BUSseq: Batch Effects Correction with Unknown Subtypes for scRNA-seq data User's Guide

Fangda Song, **Ga Ming Chan and Yingying Wei***

The Chinese University of Hong Kong

*sfd1994895@gmail.com

September 9, 2024

Contents

BUSseq: Batch Effects Correction with Unknown Subtypes for scRNA-seq data User's Guide

1 Introduction

Single-cell RNA-sequencing (scRNA-seq) technologies enable the measurement of the transcriptome of individual cells, which provides unprecedented opportunities to discover cell types and understand cellular heterogeneity [1]. Despite their widespread applications, single-cell RNA-sequencing (scRNA-seq) experiments are still plagued by batch effects and dropout events.

One of the major tasks of scRNA-seq experiments is to identify cell types for a population of cells [1]. Therefore, the cell type of each individual cell is always unknown and is the target of inference. However, most existing methods for batch effects correction, such as Combat space [2] and the surrogate variable analysis (SVA)([3], [4]), are designed for bulk experiments and require knowledge of the subtype information, which corresponds to cell type information for scRNA-seq data, of each sample a priori.

Here, the R package *BUSseq* fits an interpretable Bayesian hierarchical model—the Batch Effects Correction with Unknown Subtypes for scRNA seq Data(BUSseq)—to correct batch effects in the presence of unknown cell types [5]. BUSseq is able to simultaneously correct batch effects, clusters cell types, and takes care of the count data nature, the overdispersion, the dropout events, and the cell-specific sequencing depth of scRNA-seq data. After correcting the batch effects with BUSseq, the corrected value can be used for downstream analysis as if all cells were sequenced in a single batch. BUSseq can integrate the read count matrices measured from different platforms and allow cell types to be measured in some but not all of the batches as long as the experimental design fulfills the conditions listed in [5].

This guide provides step-by-step instructions for applying the BUSseq model to correct batch effects and identify the unknown cell type indicators for each cell for scRNA-seq data.

2 Methodolgy

BUSseq is a hierarchical model that closely mimics the data generating mechanism of scRNA-seq experiments [5]. The hierarchical structure of BUSseq can be illustrated by the following diagram.

BUSseq: Batch Effects Correction with Unknown Subtypes for scRNA-seq data
User's Guide

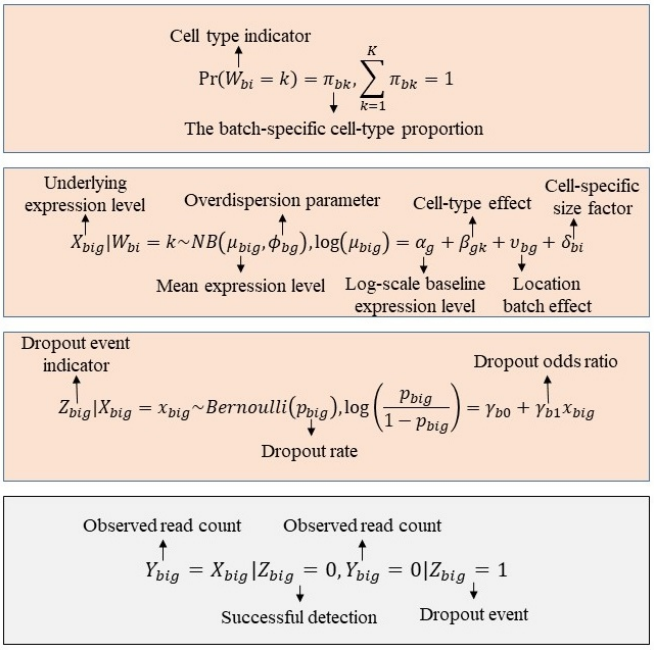

Figure 1: The hierarchical stucture of BUSseq model. Only Y_{big} in the gray rectangle is observed.

