Package 'batchelor'

June 18, 2019

Version 1.1.4

Date 2019-05-30

Title Single-Cell Batch Correction Methods

Depends SingleCellExperiment

Imports SummarizedExperiment, S4Vectors, BiocGenerics, Rcpp, stats, methods, BiocNeighbors, BiocSingular, Matrix, DelayedArray, DelayedMatrixStats, scater, BiocParallel

Suggests testthat, BiocStyle, knitr, beachmat, scRNAseq

biocViews Sequencing, RNASeq, Software, GeneExpression, Transcriptomics, SingleCell, BatchEffect, Normalization

LinkingTo Rcpp, beachmat

Description

Implements a variety of methods for batch correction of single-cell (RNA sequencing) data. This includes methods based on detecting mutually nearest neighbors, as well as a simple sparsity-preserving translation of the population means. Functions are also provided for global rescaling to remove differences in depth between batches, and to perform a principal components analysis that is robust to differences in the numbers of cells across batches.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

SystemRequirements C++11

RoxygenNote 6.1.1

git_url https://git.bioconductor.org/packages/batchelor

git_branch master

git_last_commit 3e7511c

git_last_commit_date 2019-05-31

Date/Publication 2019-06-17

Author Aaron Lun [aut, cre], Laleh Haghverdi [ctb]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

`batchCorrect` *Batch correction methods*

Description

A common interface for single-cell batch correction methods.

Usage

```
batchCorrect(..., batch = NULL, restrict = NULL, subset.row = NULL,
  correct.all = FALSE, assay.type = NULL, get.spikes = FALSE, PARAM)

## S4 method for signature 'ClassicMnnParam'
batchCorrect(..., batch = NULL,
  restrict = NULL, subset.row = NULL, correct.all = FALSE,
  assay.type = "logcounts", get.spikes = FALSE, PARAM)

## S4 method for signature 'FastMnnParam'
batchCorrect(..., batch = NULL,
  restrict = NULL, subset.row = NULL, correct.all = FALSE,
  assay.type = "logcounts", get.spikes = FALSE, PARAM)

## S4 method for signature 'RescaleParam'
batchCorrect(..., batch = NULL,
  restrict = NULL, subset.row = NULL, correct.all = FALSE,
  assay.type = "logcounts", get.spikes = FALSE, PARAM)
```

Arguments

`...`	Named data-dependent parameters to pass to the dispatched batch correction methods. This should contain one or more matrix-like objects containing single-cell gene expression matrices. Alternatively, one or more SingleCellExperiment objects can be supplied.
`batch`	A factor specifying the batch of origin for each cell if only one batch is supplied. This will be ignored if two or more batches are supplied.

`restrict`	A list of length equal to the number of objects in `...`. Each entry of the list corresponds to one batch and specifies the cells to use when computing the correction.
`subset.row`	A vector specifying the subset of genes to use for correction. Defaults to `NULL`, in which case all genes are used.
`correct.all`	A logical scalar indicating whether to return corrected expression values for all genes, even if `subset.row` is set. Used to ensure that the output is of the same dimensionality as the input.
`assay.type`	A string or integer scalar specifying the assay to use for correction. Only used for SingleCellExperiment inputs.
`get.spikes`	A logical scalar indicating whether to retain rows corresponding to spike-in transcripts. Only used for SingleCellExperiment inputs.
`PARAM`	A BatchelorParam object specifying the batch correction method to dispatch to. ClassicMnnParam will dispatch to `mnnCorrect`; FastMnnParam will dispatch to `fastMNN`; and RescaleParam will dispatch to `rescaleBatches`.

Details

Users can pass parameters to each method directly via `...` or via the constructors for `PARAM`. While there is no restriction on which parameters go where, we recommend only passing data-agnostic and method-specific parameters to `PARAM`. Data-dependent parameters - and indeed, the data themselves - should be passed in via `...`. This means that different data sets can be used without modifying `PARAM`, and allows users to switch to a different algorithm by only changing `PARAM`.

Note that `get.spikes=FALSE` effectively modifies `subset.row` to exclude spike-in transcripts when SingleCellExperiment inputs are supplied. This means that the reported SingleCellExperiment will not, by default, contain corrected expression values for spike-in transcripts unless `get.spikes=TRUE`.

Value

A SingleCellExperiment where the first assay contains corrected gene expression values for all genes. Corrected values should be returned for all genes if `subset.row=NULL` or if `correct.all=TRUE`; otherwise they should only be returned for the genes in the subset.

Cells should be reported in the same order that they are supplied. In cases with multiple batches, the cell identities are simply concatenated from successive objects in their specified order, i.e., all cells from the first object (in their provided order), then all cells from the second object, and so on. For a single input object, cells should be reported in the same order as the input.

The `colData` slot should contain `batch`, a vector specifying the batch of origin for each cell.

Author(s)

Aaron Lun

See Also

BatchelorParam classes to determine dispatch.

Examples

```
B1 <- matrix(rnorm(10000), ncol=50) # Batch 1
B2 <- matrix(rnorm(10000), ncol=50) # Batch 2
```

```
# Switching easily between batch correction methods.
m.out <- batchCorrect(B1, B2, PARAM=ClassicMnnParam())
f.out <- batchCorrect(B1, B2, PARAM=FastMnnParam(d=20))
r.out <- batchCorrect(B1, B2, PARAM=RescaleParam(pseudo.count=0))
```

`BatchelorParam-class` *BatchelorParam methods*

Description

Constructors and methods for the batchelor parameter classes.

Usage

```
ClassicMnnParam(...)

FastMnnParam(...)

RescaleParam(...)
```

Arguments

`...`	Named arguments to pass to individual methods upon dispatch. These should not include arguments named in the `batchCorrect` generic.

Details

BatchelorParam objects are intended to store method-specific parameter settings to pass to the `batchCorrect` generic. These values should refer to data-agnostic parameters; parameters that depend on data (or the data itself) should be specified directly in the `batchCorrect` call.

The BatchelorParam classes are all derived from SimpleList objects and have the same available methods, e.g., `[[`, `$`. These can be used to access or modify the object after construction.

Note that the BatchelorParam class itself is not useful and should not be constructed directly. Instead, users should use the constructors shown above to create instances of the desired subclass.

Value

The constructors will return a BatchelorParam object of the specified subclass, containing parameter settings for the corresponding batch correction method.

Author(s)

Aaron Lun

See Also

`batchCorrect`, where the BatchelorParam objects are used for dispatch to individual methods.

