The Journal of Open Source Software

# 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:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:**

**Submitted:** 22 October 2025
**Published:** unpublished

## Summary

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique for identifying a small number of components that capture the maximum variance in high-dimensional data (Meng, 2016; Stein-O'Brien, 2018). PCA has been applied across various fields of data science, including face recognition (Pablo, 2002), animal behavior analysis (Stephens, 2008), omics studies (Meng, 2016; Stein-O'Brien, 2018; Tsuyuzaki, 2020), population genetics (Li, 2023; Novembre, 2008), and molecular dynamics simulations (David, 2014).

Despite its broad applicability, PCA becomes computationally prohibitive for large data matrices, making it difficult to apply in practice. In particular, recent advances in single-cell omics have led to datasets with millions of cells, for which standard PCA implementations often fail to scale. To meet this requirement, I originally developed `OnlinePCA.jl`, which is a Julia package to perform some PCA algorithms (https://github.com/rikenbit/OnlinePCA.jl).

## Statement of need

PCA is a workhorse algorithm for most data science tasks. However, as the size of the data matrix increases, it often becomes too large to fit into memory. In such cases, an out-of-core (OOC) implementation — where only subsets of data stored on disk are loaded into memory for computation — is desirable. Additionally, representing the data in a sparse matrix format, where only non-zero values and their coordinates are stored, is computationally advantageous (Tsuyuzaki, 2020). Therefore, a PCA implementation that supports both OOC computation and sparse data handling is highly desirable.

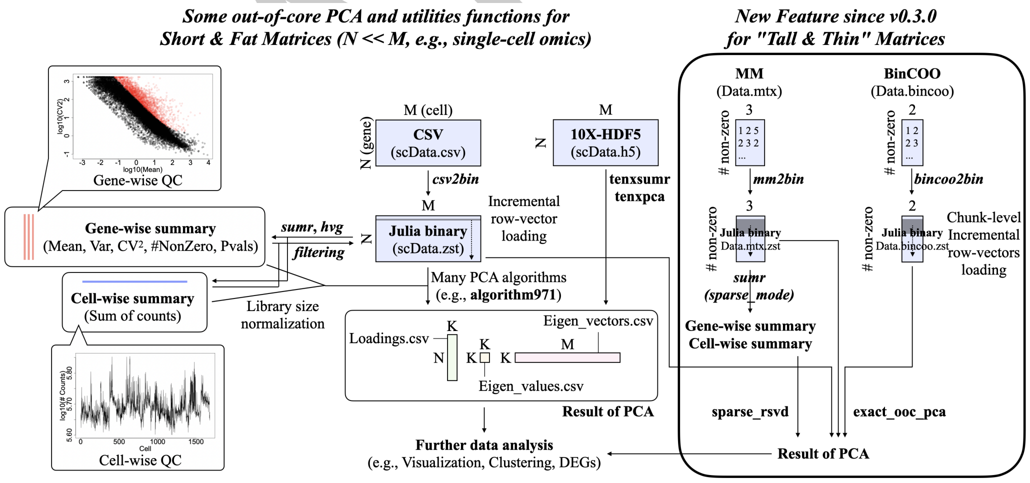
### New features since version 0.3.0

`OnlinePCA.jl` has been provided some OOC PCA functions such as `oja`, `ccipca`, `gd`, `rsgd`, `svrg`, `rsvrg`, `orthiter`, `arnoldi`, `lanczos`, `halko`, `algorithm971`, `rbkiter`, `singlepass`, and `singlepass2` and an OOC/Sparse PCA function (`tenxpca`) only for a sparse matrix in HDF5 files generated by 10X Cell Ranger (10X-HDF5) (Tsuyuzaki, 2020). These implementations were designed to take "CSV" files containing "short and fat" matrices—that is, matrices with relatively few rows (samples) and a large number of columns (features)—as typically seen in large-scale single-cell datasets.

Since version 0.3.0, the following new features have been introduced:

- *Support for Matrix Market and Binary COO formats*: Many large-scale datasets are represented as extremely sparse matrices, where only a small fraction of entries are non-zero. One widely used format for such sparse data is the Matrix Market (MM) format, which stores non-zero values along with their corresponding (x, y) coordinates in three columns. To handle this format, we implemented the `mm2bin` function for binary conversion and extended the `sumr` function with a `sparse_mode` option to extract summary statistics efficiently. Furthermore, we introduced a new format called Binary COO (BinCOO), which assumes all non-zero values are 1 and stores only the (x, y) coordinates. The corresponding function `bincoo2bin` has also been newly implemented.