Assuming that there are totally B batches and n_b cells in b-th batch, $b = 1, 2, \cdots, B$, we define the underlying gene expression level of gene g in cell i of batch b as X_{big}. Given the cell type $W_{bi} = k$, X_{big} follows a negative binomial distribution with mean expression level μ_{big} and a gene-specific and batch-specific overdispersion parameter ϕ_{bg}. The mean expression level μ_{big} is determined by the log-scale baseline expression level α_g, the cell type effect β_{gk}, the location batch effect ν_{bg}, and the cell-specific size factor δ_{bi}. It is of note that the cell type W_{bi} of each individual cell is unknown and is our target of inference. Therefore, we assume that a cell on batch b comes from cell type k with probability $\Pr(W_{bi} = k) = \pi_{bk}$, and the proportions of cell types $(\pi_{b1}, \cdots, \pi_{bK})$ can vary across batches.

Unfortunately, it is not always possible to observe the expression level X_{big} due to dropout events. Without dropout ($Z_{big} = 0$), we can directly observe $Y_{big} = X_{big}$. However, if a dropout event occurs ($Z_{big} = 1$), then we observe $Y_{big} = 0$ instead of X_{big}. In other words, when we observe a zero read count $Y_{big} = 0$, there are two circumstances: a non-expressed gene—biological zeros— or a dropout event. When gene g is not expressed in cell i of batch b ($X_{big} = 0$), we always have $Y_{big} = 0$; when gene g is actually expressed in cell i of batch b ($X_{big} > 0$) but a dropout event occurs, we can only observe $Y_{big} = 0$, and hence $Z_{big} = 1$. It has been noted that highly expressed genes are less-likely to suffer from dropout events [6].

We thus model the dependence of the dropout rate $Pr(Z_{big} = 1|X_{big})$ on the expression level using a logistic regression with batch-specific intercept γ_{b0} and log-odds ratio γ_{b1}. Noteworthy, BUSseq includes the negative binomial distribution without zero inflation as a special case. When all cells are from a single cell type and the cell-specific size factor δ_{bi} is estimated a priori according to spike-in genes, BUSseq can reduce to a form similar to BASiCS [7].

3 Entire workflow

3.1 Data Preparation

The input data of our MCMC algorithm should be a `SingleCellExperiment` object or a `list`. The input `SingleCellExperiment` object should include a `counts` assay of raw read count data and `colData` indicating the corresponding batch of each cell. On the other hand, if the input is a `list`, then each element of the list corresponds to the read count matrix of a batch, where each row represents a gene and each column corresponds to a cell. Here, we take the raw read count data `assay(BUSseqfits_example, "counts")` stored in our package as an example to illustrate how to prepare the input.

```
library(BUSseq)
library(SingleCellExperiment)

## Loading required package:  SummarizedExperiment

## Loading required package:  MatrixGenerics

## Loading required package:  matrixStats

##
## Attaching package:  'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
```

```
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs,
##     rowVars, rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## Loading required package:  GenomicRanges

## Loading required package:  stats4

## Loading required package:  BiocGenerics

##
## Attaching package:  'BiocGenerics'

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, aperm,
##     append, as.data.frame, basename, cbind, colnames, dirname,
##     do.call, duplicated, eval, evalq, get, grep, grepl, intersect,
##     is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
##     setdiff, table, tapply, union, unique, unsplit, which.max,
##     which.min

## Loading required package:  S4Vectors

##
## Attaching package:  'S4Vectors'

## The following object is masked from 'package:utils':
##
##     findMatches
```

```
## The following objects are masked from 'package:base':
##
##     I, expand.grid, unname

## Loading required package:  IRanges

## Loading required package:  GenomeInfoDb

## Loading required package:  Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'.  To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

##
## Attaching package:  'Biobase'

## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians

## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians
```

```
#Input data is should be a SingleCellExperiment object or a list
CountData <- assay(BUSseqfits_example, "counts")
batch_ind <- unlist(colData(BUSseqfits_example))

# Construct a SingleCellExperiment object with colData as batch indicators
sce_input <- SingleCellExperiment(assays = list(counts = CountData),
                          colData = DataFrame(Batch_ind = factor(batch_ind)))

# Or, construct a list with each element represents the count data matrix
# of a batch
num_cell_batch <- table(batch_ind)
list_input <- list(Batch1 = CountData[,1:num_cell_batch[1]],
              Batch2 = CountData[,1:num_cell_batch[2] + num_cell_batch[1]])

# Cell numbers within each batch
```

```
print(num_cell_batch)
```

```
## batch_ind
##   1   2
## 150 150
```

```
#Peek at the read counts
print(CountData[1:5,1:5])
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    5   11    5    6   12
## [2,]    6   16    8    7    5
## [3,]    7    6    7   10   12
## [4,]   13    9    8   14    2
## [5,]    4   12    5    7    3
```