Examples

```
# Specifying the number of neighbors, dimensionality.
fp <- FastMnnParam(k=20, d=10)
fp

# List-like behaviour:
fp$k
fp$k <- 10
fp$k
```

---

checkBatchConsistency *Check batch inputs*

Description

Utilities to check inputs into batch correction functions.

Usage

```
checkBatchConsistency(batches, cells.in.columns = TRUE)

checkSpikeConsistency(batches)

checkIfSCE(batches)

checkRestrictions(batches, restrictions, cells.in.columns = TRUE)
```

Arguments

batches	A list of batches, usually containing gene expression matrices or SingleCellExperiment objects.
cells.in.columns	A logical scalar specifying whether batches contain cells in the columns.
restrictions	A list of length equal to batches, specifying the cells in each batch that should be used for correction.

Details

These functions are intended for internal use and other package developers.

checkBatchConsistency will check whether the input batches are consistent with respect to the size of the dimension containing features (i.e., not cells). It will also verify that the dimension names are consistent, to avoid problems from variable ordering of rows/columns in the inputs.

checkSpikeConsistency will check whether the spike-in information is consistent across all batches. This only works for SingleCellExperiment objects, so one should only run this function if checkIfSCE returns TRUE.

checkRestrictions will check whether restrictions are consistent with the supplied batches, in terms of the length and names of the two lists. It will also check that each batch contains at least one usable cell after restriction.

Value

checkBatchConsistency and checkSpikeConsistency will return an invisible NULL if there are no errors.

checkIfSCE will return a logical vector specifying whether each element of batches is a SingleCellExperiment objects.

checkRestrictions will return NULL if restrictions=NULL. Otherwise, it will return a list by taking restrictions and converting each non-NULL element into an integer subsetting vector.

Author(s)

Aaron Lun

See Also

divideIntoBatches

Examples

```
checkBatchConsistency(list(cbind(1:5), cbind(1:5, 2:6)))
try( # fails
    checkBatchConsistency(list(cbind(1:5), cbind(1:4, 2:5)))
)
```

---

cosineNorm *Cosine normalization*

Description

Perform cosine normalization on the column vectors of an expression matrix.

Usage

```
cosineNorm(x, mode = c("matrix", "all", "l2norm"))
```

Arguments

x	A gene expression matrix with cells as columns and genes as rows.
mode	A string specifying the output to be returned.

Details

Cosine normalization removes scaling differences between expression vectors. In the context of batch correction, this is usually applied to remove differences between batches that are normalized separately. For example, fastMNN uses this function on the log-expression vectors by default.

Technically, separate normalization introduces scaling differences in the normalized expression, which should manifest as a shift in the log-transformed expression. However, in practice, single-cell data will contain many small counts (where the log function is near-linear) or many zeroes (which remain zero when the pseudo-count is 1). In these applications, scaling differences due to separate normalization are better represented as scaling differences in the log-transformed values.

If applied to the raw count vectors, cosine normalization is similar to library size-related (i.e., L1) normalization. However, we recommend using dedicated methods for computing size factors to normalize raw count data.

While the default is to directly return the cosine-normalized matrix, it may occasionally be desirable to obtain the L2 norm, e.g., to apply an equivalent normalization to other matrices. This can be achieved by setting `mode` accordingly.

The function will return a DelayedMatrix if `x` is a DelayedMatrix. This aims to delay the calculation of cosine-normalized values for very large matrices.

Value

If `mode="matrix"`, a double-precision matrix of the same dimensions as `X` is returned, containing cosine-normalized values.

If `mode="l2norm"`, a double-precision vector is returned containing the L2 norm for each cell.

If `mode="all"`, a named list is returned containing the fields `"matrix"` and `"l2norm"`, which are as described above.

Author(s)

Aaron Lun

See Also

`mnnCorrect` and `fastMNN`, where this function gets used.

Examples

```
A <- matrix(rnorm(1000), nrow=10)
str(cosineNorm(A))
str(cosineNorm(A, mode="l2norm"))
```

---

divideIntoBatches *Divide into batches*

Description

Divide a single input object into multiple separate objects according to their batch of origin.

Usage

```
divideIntoBatches(x, batch, byrow = FALSE, restrict = NULL)
```

Arguments

`x`	A matrix-like object where one dimension corresponds to cells and another represents features.
`batch`	A factor specifying the batch to which each cell belongs.
`byrow`	A logical scalar indicating whether rows correspond to cells.
`restrict`	A subsetting vector specifying which cells should be used for correction.

Details

This function is intended for internal use and other package developers. It splits a single input object into multiple batches, allowing developers to use the same code for the scenario where `batch` is supplied with a single input.

Value

A list containing:

- `batches`, a named list of matrix-like objects where each element corresponds to a level of `batch` and contains all cells from that batch.
- `reorder`, an integer vector to be applied to the combined `batches` to recover the ordering of cells in `x`.
- `restricted`, a named list of integer vectors specifying which cells are to be used for correction. Set to `NULL` if the input `restrict` was also `NULL`.

Author(s)

Aaron Lun

Examples

```
X <- matrix(rnorm(1000), ncol=100)
out <- divideIntoBatches(X, sample(3, 100, replace=TRUE))
names(out)

# Recovering original order.
Y <- do.call(cbind, out$batches)
Z <- Y[,out$reorder]
all.equal(Z, X) # should be TRUE.
```

`fastMNN` *Fast mutual nearest neighbors correction*

Description

Correct for batch effects in single-cell expression data using a fast version of the mutual nearest neighbors (MNN) method.

