.1109/tit.2009.2016006

Daubechies, I., Defrise, M., & De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, *57*(11), 1413–1457. https://doi.org/10.1002/cpa.20042

Davenport, M. A., & Wakin, M. B. (2010). Analysis of orthogonal matching pursuit using the restricted isometry property. *Information Theory, IEEE Transactions on*, *56*(9), 4395–4401. https://doi.org/10.1109/tit.2010.2054653

Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, *52*(4), 1289–1306. https://doi.org/10.1109/tit.2006.871582

Elad, M. (2010). *Sparse and redundant representations*. Springer. https://doi.org/10.1007/978-1-4419-7011-4

Foucart, S. (2011). Recovering jointly sparse vectors via hard thresholding pursuit. *Proc. Sampling Theory and Applications (SampTA)],(May 2-6 2011)*.

Foucart, S., & Rauhut, H. (2013). *A mathematical introduction to compressive sensing*. Springer New York. https://doi.org/10.1007/978-0-8176-4948-7

Kim, S.-J., Koh, K., Lustig, M., Boyd, S., & Gorinevsky, D. (2007). An interior-point method for large-scale l1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, *1*(4), 606–617. https://doi.org/10.1109/JSTSP.2007.910971

Kumar, S. (2021). *CR-sparse companion* (0.2.1-docs) [Computer software]. https://doi.org/10.5281/zenodo.5749656

Lee, G., Gommers, R., Waselewski, F., Wohlfahrt, K., & O'Leary, A. (2019). PyWavelets: A python package for wavelet analysis. *Journal of Open Source Software*, *4*(36), 1237. https://doi.org/10.21105/joss.01237

Mallat, S. (2009). *A wavelet tour of signal processing: The sparse way*. Elsevier. https://doi.org/10.1016/b978-0-12-374370-1.x0001-8

Marques, E. C., Maciel, N., Naviner, L., Cai, H., & Yang, J. (2018). A review of sparse recovery algorithms. *IEEE Access*, *7*, 1300–1322. https://doi.org/10.1109/ACCESS.2018.2886471

MATLAB. (2018). *9.7.0.1190202 (R2019b)*. The MathWorks Inc.

Needell, D., & Tropp, J. A. (2009). CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, *26*(3), 301–321. https://doi.org/10.1016/j.acha.2008.07.002

Paige, C. C., & Saunders, M. A. (1982). LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, *8*(1), 43–71. https://doi.org/10.1145/355984.355989

Parikh, N., & Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in Optimization*, *1*(3), 127–239. https://doi.org/10.1561/2400000003

Pati, Y. C., Rezaiifar, R., & Krishnaprasad, P. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. *Signals, Systems and Computers, 1993. 1993 Conference Record of the Twenty-Seventh Asilomar Conference on*, 40–44. https://doi.org/10.1109/acssc.1993.342465

Pourkamali-Anaraki, F., Folberth, J., & Becker, S. (2020). Efficient solvers for sparse subspace clustering. *Signal Processing*, *172*, 107548. https://doi.org/10.1016/j.sigpro.2020.107548

Qaisar, S., Bilal, R. M., Iqbal, W., Naureen, M., & Lee, S. (2013). Compressive sensing: From theory to applications, a survey. *Journal of Communications and Networks*, *15*(5), 443–456. https://doi.org/10.1109/JCN.2013.000083

Ravasi, M., & Vasconcelos, I. (2019). PyLops–a linear-operator python library for large scale optimization. *arXiv Preprint arXiv:1907.12349*. https://arxiv.org/abs/1907.12349

Simpson, L., Combettes, P. L., & Müller, C. L. (2021). C-lasso - a python package for constrained sparse and robust regression and classification. *Journal of Open Source Software*, *6*(57), 2844. https://doi.org/10.21105/joss.02844

Torrence, C., & Compo, G. P. (1998). A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, *79*(1), 61–78. https://doi.org/10.1175/1520-0477(1998)079%3C0061:APGTWA%3E2.0.CO;2

Tropp, J. A. (2004). Greed is good: Algorithmic results for sparse approximation. *Information Theory, IEEE Transactions on*, *50*(10), 2231–2242. https://doi.org/10.1109/TIT.2004.834793

Yang, J., & Zhang, Y. (2011). Alternating direction algorithms for l1-problems in compressive sensing. *SIAM Journal on Scientific Computing*, *33*(1), 250–278. https://doi.org/10.1137/090777761

You, C., Robinson, D., & Vidal, R. (2016). Scalable sparse subspace clustering by orthogonal matching pursuit. *IEEE Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/CVPR.2016.425

Zhang, Y., Yang, J., & Yin, W. (2010). *User's guide for YALL1: Your algorithms for L1 optimization*. https://www.caam.rice.edu/~optimization/L1/YALL1/User_Guide/YALL1v1.0_User_Guide.pdf