The example raw count data `CountData` consist of two batches. Each batch includes 150 cells, and there are 300 genes measured in each cell. Because it is a simulated dataset, we actually know that all of the cells come from 4 cell types.

In a nutshell, users can use a `SingleCellExperiment` object or a `list` as the input of our MCMC algorithm. Note that the gene numbers of all batches need to be the same.

3.2 Model Fitting

Once we have prepared the input data and specified the cell type number, we are able to fit the BUSseq model by running the `BUSseq_MCMC` function.

```
# Conduct MCMC sampling and posterior inference for BUSseq model
BUSseqfits_res <- BUSseq_MCMC(ObservedData = sce_input,
                          seed = 1234, n.cores = 2,
                          n.celltypes = 4, n.iterations = 500)
```

```
##    conducting the posterior sampling...
```

```
## [>                                        ] Finish 0.00k/0.50k iterations.
[====>                                    ] Finish 0.05k/0.50k iterations.
[========>                                ] Finish 0.10k/0.50k iterations.
```

```
[============>                            ] Finish 0.15k/0.50k iterations.
[================>                        ] Finish 0.20k/0.50k iterations.
[====================>                    ] Finish 0.25k/0.50k iterations.
[========================>                ] Finish 0.30k/0.50k iterations.
[============================>            ] Finish 0.35k/0.50k iterations.
[================================>        ] Finish 0.40k/0.50k iterations.
[====================================>    ] Finish 0.45k/0.50k iterations.
[========================================] Finish 0.50k/0.50k iterations.

##    The MCMC sampling takes:  0.166 mins

##    conducting the posterior inference...

##    Posterior inference takes:  0.003 mins
```

The first argument, `ObservedData`, of `BUSseq_MCMC` should be a `SingleCellExperiment` object or a `list` as we discuss before.

The second argument, `seed`, allows the users to obtain reproducible results.

The third argument, `n.celltypes`, is the number of cell types among all cells, which needs to be specified by the user in advance. As discussed later, if `n.celltypes` is unknown, we can vary the cell type number and use the Bayesian Information Criterion (BIC) to select the optimal number.

The forth argument, `n.iterations`, is the total number of iterations of the MCMC algorithm for the posterior inference of the BUSseq model. Users can also set the number of burnin iterations by the argument, `n.burnin`. Given `n.iterations`, the default number of burnins is `n.iterations`/2 iterations. The parameters are inferred by samples after the burnin iterations.