Usage

```
fastMNN(..., batch = NULL, k = 20, restrict = NULL,
  cos.norm = TRUE, ndist = 3, d = 50, auto.order = FALSE,
  min.batch.skip = 0, subset.row = NULL, correct.all = FALSE,
  pc.input = FALSE, assay.type = "logcounts", get.spikes = FALSE,
  use.dimred = NULL, BSPARAM = IrlbaParam(deferred = TRUE),
  BNPARAM = KmknnParam(), BPPARAM = SerialParam())
```

Arguments

...	One or more log-expression matrices where genes correspond to rows and cells correspond to columns, if pc.input=FALSE. Each matrix should contain the same number of rows, corresponding to the same genes in the same order. Alternatively, one or more matrices of low-dimensional representations can be supplied if pc.input=TRUE, where rows are cells and columns are dimensions. Each object should contain the same number of columns, corresponding to the same dimensions. Alternatively, one or more SingleCellExperiment objects can be supplied containing a log-expression matrix in the assay.type assay. Note the same restrictions described above for gene expression matrix inputs. Alternatively, the SingleCellExperiment objects can contain reduced dimension coordinates in the reducedDims slot if use.dimred is specified. Note the same restrictions described above for low-dimensional matrix inputs. Alternatively, one or more DataFrame objects produced by previous calls to fastMNN. This should contain a corrected field of low-dimensional corrected coordinates, along with information required for orthogonalization in the metadata. In all cases, each object contains cells from a single batch; multiple objects represent separate batches of cells. Objects of different types can be mixed together if all or none are low-dimensional.
batch	A factor specifying the batch of origin for all cells when only a single object is supplied in .... This is ignored if multiple objects are present.
k	An integer scalar specifying the number of nearest neighbors to consider when identifying MNNs.
restrict	A list of length equal to the number of objects in .... Each entry of the list corresponds to one batch and specifies the cells to use when computing the correction.
cos.norm	A logical scalar indicating whether cosine normalization should be performed on the input data prior to PCA.
ndist	A numeric scalar specifying the threshold beyond which neighbours are to be ignored when computing correction vectors. Each threshold is defined as a multiple of the number of median distances.
d	Number of dimensions to use for dimensionality reduction in multiBatchPCA.
auto.order	Logical scalar indicating whether re-ordering of batches should be performed to maximize the number of MNN pairs at each step. Alternatively, an integer vector containing a permutation of 1:N where N is the number of batches.
min.batch.skip	Numeric scalar specifying the minimum relative magnitude of the batch effect, below which no correction will be performed at a given merge step.
subset.row	A vector specifying which features to use for correction. Only relevant for gene expression inputs (i.e., pc.input=FALSE and use.dimred=NULL).
correct.all	Logical scalar indicating whether a rotation matrix should be computed for genes not in subset.row. Only used for gene expression inputs, i.e., when pc.input=FALSE.
pc.input	Logical scalar indicating whether the values in ... are already low-dimensional, e.g., the output of multiBatchPCA. Only used when ... does *not* contain SingleCellExperiment objects - in those cases, set use.dimred instead. This is also assumed to be TRUE if any element of ... is a DataFrame.

assay.type	A string or integer scalar specifying the assay containing the log-expression values. Only used for SingleCellExperiment inputs with use.dimred=NULL.
get.spikes	A logical scalar indicating whether to retain rows corresponding to spike-in transcripts. Only used for SingleCellExperiment inputs with use.dimred=NULL.
use.dimred	A string or integer scalar specifying which reduced dimension result to use, if any. Only used for SingleCellExperiment inputs.
BSPARAM	A BiocSingularParam object specifying the algorithm to use for PCA. This uses a fast approximate algorithm from **irlba** by default, see multiBatchPCA for details.
BNPARAM	A BiocNeighborParam object specifying the nearest neighbor algorithm.
BPPARAM	A BiocParallelParam object specifying whether the PCA and nearest-neighbor searches should be parallelized.

Details

This function provides a variant of the mnnCorrect function, modified for speed and more robust performance. In particular:

- It performs a multi-sample PCA via multiBatchPCA and subsequently performs all calculations in the PC space. This reduces computational work and provides some denoising for improved neighbour detection. As a result, though, the corrected output cannot be interpreted on a gene level and is useful only for cell-level comparisons, e.g., clustering and visualization.
- The correction vector for each cell is directly computed from its k nearest neighbours in the same batch. Specifically, only the k nearest neighbouring cells that *also* participate in MNN pairs are used. Each MNN-participating neighbour is weighted by distance from the current cell, using a tricube scheme with bandwidth equal to the median distance multiplied by ndist. This ensures that the correction vector only uses information from the closest cells, improving the fidelity of local correction.
- Issues with "kissing" are avoided with a two-step procedure that removes variation along the batch effect vector. First, the average correction vector across all MNN pairs is computed. Cell coordinates are adjusted such that all cells in a single batch have the same position along this vector. The correction vectors are then recalculated with the adjusted coordinates (but the same MNN pairs).

The default setting of cos.norm=TRUE provides some protection against differences in scaling between log-expression matrices from batches that are normalized separately (see cosineNorm for details). However, if possible, we recommend using the output of multiBatchNorm as input to fastMNN. This will equalize coverage on the count level before the log-transformation, which is a more accurate rescaling than cosine normalization on the log-values.

The batch argument allows users to easily perform batch correction when all cells have already been combined into a single object. This avoids the need to manually split the matrix or SingleCellExperiment object into separate objects for input into fastMNN. In this situation, the order of input batches is defined by the order of levels in batch.

Value

The output of this function depends on whether a PCA is performed on the input .... This will be the case if pc.input=FALSE for matrix inputs or if use.dimred=NULL for SingleCellExperiment inputs.

If a PCA is performed, a SingleCellExperiment is returned where each row is a gene and each column is a cell. This contains:

- A corrected matrix in the reducedDims slot, containing corrected low-dimensional coordinates for each cell. This has number of columns equal to d and number of rows equal to the total number of cells in ....
- A batch column in the colData slot, containing the batch of origin for each row (i.e., cell) in corrected.
- A rotation column the rowData slot, containing the rotation matrix used for the PCA. This has d columns and number of rows equal to the number of genes to report (see the "Choice of genes" section).
- A reconstructed matrix in the assays slot, containing the low-rank reconstruction of the original expression matrix. This can be interpreted as per-gene corrected log-expression values (after cosine normalization, if cos.norm=TRUE) but should not be used for quantitative analyses.

Otherwise, a DataFrame is returned where each row corresponds to a cell. It contains:

- corrected, the matrix of corrected low-dimensional coordinates for each cell.
- batch, the Rle specifying the batch of origin for each row.

Cells in the output object are always ordered in the same manner as supplied in .... For a single input object, cells will be reported in the same order as they are arranged in that object. In cases with multiple input objects, the cell identities are simply concatenated from successive objects, i.e., all cells from the first object (in their provided order), then all cells from the second object, and so on. This is true regardless of the value of auto.order, which only affects the internal merge order of the batches.

