# BATCH-EFFECT CORRECTION IN SINGLE-CELL RNA SEQUENCING DATA USING JIVE

JOSEPH HASTINGS, DONGHYUNG LEE, & MICHAEL J. O'CONNELL*

## ABSTRACT

Correcting for batch effects is an important step in preprocessing scRNA-seq data prior to analysis. Batch effects are technical artifacts in the data that arise from a multitude of factors: different sequencing technologies, equipment used, or even capture times. These effects are not of interest and obfuscate the true underyling biological signal. In this paper we introduce a novel application of the JIVE method which we use to perform batch-effect correction on multiple scRNA-seq datasets. We expect that the JIVE algorithm will be able to capture the common structure between the datasets which will be representative of the true biological variability of the cells and not be obscured by batch effects. We employ four evaluation metrics and compare the results against two other tools, Seurat v3 and Harmony, which were developed for this purpose. We found that JIVE performed best in metrics that consider local neighborhoods (kBET and LISI) and in scenarios in which the original data contained distinct differences between batches and cell types.

* *Department of Statistics, Miami University, Oxford, OH, USA, e-mail: oconnemj@miamioh.edu*

# CONTENTS

1 Introduction 4
1.1 Batch-Effect Correction Methods . . . 5
2 Methods 7
2.1 scRNA-seq Datasets . . . 7
2.2 JIVE Algorithm Improvements . . . 8
2.3 Batch Correction Evaluation Metrics . . . 10
3 Results 12
3.1 JIVE Runtime Improvements . . . 12
3.2 Simulated Data . . . 14
3.3 Assessments Using Real Data . . . 16
4 Discussion 21
4.1 JIVE Batch-Effect Correction Performance . . . 22
4.2 Limitations and Future Work . . . 22

# LIST OF TABLES

Table 1 Batch and Cell Type Frequency for Simulated Data . . . 7
Table 2 Batch and Cell Type Frequency for Bacher T-Cell Data . . . 8
Table 3 Batch and Cell Type Frequency for Zilionis Mouse Lung Data . . . 8
Table 4 Proportion of Variance Attributed to JIVE Decomposition . . . 14

# LIST OF FIGURES

Figure 1 Simulation Data for JIVE Benchmarks . . . 9
Figure 2 Benchmark for Partial SVD and Matrix Multiplication Runtimes. . . . 13
Figure 3 Benchmark for JIVE Improvements . . . 13
Figure 4 PVCA Breakdown for Simulated Data . . . 14
Figure 5 t-SNE Plots for Simulated Data . . . 15
Figure 6 UMAP Plots for Simulated Data . . . 15
Figure 7 Metrics for Simulated Data . . . 16
Figure 8 PVCA Breakdown for the Bacher T-Cell Data . . . 17
Figure 9 t-SNE Plots for the Bacher T-Cell Data . . . 17
Figure 10 UMAP Plots for the Bacher T-Cell Data . . . 18
Figure 11 Metrics for the Bacher T-Cell Data . . . 18
Figure 12 PVCA Breakdown for the Zilionis Mouse Lung Data . . . 19
Figure 13 t-SNE Plots for the Zilionis Mouse Lung Data . . . 20
Figure 14 UMAP Plots for the Zilionis Mouse Lung Data . . . 20

Figure 15 Metrics for the Zilionis Mouse Lung Data . . . . . 21

# 1 INTRODUCTION

There have been significant advancements in recent years in single-cell RNA sequencing (scRNA-seq) [Luecken and Theis, 2019]. Single-cell sequencing provides a higher resolution view of genomic data when compared to bulk RNA sequencing (bulk RNA-seq) and allows for the heterogeneity of different cell populations to be preserved. Bulk RNA-seq measures the average level gene expression in a population of cells [Wang et al., 2009], while scRNA-seq measures the gene expression for each cell individually [Wang and Navin, 2015]. Another prominent feature in scRNA-seq data is the high proportion of zero counts [Vallejos et al., 2017]. This is due to both biological and technical reasons. One biological cause of this zero-inflation could be due to a particular cell type having very little or no gene expression. A technical cause could be due to what is referred to as a technical dropout, where a certain gene is expressed but does not get detected by the sequencing technology. Multiple sequencing protocols have been developed to capture this information, with prominent examples including CEL-seq2, Drop-seq, MARS-seq, SCRB-seq, Smart-seq, and Smart-seq2 [Ziegenhain et al., 2017]. Sequencing data consists of a count matrix for a given set of genes obtained from a sample of cells. These counts represent the number of times that a specific gene is detected in a single cell [Grün and van Oudenaarden, 2015]. Each row represents a gene and each column is a cell. The library size for a cell is the sum of all counts across all genes.

Once the count data are obtained, it is typically normalized to help account for any variability caused by sampling effects within the given sequencing protocol [Grün and van Oudenaarden, 2015]. A few examples of normalization methods include counts per million (CPM) normalization, upper quartile (UQ) normalization, and trimmed mean of M values (TMM) normalization [Vallejos et al., 2017]. In CPM normalization, each count is divided by its cell's library size and multiplied by a million. UQ normalization uses a scaling factor proportional to the 75th percentile of the counts for a given cell. TMM normalization seeks to trim away cell counts that exhibit large log fold differences within a cell.

Another common preprocessing step is the integration of multiple data sets that are obtained from different batches. Unwanted technical variation and differences between count data are known as batch effects [Zhang et al., 2020]. These effects can arise due different sequencing technologies being used or cells being sequenced at different times. There have been over a dozen methods developed to integrate multiple sources of scRNA-seq data that aim to remove these unwanted batch effects [Tran et al., 2020]. Each method produces a "batch-corrected" data set which is then used for downstream analyses.

## 1.1 Batch-Effect Correction Methods

### *JIVE*

The joint and individual variation explained (JIVE) method [Lock et al., 2013] decomposes two or more biological datasets into three low-rank approximation components: a joint structure among the datasets, individual structures unique to each distinct dataset, and residual noise. JIVE was originally created for integrating different types of bulk sequencing data. For example, Lock et al. [2013] used gene expression and miRNA data from a set of 234 Glioblastoma Multiforme tumor cells in an attempt to identify any joint or individual variation between the data types.

JIVE can be expressed in terms of principal component analysis (PCA), a common dimension reduction technique. In PCA, given a dataset $X$ of size $p \times n$, we perform an eigendecomposition on the variance-covariance matrix $\Sigma_{p\times p}$ in the following way:

$$\Sigma_{p\times p} = U_{p\times p} D_{p\times p} U^{T}_{p\times p} \tag{1}$$

The matrix $U_{p\times p}$ in Equation 1 denotes the $p$ eigenvectors (known as loadings) and $D_{p\times p}$ is a diagonal matrix containing the $p$ eigenvalues of $\Sigma$. We can then calculate a rank $r$ approximation of $X$ in the following way:

$$X_{p\times r} \approx U_{p\times r} S_{r\times n} \tag{2}$$

In Equation 2, $U_{p\times r}$ is the first $r$ columns from the loading matrix and $S_{r\times n}$ is the first $r$ rows from the scores matrix. The $i$-th row of $S_{r\times n}$ is called the $i$-th principal component of $X$, and each principal component is constructed to be orthogonal to all others. Since each principal component is orthogonal, the scores do not suffer from any issues due to multicollinearity. The JIVE decomposition can then be written in the following manner:

$$\begin{aligned} X_1 &\approx U_1 S + W_1 S_1 + R_1 \\ X_2 &\approx U_2 S + W_2 S_2 + R_2 \\ &\vdots \\ X_k &\approx U_k S + W_k S_k + R_k \end{aligned} \tag{3}$$

$X_1, X_2, \ldots, X_k$ with $k \geqslant 2$ denote the original datasets of dimension $p_i \times n$ where $n$ is a common set of objects and $p_i$ is a given set of measurements that can vary for the $k$ matrices. We will denote $p_1 + p_2 + \ldots + p_k = p$. Let $r$ denote the chosen rank for the joint structure and $r_i$ denote the chosen ranks for each $X_i$. The matrix $U$ of size $p \times r$ is equal to $\begin{pmatrix} U_1 & U_2 & \ldots & U_k \end{pmatrix}^T$, where $U_i$ are the loadings of the joint structure for the individual datasets $X_1, \ldots, X_k$, and $S$ is an $r \times n$ score matrix. The $W_i$ are $p_i \times r_i$ loading matrices and the $S_i$ are $r_i \times n$ score matrices for each $X_i$. The $R_i$ represent any remaining residual

noise. In this paper, we will consider genes as the $n$ common set of objects because each cell can be sequenced for the same gene. We will consider the $p_i$ measurements to be the different cells present within each batch. The JIVE decomposition enforces orthogonality between the joint and individual structures, which ensures that the two structures capture distinct directions of variation within the data.

The JIVE decomposition estimates the joint and individual structures by minimizing the sum of squared error of the residual matrix. Given an initial estimate for the joint structure, it finds the individual structures to minimize the sum of squared error. Then, given the new individual structures, it finds a new estimate for the joint structure which minimizes the sum of squared error. This process is repeated until a given threshold for convergence is reached. The ranks are estimated by one of two different methods: a permutation test rank selection and a BIC rank selection. Our default rank estimation method is via the permutation test.

