The Journal of Open Source Software

FastPCA: An R package for fast singular value decomposition

Kimberly R. Ward[1], **Mitchell Hayes**[2], **Lauren C Peres**[3], **Brooke L Fridley**[4], **Steven Eschrich**[5], **and Alex C Soupir**[2, 5]

1 Department of Cutaneous Oncology, Moffitt Cancer Center **2** Department of Genitourinary Oncology, Moffitt Cancer Center **3** Department of Cancer Epidemiology, Moffitt Cancer Center **4** Health Services & Outcomes Research, Children's Mercy **5** Department of Bioinformatics and Biostatistics, Moffitt Cancer Center

DOI:

Software

- Review
- Repository
- Archive

Submitted:
Published:

Summary

The FastPCA package provides an interface to optimized matrix multiplication libraries (libtorch) for the purpose of singular value decomposition. Using FastPCA to perform randomized singular value decomposition (SVD, Halko, Martinsson, & Tropp (2011)) with the torch or pytorch backend drastically reduces the computational time compared to base R prcomp and other truncated singular value decomposition. Developed for biological data such as single-cell RNA-sequencing, spatial transcriptomics, or matrix-assisted laser desorption/ionization (MALDI imaging), FastPCA can efficiently and accurately identify the leading singular values in high-dimensional data.

Statement of Need

Over the last decade, advances in single-cell and spatial profiling technologies have dramatically increased the scale and resolution of biological data (Lim, Wang, Buzdin, & Li (2025)). Multiple custom and commercially available technologies produce tens of thousands of samples (e.g., spots, cells, pixels) and thousands of measured features (i.e., genes, peaks, etc) per sample which can require tens to hundreds of gigabytes (Zaima, Hayasaka, Goto-Inoue, & Setou (2010)). Creating linear or non-linear combinations of the input space in lower-dimensional space enables more efficient downstream analysis. However, this is typically done with packages like irlba (Baglama, Reichel, & Lewis (2011)) where the process of identifying singular values is iterative and single-thread limited. Other packages can improve efficiency by using an API for R (such as reticulate for python) but is not multithreaded without modification to the source code (Li, Meisner, & Albrechtsen (2023)). The qrpca R package (Souza, Quanfeng, Shen, Peng, & Mu (2022)) uses torch (Falbel & Luraschi (2020); Paszke et al. (2019)) to perform QR-based principal component analysis (PCA), but doesn't produce truncated singular values resulting in incredibly large memory requirements for large matrices. For biological studies, parameter tuning and rapid iteration (e.g., normalization methods, subclustering for identifying specific cell types) are essential to identify biologically meaningful signals, which typically reside in early dimensions, enabling the avoidance of the full decomposition.

The FastPCA R package was developed to address this critical need. FastPCA has access to the torch backend through R, as well as pytorch with reticulate (Ushey, Allaire, & Tang (2017)). We've also included vignettes to demonstrate FastPCA's utility and functionality (https://acsoupir.github.io/FastPCA/).

Functionality

FastPCA uses irlba as the default backend for calculating singular values for compatability, but offers access to libtorch via python pytorch or R torch packages. Using the pytorch or torch backends support setting the number of threads where the backend provides it. There are two functions within FastPCA that aid in the setup of the environments:

- `setup_py_env()`: creates a python environment (either with 'conda' or 'virtualenv') for using in the event that pytorch is wanted for the backend
- `start_FastPCA_env()`: starts the environment created with `setup_py_env()` to use for pytorch backend

Then, there are processing functions that work on input matrices or intermediate lists produced by FastPCA:

- `prep_matrix()`: converts expected data types (rows/columns) into FastPCA format (rows as samples, columns as features). Also supports transformations such as log2 and scaling.
- `FastPCA()`: performs either exact SVD (though not recommended other than benchmarking) or truncated singular value decomposition with irlba or randomized SVD. Returns a list containing matrices of singular vectors U and V^T, and vector of singular values S
- `get_pc_scores()`: calculates the principal component scores from the output of `FastPCA()` (U matrix which contains left singular vectors, S vector containing the singular values, and V^T matrix of the right singular vectors) which are typically expected for downstream analyses
- `umap()`: uses either uwot (R) or umap-learn (python) for visualization of the principal component scores

Demonstration of Improvement

To demonstrate the improvements of FastPCA, we use our previous data set of single-cell spatial transcriptomics of kidney cancer. The dataset is publicly available on Zenodo and can be downloaded locally to be used (Soupir et al. (2024)). Since the main benefits of FastPCA are in its application of singular value decomposition, that is what will be focused on. The counts for each cell were extracted from the Seurat object in the Nanostring assay, which contains the expression of 978 probes (959 genes, 19 negative control probes) in 199,112 cells. Normalization was performed using `prep_matrix()`, applying a log transformation, scaling (mean centering and unit variance), and transposing for samples to be rows and columns to be the gene features. The full script can be found https://github.com/ACSoupir/FastPCA/docs_acs/paper/benchmarking_script.R.

Experiments were performed using FastPCA, both randomized SVD and exact, irlba, pcaone, and bigstatsr, both partial and randomized partial SVD. Performance measures were collected from a M3 Pro MacBook Pro with 36GB unified RAM. FastPCA (4 cores), bigstatsr partial (1 core) and random (4 cores), irlba (1 core), and pcaone (1 core) were assessed. Time was profiled with `bench::mark()` function. The elapsed time of FastPCA calculating the singular value decomposition on the full data took 10.74 seconds on average and the randomized SVD with FastPCA taking 8.46 seconds on average (**Table 1**, **Figure 1**). The next fastest were the methods with PCAone which took 27.29 and 29.58 seconds on average for "alg1" and "alg2", respectively.

Table 1: Time to calculate singular value decomposition using different R packages and methods.

Method	Min	MEDIAN	MEAN	MAX
`FastPCA` (rSVD)	8.35s	8.43s	8.46s	8.79s
`FastPCA` (Exact)	10.61s	10.65s	10.74s	11.27s
`Irlba`	6.89m	7.06m	7.17m	7.78m
`pcaone` (Alg1)	27.1s	27.27s	27.29s	27.54s
`pcaone` (Alg2)	28.59s	29.01s	29.58s	32.59s
`bigstatsr` (rSVD)	50.0s	50.8s	51.15s	54.34s
`bigstatsr` (Partial)	1.74m	1.78m	1.77m	1.8m
`BiocSingular` (rSVD)	10.11m	10.26m	10.39m	10.78m

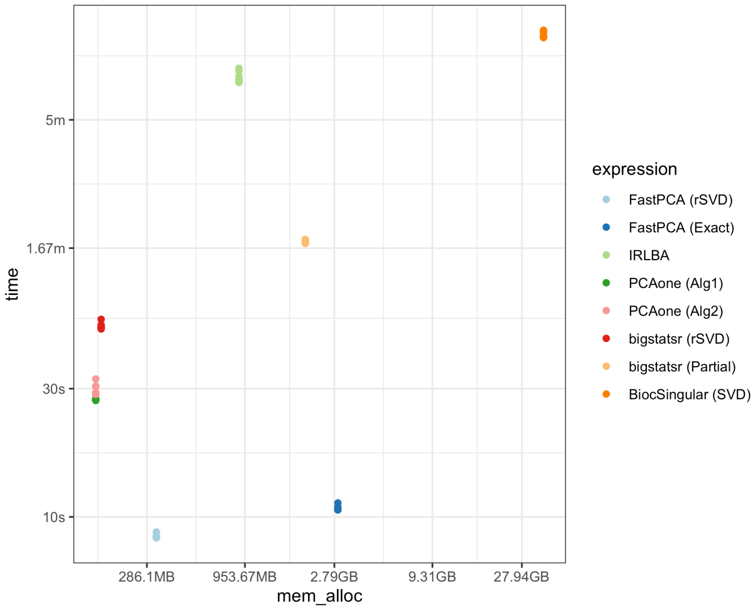

Figure 1: Time and memory usage of singular value decomposition methods.