Now, let us take a look at the output:

```
# The output is a SingleCellExperiment object
class(BUSseqfits_res)

## [1] "SingleCellExperiment"
## attr(,"package")
## [1] "SingleCellExperiment"
```

```
  # Peek at the output
  BUSseqfits_res
```

```
## class: SingleCellExperiment
## dim: 300 300
## metadata(0):
## assays(2): counts imputed_data
## rownames: NULL
## rowData names(0):
## colnames(300): Batch_ind Batch_ind ... Batch_ind Batch_ind
## colData names(1): Batch_ind
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

`BUSseqfits_res` is a `SingleCellExperiment` object. Compared with the input, `BUSseq_MCMC` incoporates the inferred underlying true read counts after imputing the dropout events and the posterior inference of parameters into `BUSseqfits_res`. The posterior inference includes the posterior mode of the cell-type indicators for each cell, and the posterior mean and variance of the cell-type proportions within each batch, the cell-type-specific mean expression levels, the location batch effects, the overdispersion parameters and the log-odds ratios of the logistic regression for dropout events. Here, we show how to extract the imputed data from the output.

```
  # Extract the imputed read counts
  Imputed_count <- assay(BUSseqfits_res, "imputed_data")
```

We will further explain how to obtain the parameter estimation from the output `BUSseqfits_res` in the next section.

3.3 Estimated Cell Type, Batch and Cell-Specific Effect Extraction

Our main interests are the estimation of the cell type for each cell and the estimation of the batch effects. We can call the `celltypes` function to extract the estimated cell type labels $\widehat{W}_{bi}$ from `BUSseqfits_res`.

```
  celltyes_est <- celltypes(BUSseqfits_res)
## Batch 1 cells' cell type indicators:  1,1,1...  ...
## Batch 2 cells' cell type indicators:  1,1,1...  ...
## The output format is an N-dimensional verctor.
```

There is a message from the `celltypes` function to remind the format of it output.

Similarly, you can call the `location_batch_effects` and `overdispersions` functions to get the estimated location batch effects $\widehat{\nu}_{bg}$ and batch-specific and gene-specific overdispersion parameters $\widehat{\phi}_{bg}$. Note that the first batch is taken as the reference batch, so its location batch effects are zeros for all genes.

```
  location_batch_effects_est <- location_batch_effects(BUSseqfits_res)
## The output format is a matrix.
## Each row represents a gene, and each column corresponds to a batch.
  head(location_batch_effects_est)
##      [,1]     [,2]
## [1,]    0 1.825914
## [2,]    0 1.856586
## [3,]    0 1.817112
## [4,]    0 1.772725
## [5,]    0 1.850341
## [6,]    0 1.800893
  overdispersion_est <- overdispersions(BUSseqfits_res)
## The output format is a matrix.
##
## Each row represents a gene, and each column corresponds to a batch.
  head(overdispersion_est)
##          [,1]      [,2]
## [1,] 11.01228  7.843142
## [2,] 14.60814 11.325966
```

```
## [3,] 15.32366 12.675340
## [4,] 18.04462 10.543350
## [5,] 16.99327 15.770450
## [6,] 14.27186  6.242549
```

The estimated cell-specific size factors $\widehat{\delta}_{bi}$ are available by calling the `cell_effect_values` function. Here, the first cell in each batch is regarded as the reference cell, and its size factor is set as zero.

```
  cell_effects_est <- cell_effect_values(BUSseqfits_res)

## The output format is an N-dimensional vector.

  head(cell_effects_est)

## [1]  0.0000000 -0.3329874 -0.3004243 -0.3180968 -0.3163159 -0.4345818
```

The `celltype_mean_expression` function provides the estimated cell-type-specific mean expression levels $exp(\widehat{\alpha}_g + \widehat{\beta}_{gk})$. The estimates remove the technical artifacts, including the batch effects and the cell-spcific size factors, but retain the biological variability across different cell types. Moreover, the estimated cell type effects $\widehat{\beta}_{gk}$ can be obtained by the `celltype_effects` function. Notice that the first cell type is regarded as the reference cell type implying all zeros in the first column of `celltype_effects_est`.

