Package 'SuperCellCyto'

November 12, 2026

Title SuperCell For Cytometry Data

Version 1.2.0

Description SuperCellCyto provides the ability to summarise cytometry data into supercells by merging together cells that are similar in their marker expressions using the SuperCell package.

biocViews CellBiology, FlowCytometry, Software, SingleCell

License GPL-3 + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports SuperCell, data.table, Matrix, BiocParallel

Suggests flowCore, knitr, rmarkdown, usethis, testthat (>= 3.0.0), BiocSingular, bluster, scater, scran, Seurat, SingleCellExperiment, BiocStyle, magick, qs2

VignetteBuilder knitr

Config/testthat/edition 3

URL https://phipsonlab.github.io/SuperCellCyto/

BugReports https://github.com/phipsonlab/SuperCellCyto/issues

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

git_branch RELEASE_3_23

git_last_commit ad1bb6a

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Givanna Putri [aut, cre] (ORCID: <https://orcid.org/0000-0002-7399-8014>), George Howitt [aut], Felix Marsh-Wakefield [aut], Thomas Ashhurst [aut], Belinda Phipson [aut]

Maintainer Givanna Putri <givanna.h@gmail.com>

Contents

recomputeSupercells 2
runSuperCellCyto 3
simCytoData 7

Index **8**

`recomputeSupercells` *Recompute supercells*

Description

Given a supercell object, recreate the supercells using a different gamma value.

Gamma value controls the number of supercells generated. The smaller the value, the more supercells you get, and vice versa.

For this function to run, you need to have at least run `runSuperCellCyto()` function **once**!

Usage

```
recomputeSupercells(
  dt,
  sc_objects,
  markers,
  sample_colname,
  cell_id_colname,
  aggregation_method = c("mean", "median"),
  gam = 20
)
```

Arguments

`dt`	A **data.table** object containing cytometry data where rows represent cells and columns represent markers.
`sc_objects`	The `supercell_object` returned by `runSuperCellCyto()` function.
`markers`	A character vector identifying the markers to create supercells with.
`sample_colname`	A character string identifying the column in `dt` that denotes the sample of a cell.
`cell_id_colname`	A character string identifying the column in `dt` representing each cell's unique ID.
`aggregation_method`	A character string indicating how to aggregate the cells in supercells to get expression matrix. Options are "mean" or "median". Defaults to "mean". If "mean", the mean of the marker expressions across all cells within each individual supercell is computed. If "median", the median of the marker expressions across all cells within each individual supercell is computed.
`gam`	A numeric value specifying the gamma value which regulates the number of supercells generated. Defaults to 20.

Value

A list with the following components:

- `supercell_expression_matrix`: A **data.table** object that contains the marker expression for each supercell. These marker expressions are computed by calculating the mean of the marker expressions across all cells within each individual supercell.
- `supercell_cell_map`: A **data.table** that maps each cell to its corresponding supercell. This table is essential for identifying the specific supercell each cell has been allocated to. It proves particularly useful for analyses that require one to expand the supercells to the individual cell level.

Author(s)

Givanna Putri

Examples

```
set.seed(42)
cyto_dat <- simCytoData(10, rep(1000, 3))
markers <- paste0("Marker_", seq_len(10))
out_gam20 <- runSuperCellCyto(
    dt = cyto_dat,
    markers = markers,
    sample_colname = "Sample",
    cell_id_colname = "Cell_Id",
    gam = 20
)
recomputed_sc <- recomputeSupercells(
    dt = cyto_dat,
    sc_objects = out_gam20$supercell_object,
    markers = markers,
    sample_colname = "Sample",
    cell_id_colname = "Cell_Id",
    gam = 50
)
```

---

runSuperCellCyto *Run SuperCellCyto on Cytometry Data*

Description

This function creates supercells for a cytometry data formatted as a **data.table** object using the SuperCellCyto algorithm.

Please make sure you read additional details below to better understand what the function does and how it works.

Usage

```
runSuperCellCyto(
  dt,
  markers,
  sample_colname,
  cell_id_colname,
  n_pc = 10,
  aggregation_method = c("mean", "median"),
  gam = 20,
  k_knn = 5,
  BPPARAM = SerialParam(),
  load_balancing = FALSE
)
```

Arguments

dt	A **data.table** object containing cytometry data where rows represent cells and columns represent markers. If this is a `data.frame` object, the function will try to convert it to a **data.table** object. A warning message will be displayed when this happens. Otherwise, it will terminate.
markers	A character vector identifying the markers to create supercells with.
sample_colname	A character string identifying the column in `dt` that denotes the sample of a cell.
cell_id_colname	A character string identifying the column in `dt` representing each cell's unique ID.
n_pc	A numeric value specifying the number of principal components (PCs) to compute from the marker expression matrix. Defaults to 10. If there are less than 10 markers in the `markers` parameter, then the number of PCs is set to however many markers there are in the `markers` parameter.
aggregation_method	A character string specifying the method to be used for calculating the marker expression of the supercells. Accepted values are "mean" and "median". Based on the choice, the supercells' marker expression are computed by computing either the mean or median of the marker expression of the cells therein. The default value is "mean". If any other value is provided, the function will return an error.
gam	A numeric value specifying the gamma value which regulates the number of supercells generated. Defaults to 20.
k_knn	A numeric value specifying the k value (number of neighbours) used to build the kNN network. Defaults to 5.
BPPARAM	A BiocParallelParam-class object specifying the parallel processing settings. Defaults to SerialParam-class, meaning the samples will be processed sequentially one after the other. Refer to additional details section below on parallel processing for more details.
load_balancing	A logical value indicating whether to use a custom load balancing scheme when processing multiple samples in parallel. Defaults to FALSE. Refer to additional details section below on parallel processing for more details.