Our main interest is how well this method performs when we apply it in the context of scRNA-seq data batch-effect correction. We expect the joint structure matrix to capture the shared biological structure between scRNA-seq data from different batches and to serve as the batch-corrected dataset.

### *Seurat v3*

Seurat v3 [Stuart et al., 2019] is software package developed by the Satija lab which provides a comprehensive set of tools for single-cell data analysis and integration. The Seurat v3 integration method employs a strategy which anchors diverse datasets together. First, log-normalization is performed on all datasets and expression values are standardized for each gene. A subset of features are selected which exhibit high variance across all datasets. Then an initial dimension reduction method utilizing canonical correlation analysis (CCA) is performed to ensure similarities across datasets are preserved. Canonical correlation vectors (CCV) are then approximated and used to identify K-nearest neighbors (KNN) for each cell within their paired dataset. Mutual nearest neighbors (MNN) are then identified to act as anchors between datasets. These anchors are then filtered, scores, weighted, and used to perform the batch correction.

### *Harmony*

The Harmony integration method [Korsunsky et al., 2019] utilizes PCA for dimensionality reduction and then iterates between two algorithms until convergence is reached. The first algorithm clusters cells from multiple batches but ensures that the diversity of batches within each cluster are maximized (i.e., maximum diversity clustering). The second algorithm then uses a mixture model based approach to perform linear batch correction from a given vector of the known batches. The clustering step assigns soft clusters to cells and the correction step uses these clusters to compute new cell embeddings from the previous iteration.

# 2 METHODS

## 2.1 scRNA-seq Datasets

One simulated dataset and two real scRNA-seq datasets were used to evaluate the performance of the batch correction methods. The two real data sets were acquired using the scRNAseq R package [Risso and Cole, 2022]. Principal variance component analysis (PVCA) [Li et al., 2009] was performed for each raw dataset to get an estimation of how much variability is attributed to batch effects, cell type effects, and random error. PVCA uses principal components from PCA and variance components analysis (VCA) to fit a mixed linear model using the factors of interest as random effects to partition the total observed variability into varability due to batches, cell types, or residual error. This helps to quantify the impact of each source of variation on subsequent analyses.

### *Simulated Data*

The simulated data was created using the Splatter R package [Zappia et al., 2017] which allows the user to implement a Splat model. The core of the Splat model is a gamma-Poisson distribution which is used to generate a matrix of cell counts for a given number of genes. More than 20 different parameters are available for modifying, including parameters that affect library size, gene means, expression outliers, the presence of batch effects, the size of batch effects, and more. In our dataset, we simulate data for 5000 genes with two batches containing 500 cells each consisting of three different cell types in similar proportions between batches. The frequency of cells in each batch and cell group can be seen in Table 1.

**Table 1:** Batch and Cell Type Frequency for Simulated Data

| Batch Name | Group 1 | Group 2 | Group 3 | Total |
|---|---|---|---|---|
| Batch1 | 312 | 136 | 52 | 500 |
| Batch2 | 294 | 148 | 58 | 500 |

### *Bacher T-Cell Data*

The Bacher CD4+ T-cell RNA sequencing data [Bacher et al., 2020] was obtained from six unexposed and fourteen COVID-19 patients. There are a total of fifteen different batches and six different cell clusters provided. This data was chosen as there are not very large distinctions between batches and clusters, so we wished to see how the methods would perform in this type of scenario. The frequency of cells in each batch and cell group can be seen in Table 2.

**Table 2:** Batch and Cell Type Frequency for Bacher T-Cell Data

| Batch Name | Central Memory | Cycling | Cytotoxic / Th1 | Tfh-like | Transitional Memory | Type-1 IFN signature | Total |
|---|---|---|---|---|---|---|---|
| 14 | 230 | 1 | 76 | 250 | 190 | 7 | 754 |
| 15 | 239 | 4 | 143 | 425 | 289 | 13 | 1113 |

### *Zilionis Mouse Lung Data*

The Zilionis mouse lung data [Zilionis et al., 2019] analyzed tumor-infiltrating myeloid cells in mouse lung cancers. There are a total of three different batches and seven different cell clusters provided. This data was chosen as there is some distinct separation due to a batch effect and the cell clusters are well seperated. The frequency of cells in each batch and cell group can be seen in Table 3.

**Table 3:** Batch and Cell Type Frequency for Zilionis Mouse Lung Data

| Batch Name | B cells | T cells | Total |
|---|---|---|---|
| round1_20151128 | 641 | 579 | 1220 |
| round2_20151217 | 879 | 434 | 1313 |

In this study, we evaluate the performance of the three batch-effect correction methods above across three scRNA-seq datasets with four different evaluation metrics. We also present multiple improvements to the JIVE computation algorithms which help significantly decrease runtime compared to the current implementation in the R.JIVE R package.

## 2.2 JIVE Algorithm Improvements

The JIVE algorithm was implemented into the R.JIVE R package [O'Connell and Lock, 2016] from the original MATLAB code. The base functions provided in this package can take a substantial amount of runtime to get results (taking upwards of 12+ hours depending on data). We improved the speed of these base functions in two main ways: utilizing partial singular value decomposition in the RSpectra R package [Qiu and Mei, 2022] and converting frequently used matrix operations into precompiled C++ code using the Rcpp R package [Eddelbuettel and François, 2011] and the RcppEigen R package [Bates and Eddelbuettel, 2013] which provides access to the Eigen C++ linear algebra library.

The original R.JIVE code utilizes singular value decompositions (SVD) in many different areas, however only the largest singular values/vectors are used. A full decomposition takes a lot of time and resources to compute and the majority of the output is not used. We switched to using a partial SVD function in the RSpectra R package which

returns the largest singular values/vectors of a given matrix. We compared the partial SVD function to the base SVD function that is used in the R.JIVE package. A benchmark was performed on a dataset of size $1000 \times 1000$ generated from a standard normal distribution in which each function call was repeated 100 times. The top 1, 5, and 10 singular values/vectors were computed from each function call.

The other area in which we made significant improvements was in basic matrix operations. We tested two different functions which implemented C++ code to perform matrix multiplication and compared their performance to the default %*% operator in R. One function uses the Armadillo C++ library via the RcppArmadillo R package [Eddelbuettel and Sanderson, 2014] and the other uses the Eigen C++ library via the RcppEigen R package. The function using RcppEigen allows us to specify the number of CPU cores to utilize when performing computations. We compared the default matrix multiplication operator %*% in R to the implementations in RcppArmadillo and RcppEigen. A benchmark was performed by multiplying two matrices A and B of size $1000 \times 1000$ generated from a standard normal distribution in which each function call was performed 100 times.

We compared the runtimes of the original R.JIVE functions to the updated versions which implement the changes in 2.2 to assess overall improvements. Two matrices A and B of size $200 \times 1000$ were generated from a common joint structure matrix, two unique individual structure matrices, and two residual error matrices generated from a standard normal distribution. A visualization of these datasets can be seen in Figure 1.

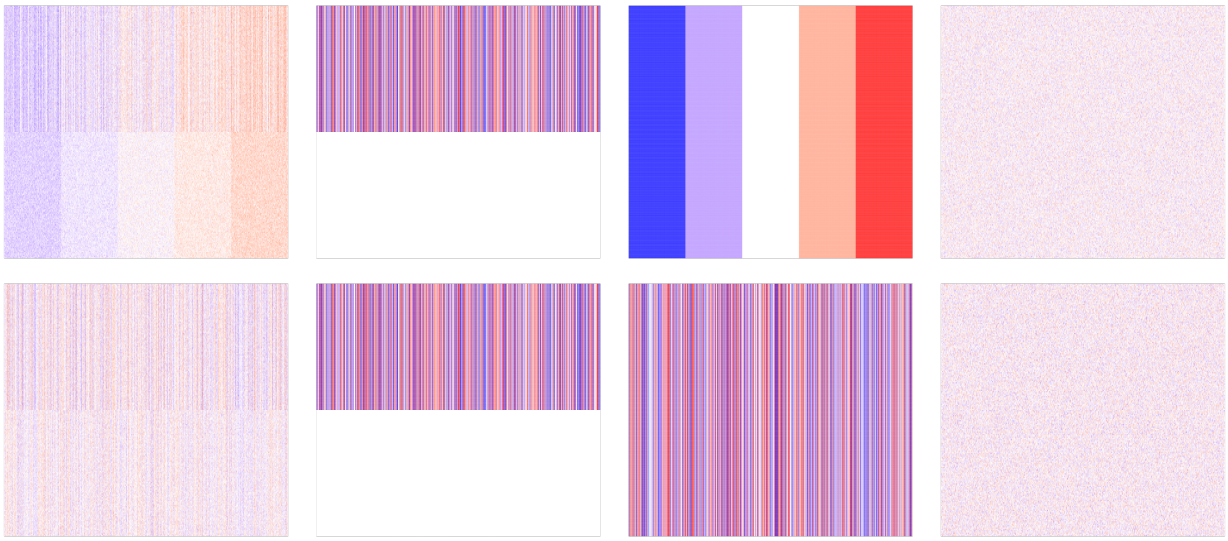

**Figure 1:** Simulation Data for JIVE Benchmarks.

The first and second row contain the final matrix in the first column, the joint matrix in the second column, the individual matrix in the third column, and the error matrix in the fourth column for A and B, respectively. Matrix values range from between approximately -2 (represented by the color blue) and 2 (represented by the color red). A benchmark was performed in which both the old R.JIVE implementation and the up-

dated version were repeated 20 times to record runtimes. We used given ranks of 1 for the joint structure and each individual structure.