```
  celltype_mean_expression_est <- celltype_mean_expression(BUSseqfits_example)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a cell type.

  head(celltype_mean_expression_est)

##           [,1]     [,2]     [,3]     [,4]
## [1,] 10.376815 1.250272 1.274214 1.409057
## [2,]  9.954963 1.273692 1.341472 1.109019
## [3,] 10.406628 1.331637 1.252357 1.356293
## [4,] 10.393802 1.423639 1.629966 1.639577
## [5,]  8.982671 1.365512 1.315853 1.303614
## [6,] 10.825665 1.534218 1.330363 1.483122

  celltype_effects_est <- celltype_effects(BUSseqfits_res)
```

```
## The output format is a matrix.
##
## Each row represents a gene, and each column corresponds to a cell type.
```

```
head(celltype_effects_est)
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,]    0 -2.116213 -2.097245 -1.996653
## [2,]    0 -2.056152 -2.004303 -2.194595
## [3,]    0 -2.056034 -2.117415 -2.037687
## [4,]    0 -1.987993 -1.852650 -1.846771
## [5,]    0 -1.883768 -1.920812 -1.930157
## [6,]    0 -1.953899 -2.096468 -1.987770
```

3.4 Intrinsic Gene Identification

```
#obtain the intrinsic gene indicators
intrinsic_gene_indicators <- intrinsic_genes_BUSseq(BUSseqfits_res)
print(intrinsic_gene_indicators)
```

```
## DataFrame with 300 rows and 1 column
##     IntrinsicGene
##       <character>
## 1             Yes
## 2             Yes
## 3             Yes
## 4             Yes
## 5             Yes
## ...           ...
## 296            No
## 297            No
## 298            No
## 299            No
## 300            No
```

```
#The estimated FDR, the first 240 genes are known as intrinsic
#genes in the simulation setting.
index_intri <- which(unlist(intrinsic_gene_indicators) == "Yes")
false_discovery_ind <- !(index_intri %in% 1:240)
```

```
  fdr_est <- sum(false_discovery_ind)/length(index_intri)
  print(fdr_est)
```

```
## [1] 0.004149378
```

Therefore, the true FDR is 0.0041494 much less than the estimated FDR, 0.05.

3.5 Corrected Read Count Data and Visualization

The `BUSseq_MCMC` function not only conducts MCMC sampling and posterior inference, but also imputes the missing data caused by dropout events. Furthermore, based on the imputed data, we expect to correct batch effects as if all the batches were measured in a single scRNA-seq experiment. The `corrected_read_counts` function adjusts to the imputed data and adds the corrected read count data into the input `SingleCellExperiment`.

```
  # Obtain the corrected read count data
  BUSseqfits_res <- corrected_read_counts(BUSseqfits_res)
```

```
##    correcting read counts...
```

```
## The corrected read count matrix is added into the output "SingleCellExperiment"
object.
```

```
  # An new assay "corrected_data" is incorporated
  BUSseqfits_res
```

```
## class: SingleCellExperiment
## dim: 300 300
## metadata(0):
## assays(3): counts imputed_data corrected_data
## rownames: NULL
## rowData names(0):
## colnames(300): Batch_ind Batch_ind ... Batch_ind Batch_ind
## colData names(1): Batch_ind
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

Subsequently, we visualize the raw count data that suffer from batch effects and dropout events, the inferred true expression levels after imputing dropout events, and the corrected count data which impute the dropout events and remove the batch effects. The `heatmap_data_BUSseq` function draws the heatmap for the count data across batches for the output `SinleCellExperiment` object of the functions `BUSseq_MCMC` and `corrected_read_counts`.