- `sparse_rsvd`: In earlier versions of `OnlinePCA.jl` (Tsuyuzaki, 2020), the `tenxpca` function was implemented specifically for performing PCA on a sparse matrix in 10X-HDF5. The `sparse_rsvd` function generalizes this approach to a sparse matrix as Matrix Market (MM) format. It assumes as input a binary-converted sparse matrix generated by `mm2bin`, along with summary statistics computed using `sumr` with the `sparse_mode` option.
- `exact_ooc_pca`: The `exact_ooc_pca` function is designed for "tall and thin" matrices, where it performs OOC computation of the covariance matrix followed by eigendecomposition. Unlike other functions, it does not require precomputed summary statistics via `sumr` and instead directly accepts a binary-converted data matrix as input. Supported input formats include CSV, Matrix Market (MM), and BinCOO. The eigendecomposition is performed using Julia's standard `eigen` function, and since this method does not rely on approximated solutions such as IRLBA or Randomized SVD, the resulting principal components are mathematically equivalent to those obtained by running offline PCA with the full data loaded into memory (Tsuyuzaki, 2020).
- Adjustable chunk size: The `sparse_rsvd` and `exact_ooc_pca` functions include a `chunksize` option, which allows users to control the amount of data loaded into memory at once. This feature is based on our previous benchmarking with large-scale single-cell datasets (Tsuyuzaki, 2020), where we found that processing data in moderately sized chunks, rather than row by row, significantly improved computational efficiency.



Some out-of-core PCA and utilities functions for Short & Fat Matrices (N << M, e.g., single-cell omics)
New Feature since v0.3.0 for "Tall & Thin" Matrices
Gene-wise QC
Gene-wise summary (Mean, Var, CV², #NonZero, Pvals)
Cell-wise summary (Sum of counts)
Cell-wise QC
sumr, hvg
filtering
Library size normalization
M (cell)
N (gene)
CSV (scData.csv)
csv2bin
Julia binary (scData.zst)
10X-HDF5 (scData.h5)
tenxsumr
tenxpca
Incremental row-vector loading
Many PCA algorithms (e.g., algorithm971)
Loadings.csv
Eigen_vectors.csv
Eigen_values.csv
Result of PCA
Further data analysis (e.g., Visualization, Clustering, DEGs)
MM (Data.mtx)
BinCOO (Data.bincoo)
# non-zero
mm2bin
bincoo2bin
Julia binary Data.mtx.zst
Julia binary Data.bincoo.zst
sumr (sparse_mode)
Gene-wise summary
Cell-wise summary
Chunk-level Incremental row-vectors loading
sparse_rsvd
exact_ooc_pca
Result of PCA

**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:

```julia
# 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. 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)
```

## Setting for plot

Define a helper function to visualize the results of PCA using the `PlotlyJS.jl` package. It generates two subplots: PC1 vs PC2 and PC2 vs PC3, with color-coded groups.

```
using DataFrames
using PlotlyJS

function subplots(respca, group)
  # data frame
  data_left = DataFrame(pc1=respca[:,1], pc2=respca[:,2], group=group)
  data_right = DataFrame(pc2=respca[:,2], pc3=respca[:,3], group=group)
  # plot
  p_left = Plot(data_left, x=:pc1, y=:pc2, mode="markers",
      marker_size=10, group=:group)
  p_right = Plot(data_right, x=:pc2, y=:pc3, mode="markers",
      marker_size=10,
  group=:group, showlegend=false)
  p_left.data[1]["marker_color"] = "red"
  p_left.data[2]["marker_color"] = "blue"
  p_left.data[3]["marker_color"] = "green"
  p_right.data[1]["marker_color"] = "red"
  p_right.data[2]["marker_color"] = "blue"
  p_right.data[3]["marker_color"] = "green"
  p_left.data[1]["name"] = "group1"
  p_left.data[2]["name"] = "group2"
  p_left.data[3]["name"] = "group3"
  p_left.layout["title"] = "PC1 vs PC2"
  p_right.layout["title"] = "PC2 vs PC3"
  p_left.layout["xaxis_title"] = "pc1"
  p_left.layout["yaxis_title"] = "pc2"
  p_right.layout["xaxis_title"] = "pc2"
  p_right.layout["yaxis_title"] = "pc3"
  plot([p_left p_right])
end

group=vcat(repeat(["group1"],inner=33), repeat(["group2"],inner=33),
    repeat(["group3"],inner=33))
```

## PCA using Halko's method on CSV input

This example shows how to perform PCA using Halko's randomized SVD method on dense CSV input. The result is visualized in the PC1–PC3 space.