All benchmarks were performed on a laptop with a 3.20 GHz AMD Ryzen 7 processor with 16.0 GB RAM.

## 2.3 Batch Correction Evaluation Metrics

We employed four tools/metrics to evaluate the performance of each of the batch correction methods: visual inspection of t-distributed stochastic neighbor embedding (t-SNE) [Van der Maaten and Hinton, 2008] and uniform manifold approximation and projection (UMAP) [McInnes et al., 2018] dimension reduction plots, k-nearest neighbor batch effect tests (kBET) [Büttner et al., 2019], average silhouette width (ASW) [Rousseeuw, 1987], and local inverse Simpson's index (LISI) [Korsunsky et al., 2019]. For the visual inspections, we expect to see cells from different batches overlapping each other in the plots with distinct cell type clusters. This is indicative of well-mixed (i.e., integrated) batches that preserve cell type heterogeneity.

While dimension reduction plots are a popular method for evaluating the performance of scRNA-seq batch-effect correction, any conclusions made from them are subjective. We include three numeric evaluation metrics to provide an objective sense of the performance for the three methods.

### *t-Distributed Stochastic Neighbor Embedding*

t-SNE is a non-linear dimension reduction technique [Van der Maaten and Hinton, 2008] that aids in visualizing high-dimensional data by assigning each data point a location in a two or three-dimensional map. It aims to preserve as much of the local structure of the original data as possible while also revealing global structure such as clusters. High dimensional Euclidean distances between points are used to create conditional probabilities of one point picking the other as its neighbor. A similar conditional probability is calculated for a low dimensional representation of the data. The goal of t-SNE is to find a low dimensional (i.e., two or three dimensions) representation that matches the two probabilities as best as possible by minimizing a certain objective function. We performed t-SNE using the scater R package [McCarthy et al., 2017] version 1.26.1 and the Seurat R package [Stuart et al., 2019] version 4.3.0.

### *Uniform Manifold Approximation and Projection*

UMAP is a non-linear dimension reduction technique [McInnes et al., 2018] that is based in manifold theory and topological data analysis. It can be separated into two main phases: graph construction and graph layout. In the graph construction phase, a weighted k-nearest neighbor graph is created, transformations are applied to the graph's edges, and asymmetry is dealt with. In short, it ensures that the underlying geometric structure of the data is captured. In the graph layout phase, an objective function is

defined that preserves important characteristics present in the k-nearest neighbor graph, and the final UMAP representation is the one which minimizes this function. We performed UMAP using the scater R package [McCarthy et al., 2017] version 1.26.1 and the Seurat R package [Stuart et al., 2019] version 4.3.0.

### *k-Nearest Neighbor Batch Effect Test*

The kBET metric [Büttner et al., 2019] was constructed with the following premise in mind: a subset of a well-mixed dataset with batch-effects removed should have the same distribution of batch labels as the full dataset. A $\chi^2$ -based test is performed for random subsets of a fixed size neighborhood and results from each test (i.e., reject or fail to reject) is averaged over to provide an overall rejection rate. If the rejection rates are low, then we failed to reject most of the tests, and thus the distribution of batch labels in the small neighborhoods were not significantly different from the entire data's distribution of batch labels. We performed kBET using the kBET R package [Büttner et al., 2017] version 0.99.6.

We calculate the rejection rates with neighborhood sizes equal to 5%, 10%, 15%, 20%, and 25% of the number of cells in each dataset. We then use the first 30 principal components from the batch-effect corrected datasets to perform the kBET at each neighborhood size. We then calculate the acceptance rate (1 - rejection rate) so that larger values are more desirable. The acceptance rates are then used for comparison across all methods.

### *Average Silhouette Width*

A silhouette is a measure of consistency within clusters of a given dataset [Rousseeuw, 1987]. For each data point in a given cluster, we calculate the mean distance between itself and all other points within the same cluster. We also calculate the smallest mean distance between itself and any other data point not in the same cluster. Then a silhouette is the difference of these two values scaled by the largest of the two. A silhouette takes on values between -1 and 1, with values close to 1 indicating that a particular point is appropriately clustered and values close to -1 indicating the opposite. The ASW is the average of all silhouette values which gives an measure of how well-clustered the data are as a whole. We performed ASW calculations using the cluster R package [Maechler et al., 2022] version 2.1.4.

For our purposes, we use the Euclidean distance metric for all calculations. We then subsample our data down to 80% of the original and use the first 30 principal components from the subsampled batch-effect corrected datasets. We calculate two ASW metrics: ASW batch (the batch labels are the clusters) and ASW cell type (cell type labels are the clusters), and this process is repeated 20 times for each method. ASW batch and ASW cell type results from all methods are separately scaled to be between 0 and 1. We report 1 - ASW batch values so that large values are more desirable. The median values of each of these scores are then used for comparison across all methods.

#### *Local Inverse Simpson's Index*

The local inverse Simpon's index [Korsunsky et al., 2019] first builds local Gaussian kernel-based distributions of neighborhoods around each cell. These neighborhoods are then used in conjuction with the inverse Simpon's index to calculate a diversity score which corresponds to the effective number of clusters in a particular cell's neighborhood. We performed LISI calculations using the lisi R package [Korsunsky, 2019] version 1.0.

We calculate two LISI metrics: LISI for batch label clusters (iLISI batch) and LISI for cell type clusters (cLISI cell type). Both LISI scores are calculated for each cell in the batch-effect corrected datasets for each method. iLISI and cLISI results from all methods are separately scaled to be between 0 and 1. We report 1 - cLISI cell type so that large values are more desirable. The median values of each of these scores are then used for comparison across all methods.

# 3 RESULTS

## 3.1 JIVE Runtime Improvements

The runtimes comparing the new implementations of partial SVD and matrix multiplication can be seen in Figure 2. We see that the original functions perform the decompositions in approximately 1.67 seconds on average, while partial SVD functions are performed in 0.08, 0.12, and 0.15 seconds on average. The implementation of a partial SVD function had significant improvements on the overall runtime of the JIVE algorithm. We observed a 95.4% shorter runtime when estimating one singular value/vector, a 93.1% shorter runtime when estimating five singular values/vectors, and a 91.0% shorter runtime when estimating ten singular values/vectors. This is of particular importance because a partial SVD is computed for the joint structure matrix and each individual structure matrix in every iteration of the JIVE estimation algorithm.

We can also see the original multiplication operator and the function using RcppArmadillo both averaged just under 0.3 seconds. The next four functions use the RcppEigen utilizing 1, 2, 4, and 8 CPU cores with runtimes of 0.093, 0.047, 0.027, and 0.02 on average, respectively. Performing matrix multiplication using precompiled C++ code also provided a sizeable decrease in runtime. The difference between the %*% operator and the function implemented in RcppArmadillo were neglible. The function implemented in RcppEigen not only provided singificant improvements over the base R operator, but it also allows for the user to specify the amount of CPU cores to utilize during runtime. We observed 67.5% shorter runtime when using the function with one core, a 83.2% shorter runtime when using two cores, a 90.6% shorter runtime when using four cores, and a 92.9% shorter runtime when using eight cores. Note that using a larger number of available CPU cores does not always provide an increase in speed. Computation on

smaller matrices tend to be faster without using multiple cores, but computations on large matrices typically run faster on multiple cores.

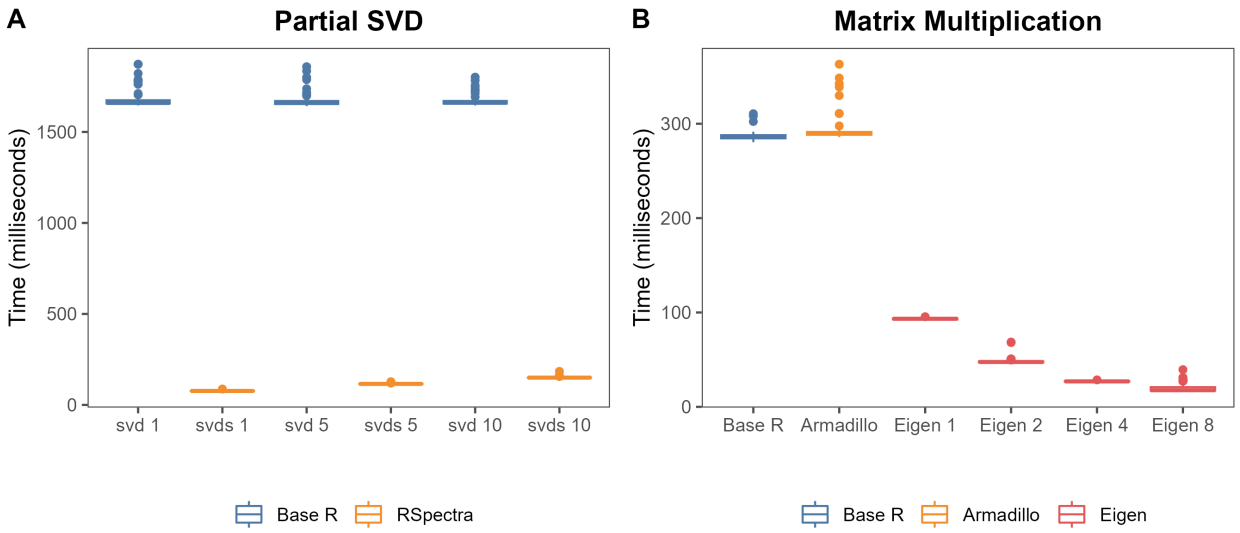


**Figure 2:** Benchmark for Partial SVD and Matrix Multiplication Runtimes.

Overall runtime improvements can be seen in Figure 3.