First, the used assay to draw the heatmap is determined by the arugment `data_type`, including `Raw`, `Imputed` and `Corrected`. Moreover, the heatmap will be stored in the local folder according to the argument `image_dir`. The image name can be modified by the argument `project_name`. Besides, the user can specify the argument `gene_set` to only display a subset of genes in the heatmap.

```
  #generate the heatmap of raw read count data
  heatmap_data_BUSseq(BUSseqfits_res, data_type = "Raw",
                          project_name = "BUSseq_raw_allgenes",
                          image_dir = "./heatmap")
```

```
## pdf
##   2
```

```
  #display only the first 100 genes in the heatmap
  heatmap_data_BUSseq(BUSseqfits_res, data_type = "Raw",
                          gene_set = 1:100,
                          project_name = "BUSseq_raw_100genes",
                          image_dir = "./heatmap")
```

```
## pdf
##   2
```

```
  #generate the heatmap of imputed read count data
  heatmap_data_BUSseq(BUSseqfits_res, data_type = "Imputed",
                          project_name = "BUSseq_imputed_allgenes",
                          image_dir = "./heatmap")
```

```
## pdf
##   2
```

```
  #generate the heatmap of corrected read count data
  heatmap_data_BUSseq(BUSseqfits_res, data_type = "Corrected",
                          project_name = "BUSseq_corrected_allgenes",
```

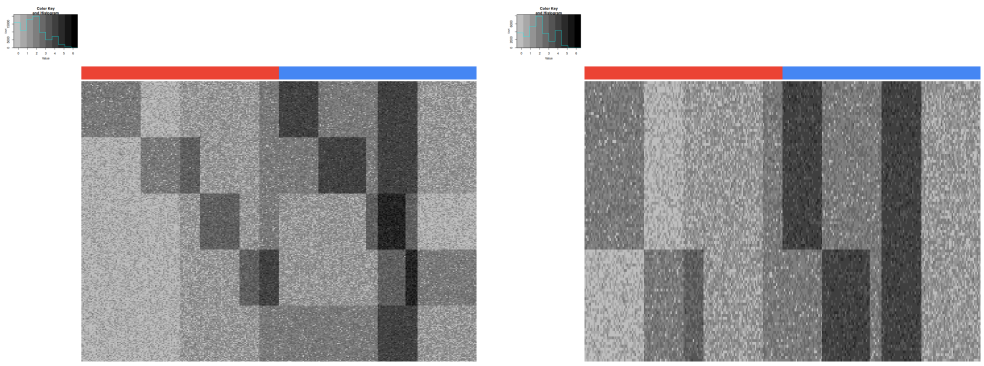

Figure 2: The heatmap of the raw count data of all genes (left) and the first 100 genes (right). Each row represents a gene, and each column denotes a cell. The color bar indicates the corresponding batch of each cell.

```
                                image_dir = "./heatmap")

## pdf
##   2
```

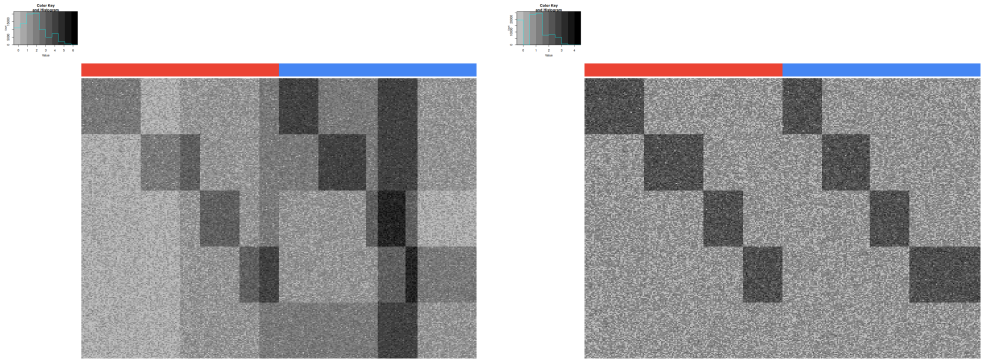

Figure 3: The heatmap for the imputed (left) and corrected (right) count data of all genes.

