Package 'staRgate'

April 22, 2026

Title Automated gating pipeline for flow cytometry analysis to characterize the lineage, differentiation, and functional states of T-cells

Version 0.99.8

Description An R-based automated gating pipeline for flow cytometry data designed to mimic the manual gating strategy of defining flow biomarker positive populations relative to a unimodal background population to include cells with varying intensities of marker expression. The pipeline's main feature is a flexible density-based gating strategy capable of capturing varying scenarios based on marker expression patterns to analyze a 29-marker flow panel that characterizes T-cell lineage, differentiation, and functional states.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews FlowCytometry, Preprocessing, ImmunoOncology

Depends R (>= 4.3.0)

Imports dplyr, janitor, purrr, rlang, stringr, tidyr, flowCore, flowWorkspace, glue, tibble

Suggests flowAI, ggplot2, gt, knitr, openCyto, ggcyto, rmarkdown, data.table, here, testthat (>= 3.0.0), BiocStyle

VignetteBuilder knitr

BugReports https://github.com/leejasme/staRgate/issues

URL https://bioconductor.org/packages/staRgate, https://leejasme.github.io/staRgate

Config/testthat/edition 3

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

git_branch devel

git_last_commit f007905

git_last_commit_date 2026-04-17

Repository Bioconductor 3.24

Date/Publication 2026-04-21

Author Jasme Lee [aut, cre] (ORCID: <https://orcid.org/0009-0006-4492-4872>),
Matthew Adamow [aut],
Colleen Maher [aut],
Xiyu Peng [aut],
Phillip Wong [aut],
Fiona Ehrich [aut],
Michael A Postow [aut],
Margaret K Callahan [aut],
Ronglai Shen [aut],
Katherine S Panageas [aut],
V foundation [fnd],
MSK-MIND [fnd],
NIH R01CA276286 [fnd],
NIH P30CA008748 [fnd]

Maintainer Jasme Lee <leej22@mskcc.org>

Contents

`staRgate-package` *staRgate: Automated gating pipeline for flow cytometry analysis to characterize the lineage, differentiation, and functional states of T-cells*

Description

An R-based automated gating pipeline for flow cytometry data designed to mimic the manual gating strategy of defining flow biomarker positive populations relative to a unimodal background population to include cells with varying intensities of marker expression. The pipeline's main feature is a flexible density-based gating strategy capable of capturing varying scenarios based on marker expression patterns to analyze a 29-marker flow panel that characterizes T-cell lineage, differentiation, and functional states.

Author(s)

Maintainer: Jasme Lee <leej22@mskcc.org> (ORCID)

Authors:

- Matthew Adamow
- Colleen Maher

- Xiyu Peng
- Phillip Wong
- Fiona Ehrich
- Michael A Postow
- Margaret K Callahan
- Ronglai Shen
- Katherine S Panageas

Other contributors:

- V foundation [funder]
- MSK-MIND [funder]
- NIH R01CA276286 [funder]
- NIH P30CA008748 [funder]

See Also

Useful links:

- https://bioconductor.org/packages/staRgate
- https://leejasme.github.io/staRgate
- Report bugs at https://github.com/leejasme/staRgate/issues

`getBiexpTransformGS` *Applies Biexpeonential Transformation using specifications in csv file provided at* `path_biexp_params`

Description

The csv file at `path_biexp_params` should specify the channels to apply the transformation to and the parameters (negative decades, width basis and positive decades). The default is negative decades=0.5, width basis=-30 and positive decades=4.5. The Transformation can be applied to only a subset of the channels included in the GatingSet.

Usage

```
getBiexpTransformGS(gs, path_biexp_params)
```

Arguments

`gs`	GatingSet to apply Biexponential Transformation to
`path_biexp_params`	file path for .csv file that specifies the Biexponential Transformation

Details

An example table is provided in the extdata/biexp_transf_parameters_x50.csv

Value

GatingSet with Biexponentially Transformed data

Examples

```
# This example does not contain all the pre-processing steps required in
# getting the GatingSet (gs) ready for Biexp transformation.
# To see the steps that are required to creating the (gs),
# please see the vignette for a full tutorial

# To make this a runnable example, read in the FCS file to create gs and
# directly apply

# File path to the FCS file
path_fcs <- system.file("extdata",
                        "example_fcs.fcs",
                        package="staRgate",
                        mustWork=TRUE)
path_biexp_params <- system.file("extdata",
                                 "biexp_transf_parameters_x50.csv",
                                 package="staRgate",
                                 mustWork=TRUE)

# Create a cytoset then convert to gs
cs <- flowWorkspace::load_cytoset_from_fcs(path_fcs)
gs <- flowWorkspace::GatingSet(cs)

# gs must be a GatingSet object
gs <- getBiexpTransformGS(gs, path_biexp_params=path_biexp_params)

# To check the transformation parameters applied
flowWorkspace::gh_get_transformations(gs)
```