Value

A list with the following components:

- `supercell_object`: A list containing the object returned by SCimplify function. One object per sample. This object is critical for recomputing supercells in the future. Hence do not discard it.
- `supercell_expression_matrix`: A **data.table** object that contains the marker expression for each supercell. These marker expressions are computed by calculating the mean of the marker expressions across all cells within each individual supercell.
- `supercell_cell_map`: A **data.table** that maps each cell to its corresponding supercell. This table is essential for identifying the specific supercell each cell has been allocated to. It proves particularly useful for analyses that require one to expand the supercells to the individual cell level.

Parallel Processing

SuperCellCyto can process multiple samples simultaneously in parallel. This can drastically bring down processing time for dataset with a large number of samples. To enable this feature, set the `BPPARAM` parameter to either a MulticoreParam-class object or a SnowParam-class object. Importantly, it is also recommended to set the number of tasks (i.e., the `task` parameter in either MulticoreParam-class or SnowParam-class object) to **the number of samples in the dataset**.

Furthermore, we also recommend setting `load_balancing` parameter to TRUE. This ensures optimal distribution of samples across multiple cores, and is particularly important if your samples are of varying sizes (number of cells).

Cell ID and Sample Definitions

The `cell_id_colname` parameter specifies the column in `dt` that denotes the unique identifier for each cell. It is perfectly normal to not have this column in your dataset by default. The good news is that **it is trivial to create one**. You can create a new vector containing a sequence of numbers from 1 to however many cells you have, and append this vector as a new column in your dataset. Refer to our vignette on how to do this.

The `sample_colname` parameter specifies the column in `dt` that denotes the sample a cell came from. By default, SuperCellCyto creates supercells for each sample independent of other samples. This ensures each supercell to only contain cells from **exactly** one sample.

What constitute a sample? For most purposes, a sample represents a biological sample in your experiment. You may be thinking, is it then possible to use this in a different context, say creating supercells for each population or cluster rather than a biological sample? The short answer is yes, and we address this in our vignette.

Computing PCA

The function will start by computing PCA from all the markers specified in `markers` parameter. By default, the number of PCs calculated is set to 10. If there is less than 10 markers in the `markers` parameter, then the number of PCs is set to however many markers there are in the `markers` parameter.

Notably, *no* scaling or transformation were done on the markers' expressions prior to computing the PCs.

The function does not use `irlba` to calculate PCA. There is very little gain to use it for cytometry data because of the relatively tiny number of features (markers) in the data.

Setting Supercell's Resolution

The `gam` parameter influences the number of supercells created per sample. A lower `gam` value results in more, and thus generally smaller supercells, and vice versa.

To estimate how many supercells we will get for our dataset, it is important to understand how the `gam` value is interpreted in the context of number of cells in a sample.

`gam=n_cells/n_supercells` where `n_cells` denotes the number of cells and `n_supercells` denotes the number of supercells to be created.

By resolving the formula above, we can roughly estimate how many supercells we will get per sample. For example, say we have 2 samples, sample A and B. Sample A has 10,000 cells, while sample B has 5,000 cells:

- If `gam` is set to 10, we will end up with 1,000 supercells for sample A and 500 supercells for sample B, a total of 1,500 supercells.
- If `gam` is set to 50, we will end up with 200 supercells for sample A and 100 supercells for sample B, a total of 300 supercells.

Importantly, one cannot expect all the supercells to be of the same size. Some will capture more/less cells than others. It is not trivial to estimate how many cells will be captured in each supercell beforehand.

Computing kNN network

To create supercells, a kNN (k-Nearest Neighbors) network is constructed based on the `k_knn` parameter which dictates the number of neighbours (for each cell) used to create the network. An actual (not approximate) kNN network is created.

A walktrap algorithm then uses this network to group cells into supercells.

Author(s)

Givanna Putri

Examples

```
# Simulate some data
set.seed(42)
cyto_dat <- simCytoData(nmarkers = 10, ncells = rep(2000,2))

# Setup the columns designating the markers, samples, and cell IDs
marker_col <- paste0("Marker_", seq_len(10))
sample_col <- "Sample"
cell_id_col <- "Cell_Id"

supercell_dat <- runSuperCellCyto(
```

```
    cyto_dat, marker_col,
    sample_col, cell_id_col
)
```

`simCytoData` *Simulate cytometry data*

Description

Simulate some cytometry data for use in testing or documenting functions which require some cytometry data. Please run `set.seed` before running the function if you want to ensure reproducibility.

Usage

```
simCytoData(nmarkers = 10, ncells = rep(10000, 2))
```

Arguments

`nmarkers`	A numeric value specifying number of markers to simulate.
`ncells`	A numeric vector specifying the number of cells to simulate per sample. 1 vector element per sample.

Value

A **data.table** object containing the simulated cytometry data where rows represent cells and columns represent markers.

Author(s)

Givanna Putri

Examples

```
set.seed(42)
cyto_dat <- simCytoData()
head(cyto_dat)
dim(cyto_dat)
```

Index

BiocParallelParam-class, *4*

MulticoreParam-class, *5*

recomputeSupercells, 2
runSuperCellCyto, 3
runSuperCellCyto(), *2*

SerialParam-class, *4*
simCytoData, 7
SnowParam-class, *5*