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

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

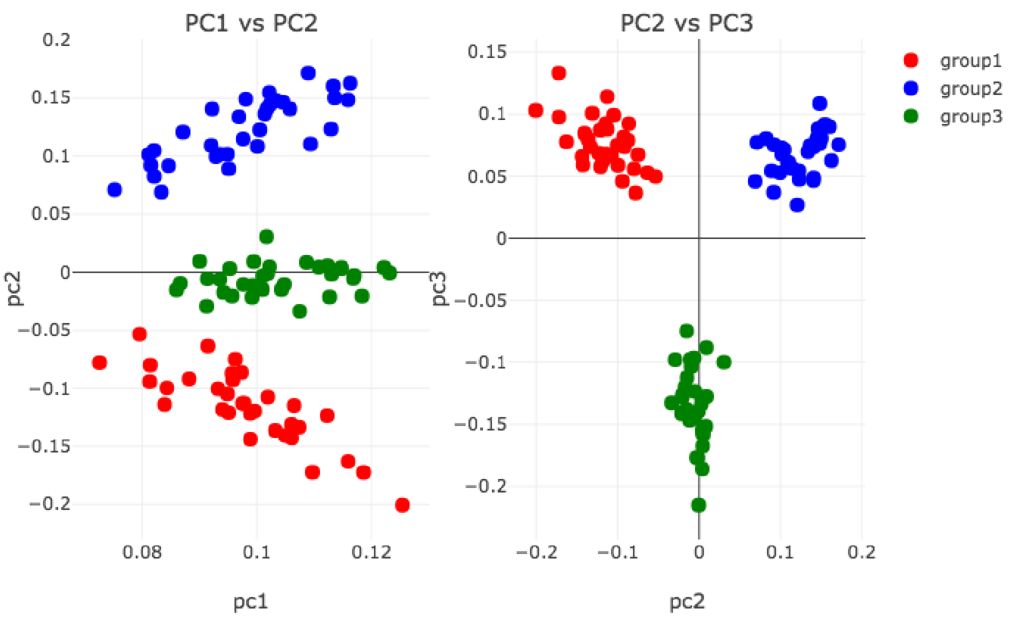


**Figure 2:** Output of halko against CSV format.

## 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)
```

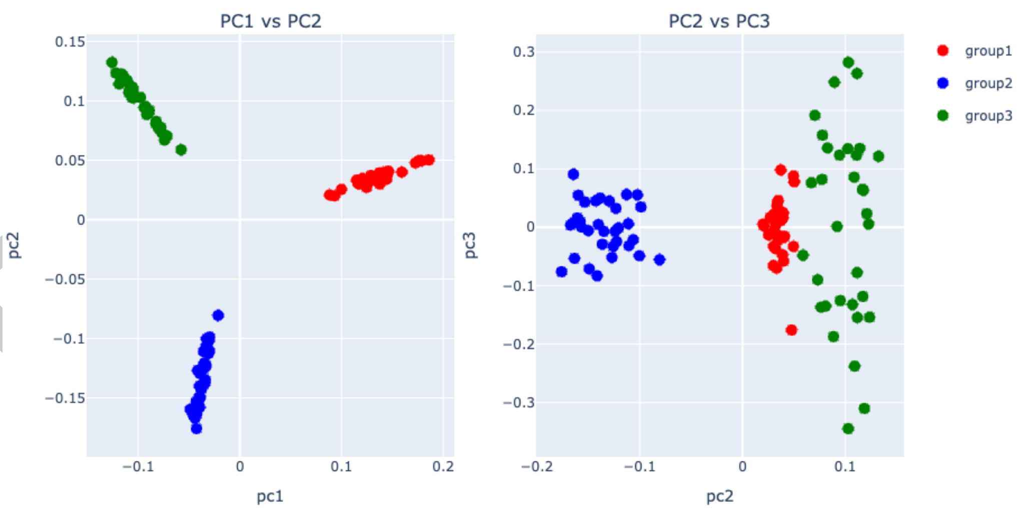


**Figure 3:** Output of sparse_rsvd against MM format.

## 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)
```

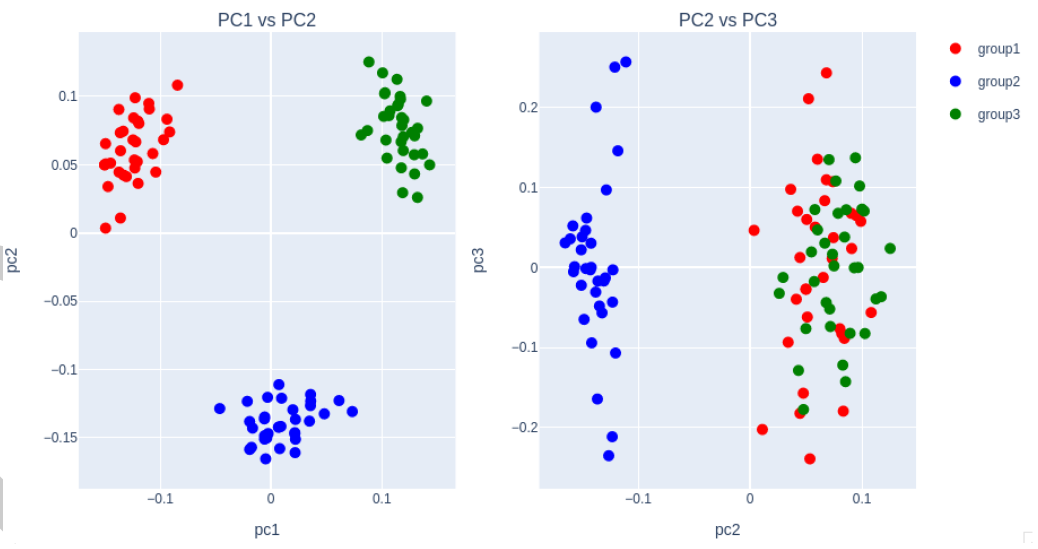


**Figure 4:** Output of exact_ooc_pca against BinCOO format.

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 | OOC | 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