

OnlinePCA.jl: A Julia Package for Out-of-core and Sparse Principal Component Analysis

Koki Tsuyuzaki^{1, 2}

¹ Department of Artificial Intelligence Medicine, Graduate School of Medicine, Chiba University, Japan ² Laboratory for Bioinformatics Research, RIKEN Center for Biosystems Dynamics Research, Japan

DOI: [DOIunavailable](#)

Software

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

Editor: [Pending Editor](#) ↗

Reviewers:

- [@Pending Reviewers](#)

Submitted: N/A

Published: N/A

License

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

Summary

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique (Meng, 2016; Stein-O'Brien, 2018), but becomes computationally prohibitive for large data matrices. Recent advances in single-cell omics have led to datasets with millions of cells, for which standard PCA implementations often fail to scale. `OnlinePCA.jl` is a Julia package that addresses this challenge by providing scalable PCA algorithms (<https://github.com/rikenbit/OnlinePCA.jl>).

Statement of need

PCA is widely used across diverse fields including face recognition (Pablo, 2002), animal behavior (Stephens, 2008), genomics (Meng, 2016; Stein-O'Brien, 2018), population genetics (Li, 2023; Novembre, 2008), and molecular dynamics (David, 2014). However, large data matrices often exceed available memory. An out-of-core (OOC) implementation—where data subsets are loaded from disk—combined with sparse matrix support is highly desirable (Tsuyuzaki, 2020).

New features since version 0.3.0

`OnlinePCA.jl` previously provided OOC PCA functions and `tenxpca` for 10X-HDF5 sparse matrices (Tsuyuzaki, 2020). These implementations were designed for “short and fat” matrices with few rows (samples) and many columns (features). Since version 0.3.0, the following features have been introduced:

- *Support for Matrix Market and Binary COO formats*: We implemented `mm2bin` and `bin_coo2bin` for binary conversion of sparse data, and extended `sumr` with a `sparse_mode` option.
- *`sparse_rsvd`*: Generalizes `tenxpca` to Matrix Market format.
- *`exact_ooc_pca`*: Designed for “tall and thin” matrices, this function computes the covariance matrix in OOC manner followed by eigendecomposition. Supports CSV, MM, and BinCOO formats. Results are mathematically equivalent to offline PCA.
- *Adjustable chunk size*: Both `sparse_rsvd` and `exact_ooc_pca` include a `chunksize` option for memory control.

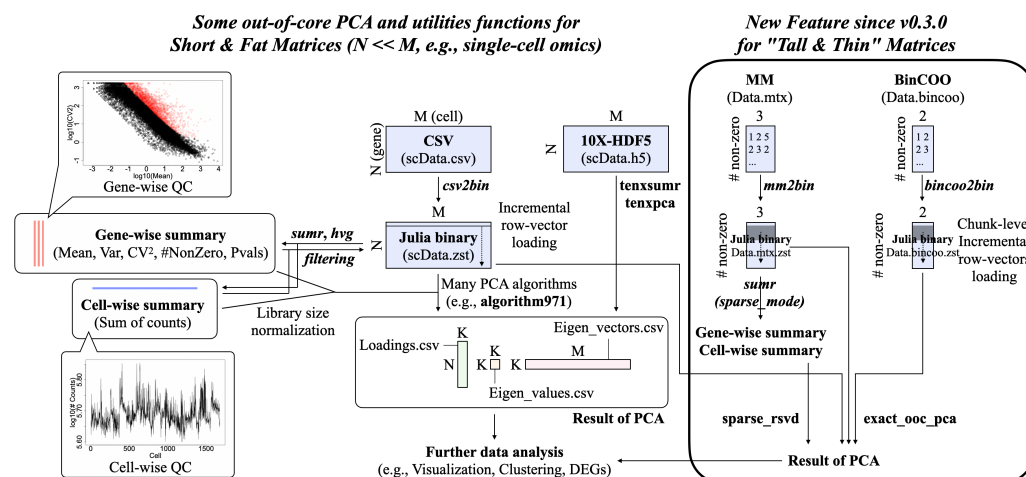


Figure 1: Overview of workflow in OnlinePCA.jl since v0.3.0.