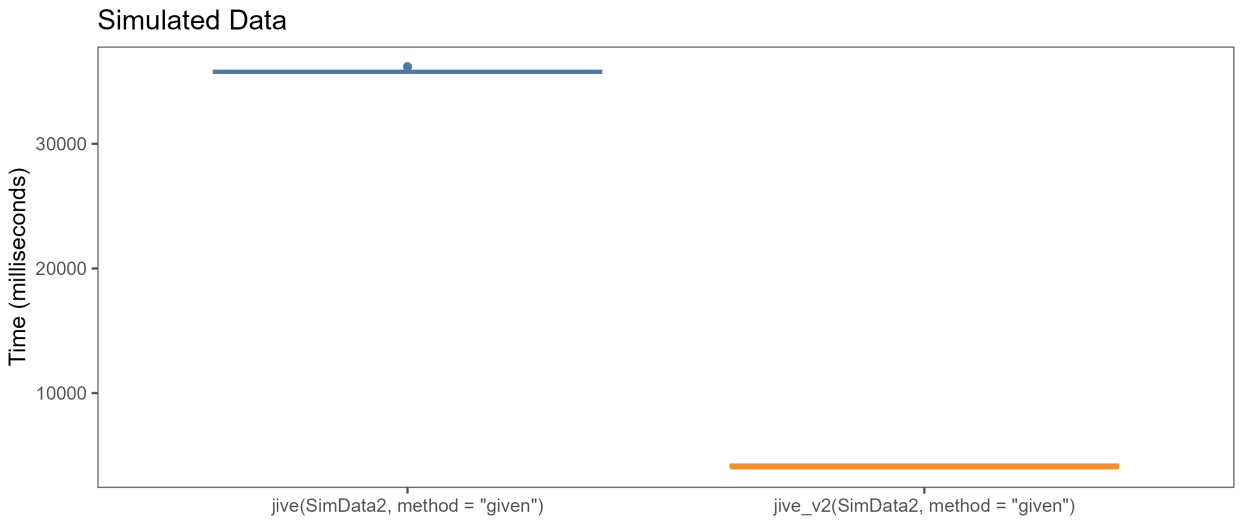


**Figure 3:** Benchmark for JIVE Improvements.

We see that the original R.JIVE function performs the decomposition in about 35.8 seconds on average, while the improved function completes it in 4.1 seconds on average.

The two procedures produced close to identical results. Table 4 shows the proportion of variance attributable to joint structure, individual structure, and residual variance for the two methods.

**Table 4:** Proportion of Variance Attributed to JIVE Decomposition

| Original | Data1 | Data 2 | Updated | Data1 | Data 2 |
|---|---|---|---|---|---|
| Joint | 0.346 | 0.161 | Joint | 0.346 | 0.161 |
| Individual | 0.400 | 0.582 | Individual | 0.400 | 0.582 |
| Residual | 0.254 | 0.256 | Residual | 0.254 | 0.256 |

## 3.2 Simulated Data

The only preprocessing step performed for the simulated data is a log-normalization [McCarthy et al., 2017]. Each cell count is divided by factor proportional to its library size, a pseudo-count of 1 is added (for zero counts), and a log2-transformation is applied. The PVCA plot for the simulated dataset can be seen in Figure 4.

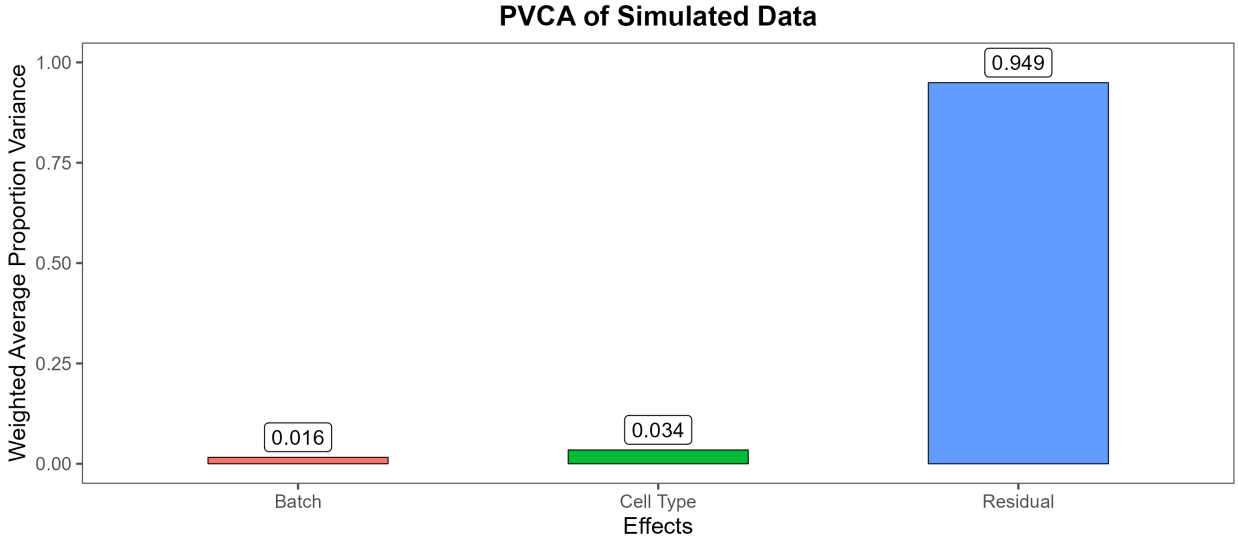


**Figure 4:** PVCA breakdown for simulated data.

We observe that almost 95% of the variability within the data is not due to the batch or cell clusters which is consistent with most scRNA-seq data. The t-SNE plots can be seen in Figure 5 and the UMAP plots can be seen in in Figure 6.

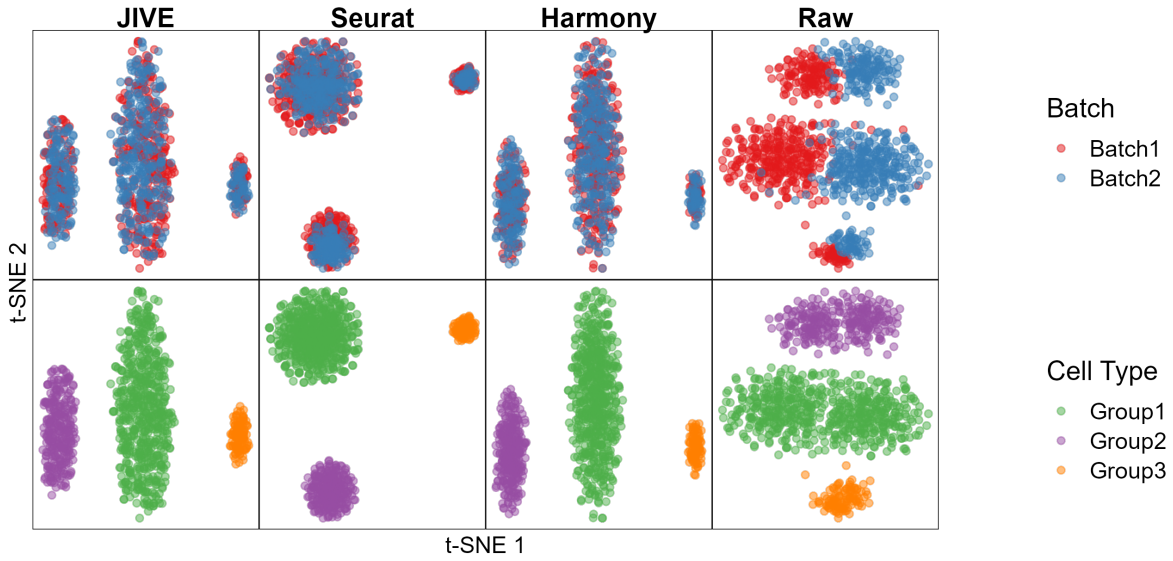


**Figure 5:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using t-SNE plots for the simulated data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