The metadata of the output object contains:

- merge.order, a vector of batch names or indices, specifying the order in which batches were merged.
- merge.info, a DataFrame of information about each merge step (corresponding to each row). This contains the following fields:
  - pairs, a List of DataFrames specifying which pairs of cells in corrected were identified as MNNs at each step.
  - batch.vector, a List of numeric vectors specifying the average batch vector at each step.
  - batch.size, a numeric vector specifying the relative magnitude of the batch effect at each merge.
  - skipped, a logical vector indicating whether the correction was skipped if the magnitude was below min.batch.skip.
  - lost.var, a numeric matrix specifying the percentage of variance lost due to orthogonalization at each merge step. This is reported separately for each batch (columns, ordered according to the input order, *not* the merge order).
- pre.orthog, a DataFrame containing information about pre-correction orthogonalization. This is only reported if ... contains one or more DataFrames. Each row corresponds to a vector used for orthogonalization in one of the DataFrames in .... The vector is stored in batch.vector and the variance lost due to orthogonalization in each batch is reported in lost.var.

Controlling the merge order

By default, batches are merged in the user-supplied order. However, if auto.order=TRUE, batches are ordered to maximize the number of MNN pairs at each step. The aim is to improve the stability of the correction by first merging more similar batches with more MNN pairs. This can be somewhat

time-consuming as MNN pairs need to be iteratively recomputed for all possible batch pairings. It is often more convenient for the user to specify an appropriate ordering based on prior knowledge about the batches.

If `auto.order` is an integer vector, it is treated as an ordering permutation with which to merge batches. For example, if `auto.order=c(4,1,3,2)`, batches 4 and 1 in `...` are merged first, followed by batch 3 and then batch 2. This is often more convenient than changing the order manually in `...`, which would alter the order of batches in the output `corrected` matrix. Indeed, no matter what the setting of `auto.order` is, the order of cells in the output corrected matrix is *always* the same.

Further control of the merge order can be achieved by performing the multi-sample PCA outside of this function with `multiBatchPCA`. Batches can then be progressively merged by repeated calls to `fastMNN` with low-dimensional inputs (see below). This is useful in situations where the batches need to be merged in a hierarhical manner, e.g., combining replicate samples before merging them across different conditions. For example, we could merge batch 1 with 4 to obtain a corrected 1+4; and then batch 2 with 3 to obtain a corrected 2+3; before merging the corrected 1+4 and 2+3 to obtain the final set of corrected values.

Choice of genes

All genes are used with the default setting of `subset.row=NULL`. Users can set `subset.row` to subset the inputs to highly variable genes or marker genes. This improves the quality of the PCA and identification of MNN pairs by reducing the noise from irrelevant genes. Note that users should not be too restrictive with subsetting, as high dimensionality is required to satisfy the orthogonality assumption in MNN detection.

For SingleCellExperiment inputs, spike-in transcripts are automatically removed unless `get.spikes=TRUE`. If `subset.row` is specified and `get.spikes=FALSE`, only the non-spike-in specified features will be used. All SingleCellExperiment objects should have the same set of spike-in transcripts.

By default, only the selected genes are used to compute rotation vectors and a low-rank representation of the input matrix. However, rotation vectors can be obtained that span all genes in the supplied input data with `correct.all=TRUE`. This will not affect the corrected low-dimension coordinates or the output for the selected genes.

Note that these settings for the choice of genes are completely ignored when using low-dimensional inputs (see below).

Using low-dimensional inputs

Low-dimensional inputs can be supplied directly to `fastMNN` if the PCA (or some other projection to low-dimensional space) is performed outside the function. This intructs the function to skip the `multiBatchPCA` step. To enable this, set `pc.input=TRUE` for matrix-like inputs in `...`, or specify `use.dimred` with SingleCellExperiment inputs.

If `...` contains any DataFrame objects, these are assumed to be the output of a previous `fastMNN` call. All inputs are subsequently treated as low-dimensional inputs and any other setting of `pc.input` is ignored. If any SingleCellExperiment objects are also present in `...`, `use.dimred` must be specified.

Note that `multiBatchPCA` will not perform cosine-normalization, so it is the responsibility of the user to cosine-normalize each batch beforehand with `cosineNorm` to recapitulate results with `cos.norm=TRUE`. In addition, `multiBatchPCA` must be run on all samples at once, to ensure that all cells are projected to the same low-dimensional space.

Users are referred to the Examples for a demonstration of this functionality.

Using restriction

It is possible to compute the correction using only a subset of cells in each batch, and then extrapolate that correction to all other cells. This may be desirable in experimental designs where a control set of cells from the same source population were run on different batches. Any difference in the controls must be artificial in origin and can be directly removed without making further biological assumptions.

To do this, users should set `restrict` to specify the subset of cells in each batch to be used for correction. This should be set to a list of length equal to the length of `...`, where each element is a subsetting vector to be applied to the columns of the corresponding batch. A `NULL` element indicates that all the cells from a batch should be used. In situations where one input object contains multiple batches, `restrict` is simply a list containing a single subsetting vector for that object.

`fastMNN` will only use the restricted subset of cells in each batch to identify MNN pairs and the center of the orthogonalization. However, it will apply the correction to all cells in each batch - hence the extrapolation. This means that the output is always of the same dimensionality, regardless of whether `restrict` is specified.

Note that *all* cells are used to perform the PCA, regardless of whether `restrict` is set. Constructing the projection vectors with only control cells will not guarantee resolution of unique non-control populations in each batch. The function will only completely ignore cells that are not in `restrict` if `pc.input=TRUE` or, for SingleCellExperiment inputs, `use.dimred` is set.

Orthogonalization details

`fastMNN` will compute the percentage of variance that is lost from each batch during orthogonalization at each merge step. This represents the variance in each batch that is parallel to the average correction vectors (and hence removed during orthogonalization) at each merge step. Large proportions suggest that there is biological structure that is parallel to the batch effect, corresponding to violations of the assumption that the batch effect is orthogonal to the biological subspace.

If `fastMNN` is called with DataFrame inputs, each DataFrame is assumed to be the result of a previous `fastMNN` call and have a set of vectors used for orthogonalization in the merge steps of that previous call. In the current call, `fastMNN` will gather all such batch vectors across all DataFrame inputs. Each batch is then re-orthogonalized with respect to each of these vectors. This ensures that the same variation is removed from each batch prior to merging. The variance lost due to this pre-correction orthogonalization is reported in the `pre.orthog` field in the output metadata.