In all these heatmaps, the top bar indicates the corresponding batch of each cell. That's to say, cells under the same color are from the same batch. The batch effects present in the raw data are correctly removed in the corrected count data, and only the biological variabilities are kept. We can also only display the identified intrinsic genes in the corrected count data by setting the argument `gene_set` as the indices of the identified intrinsic genes `index_intri` in the last section.

```
#Only show the identified intrinsic genes
heatmap_data_BUSseq(BUSseqfits_res, data_type = "Corrected",
                    gene_set = index_intri,
                    project_name = "BUSseq_corrected_intrinsic_genes",
                    image_dir = "./heatmap")

## pdf
##   2
```

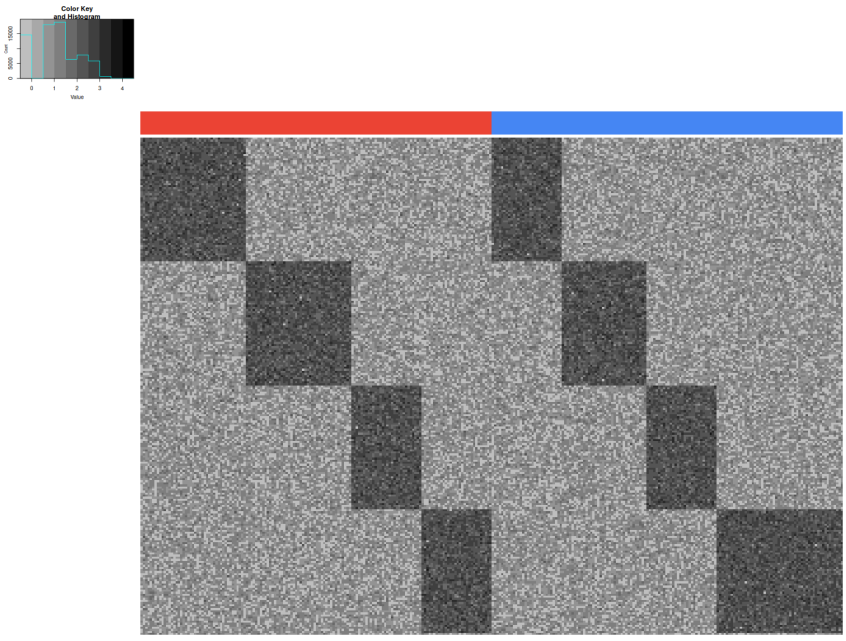

Figure 4: The heatmap for the corrected count data of the identified intrinsic genes.

4 Performance of BUSseq in real data analysis

In the scRNA-seq experiments, the unknown cell types of individual cells are usually the target of inference. Because of severe batch effects, if we directly pool cells from different experiments together, the cells are often clustered by batch or experiment rather than by cell type. After correcting batch effects, cells can be clustered based on the corrected read count data or their corresponding low-dimensional embedding. Therefore, to benchmark different batch effects correct methods, we expect that the estimated cell-type labels are highly consistent with the reference cell type labels generated by the fluorescence-activated cell sorting (FACS)

technique. The adjusted Rand index (ARI) can measure the consistency between the estimated labels and the reference ones. ARI ranges from 0 to 1, and the higher value means the better consistency [8].

[5] benchmarked BUSseq with the state-of-the-art methods of batch effects correction for scRNA-seq data, including LIGER [9], MNN [10], Scanorama [11], scVI [12], Seurat [13] and ZINB-WaVE [14]. We applied all these methods to integrate multiple scRNA-seq experiments in a mouse hematopoietic study and a human pancreatic study, respectively.

Method	ARI on hematopoietic study	ARI on pancreatic study
BUSseq	0.582	0.608
LIGER	0.307	0.542
MNN	0.575	0.279
Scanorama	0.518	0.527
scVI	0.197	0.282
Seurat 3.0	0.266	0.287
ZINB-WaVE	0.348	0.380

Table 1: The comparison of different methods in the cell type clustering.

According to the above table, BUSseq outperforms all of the other methods in being consistent with the reference cell-type labels for these two real studies.