---

getCompGS *Applies Compensation using specifications in csv file provided at* path_comp_mat

---

Description

The csv file at path_comp_mat should specify the channels to apply the compensation to. The format is a matrix where the col and row names correspond to the channel names

Usage

```
getCompGS(gs, path_comp_mat)
```

Arguments

gs	GatingSet to apply Biexponential Transformation to
path_comp_mat	file path for .csv file that specifies the Compensation Matrix

Details

An example matrix is provided in the extdata/comp_mat_example_fcs.csv

Value

GatingSet with compensated data

Examples

```
# This example does not contain all the pre-processing steps required in
# getting the GatingSet (gs) ready for compensation step
# To see the steps that are required to creating the (gs),
# please see the vignette for a full tutorial

# To make this a runnable example, read in the FCS file to create gs and
# directly apply

# File path to the FCS file
path_fcs <- system.file("extdata",
                        "example_fcs.fcs",
                        package="staRgate",
                        mustWork=TRUE)

path_biexp_params <- system.file("extdata",
                                 "biexp_transf_parameters_x50.csv",
                                 package="staRgate",
                                 mustWork=TRUE)

# Create a cytoset then convert to gs
cs <- flowWorkspace::load_cytoset_from_fcs(path_fcs)
gs <- flowWorkspace::GatingSet(cs)

path_comp_mat <- system.file("extdata", "comp_mat_example_fcs.csv",
                             package="staRgate", mustWork=TRUE)

# gs is a GatingSet object
gs <- getCompGS(gs, path_comp_mat=path_comp_mat)

# Checks the comp mat was successfully applied
flowWorkspace::gh_get_compensations(gs)
```

---

getDensityDerivs *Internal function: Estimate derivatives for density of* `marker` *for each unique subset of* `subset_col`

Description

Internal function for `get_density_gates` For each unique value in `subset_col`, estimate the derivatives for `marker` (intensity values)

Usage

```
getDensityDerivs(dens)
```

Arguments

`dens`	`density` object from the density

Value

list of dataframe with density estimation and corresponding 1st-4th derivatives, indicators of local peaks, plateau_pre
each element corresponds to each unique value of `subset_col`
for each dataframe: rows correspond to each of the bins

`getDensityGates` *Density gating of intensity values in* `marker` *for each unique subset of* `subset_col`

Description

For each unique value in `subset_col`, gate using density and estimated derivatives to identify cutoff at shoulder (i.e., point of tapering off) relative to the peak for `marker` (intensity values). The strategy of cutting at the shoulder mimics the strategy to gate relative to a unimodal background negative subpopulation, which is capable of capturing dim subpopulations.

Usage

```
getDensityGates(
  intens_dat,
  marker,
  subset_col,
  bin_n = 512,
  peak_detect_ratio = 10,
  pos_peak_threshold = 1800,
  neg_intensity_threshold = -1000
)
```

Arguments

`intens_dat`	dataframe of pre-gated (compensated, biexp. transf, openCyto steps) intensity values where cols=intensity value per marker, rows=each sample
`marker`	string for the marker(s) to gate on the names need to match exactly the column name in `intens_dat`
`subset_col`	string for the column name to indicate the subsets to apply density gating on will perform operation on subsets corresponding to each unique value in column
`bin_n`	numeric to be passed to `n` parameter of `density(n=bin_n)` for number of equally spaced points at which the density is to be estimated Default is 512, which is the default of `density(n=512)`
`peak_detect_ratio`	numeric threshold for eliminating small peaks where a peak that is < than the highest peak by `peak_detect_ratio` times will be ignored Default=10
`pos_peak_threshold`	either: • numeric for threshold to identify a positive peak for all or • a dataframe if supplying multiple `marker` to gate. The dataframe needs to be supplied with 2 columns named `marker` and `pos_peak_threshold` and rows for the `marker` to gate

Default is 1800 (note this is on the biexponential scale) for all `marker`

`neg_intensity_threshold`
: numeric for threshold to filter out any "very negatively" expressed cells in the density estimation to avoid over-compression and difficulty in distinguishing peaks and the gates
This is only applied as a filter for the density estimation, the cells < `neg_intensity_threshold` are retained in the intensity matrix for other steps
Expects the `neg_intensity_threshold` is on the same scale as the transformed data in `intens_dat`
Default is `NULL`: no filters applied and density estimation based on all cells in corresponding subsets.
Suggested for biexp. transformed data is -1000 which corresponds to ~-3300 on the original intensity scale)