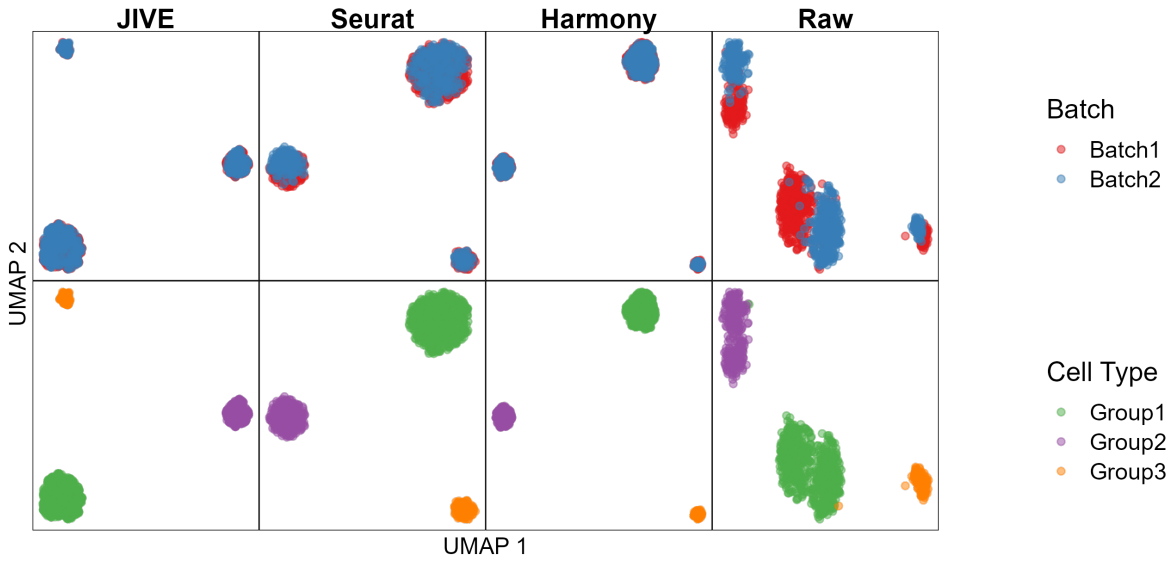


**Figure 6:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using UMAP plots for the simulated data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

The top half of each plot has cells colored by batch label and the bottom half has cells colored by their respective cell type/cluster label. Each method has the batches overlapping while the cell clusters are still preserved and distinct from one another.

In contrast, the raw plots show some clear seperation between batches. The numeric evaluation metrics can be seen in Figure 7.

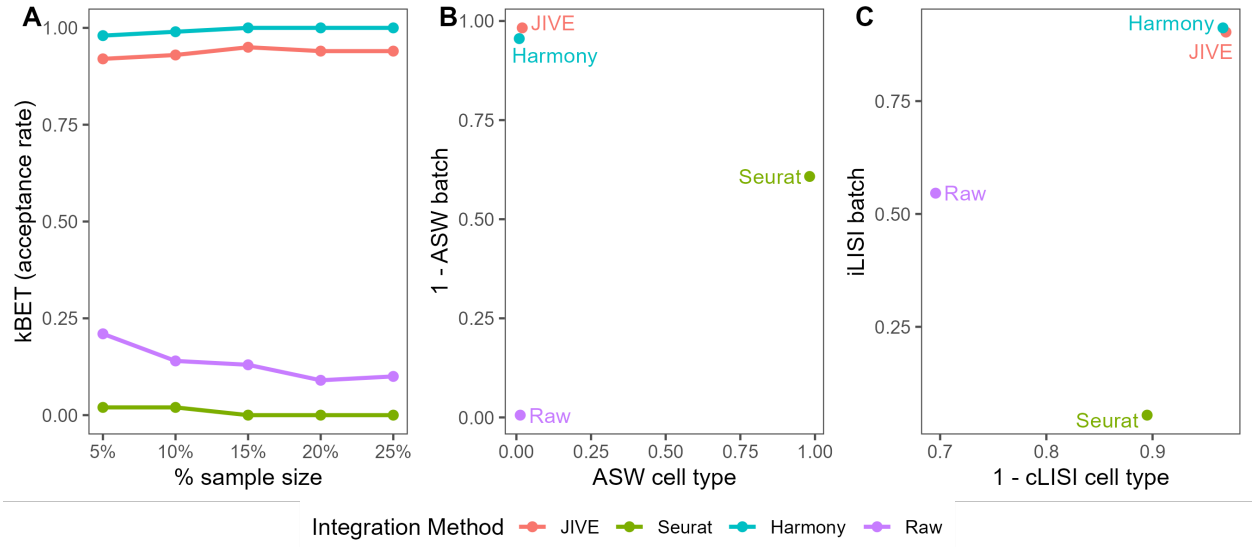


**Figure 7:** Quantitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using (A) kBET, (B) ASW, and (C) LISI for the simulated data. Methods with higher kBET acceptance rates performed best. Methods in the top right of the ASW and LISI plots peformed best.

The best performing method for the kBET metric is Harmony followed by closely JIVE. Seurat surprisingly performed worse than the data without any batch correction performed. JIVE and Harmony performed well for the ASW batch metric, closely followed by Seurat. However, Seurat outperformed all methods with regards to the ASW cell type metric. This indicates that Seurat was much better at preserving the different cell types within its cell embeddings than JIVE and Harmony, but not able to distinguish between batches. Harmony and JIVE were the top performers on both the iLISI batch and cLISI cell type metrics. Seurat did not do well with regards to iLISI batch, but was serviceable for the cLISI cell type metric. Overall, JIVE and Harmony were the best performing batch-effect correction methods for the simulated data.

## 3.3 Assessments Using Real Data

### *Bacher T-Cell Data*

Data preprocessing for the Bacher T-cell data consisted of log-normalization (as described for the simulated data), and the top 2000 genes with the highest variability across all batches were selected for analysis. This was performed using using a standard workflow provided in the Seurat R package. We also chose to select only two of the fifteen total batches. This reduced our total dataset size from $33538 \times 104417$ to $2000 \times 1867$. The PVCA plot for the Bacher T-cell dataset can be seen in Figure 8.

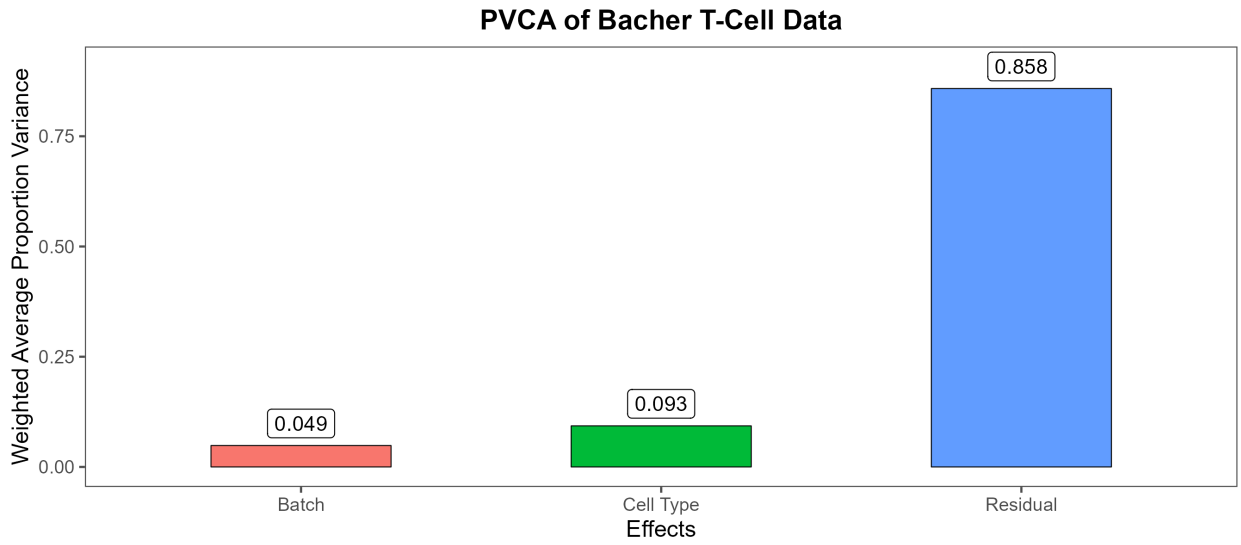


**Figure 8:** PVCA breakdown for the Bacher T-cell data.

We see more variability attributed to the batch and cell clusters than in our simulated data, with almost 5% and 10%, respectively. The t-SNE plots can be seen in Figure 9 and the UMAP plots can be seen in in Figure 10.