5 Session information

```
  sessionInfo()
```

```
## R version 4.4.1 (2024-06-14)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 24.04.1 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK versio
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
```

```
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: Etc/UTC
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] SingleCellExperiment_1.27.2 SummarizedExperiment_1.35.1
##  [3] Biobase_2.65.1              GenomicRanges_1.57.1
##  [5] GenomeInfoDb_1.41.1         IRanges_2.39.2
##  [7] S4Vectors_0.43.2            BiocGenerics_0.51.1
##  [9] MatrixGenerics_1.17.0       matrixStats_1.4.1
## [11] BUSseq_1.11.0
##
## loaded via a namespace (and not attached):
##  [1] Matrix_1.7-0            BiocStyle_2.33.1        jsonlite_1.8.8
##  [4] highr_0.11              compiler_4.4.1          BiocManager_1.30.25
##  [7] crayon_1.5.3            gtools_3.9.5            bitops_1.0-8
## [10] yaml_2.3.10             fastmap_1.2.0           lattice_0.22-6
## [13] R6_2.5.1                XVector_0.45.0          S4Arrays_1.5.7
## [16] knitr_1.48              DelayedArray_0.31.11    maketools_1.3.0
## [19] GenomeInfoDbData_1.2.12 rlang_1.1.4             xfun_0.47
## [22] caTools_1.18.3          sys_3.4.2               SparseArray_1.5.31
## [25] cli_3.6.3               zlibbioc_1.51.1         digest_0.6.37
## [28] grid_4.4.1              KernSmooth_2.23-24      evaluate_0.24.0
## [31] buildtools_1.0.0        abind_1.4-5             rmarkdown_2.28
## [34] httr_1.4.7              tools_4.4.1             htmltools_0.5.8.1
## [37] UCSC.utils_1.1.0        gplots_3.1.3.1
```

References

[1] Rhonda Bacher and Christina Kendziorski. Design and computational analysis of single-cell rna-sequencing experiments. *Genome Biology*, 17(1):63, 2016.

[2] W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

[3] Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genetics*, 3(9):e161, 2007.

[4] Jeffrey T Leek. svaseq: removing batch effects and other unwanted noise from sequencing data. *Nucleic Acids Research*, page gku864, 2014.

[5] Fangda Song, Ga Ming Angus Chan, and Yingying Wei. Flexible experimental designs for valid single-cell rna-sequencing experiments allowing batch effects correction. *Nature communications*, 11(1):1–15, 2020.

[6] Peter V Kharchenko, Lev Silberstein, and David T Scadden. Bayesian approach to single-cell differential expression analysis. *Nature Methods*, 11(7):740, 2014.

[7] Catalina A Vallejos, John C Marioni, and Sylvia Richardson. BASiCS: Bayesian analysis of single-cell sequencing data. *PLoS Computational Biology*, 11(6):e1004333, 2015.

[8] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[9] Joshua D Welch, Velina Kozareva, Ashley Ferreira, Charles Vanderburg, Carly Martin, and Evan Z Macosko. Single-cell multi-omic integration compares and contrasts features of brain cell identity. *Cell*, 177(7):1873–1887.e17, 2019.

[10] Laleh Haghverdi, Aaron TL Lun, Michael D Morgan, and John C Marioni. Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature Biotechnology*, 36(5):421–427, 2018.

[11] Brian Hie, Bryan Bryson, and Bonnie Berger. Efficient integration of heterogeneous single-cell transcriptomes using Scanorama. *Nature Biotechnology*, 37(6):685, 2019.

[12] Romain Lopez, Jeffrey Regier, Michael B Cole, Michael I Jordan, and Nir Yosef. Deep generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12):1053, 2018.

[13] 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.e21, 2019.

[14] Davide Risso, Fanny Perraudeau, Svetlana Gribkova, Sandrine Dudoit, and Jean-Philippe Vert. A general and flexible method for signal extraction from single-cell RNA-seq data. *Nature Communications*, 9(1):284, 2018.