batchelor: a Bioconductor package for batch correction in single-cell data.

Design principles:

Signatures:

Each method should have the following signature:

FUN(..., batch=NULL)

The first set allows for individual batch-specific objects to be passed into the function, while the second set allows for a single object containing all of the batches to be passed. One or the other of these may be more convenient depending on the circumstances.

All original expression inputs are assumed to have genes as rows and cells as columns, following existing convention in the single-cell ‘omics field.

Corrected expression values:

Some batch correction methods will report corred values for all genes and all cells. Such functions should aim to return a SummarizedExperiment where the first assay is the matrix of corrected values (called *corrected*). The first field of the column metadata should be *batch*, specifying the batch of origin for each cell. The order of cells in the output should always be the same as the input order of cells within and across all supplied objects in “…”.

With dimensionality reduction:

If dimensionality reduction is performed internally as part of the method, we expect further arguments to indicate whether the inputs are low-dimensional or not. This allows the user to avoid a redundant dimensionality reduction step if, e.g., PCs are already supplied.

FUN(..., batch=NULL, pc.input=FALSE, use.dimred=NULL) # for matrix inputs

The *pc.input* argument indicates whether non-SingleCellExperiment objects in “...” are low-dimensional. The *use.dimred* argument only applies when SingleCellExperiment objects are in “...”, and indicates whether to retrieve low-dimensional data from the reducedDims slots instead of the using the gene expression assays for correction.

All reduced-dimension inputs are assumed to have cells as rows and dimensions as columns. This follows existing convention by considering reduced dimensions as “metadata” of sorts.

All reduced dimension methods should strive to return a DataFrame with *corrected*, a matrix of the corrected coordinates with one cell in each row; and *batch*, a vector specifying the batch of

origin for each cell. The order of cells should reflect their input order, under the principle of least surprise. Any other method-specific fields can be stored in the metadata.

Core procedures:

- Use *BiocNeighbors* for nearest neighbour searching.
- Use *BiocSingular* for PCA or SVD.
- Use *BiocParallel* for any type of parallelization.

If your algorithm of choice is not present in either of these packages, consider making a PR there (where it will be generally useful to others) rather than re-implementing things here.

Contribution guidelines:

Coding style:

- Use four space indents.
- Use Roxygen documentation.
    - Every function should @importFrom its necessary methods, even if it is already imported elsewhere.
    - Each sentence should be a line.
- Keep functions small for easier testing.
- Ignore the 80 character line space limit, which is stupid IMO.
- No tidyverse structures or grammar.
- No plotting code of any sort. These belong elsewhere, e.g., *scater*.

The above comments pertain to R code; for those planning to write C++, please contact me directly as this will require more careful inspection and coordination.

Minimum requirements:

If contributing a new method, there is a minimum of three components:

- Code itself, obviously. For each new method, this should be present in a new R source file, with the user-visible signatures defined above.
- Documentation; follow the conventions in the existing docs.
- Tests. These should test against a reference truth, and should not simply run the function without any quantitative check on the correctness of the output.

Questions:

- **Can anyone contribute a batch correction method to this package?** Yes. Pending review of the code, and obviously it would be helpful if there was a manuscript somewhere that could be read.

- **Is the goal of this package that all bioc scrna-seq batch correction methods can be put here?** Not necessarily in the package itself, but wrapped in a common interface so that switching between methods is as easy as changing an argument.
    - **Oooooh got it! So this should be more of a wrapper.** Depends on whether the authors of a particular method can be bothered to write their own package, or if they want to put it into this package. My sole motivation is to move the MNN stuff out of *scran* because the package is getting too bloated.
- **Btw, i'm happy to talk on slack if it's more useful :)** Yes, probably**.**