Example

PCA can be easily reproduced on any machine where Julia is pre-installed by using the following commands in the Julia REPL window:

Installation

First, install OnlinePCA.jl from the official Julia package registry or directly from GitHub:

```
# Install OnlinePCA.jl from Julia General
```

```
julia> Pkg.add("OnlinePCA")
```

```
# or GitHub for the latest version
```

```
julia> Pkg.add(url="https://github.com/rikenbit/OnlinePCA.jl.git")
```

Preprocess of CSV

Then, write a synthetic data as a CSV file, convert it to a compressed binary format using Zstandard, and prepare summary statistics for PCA. Matrix Market (MM) format is also supported for sparse matrices.

```
using OnlinePCA
using OnlinePCA: write_csv
using Distributions
using DelimitedFiles
using SparseArrays
using MatrixMarket

# CSV
tmp = mktempdir()
data = Int64.(ceil.(rand(NegativeBinomial(1, 0.5), 300, 99)))
data[1:50, 1:33] .= 100*data[1:50, 1:33]
data[51:100, 34:66] .= 100*data[51:100, 34:66]
data[101:150, 67:99] .= 100*data[101:150, 67:99]
write_csv(joinpath(tmp, "Data.csv"), data)

# Binarization (Zstandard)
```

```
csv2bin(csvfile=joinpath(tmp, "Data.csv"),
        binfile=joinpath(tmp, "Data.zst"))

# Matrix Market (MM)
mmwrite(joinpath(tmp, "Data.mtx"), sparse(data))

# Binarization (Zstandard)
csv2bin(csvfile=joinpath(tmp, "Data.csv"),
        binfile=joinpath(tmp, "Data.zst"))

# Summary of data for CSV/Dense Matrix
dense_path = mktempdir()
sumr(binfile=joinpath(tmp, "Data.zst"), outdir=dense_path)
```

PCA using Halko's method on CSV input

This example performs PCA using Halko's randomized SVD method on dense CSV input. Results are visualized using the subplots function defined in the README (<https://github.com/rikenbit/OnlinePCA.jl?tab=readme-ov-file#setting-for-plot>).

```
out_halko = halko(input=joinpath(tmp, "Data.zst"), dim=3,
                  scale="log",
                  rowmeanlist=joinpath(dense_path, "Feature_LogMeans.csv"))

subplots(out_halko[1], group)
```

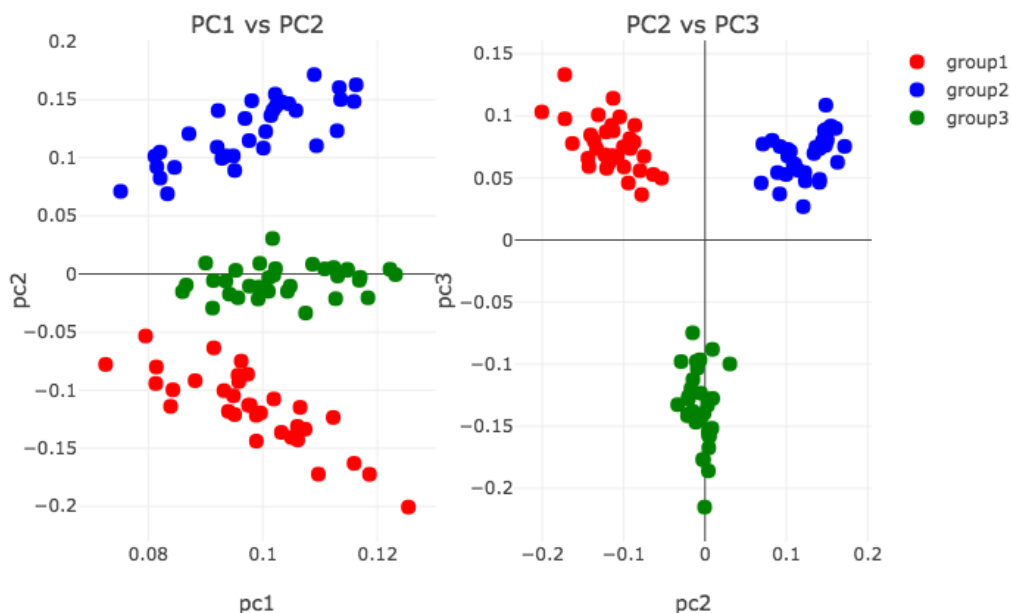


Figure 2: Output of halko against CSV format.