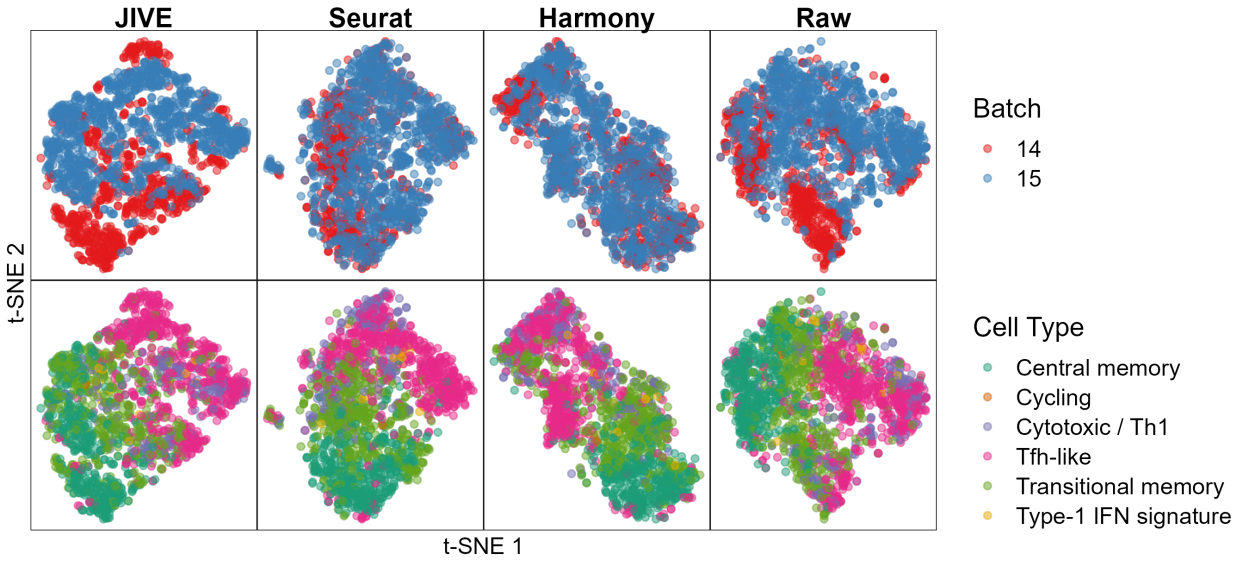


**Figure 9:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using t-SNE plots for the Bacher T-cell data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

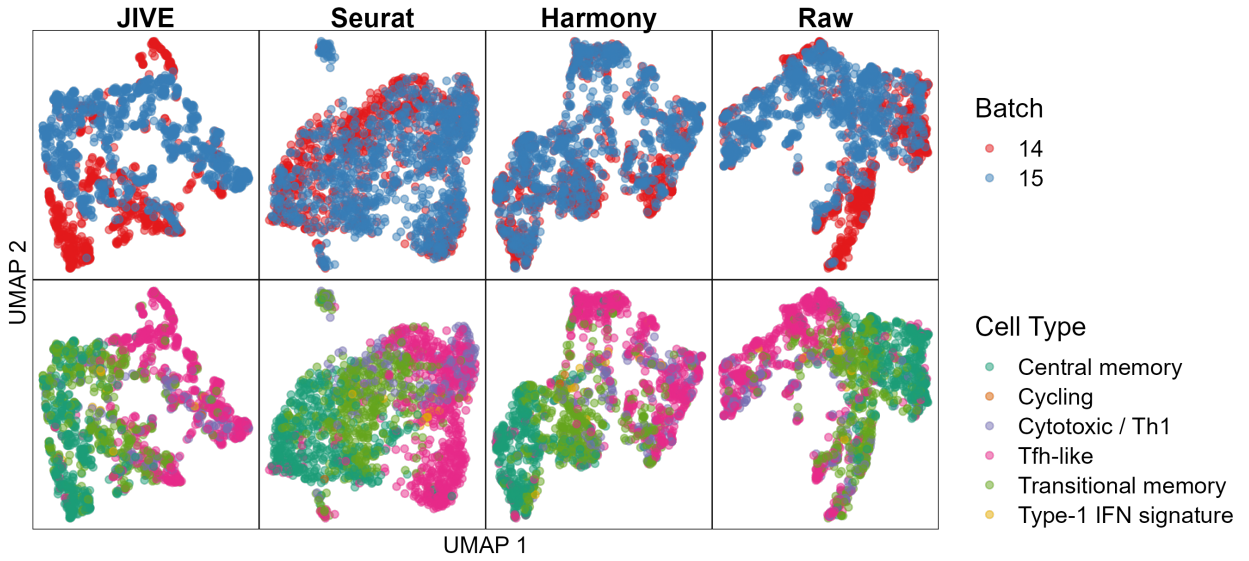


**Figure 10:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using UMAP plots for the Bacher T-cell data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

We see that Seurat and Harmony both have well-mixed batches in the dimensionality reduction plots, while JIVE does not. The cell clusters look to be well preserved in all three methods. The numeric evaluation metrics can be seen in Figure 11.

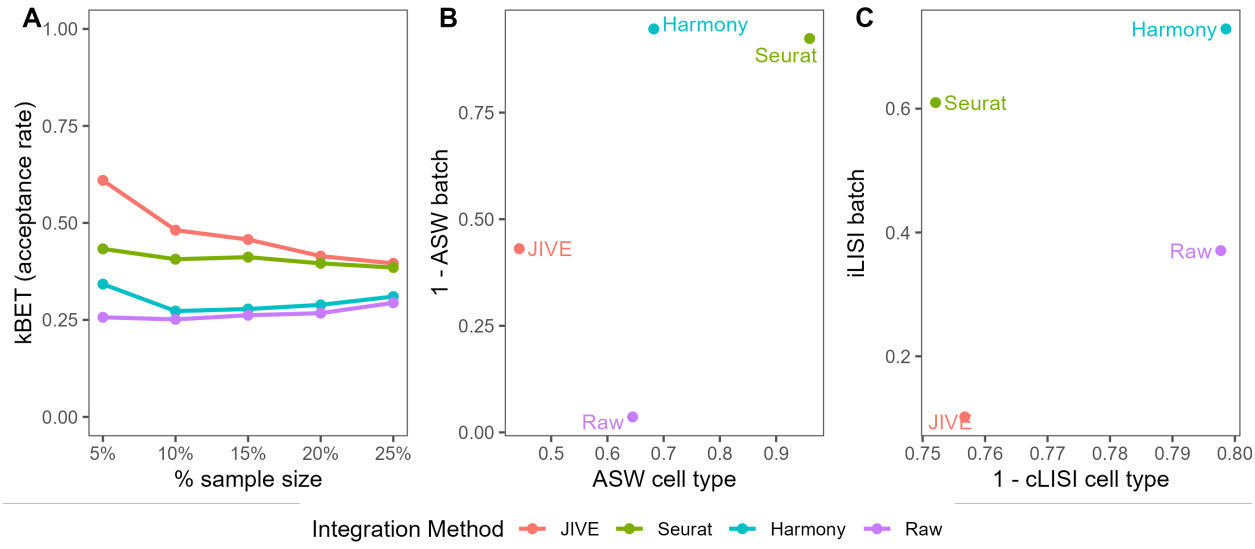


**Figure 11:** Quantitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using (A) kBET, (B) ASW, and (C) LISI for the Bacher T-cell data. Methods with higher kBET acceptance rates performed best. Methods in the top right of the ASW and LISI plots peformed best.

JIVE performs the best with regards to kBET, with JIVE close behind. The acceptance rates for Harmony are just slightly larger than the raw data. Harmony and Seurat both perform best in the ASW metrics, while JIVE only outperforms the raw data with regards to ASW batch and actually performs worse in ASW cell type. Harmony is the clear winner in the LISI metrics followed closely by Seurat. Notably, JIVE performs worse than the raw data in both metrics. It is worth noting that the cLISI cell type is on a much tighter scale than the iLISI batch metric in the plot, so the percieved differences are not as larger as they appear.

### *Zilionis Mouse Lung Data*

We perform the same preprocessing as described for the Bacher T-cell data, select only two batches, and this time selecting only two cell types/clusters (B-cells and T-cells). This helps to reduce the data dimensions from $28205 \times 17549$ to $2000 \times 2533$. The PVCA plot for the Zilionis Mouse Lung dataset can be seen in Figure 12.

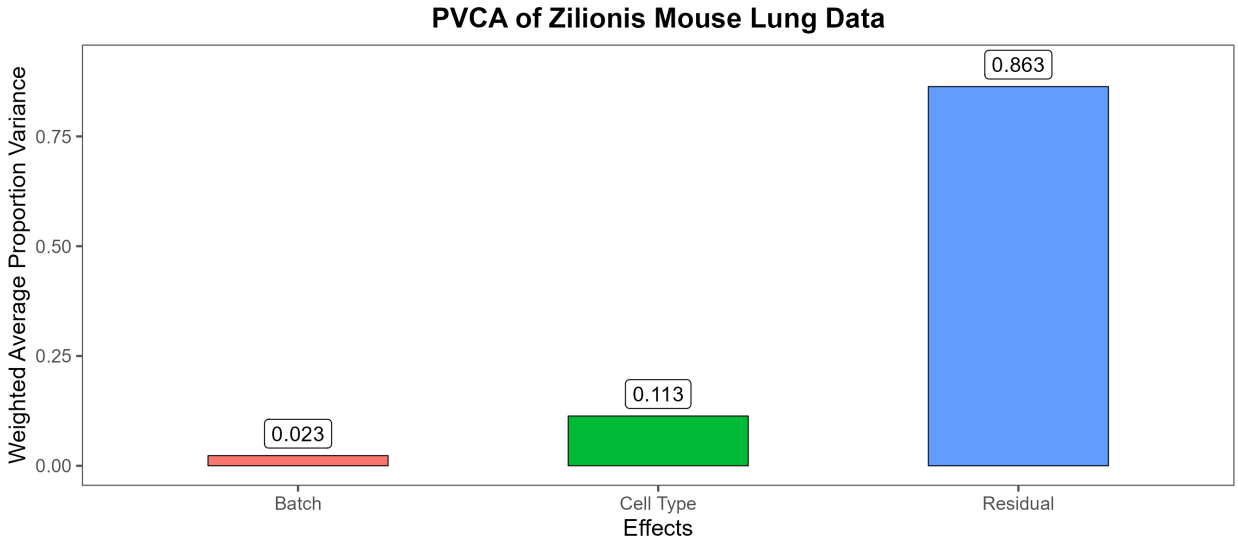


**Figure 12:** PVCA breakdown for the Zilionis mouse lung data.

We see a similar breakdown of total variability as the Bacher T-cell data except a bit more is attributed to the cell clusters. The t-SNE plots can be seen in Figure 13 and the UMAP plots can be seen in in Figure 14.

