# fnnls: An implementation of Fast Nonnegative Least Squares

Joshua Vendrow[1] **and Jamie Haddock**[1]

1 Department of Mathematics, University of California, Los Angeles

Summary

`fnnls` is a Python package that offers a fast algorithm for solving the nonnegative least squares problem. The Fast Nonnegative Least Squares (fnnls) algorithm was first presented in the paper "A fast non-negativity-constrained least squares algorithm" (Bro & De Jong, 1997). This algorithm exploits speed-up opportunities and improves upon the nonnegative least squares active-set algorithm of (Lawson & Hanson, 1995).

Given a matrix $\mathbf{Z} \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{x} \in \mathbb{R}^n$ the goal of nonnegative least squares is to find

$$\min_{\mathbf{d} \in \mathbb{R}^m} ||\mathbf{x} - \mathbf{Z}\mathbf{d}|| \text{ subject to } \mathbf{d} \geq 0.$$

The fnnls algorithm improves upon the computational effort of (Lawson & Hanson, 1995) by precomputing the values of $\mathbf{Z}^\top \mathbf{Z}$ and $\mathbf{Z}^\top \mathbf{x}$ and using them throughout the algorithm. If we assume that $\mathbf{Z}$ is tall ($m \gg n$) then this precomputation significantly decreases the computing time by replacing matrix and vector multiplications with computations of a smaller dimension.

The nonnegative least squares problem has many applications in the field of applied math and specifically as a subproblem for various matrix and tensor factorization algorithms, including nonnegative matrix factorization (NMF) (Lee & Seung, 2001) and nonnegative tensor decomposition (NTD) (Bro & others, 1997).

`fnnls` is currently in use in ongoing research on NMF and NTD methods. Efficient solution of the nonnegative least squares problem is the backbone of the new algorithms for computing hierarchical NMF and NTD models via a forward propagation and back-propagation procedure (Gao et al., 2019; Vendrow, Haddock, & Needell, 2020; Will et al., 2020).

Comparison to Other Algorithms

The standard implementation for nonnegative least squares is the `scipy.optimize.nnls` function within the SciPy open-source Python library (Virtanen et al., 2020). SciPy uses an implementation of the Lawson and Hanson algorithm, which was first presented in 1974 (Lawson & Hanson, 1995).

Below, we test the efficiency and accuracy of `fnnls` against the SciPy `scipy.optimize.nnls` function. We measure the time taken by each method over 100 runs each on random Gaussian and sparse uniform data generated separately for each run. In each run, we randomly generate $\mathbf{Z}$ and $\mathbf{x}$ and record the time spent by `fnnls` and `scipy.optimize.nnls`. For runs on Gaussian data, $\mathbf{Z}$ and $\mathbf{x}$ are generated with the `numpy.random.rand` function which draws each entry i.i.d. from the standard normal distribution, taking the absolute value of the outcome (Oliphant, 2006). For the runs on sparse uniform data, $\mathbf{Z}$ and $\mathbf{x}$ are generated with the `scipy.sparse.random` function which constructs a matrix

with density 0.1, where the nonzero entries are drawn i.i.d. from the uniform distribution $[0, 1)$. We graph the time consumption versus the size of the matrices. Here, dimension n indicates that we generate $\mathbf{Z} \in \mathbb{R}^{10n \times n}$ and $\mathbf{x} \in \mathbb{R}^{10n}$, yielding a tall matrix.
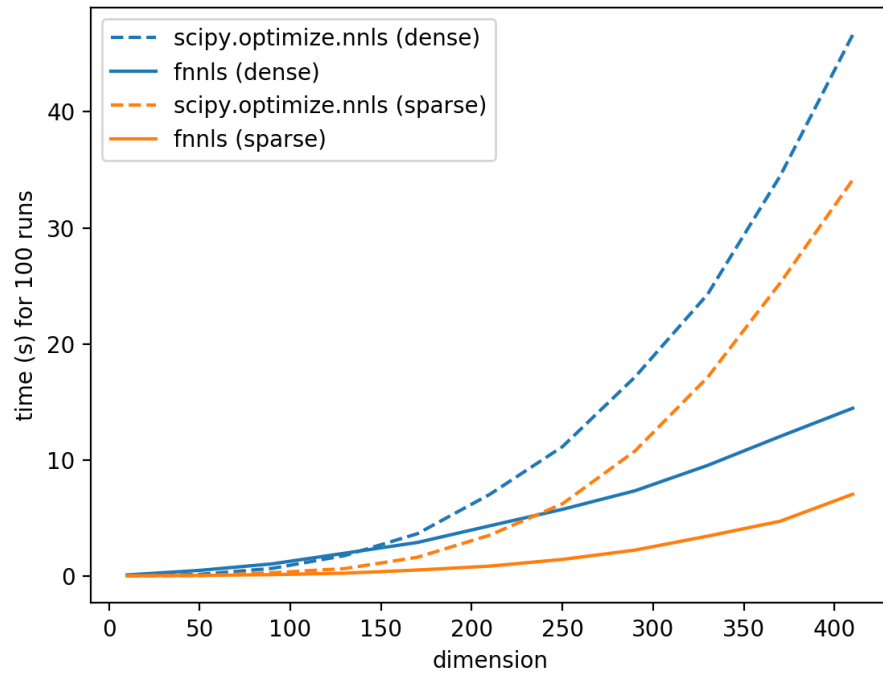

Fig. 1: The total running time over 100 runs of each algorithm on random Gaussian data (dense) and random uniform sparse data (sparse) of varying dimension. `fnnls` required significantly less time at higher dimensions for both types of matrices.

Let $\mathbf{d_f}$ and $\mathbf{d_n}$ be the solution vectors produced by `fnnls` and the SciPy `nnls` function, respectively. Then we compute the relative error between the solutions as

$$\frac{||\mathbf{d_f} - \mathbf{d_n}||_2}{||\mathbf{d_n}||_2}.$$

The average relative error between $\mathbf{d_f}$ and $\mathbf{d_n}$ across the 100 runs did not exceed 10^{-12} for any dimension of the Gaussian data, and did not exceed 10^{-14} for any dimension of the sparse uniform data.

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics: A Journal of the Chemometrics Society*, *11*(5), 393–401.

Bro, R., & others. (1997). PARAFAC. Tutorial and applications. *Chemometrics and intelligent laboratory systems*, *38*(2), 149–172.

Gao, M., Haddock, J., Molitor, D., Needell, D., Sadovnik, E., Will, T., & Zhang, R. (2019). Neural nonnegative matrix factorization for hierarchical multilayer topic modeling. In *Proc. Interational workshop on computational advances in multi-sensor adaptive processing*.

Lawson, C. L., & Hanson, R. J. (1995). *Solving least squares problems* (Vol. 15). Siam.

Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems* (pp. 556–562).

Oliphant, T. E. (2006). *A guide to numpy* (Vol. 1). Trelgol Publishing USA.

Vendrow, J., Haddock, J., & Needell, D. (2020). Neural nonnegative canonical polyadic decomposition for hierarchical tensor analysis.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. doi:https://doi.org/10.1038/s41592-019-0686-2

Will, T., Zhang, R., Sadovnik, E., Gao, M., Vendrow, J., Haddock, J., Molitor, D., et al. (2020). Neural nonnegative matrix factorization for hierarchical multilayer topic modeling.