Preprocessing sparse data in Matrix Market format

The following code converts a sparse matrix in MM format into a binary compressed format and computes summary statistics for PCA.

```
# Sparsification + Binarization (Zstandard + MM format)
mm2bin(mmfile=joinpath(tmp, "Data.mtx"),
       binfile=joinpath(tmp, "Data.mtx.zst"))

sparse_path = mktempdir()
sumr(binfile=joinpath(tmp, "Data.mtx.zst"),
     outdir=sparse_path, mode="sparse_mm")
```

PCA using sparse_rsvd on Matrix Market input

This example performs PCA using the `sparse_rsvd` method, designed for sparse input data in MM format. The top 3 components are visualized.

```
out_sparse_rsvd = sparse_rsvd(
    input=joinpath(tmp, "Data.mtx.zst"),
    scale="ftt",
    rowmeanlist=joinpath(sparse_path, "Feature_FTTMeans.csv"),
    dim=3, chunksize=100)

subplots(out_sparse_rsvd[1], group)
```

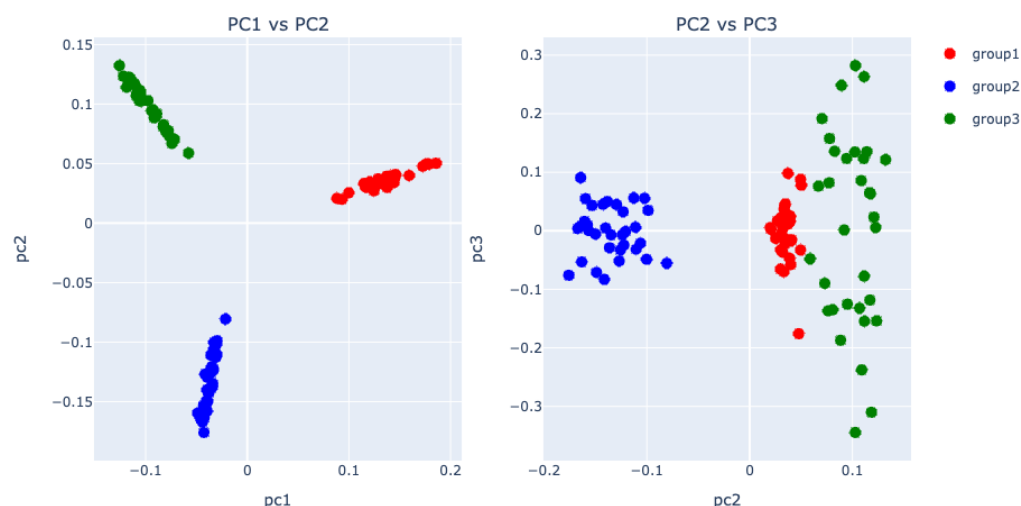


Figure 3: Output of `sparse_rsvd` against MM format.

Preparing another sparse matrix for `exact_ooc_pca`

This example generates a BinCOO format data to simulate “tall and thin” matrix and compresses it by `bincoo2bin` for use with `exact_ooc_pca`.

```
# Binary COO (BinCOO)
tmp2 = mktempdir()
data2 = Int64.(ceil.(rand(Binomial(1, 0.2), 99, 33)))
data2[1:33, 1:11] .= 1
data2[34:66, 12:22] .= 1
data2[67:99, 23:33] .= 1

bincoofile = joinpath(tmp2, "Data2.bincoo")
open(bincoofile, "w") do io
    for i in 1:size(data2, 1)
```

```

for j in 1:size(data2, 2)
    if data2[i, j] != 0
        println(io, "$i $j")
    end
end
end
end

# Binarziation (BinCOO + Zstandard)
bincoo2bin(bincoofile=bincoofile, binfile=joinpath(tmp2, "Data2.bincoo.zst"))

```

PCA using exact_ooc_pca on BinCOO input

Here, we apply `exact_ooc_pca`, which computes the full covariance matrix in a OOC manner and performs eigendecomposition.

```

# Sparse-mode (BinCOO)
out_exact_ooc_pca_sparse_bincoo = exact_ooc_pca(
    input=joinpath(tmp2, "Data2.bincoo.zst"),
    scale="raw", dim=3, chunksize=10, mode="sparse_bincoo")

subplots(out_exact_ooc_pca_sparse_bincoo[3], group)