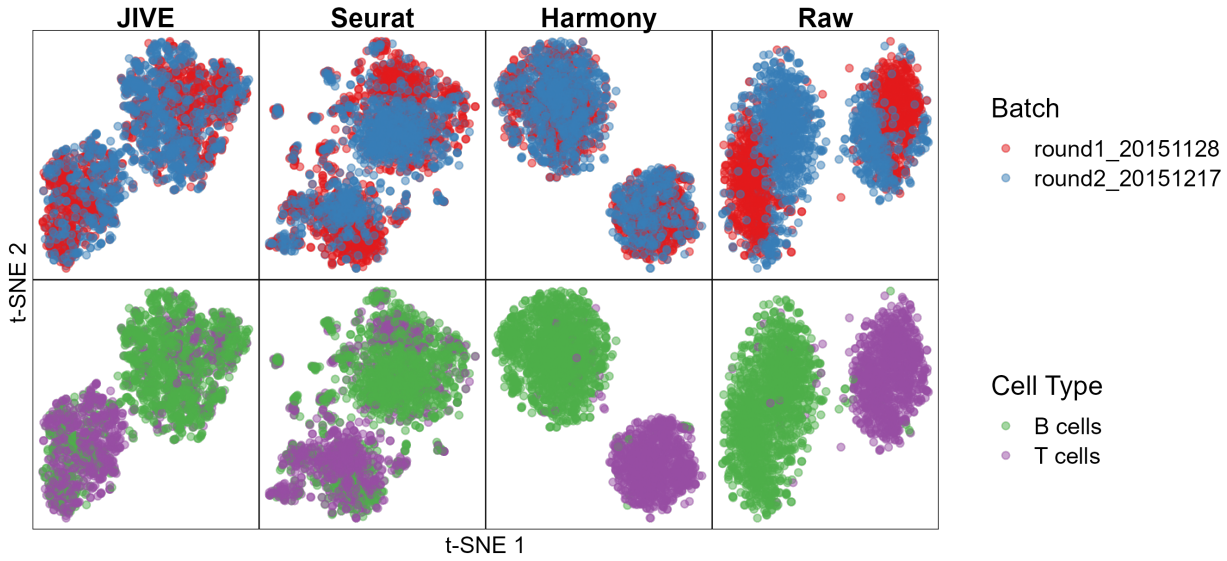


**Figure 13:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using t-SNE plots for the Zilionis mouse lung data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

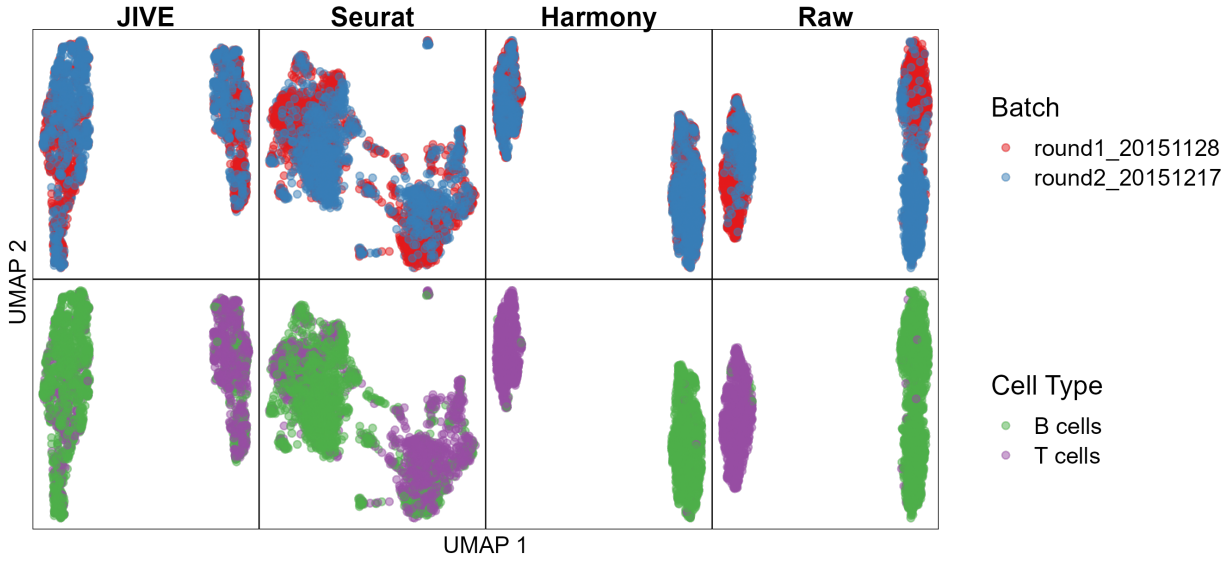


**Figure 14:** Qualitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using UMAP plots for the Zilionis mouse lung data. Each column represents a different method, with the fourth having no batch-effect correction applied. The first row has cells colored by batch and the second row has cells colored by cell type.

We see that each method has well-mixed batches and cell clusters are preserved. It is interesting to note that both JIVE and Seurat produced two distinct clusters that consist of a mix of both cell clusters, while Harmony was able to keep the cell clusters away from each other. The numeric evaluation metrics can be seen in Figure 15.

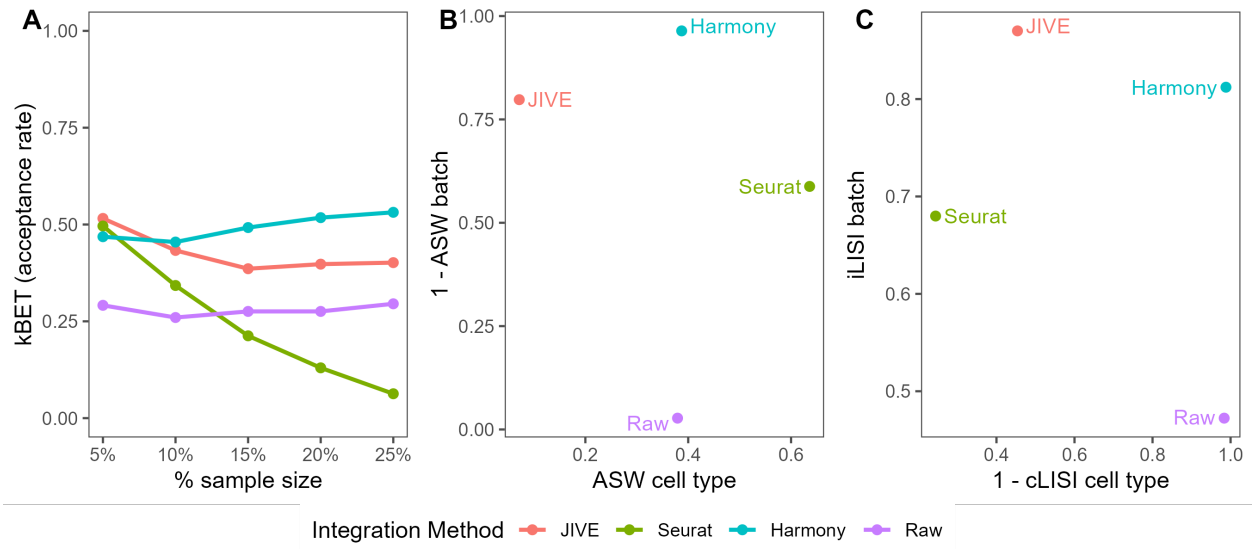


**Figure 15:** Quantitative evaluation of JIVE, Seurat, and Harmony batch-effect correction methods using (A) kBET, (B) ASW, and (C) LISI for the Zilionis mouse lung data. Methods with higher kBET acceptance rates performed best. Methods in the top right of the ASW and LISI plots peformed best.

We see that all three methods perform similarly at low neighorhood levels. However as the size increases, Seurat drops off dramatically. Harmony performs best at kBET with JIVE close behind. The ASW metric performances are a bit of a mixed bag: each method is better than another at either ASW batch or ASW cell type. Harmony performs best at ASW batch, followed by JIVE and then Seurat. Seurat performs best at ASW cell type, followed by Harmony and then JIVE. JIVE and Harmony are the best performers in the LISI metrics, with JIVE being the best with regards to iLISI batch and Harmony winning out in cLISI cell type. Seurat performs worse than both JIVE and Harmony, and only beats the raw data with regards to iLISI batch.

