fnnls: An implementation of Fast Nonnegative Least Squares

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

1 Department of Mathematics, University of California, Los Angeles

DOI:

Software

- Review
- Repository
- Archive

Submitted:
Published:

Summary

`fnnls` is a python package that offers a fast algorithm for solving the nonnegative least square 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).

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

$$\min_{\mathbf{d}} ||\mathbf{x} - \mathbf{Zd}|| \text{ subject to } \mathbf{d} \geq 0.$$

The fnnls algorithm improves the complexity of computation by precomputing the values of $\mathbf{Z^TZ}$ and $\mathbf{Z^Tx}$ and using them throughout the algorithm. If we assume that $\mathbf{Z}$ is tall, so $m \gg n$, then this precomputation significantly decreases the computational complexity by replacing matrix and vector multiplications with computations of a smaller dimension.

The nonnegative least square problem has many applications to problems in the field of applied math and specifically as a subproblem for various matrix factorization algorithms, including nonnegative matrix/tensor factorization (NMF and NTF) and tensor rank decomposition (canonical polyadic decomposition) (Bro & others, 1997).

`fnnls` is currently in use in ongoing research on nonnegative matrix factorization methods.

Comparison to Other Algorithms

The standard implementation for nonnegative least squares is the scipy.optimize.nnls function within the SciPy open-source Python library. 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 nnls function. We measure the time taken for each method over 25 repeated runs on a gaussian random $\mathbf{Z}$ and $\mathbf{x}$, generated seperately for each run. We graph the time consumption as a function of the size of the matrices. Here, a dimension of n indicates that we generate $\mathbf{Z} \in \mathbb{R}^{10nxn}$ and $\mathbf{x} \in \mathbb{R}^{10n}$, giving us a tall matrix.

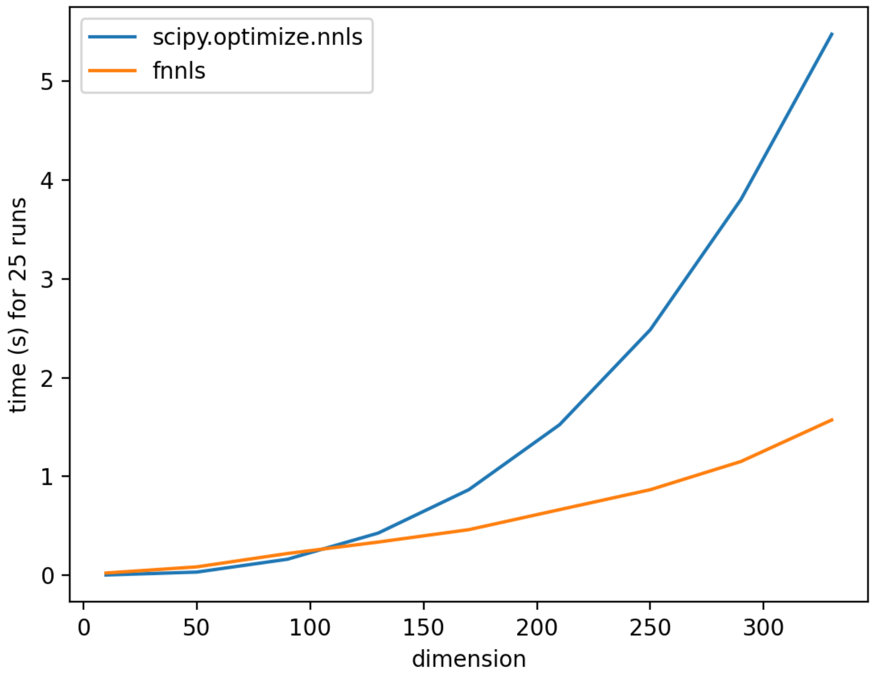

Fig. 1: The total running time over 25 runs at each dimension when running each algorithm on random data for varying dimension. We see that our method consumed significantly less time at higher dimensions, indicating a better complexity in terms of the size of the matrix.

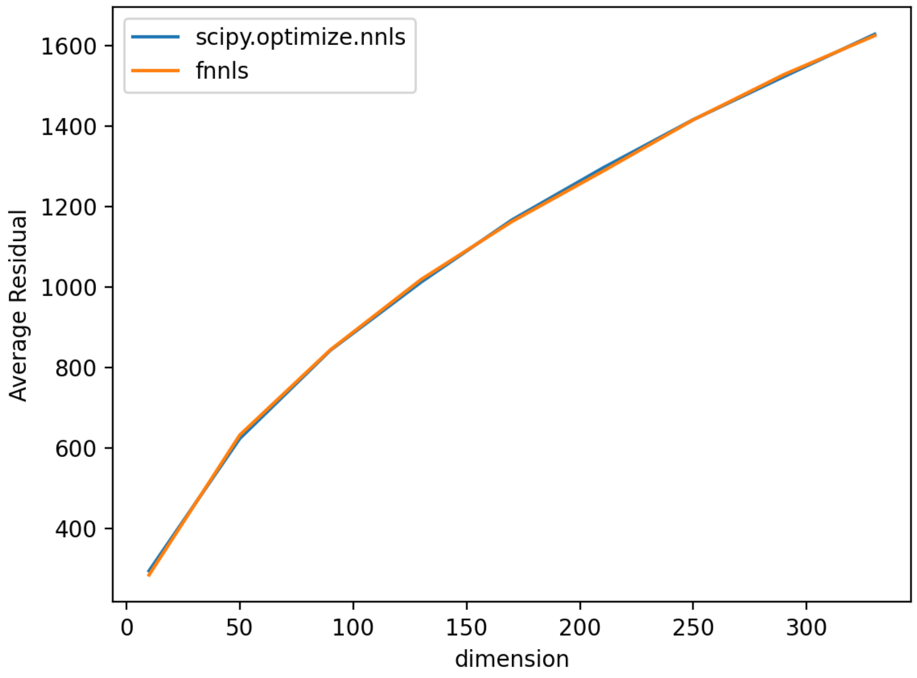

Fig. 2: The average residual produced when running each algorithm on random data

for varying dimension. We see that there is no visible difference in the results produced by these two methods.

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.

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