```

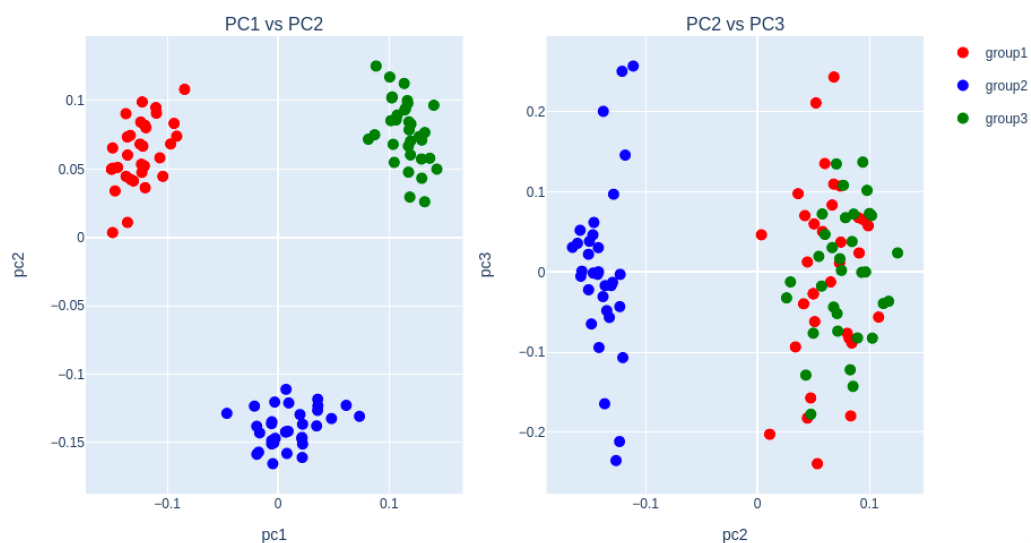


Figure 4: Output of `exact_ooc_pca` against BinCOO format.

For more details, see the README.md of `OnlinePCA.jl` at <https://github.com/rikenbit/OnlinePCA.jl>.

Related work

There are various implementations of PCA and some of them are OOC-type or sparse-type (Li, 2023; Moreno, 2022; Pedregosa, 2011) but `OnlinePCA.jl` is the only tool that supports both OOC computation and sparse data formats (e.g., 10X-HDF5, MM, BinCOO).

Function Name	Language	OOB	Sparse Format
prcomp/princomp	R	No	-
sklearn.decomposition.PCA	Python	No	-
MultivariateStats.PCA	Julia	No	-
oocRPCA::oocPCA_CSV	R	Yes	-
sklearn.decomposition.IncrementalPCA	Python	Yes	-
dask_ml.decomposition.PCA	Python	Yes	-
PCAone	R/C++	Yes	-
irlba::prcomp_irlba	R	No	dgCMatrix
sklearn.decomposition.TruncatedSVD	Python	No	scipy.sparse
tenxpca	Julia	Yes	10X-HDF5
sparse_rsvd	Julia	Yes	MM
exact_ooc_pca	Julia	Yes	CSV/MM/BinCOO

For a more comprehensive comparison, see the Figure 2 in Tsuyuzaki (2020).

References

- David, C. C. et al. (2014). Principal component analysis: A method for determining the essential dynamics of proteins. *Methods Mol Biol*, 1084, 193–226. https://doi.org/10.1007/978-1-62703-658-0_11
- Li, Z. et al. (2023). Fast and accurate out-of-core PCA framework for large scale biobank data. *Genome Research*, 33, 1599–1608. <https://doi.org/10.1101/gr.277525.122>
- Meng, C. et al. (2016). Dimension reduction techniques for the integrative analysis of multi-omics data. *Briefings in Bioinformatics*, 17(4), 628–641. <https://doi.org/10.1093/bib/bbv108>
- Moreno, M. et al. (2022). Scalable transcriptomics analysis with Dask: Applications in data science and machine learning. *BMC Bioinformatics*, 23(514). <https://doi.org/10.1186/s12859-022-05065-3>
- Novembre, J. et al. (2008). Genes mirror geography within Europe. *Nature*, 456, 98–101. <https://doi.org/10.1038/nature07331>
- Pablo, N. et al. (2002). Analysis and comparison of eigenspace-based face recognition approaches. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(7), 817–830. <https://doi.org/10.1142/S0218001402002003>
- Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85), 2825–2830.
- Stein-O'Brien, G. L. et al. (2018). Enter the matrix: Factorization uncovers knowledge from omics. *Trends in Genetics*, 34(10), 790–805. <https://doi.org/10.1016/j.tig.2018.07.003>
- Stephens, G. J. et al. (2008). Dimensionality and dynamics in the behavior of c. elegans. *PLOS Computational Biology*, 4(4), e1000028. <https://doi.org/10.1371/journal.pcbi.1000028>
- Tsuyuzaki, K. et al. (2020). Benchmarking principal component analysis for large-scale single-cell RNA-sequencing. *Genome Biology*, 21(1). <https://doi.org/10.1186/s13059-019-1900-3>