Orthogonalization may cause problems if there is actually no batch effect, resulting in large losses of variance. To avoid this, `fastMNN` will not perform any correction if the relative magnitude of the batch effect is less than `min.batch.skip`. The relative magnitude is defined as the L2 norm of the average correction vector divided by the root-mean-square of the L2 norms of the per-MNN pair correction vectors. This will be large when the per-pair vectors are all pointing in the same direction, and small when the per-pair vectors point in random directions due to the absence of a consistent batch effect. If a large loss of variance is observed along with a small batch effect in a given merge step, users can set `min.batch.skip` to simply skip correction in that step.

Author(s)

Aaron Lun

References

Haghverdi L, Lun ATL, Morgan MD, Marioni JC (2018). Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nat. Biotechnol.* 36(5):421

Lun ATL (2018). Further MNN algorithm development. https://MarioniLab.github.io/FurtherMNN2018/theory/description.html

See Also

cosineNorm and multiBatchPCA to obtain the values to be corrected.

mnnCorrect for the "classic" version of the MNN correction algorithm.

Examples

```
B1 <- matrix(rnorm(10000), ncol=50) # Batch 1
B2 <- matrix(rnorm(10000), ncol=50) # Batch 2
out <- fastMNN(B1, B2)
str(reducedDim(out)) # corrected values

# An equivalent approach with PC input.
cB1 <- cosineNorm(B1)
cB2 <- cosineNorm(B2)
pcs <- multiBatchPCA(cB1, cB2)
out2 <- fastMNN(pcs[[1]], pcs[[2]], pc.input=TRUE)

all.equal(reducedDim(out), out2$corrected) # should be TRUE

# Extracting corrected expression values for gene 10.
summary(assay(out)[10,])
```

---

findMutualNN *Find mutual nearest neighbors*

Description

Find mutual nearest neighbors (MNN) across two data sets.

Usage

```
findMutualNN(data1, data2, k1, k2 = k1, BNPARAM = KmknnParam(),
  BPPARAM = SerialParam())
```

Arguments

data1	A numeric matrix containing samples (e.g., cells) in the rows and variables/dimensions in the columns.
data2	A numeric matrix like data1 for another data set with the same variables/dimensions.
k1	Integer scalar specifying the number of neighbors to search for in data1.
k2	Integer scalar specifying the number of neighbors to search for in data2.
BNPARAM	A BiocNeighborParam object specifying the neighbour search algorithm to use.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed.

Details

The concept of a MNN pair can be explained by considering cells in each of two data sets. For each cell in data set 1, the set of k2 nearest cells in data set 2 is identified, based on the Euclidean distance in expression space. For each cell in data set 2, the set of k1 nearest cells in data set 1 is similarly identified. Two cells in different batches are considered to be MNNs if each cell is in the other's set.

The value of k can be interpreted as the minimum size of a subpopulation in each batch. Larger values allow for more MNN pairs to be obtained, which improves the stability of batch correction in fastMNN and mnnCorrect. It also increases robustness against non-orthogonality, which would otherwise result in MNN pairs being detected on the "surface" of the distribution. Obviously, though, values of k should not be too large, as this would result in MNN pairs being inappropriately identified between biologically distinct populations.

Value

A list containing the integer vectors first and second. Corresponding entries in first and second specify a MNN pair of cells from data1 and data2, respectively.

Author(s)

Aaron Lun

See Also

queryKNN for the underlying neighbor search code.

Examples

```
B1 <- matrix(rnorm(10000), ncol=50) # Batch 1
B2 <- matrix(rnorm(10000), ncol=50) # Batch 2
out <- findMutualNN(B1, B2, k1=20)
head(out$first)
head(out$second)
```

---

mnnCorrect *Mutual nearest neighbors correction*

Description

Correct for batch effects in single-cell expression data using the mutual nearest neighbors method.

Usage

```
mnnCorrect(..., batch = NULL, restrict = NULL, k = 20, sigma = 0.1,
  cos.norm.in = TRUE, cos.norm.out = TRUE, svd.dim = 0L,
  var.adj = TRUE, subset.row = NULL, correct.all = FALSE,
  auto.order = FALSE, assay.type = "logcounts", get.spikes = FALSE,
  BSPARAM = ExactParam(), BNPARAM = KmknnParam(),
  BPPARAM = SerialParam())
```

Arguments

...	Two or more log-expression matrices where genes correspond to rows and cells correspond to columns. Each matrix should contain cells from the same batch; multiple matrices represent separate batches of cells. Each matrix should contain the same number of rows, corresponding to the same genes (in the same order). Alternatively, one or more SingleCellExperiment objects can be supplied containing a log-expression matrix in the `assay.type` assay. Note the same constraints described above for matrix inputs.
batch	A factor specifying the batch of origin for all cells when only a single object is supplied in .... This is ignored if multiple objects are present.
restrict	A list of length equal to the number of objects in .... Each entry of the list corresponds to one batch and specifies the cells to use when computing the correction.
k	An integer scalar specifying the number of nearest neighbors to consider when identifying mutual nearest neighbors.
sigma	A numeric scalar specifying the bandwidth of the Gaussian smoothing kernel used to compute the correction vector for each cell.
cos.norm.in	A logical scalar indicating whether cosine normalization should be performed on the input data prior to calculating distances between cells.
cos.norm.out	A logical scalar indicating whether cosine normalization should be performed prior to computing corrected expression values.
svd.dim	An integer scalar specifying the number of dimensions to use for summarizing biological substructure within each batch.
var.adj	A logical scalar indicating whether variance adjustment should be performed on the correction vectors.
subset.row	A vector specifying which features to use for correction.
correct.all	A logical scalar specifying whether correction should be applied to all genes, even if only a subset is used for the MNN calculations.
auto.order	Logical scalar indicating whether re-ordering of batches should be performed to maximize the number of MNN pairs at each step. Alternatively, an integer vector containing a permutation of `1:N` where `N` is the number of batches.
assay.type	A string or integer scalar specifying the assay containing the log-expression values, if SingleCellExperiment objects are present in ....
get.spikes	A logical scalar indicating whether to retain rows corresponding to spike-in transcripts. Only used for SingleCellExperiment inputs.
BSPARAM	A BiocSingularParam object specifying the SVD algorithm to use.
BNPARAM	A BiocNeighborParam object specifying the neighbor search algorithm to use.
BPPARAM	A BiocParallelParam object specifying the parallelization scheme to use.

Details

This function is designed for batch correction of single-cell RNA-seq data where the batches are partially confounded with biological conditions of interest. It does so by identifying pairs of mutual nearest neighbors (MNN) in the high-dimensional log-expression space. Each MNN pair represents cells in different batches that are of the same cell type/state, assuming that batch effects are mostly

orthogonal to the biological manifold. Correction vectors are calculated from the pairs of MNNs and corrected (log-)expression values are returned for use in clustering and dimensionality reduction.