Memory is another area that may dictate methods used. We also used the same `bench::mark()` calls as assessing memory. PCAone (alg1 and alg2) and `bigstatsr` randomized partial SVD used between 152.7 MB and 163.1 MB (Table 2). `FastPCA`, with calculating randomized SVD, used 321.4 MB and `irlba`, which is often used for single-cell studies with Seurat, consumed 886.2 MB. Surprisingly, `bigstatsr::big_svd()` used just over 2000 MB and `FastPCA` calculating the exact SVD matrices used 2990.1 MB. The memory use by `BioSingular::runSVD()`, with `RandomParam()` method (for approximate SVD) and same oversampling/power interation parameters as the other random SVD methods), had a staggering 36.5GB logged with `bench::mark()`.

Table 2: Memory usage for the calculation of singular value decomposition with different R packages and methods.

Method	Memory (MB)
FastPCA (rSVD)	321.4
FastPCA (Exact)	2990.1
Irlba	886.2
PCAone (alg1)	152.7

Method	Memory (MB)
PCAone (alg2)	152.7
bigstatsr (random SVD)	163.1
bigstatsr (Partial)	2001.5
BiocSingular (rSVD)	37324.8

Lastly, to assess the quality of the top singular values, we plotted the 100 derived values from all methods against all others. Using the `irlba` as the comparator due to its common use in workflows for deriving eigenvectors and eigenvalues, we observe that the exact singular values from `FastPCA` are essentially identical, as are those from both `bigstatsr` methods (**Figure 2**). `FastPCA` using the randomized singular vector approach showed slight deviation in the lower end singular values, though, still maintained high similarity to the exact approach. PCAone with "alg1" also showed a slight deviation from `irlba` while maintaining the main dimensions' singular values.

