




fnnls: An implementation of Fast Nonnegative Least Squares

Joshua Vendrow¹ and Jamie Haddock¹

DOI:

¹ Department of Mathematics, University of California, Los Angeles

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Submitted:

Published:

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC-BY](#)).

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 $\mathbf{Z}^T \mathbf{Z}$ and $\mathbf{Z}^T \mathbf{x}$ 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.

We note that the most significant computational task within the `fnnls` algorithm is an unconstrained least squares computation that is performed at each iteration. There are many different algorithms available to calculate the least squares solution and we have found experimentally that the choice of such function can significantly impact the running time of the `fnnls` algorithm and that the optimal choice of algorithm is data-dependent. For this reason, we include the choice of least-squares algorithm as an optional parameter of our package. As our default, we solve this least squares task by manually computing the pseudoinverse and performing a matrix-vector multiplication, which has shown to work well in practice. We additionally include an implementation of two possible algorithms to efficiently approximate the least squares solution: the randomized Kaczmarz algorithm (Kaczmarz, 1937; Strohmer & Vershynin, 2009) and the randomized Gauss-Seidel algorithm (Leventhal & Lewis, 2010).

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). Often the nonnegative least squares problems encountered during the course of computing iterative NMF and NTD algorithms are extremely similar across iterations, leading one to expect that the support of the solution will also be similar. For this reason, we include the estimated support of the solution as an optional parameter for our package. In practice, this has shown to improve the speed of computation; see documentation for further information.

`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` (`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 `nnls`. For runs on Gaussian data, \mathbf{Z} and \mathbf{x} are generated with the `numpy.random.randn` function which draws each entry i.i.d. from the standard normal distribution (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 on $[0, 1)$. 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.

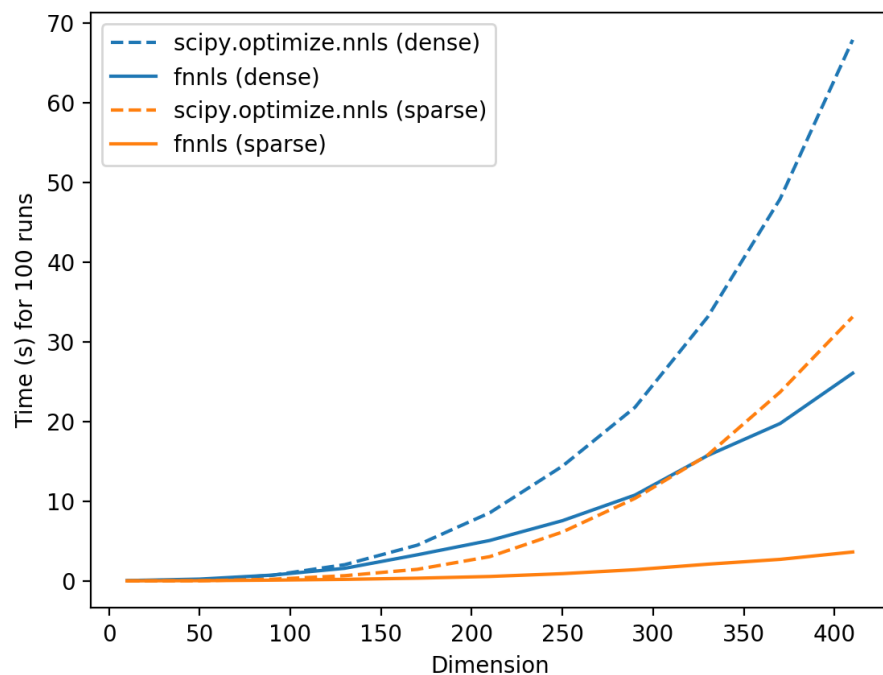


Figure 1: The total running time over 100 runs of each algorithm on random Gaussian data (dense) and random uniform sparse data (sparse) of varying dimension. `fnnls` required significantly less time at higher dimensions for both types of matrices.

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^{-14} for any dimension of the Gaussian data or the sparse uniform data.

Acknowledgements

The authors were partially supported by NSF CAREER DMS-1348721 and NSF BIG-DATA 1740325. They would also like to thank Jacob Moorman for excellent advice and guidance.

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*.
- Kaczmarz, S. (1937). Angenäherte auflösung von systemen linearer gleichungen. *Bull. Int. Acad. Polon. Sci. Lett. Ser. A*, 335–357.
- 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).
- Leventhal, D., & Lewis, A. S. (2010). Randomized methods for linear constraints: Convergence rates and conditioning. *Mathematics of Operations Research*, 35(3), 641–654.
- Oliphant, T. E. (2006). *A guide to numpy* (Vol. 1). Trelgol Publishing USA.
- Strohmer, T., & Vershynin, R. (2009). A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2), 262.
- 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.