## 4 DISCUSSION

In this paper, we explored if the JIVE method was effective at performing batch-effect correction in scRNA-seq data. We expected JIVE to provide a more flexible alternative to other commonly used methods because it estimates joint structure and individual structure simultaneously and allows the user to specify the ranks chosen for the low-rank approximations. We predicted that the simulataneous estimation of both joint and individual structure would essentially perform both batch integration and batch correction at the same time.

Initially, the use of JIVE for this purpose was impractical due to its prohibitively long runtimes with data on the scale commonly observed in scRNA-seq datasets. The signif-

icant increase in speed due to the improvements we made to the algorithm make it a much more realistic tool to use for bach-effect correction.

## 4.1 JIVE Batch-Effect Correction Performance

In each scRNA-seq dataset, we tested the performance of each batch correction method on their ability mix batches while still preserving the purity of the cell types/clusters. A commonly used method for evaluating batch integration is by visual inspection of dimension reduction plots, with the most common being PCA, t-SNE, or UMAP plots [Tran et al., 2020]. This method works well for simple cases like in the simulated data where the batch effects and cell clusters are clearly defined. However, this subjective method tends to become more difficult if there is not clear seperation between batches or when cell types are very similar, as is the case in the Bacher T-cell data. This ambiguity that stems from visual inspection is the reason we employed the use of three numeric evaluation metrics to objectively assess the performance of each method. Note that while objective measurements are useful, we still believe that visual inspection can still provide useful insight during exploratory analysis. Note that none of the metrics simultaneously test both the quality of batch mixing and preservation of cell types, and the development of such a metric would be of great interest.

Overall, Harmony performed the best in the simulated data and Zilionis mouse lung data where batch effects were distinct and cell types effets were large. It consistenly performed well on kBET and LISI metrics that take into account the structure of each cell's local neighborhood. The t-SNE and UMAP plots were also consistent with its performance in the numeric metrics. JIVE performed second best with regards to metrics concerning batch mixing, but it struggled greatly with cell type purity metrics. The only dataset where it outperformed the raw data was the cLISI metric in the simulated data. Despite this, it was encouraging to see that JIVE was able keep up with Harmony in both the simulated data and the Zilonis mouse lung data. Seurat performed second best at metrics concerning cell type purity and had its best performance in the nebulous Bacher T-cell data. The increase in neighorhood size did not have much impact on its kBET performance and it did well in both ASW metrics and both LISI metrics. The most interesting result was Seurat's poor kBET performance in the simulated data, which seem to contradict the t-SNE and UMAP plots.

## 4.2 Limitations and Future Work

A limitation that leaves room for future work may involve exploring different data normalization techniques. The JIVE decomposition is best suited to data that is normally distributed, which scRNA-seq data clearly does not adhere to even after normalization. The normalization methods in this study were chosen for simplicity and consistency between integration methods. It is therefore likely that a more sophisticated method which takes into account the zero-inflated nature of scRNA-seq data will provide a better

input for the JIVE algorithm and lead to more accurate results. Another method to address this issue would be adapting JIVE to work with other distributions (e.g., negative binomial), akin to generalized linear models. This would also help to relieve the issue of scRNA-seq not being normally distributed.

Another limitation is the narrow scope of datasets used in this study. We only compared three datasets and considered a maximum of two batches at a time with at most six cell types. Subsets of batches in the real datasets were chosen for the sake of simplicity, however in practice it may be of interest to integrate datasets from much more than two experiments or batches. Tran et al. [2020] considered a more comprehensive set of ten different datasets that analyzed five distinct scenarios: identical cell types sequenced by different technologies, non-identical cell types across batches, data from more than two batches, big data sets ($> 100,000$ cells each), and multiple simulated datasets.

The comparison of runtimes between JIVE and other integration methods is also an area for future research. While algorithmic improvements greatly reduced the runtime of JIVE, we did not compare it to Seurat or Harmony. An analysis of the memory requirements for each method would also be of interest.

## REFERENCES

Malte D Luecken and Fabian J Theis. Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6):e8746, 2019.

Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57–63, 2009.

Yong Wang and Nicholas E Navin. Advances and applications of single-cell sequencing technologies. *Molecular cell*, 58(4):598–609, 2015.

Catalina A Vallejos, Davide Risso, Antonio Scialdone, Sandrine Dudoit, and John C Marioni. Normalizing single-cell rna sequencing data: challenges and opportunities. *Nature methods*, 14(6):565–571, 2017.

Christoph Ziegenhain, Beate Vieth, Swati Parekh, Björn Reinius, Amy Guillaumet-Adkins, Martha Smets, Heinrich Leonhardt, Holger Heyn, Ines Hellmann, and Wolfgang Enard. Comparative analysis of single-cell rna sequencing methods. *Molecular cell*, 65(4):631–643, 2017.

Dominic Grün and Alexander van Oudenaarden. Design and analysis of single-cell sequencing experiments. *Cell*, 163(4):799–810, 2015.

Yuqing Zhang, Giovanni Parmigiani, and W Evan Johnson. Combat-seq: batch effect adjustment for rna-seq count data. *NAR genomics and bioinformatics*, 2(3):lqaa078, 2020.

Hoa Thi Nhu Tran, Kok Siong Ang, Marion Chevrier, Xiaomeng Zhang, Nicole Yee Shin Lee, Michelle Goh, and Jinmiao Chen. A benchmark of batch-effect correction methods for single-cell rna sequencing data. *Genome biology*, 21:1–32, 2020.

Eric F Lock, Katherine A Hoadley, James Stephen Marron, and Andrew B Nobel. Joint and individual variation explained (jive) for integrated analysis of multiple data types. *The annals of applied statistics*, 7(1):523, 2013.

Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, Yuhan Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. Comprehensive integration of single-cell data. *Cell*, 177(7):1888–1902, 2019.

Ilya Korsunsky, Nghia Millard, Jean Fan, Kamil Slowikowski, Fan Zhang, Kevin Wei, Yuriy Baglaenko, Michael Brenner, Po-ru Loh, and Soumya Raychaudhuri. Fast, sensitive and accurate integration of single-cell data with harmony. *Nature methods*, 16(12): 1289–1296, 2019.

Davide Risso and Michael Cole. scrnaseq: Collection of public single-cell rna-seq datasets. 2022. R package version 2.12.0.

Jianying Li, Pierre R Bushel, Tzu-Ming Chu, and Russell D Wolfinger. Principal variance components analysis: estimating batch effects in microarray gene expression data. *Batch Effects and Noise in Microarray Experiments: Sources and Solutions*, pages 141–154, 2009.

Luke Zappia, Belinda Phipson, and Alicia Oshlack. Splatter: simulation of single-cell rna sequencing data. *Genome biology*, 18(1):174, 2017.

Petra Bacher, Elisa Rosati, Daniela Esser, Gabriela Rios Martini, Carina Saggau, Esther Schiminsky, Justina Dargvainiene, Ina Schröder, Imke Wieters, Yascha Khodamoradi, et al. Low-avidity cd4+ t cell responses to sars-cov-2 in unexposed individuals and humans with severe covid-19. *Immunity*, 53(6):1258–1271, 2020.

Rapolas Zilionis, Camilla Engblom, Christina Pfirschke, Virginia Savova, David Zemmour, Hatice D Saatcioglu, Indira Krishnan, Giorgia Maroni, Claire V Meyerovitz, Clara M Kerwin, et al. Single-cell transcriptomics of human and mouse lung cancers reveals conserved myeloid populations across individuals and species. *Immunity*, 50 (5):1317–1334, 2019.

Michael J O'Connell and Eric F Lock. R. jive for exploration of multi-source molecular data. *Bioinformatics*, 32(18):2877–2879, 2016.

Yixuan Qiu and Jiali Mei. Rspectra: Solvers for large-scale eigenvalue and svd problems. 2022. URL https://CRAN.R-project.org/package=RSpectra. R package version 0.16-1.

Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08.

Douglas Bates and Dirk Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. doi: 10.18637/jss.v052.i05.

Dirk Eddelbuettel and Conrad Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Maren Büttner, Zhichao Miao, F Alexander Wolf, Sarah A Teichmann, and Fabian J Theis. A test metric for assessing single-cell rna-seq batch correction. *Nature methods*, 16(1): 43–49, 2019.

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Davis J. McCarthy, Kieran R. Campbell, Aaron T. L. Lun, and Quin F. Willis. Scater: pre-processing, quality control, normalisation and visualisation of single-cell RNA-seq data in R. *Bioinformatics*, 33:1179–1186, 2017. doi: 10.1093/bioinformatics/btw777.

Maren Büttner, Alex Wolf, and Fabian Theis. *kBET: k-nearest neighbour batch effect test*. Helmholtz-Zentrum München, Ingolstädter Landstraße 1, Neuherberg, Germany, 2017. More information at <https://github.com/theislab/kBET>.

Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. cluster: Cluster analysis basics and extensions. 2022. URL https://CRAN.R-project.org/package=cluster. R package version 2.1.4 — For new features, see the 'Changelog' file (in the package source).

Ilya Korsunsky. lisi: Local inverse simpson index (lisi) for scrnaseq data. 2019. R package version 1.0.