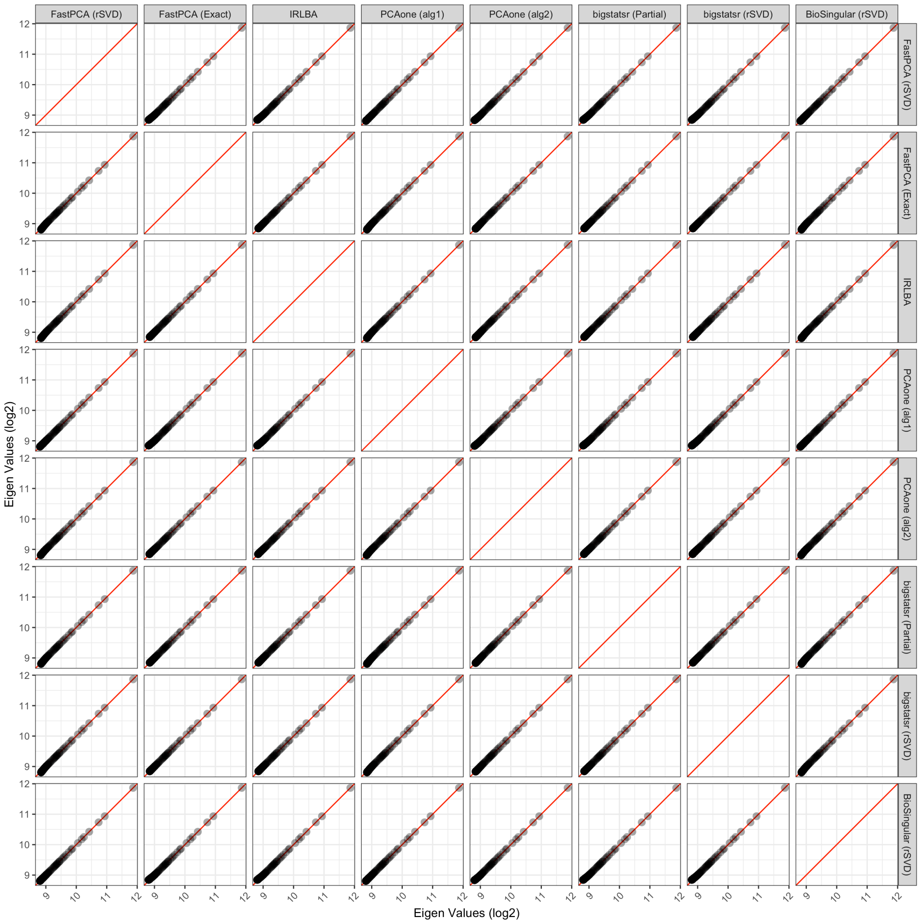

Figure 2: Singular values derived from different methods plotted against each other.

Conclusion

`FastPCA` provides an R-native, accelerated route to principal component analysis on large matrices by coupling randomized SVD with `torch`/`pytorch` backends and a familiar API. In our kidney cancer spatial-transcriptomics benchmark (199,112 cells × 978 features), FastPCA's randomized SVD achieved 8.5 s mean runtime on commodity hardware while matching the early singular values from exact decompositions and widely used methods (IRLBA in irlba). Memory use remained modest for FastPCA's randomized SVD implementation (321.4 MB), with predictable increase for exact decompositions when full matrices are returned. These results, together with setup helpers (optional Python via reticulate, otherwise R-only), make FastPCA a practical default for exploratory analyses and iterative pipelines (e.g., normalization alternatives, sub-clustering, visualization) where the top components contain the most biological information. Future work will extend functionality to sparse-matrix coverage and GPU pathways.

Acknowledgements

This work was supported in part by effort on NIH R01CA279065. We would like to thank Dr. Oscar Ospina for his testing of the FastPCA in a single-cell workflow to validate its similarity to values derived from Seurat (irlba) in real data.

References

Baglama, J., Reichel, L., & Lewis, B. W. (2011, May). Irlba: Fast truncated singular value decomposition and principal components analysis for large dense and sparse matrices. The R Foundation.

Falbel, D., & Luraschi, J. (2020, August). Torch: Tensors and neural networks with 'GPU' acceleration. The R Foundation.

Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, *53*(2), 217–288. doi:10.1137/090771806

Li, Z., Meisner, J., & Albrechtsen, A. (2023). Fast and accurate out-of-core PCA framework for large scale biobank data. *Genome Res.*, *33*(9), 1599–1608.

Lim, H. J., Wang, Y., Buzdin, A., & Li, X. (2025). A practical guide for choosing an optimal spatial transcriptomics technology from seven major commercially available options. *BMC Genomics*, *26*(1), 47.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., et al. (2019). PyTorch: An imperative style, high-performance deep learning library.

Soupir, A. C., Hayes, M. T., Peak, T. C., Ospina, O., Chakiryan, N. H., Berglund, A. E., Stewart, P. A., et al. (2024). Increased spatial coupling of integrin and collagen IV in the immunoresistant clear-cell renal-cell carcinoma tumor microenvironment. *Genome Biol.*, *25*(1), 308.

Souza, R. S. de, Quanfeng, X., Shen, S., Peng, C., & Mu, Z. (2022). Qrpca: A package for fast principal component analysis with GPU acceleration.

Ushey, K., Allaire, J. J., & Tang, Y. (2017, March). Reticulate: Interface to 'python'. The R Foundation.

Zaima, N., Hayasaka, T., Goto-Inoue, N., & Setou, M. (2010). Matrix-assisted laser desorption/ionization imaging mass spectrometry. *Int. J. Mol. Sci.*, *11*(12), 5040–5055.