The threshold to define nearest neighbors is defined by k, which is passed to findMutualNN to identify MNN pairs. The size of k can be interpreted as the minimum size of a subpopulation in each batch. Values that are too small will not yield enough MNN pairs, while values that are too large will ignore substructure within each batch. The algorithm is generally robust to various choices of k.

For each MNN pair, a pairwise correction vector is computed based on the difference in the log-expression profiles. The correction vector for each cell is computed by applying a Gaussian smoothing kernel with bandwidth sigma is the pairwise vectors. This stabilizes the vectors across many MNN pairs and extends the correction to those cells that do not have MNNs. The choice of sigma determines the extent of smoothing - a value of 0.1 is used by default, corresponding to 10% of the radius of the space after cosine normalization.

Value

A SingleCellExperiment object containing the corrected assay. This contains corrected (log-)expression values for each gene (row) in each cell (column) in each batch. A batch field is present in the column data, specifying the batch of origin for each cell.

Cells in the output object are always ordered in the same manner as supplied in .... For a single input object, cells will be reported in the same order as they are arranged in that object. In cases with multiple input objects, the cell identities are simply concatenated from successive objects, i.e., all cells from the first object (in their provided order), then all cells from the second object, and so on.

The metadata of the SingleCellExperiment contains:

- merge.order: a vector of batch names or indices, specifying the order in which batches were merged.
- merge.info, a DataFrame of information about each merge step (corresponding to each row). This contains pairs, a List of DataFrames specifying which pairs of cells in corrected were identified as MNNs at each step.

Choosing the gene set

All genes are used with the default setting of subset.row=NULL. Users can set subset.row to subset the inputs to highly variable genes or marker genes. This may provide more meaningful identification of MNN pairs by reducing the noise from irrelevant genes. Note that users should not be too restrictive with subsetting, as high dimensionality is required to satisfy the orthogonality assumption in MNN detection.

For SingleCellExperiment inputs, spike-in transcripts are automatically removed unless get.spikes=TRUE. If subset.row is specified and get.spikes=FALSE, only the non-spike-in specified features will be used. All SingleCellExperiment objects should have the same set of spike-in transcripts.

If subset.row is specified and correct.all=TRUE, corrected values are returned for *all* genes. This is possible as subset.row is only used to identify the MNN pairs and other cell-based distance calculations. Correction vectors between MNN pairs can then be computed in for all genes in the supplied matrices.

Expected type of input data

The input expression values should generally be log-transformed, e.g., log-counts, see normalize for details. They should also be normalized within each data set to remove cell-specific biases in

capture efficiency and sequencing depth. By default, a further cosine normalization step is performed on the supplied expression data to eliminate gross scaling differences between data sets.

- When `cos.norm.in=TRUE`, cosine normalization is performed on the matrix of expression values used to compute distances between cells. This can be turned off when there are no scaling differences between data sets.
- When `cos.norm.out=TRUE`, cosine normalization is performed on the matrix of values used to calculate correction vectors (and on which those vectors are applied). This can be turned off to obtain corrected values on the log-scale, similar to the input data.

The cosine normalization is achieved using the `cosineNorm` function.

Controlling the merge order

The order in which batches are corrected will affect the final results. The first batch in `auto.order` is used as the reference batch against which the second batch is corrected. Corrected values of the second batch are added to the reference batch, against which the third batch is corrected, and so on. This strategy maximizes the chance of detecting sufficient MNN pairs for stable calculation of correction vectors in subsequent batches.

If `auto.order=TRUE`, batches are ordered to maximize the number of MNN pairs at each step. The aim is to improve the stability of the correction by first merging more similar batches with more MNN pairs. This can be somewhat time-consuming as MNN pairs need to be iteratively recomputed for all possible batch pairings. It is often more convenient for the user to specify an appropriate ordering based on prior knowledge about the batches.

Note that, no matter what the setting of `auto.order` is, the order of cells in the output corrected matrix is *always* the same.

Further options

The function depends on a shared biological manifold, i.e., one or more cell types/states being present in multiple batches. If this is not true, MNNs may be incorrectly identified, resulting in over-correction and removal of interesting biology. Some protection can be provided by removing components of the correction vectors that are parallel to the biological subspaces in each batch. The biological subspace in each batch is identified with a SVD on the expression matrix to obtain `svd.dim` dimensions. (By default, this option is turned off by setting `svd.dim=0`.)

If `var.adj=TRUE`, the function will adjust the correction vector to equalize the variances of the two data sets along the batch effect vector. In particular, it avoids "kissing" effects whereby MNN pairs are identified between the surfaces of point clouds from different batches. Naive correction would then bring only the surfaces into contact, rather than fully merging the clouds together. The adjustment ensures that the cells from the two batches are properly intermingled after correction. This is done by identifying each cell's position on the correction vector, identifying corresponding quantiles between batches, and scaling the correction vector to ensure that the quantiles are matched after correction.

Using restriction

It is possible to compute the correction using only a subset of cells in each batch, and then extrapolate that correction to all other cells. This may be desirable in experimental designs where a control set of cells from the same source population were run on different batches. Any difference in the controls must be artificial in origin and can be directly removed without making further biological assumptions.

To do this, users should set `restrict` to specify the subset of cells in each batch to be used for correction. This should be set to a list of length equal to the length of `...`, where each element is a

subsetting vector to be applied to the columns of the corresponding batch. A NULL element indicates that all the cells from a batch should be used. In situations where one input object contains multiple batches, restrict is simply a list containing a single subsetting vector for that object.

mnnCorrect will only use the restricted subset of cells in each batch to identify MNN pairs (and to perform variance adjustment, if var.adj=TRUE). However, it will apply the correction to all cells in each batch - hence the extrapolation. This means that the output is always of the same dimensionality, regardless of whether restrict is specified.

Author(s)

Laleh Haghverdi, with modifications by Aaron Lun

References

Haghverdi L, Lun ATL, Morgan MD, Marioni JC (2018). Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nat. Biotechnol.* 36(5):421

See Also

fastMNN for a faster equivalent.

Examples

```
B1 <- matrix(rnorm(10000), ncol=50) # Batch 1
B2 <- matrix(rnorm(10000), ncol=50) # Batch 2
out <- mnnCorrect(B1, B2) # corrected values
```

---

multiBatchNorm *Per-batch scaling normalization*

Description

Perform scaling normalization within each batch to provide comparable results to the lowest-coverage batch.

Usage

```
multiBatchNorm(..., assay.type = "counts", norm.args = list(),
  min.mean = 1, subset.row = NULL, separate.spikes = FALSE)
```

Arguments

...	Two or more SingleCellExperiment objects containing counts and size factors. Each object is assumed to represent one batch.
assay.type	A string specifying which assay values contains the counts.
norm.args	A named list of further arguments to pass to normalize.
min.mean	A numeric scalar specifying the minimum (library size-adjusted) average count of genes to be used for normalization.
subset.row	A vector specifying which features to use for correction.
separate.spikes	Logical scalar indicating whether spike-in size factors should be rescaled separately from endogenous genes.

Details

When performing integrative analyses of multiple batches, it is often the case that different batches have large differences in coverage. This function removes systematic differences in coverage across batches to simplify downstream comparisons. It does so by resaling the size factors using median-based normalization on the ratio of the average counts between batches. This is roughly equivalent to the between-cluster normalization described by Lun et al. (2016).

This function will adjust the size factors so that counts in high-coverage batches are scaled *downwards* to match the coverage of the most shallow batch. The `normalize` function will then add the same pseudo-count to all batches before log-transformation. By scaling downwards, we favour stronger squeezing of log-fold changes from the pseudo-count, mitigating any technical differences in variance between batches. Of course, genuine biological differences will also be shrunk, but this is less of an issue for upregulated genes with large counts.

This function is preferred over running `normalize` directly when computing log-normalized values for use in `mnnCorrect` or `fastMNN`. In most cases, size factors will be computed within each batch; their direct application in `normalize` will not account for scaling differences between batches. In contrast, `multiBatchNorm` will rescale the size factors so that they are comparable across batches.

Only genes with library size-adjusted average counts greater than `min.mean` will be used for computing the rescaling factors. This improves precision and avoids problems with discreteness. Users can also set `subset.row` to restrict the set of genes used for computing the rescaling factors. However, this only affects the rescaling of the size factors - normalized values for *all* genes will still be returned.

Value

A list of SingleCellExperiment objects with normalized log-expression values in the `"logcounts"` assay (depending on values in `norm.args`).

Handling spike-ins

Spike-in transcripts should be either absent in all batches or, if present, they should be the same across all batches. Rows annotated as spike-in transcripts are not used to compute the rescaling factors for endogenous genes.

By default, the spike-in size factors are rescaled using the same scaling factor for the endogenous genes in the same batch. This preserves the abundances of the spike-in transcripts relative to the endogenous genes, which is important if the returned SingleCellExperiments are to be used to model technical noise.

If `separate.spikes=TRUE`, spike-in size factors are rescaled separately from those of the endogenous genes. This will eliminate differences in spike-in quantities across batches at the cost of losing the ability to compare between endogenous and spike-in transcripts within each batch.

Author(s)

Aaron Lun

References

Lun ATL, Bach K and Marioni JC (2016). Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biol.* 17:75

See Also

mnnCorrect and fastMNN for methods that can benefit from rescaling.

normalize for the calculation of log-transformed normalized expression values.

Examples

```
d1 <- matrix(rnbinom(50000, mu=10, size=1), ncol=100)
sce1 <- SingleCellExperiment(list(counts=d1))
sizeFactors(sce1) <- runif(ncol(d1))

d2 <- matrix(rnbinom(20000, mu=50, size=1), ncol=40)
sce2 <- SingleCellExperiment(list(counts=d2))
sizeFactors(sce2) <- runif(ncol(d2))

out <- multiBatchNorm(sce1, sce2)
summary(sizeFactors(out[[1]]))
summary(sizeFactors(out[[2]]))
```

---

multiBatchPCA *Multi-batch PCA*

Description

Perform a principal components analysis across multiple gene expression matrices to project all cells to a common low-dimensional space.

Usage

```
multiBatchPCA(..., batch = NULL, d = 50, subset.row = NULL,
  get.all.genes = FALSE, rotate.all = FALSE, get.variance = FALSE,
  preserve.single = FALSE, assay.type = "logcounts",
  get.spikes = FALSE, BSPARAM = ExactParam(),
  BPPARAM = SerialParam())
```

Arguments

...	Two or more matrices containing expression values (usually log-normalized). Each matrix is assumed to represent one batch. Alternatively, two or more SingleCellExperiment objects containing these matrices. Alternatively, one matrix or SingleCellExperiment can be supplied containing cells from all batches. This requires batch to also be specified.
batch	A factor specifying the batch identity of each cell in the input data. Ignored if ... contains more than one argument.
d	An integer scalar specifying the number of dimensions to keep from the initial multi-sample PCA.
subset.row	A vector specifying which features to use for correction.
get.all.genes	A logical scalar indicating whether the reported rotation vectors should include genes that are excluded by a non-NULL value of subset.row.

rotate.all	A deprecated synonym for get.all.genes.
get.variance	A logical scalar indicating whether to return the (weighted) variance explained by each PC.
preserve.single	A logical scalar indicating whether to combine the results into a single matrix if only one object was supplied in ....
assay.type	A string or integer scalar specifying the assay containing the expression values, if SingleCellExperiment objects are present in ....
get.spikes	A logical scalar indicating whether to retain rows corresponding to spike-in transcripts. Only used for SingleCellExperiment inputs.
BSPARAM	A BiocSingularParam object specifying the algorithm to use for PCA, see runSVD for details.
BPPARAM	A BiocParallelParam object specifying whether the SVD should be parallelized.

Details

This function is roughly equivalent to cbinding all matrices in ... and performing PCA on the merged matrix. The main difference is that each sample is forced to contribute equally to the identification of the rotation vectors. Specifically, the mean vector used for centering is defined as the grand mean of the mean vectors within each batch. Each batch's contribution to the gene-gene covariance matrix is also divided by the number of cells in that batch.

Our approach is to effectively weight the cells in each batch to mimic the situation where all batches have the same number of cells. This ensures that the low-dimensional space can distinguish subpopulations in smaller batches. Otherwise, batches with a large number of cells would dominate the PCA, i.e., the definition of the mean vector and covariance matrix. This may reduce resolution of unique subpopulations in smaller batches that differ in a different dimension to the subspace of the larger batches.

If ... contains SingleCellExperiment objects, any spike-in transcripts should be the same across all batches. These will be removed prior to PCA unless get.spikes=TRUE. If subset.row is specified and get.spikes=FALSE, only the non-spike-in specified features will be used.

Setting get.all.genes=TRUE will report rotation vectors that span all genes, even when only a subset of genes are used for the PCA. This is done by projecting all non-used genes into the low-dimensional "cell space" defined by the first d components.

If BSPARAM is defined with deferred=TRUE, the per-gene centering and per-cell scaling will be manually deferred during matrix multiplication. This can greatly improve speeds when the input matrices are sparse, as deferred operations avoids loss of sparsity (at the cost of numerical precision).

Value

A List of numeric matrices is returned where each matrix corresponds to a batch and contains the first d PCs (columns) for all cells in the batch (rows).

If preserve.single=TRUE and ... contains a single object, the List will only contain a single matrix. This contains the first d PCs (columns) for all cells in the same order as supplied in the single input object.

The metadata contains rotation, a matrix of rotation vectors, which can be used to construct a low-rank approximation of the input matrices. This has number of rows equal to the number of genes after any subsetting, except if rotate.all=TRUE, where the number of rows is equal to the genes before subsetting.

If `get.variance=TRUE`, the metadata will also contain `var.explained`, the weighted variance explained by each PC; and `var.total`, the total variance after weighting.

Author(s)

Aaron Lun

See Also

`runSVD`

Examples

```
d1 <- matrix(rnorm(5000), ncol=100)
d1[1:10,1:10] <- d1[1:10,1:10] + 2 # unique population in d1
d2 <- matrix(rnorm(2000), ncol=40)
d2[11:20,1:10] <- d2[11:20,1:10] + 2 # unique population in d2

out <- multiBatchPCA(d1, d2)

xlim <- range(c(out[[1]][,1], out[[2]][,1]))
ylim <- range(c(out[[1]][,2], out[[2]][,2]))
plot(out[[1]][,1], out[[1]][,2], col="red", xlim=xlim, ylim=ylim)
points(out[[2]][,1], out[[2]][,2], col="blue")
```

---

rescaleBatches *Scale counts across batches*

Description

Scale counts so that the average count within each batch is the same for each gene.

Usage

```
rescaleBatches(..., batch = NULL, restrict = NULL, log.base = 2,
  pseudo.count = 1, subset.row = NULL, assay.type = "logcounts",
  get.spikes = FALSE)
```

Arguments

`...`	Two or more log-expression matrices where genes correspond to rows and cells correspond to columns. Each matrix should contain cells from the same batch; multiple matrices represent separate batches of cells. Each matrix should contain the same number of rows, corresponding to the same genes (in the same order). Alternatively, one or more SingleCellExperiment objects can be supplied containing a count matrix in the `assay.type` assay. Note the same restrictions described above for matrix inputs.
`batch`	A factor specifying the batch of origin for all cells when only a single object is supplied in `...`. This is ignored if multiple objects are present.

`restrict`	A list of length equal to the number of objects in `...`. Each entry of the list corresponds to one batch and specifies the cells to use when computing the correction.
`log.base`	A numeric scalar specifying the base of the log-transformation.
`pseudo.count`	A numeric scalar specifying the pseudo-count used for the log-transformation.
`subset.row`	A vector specifying which features to use for correction.
`assay.type`	A string or integer scalar specifying the assay containing the log-expression values, if SingleCellExperiment objects are present in `...`.
`get.spikes`	A logical scalar indicating whether to retain rows corresponding to spike-in transcripts. Only used for SingleCellExperiment inputs.

Details

This function assumes that the log-expression values were computed by a log-transformation of normalized count data, plus a pseudo-count. It reverses the log-transformation and scales the underlying counts in each batch so that the average (normalized) count is equal across batches. The assumption here is that each batch contains the same population composition. Thus, any scaling difference between batches is technical and must be removed.

This function is equivalent to centering in log-expression space, the simplest application of linear regression methods for batch correction. However, by scaling the raw counts, it avoids loss of sparsity that would otherwise result from centering. It also mitigates issues with artificial differences in variance due to log-transformation.

The output values are always re-log-transformed with the same `log.base` and `pseudo.count`. These can be used directly in place of the input values for downstream operations.

Value

A SingleCellExperiment object containing the `corrected` assay. This contains corrected log-expression values for each gene (row) in each cell (column) in each batch. A `batch` field is present in the column data, specifying the batch of origin for each cell.

Cells in the output object are always ordered in the same manner as supplied in `...`. For a single input object, cells will be reported in the same order as they are arranged in that object. In cases with multiple input objects, the cell identities are simply concatenated from successive objects, i.e., all cells from the first object (in their provided order), then all cells from the second object, and so on.

Choice of genes

All genes are used with the default setting of `subset.row=NULL`. Users can set `subset.row` to subset the inputs, though this is purely for convenience as each gene is processed independently of other genes.

For SingleCellExperiment inputs, spike-in transcripts are automatically removed unless `get.spikes=TRUE`. If `subset.row` is specified and `get.spikes=FALSE`, only the non-spike-in specified features will be used. All SingleCellExperiment objects should have the same set of spike-in transcripts.

Using restriction

It is possible to compute the correction using only a subset of cells in each batch, and then extrapolate that correction to all other cells. This may be desirable in experimental designs where a control set of cells from the same source population were run on different batches. Any difference in the

controls must be artificial in origin and can be directly removed without making further biological assumptions.

To do this, users should set `restrict` to specify the subset of cells in each batch to be used for correction. This should be set to a list of length equal to the length of `...`, where each element is a subsetting vector to be applied to the columns of the corresponding batch. A `NULL` element indicates that all the cells from a batch should be used. In situations where one input object contains multiple batches, `restrict` is simply a list containing a single subsetting vector for that object.

The function will compute the scaling differences using only the specified subset of cells. However, the re-scaling will then be applied to all cells in each batch - hence the extrapolation. This means that the output is always of the same dimensionality, regardless of whether `restrict` is specified.

Author(s)

Aaron Lun

Examples

```
means <- 2^rgamma(1000, 2, 1)
A1 <- matrix(rpois(10000, lambda=means), ncol=50) # Batch 1
A2 <- matrix(rpois(10000, lambda=means*runif(1000, 0, 2)), ncol=50) # Batch 2

B1 <- log2(A1 + 1)
B2 <- log2(A2 + 1)
out <- rescaleBatches(B1, B2)
```

Index