Value

tibble of gates/cutoffs for `marker` for each unique subset found in `subset_col` where

- rows correspond to unique values in `subset_col`
- , columns correspond to`marker`

Examples

```
# Create a fake dataset
set.seed(100)
intens_dat<-tibble::tibble(
               CD3_pos=rep(c(0, 1), each=50),
               CD4=rnorm(100, 100, 10),
               CD8=rnorm(100, 100, 10)
)

# Run density gating, leaving other params at suggested defaults
# number of bins suggested is 40 but default is at `bin_n=512`,
# which is the default for the R base density() function
getDensityGates(intens_dat, marker="CD4", subset_col="CD3_pos", bin_n=40)
```

---

getDensityMats *Internal function: Matrix of calculations for density gating of intensity values in* `marker` *for each unique subset of* `subset_col`

Description

Internal function for `getDensityGates` For each unique value in `subset_col`, there is a matrix for storing calculations for density gating contains: first to fourth derivatives of density, indicators for local peaks, "real peaks", plateau_pre and cutoff

Usage

```
getDensityMats(
  intens_dat,
  marker,
  subset_col,
```

```
  bin_n = 512,
  peak_detect_ratio = 10,
  pos_peak_threshold = 1800
)
```

Arguments

intens_dat	dataframe of pre-gated (compensated, biexp. transf, gated CD4/CD8) intensity values where cols = intensity value per marker, rows = each sample
marker	string for the marker to gate on the name needs to match exactly the column name in intens_dat
subset_col	string for the column name to indicate the subsets to apply density gating on will perform operation on subsets corresponding to each unique value in column
bin_n	numeric to be passed to n parameter of density(n = bin_n) for number of equally spaced points at which the density is to be estimated default is 512, which is the default of density(n = 512)
peak_detect_ratio	numeric threshold for eliminating small peaks where a peak that is < than the highest peak by peak_detect_ratio times will be ignored default = 10
pos_peak_threshold	numeric for threshold to identify a positive peak ' default is 1800, which is on the biexponential scale

Value

tibble of matrices for marker containing calculations for density gating for each unique subset found in subset_col
rows correspond to unique values in subset_col,
cols correspond to the information for density gating

---

getDensityPeakCutoff *Internal function: Determine the "real peaks" and cutoff based on the density estimation and its derivs*

Description

Internal function for getDensityGates

Usage

```
getDensityPeakCutoff(
  dens_binned_dat,
  marker,
  subset_col,
  bin_n = 512,
  peak_detect_ratio = 10,
  pos_peak_threshold = 1800,
  dens_flip = FALSE
)
```

Arguments

dens_binned_dat	list of dataframe output from the getDensityDerivs
marker	string for the marker to gate on the name needs to match exactly the column name in dens_binned_dat
subset_col	string for the column name to indicate the subsets to apply density gating on will perform operation on subsets corresponding to each unique value in column
bin_n	numeric to be passed to n parameter of density(n = bin_n) for number of equally spaced points at which the density is to be estimated default is 512, which is the default of density(n = 512)
peak_detect_ratio	numeric threshold for eliminating small peaks where a peak that is < than the highest peak by peak_detect_ratio times will be ignored default = 10
pos_peak_threshold	numeric for threshold to identify a positive peak default is 1800, which is on the biexponential scale
dens_flip	logical for whether the gating should be applied "backwards" where the peak is a positive peak and want to gate to the left of peak instead of right

Value

list of dataframe dens_binned_dat with additional columns added for peak(s) identified and the cutoff each element corresponds to each unique value of subset_col for each dataframe: rows correspond to each of the bins

---

getGatedDat *Attach indicator columns to* intens_dat *based on gates provided in* cutoffs

---

Description

Adds an indicator column (0/1) to intens_dat for each marker in cutoffs as indicated by the columns in cutoffs

Usage

```
getGatedDat(intens_dat, cutoffs, subset_col)
```

Arguments

intens_dat	dataframe of pre-gated (compensated, biexp. transf, openCyto steps) intensity values where rows=each cell and cols are the intensity values for each marker
cutoffs	tibble of gates/cutoffs for all markers to gate Expects cutoffs to match format of output from getDensityGates() with column corresponding to a marker, and rows to the subsets defined in the subset_col
subset_col	string for the column name to indicate the subsets to apply density gating on will perform operation on subsets corresponding to each unique value in column

Details

The naming convention for the tagged on indicator columns will be `tolower(<marker_name>_pos)` where 0 indicates negativity or intensity < gate provided 1 indicates positivity or intensity > gate provided

Value

`intens_dat` with additional columns attached for each marker in `cutoffs`

Examples

```
# Create a fake dataset
set.seed(100)
intens_dat <- tibble::tibble(
               CD3_pos=rep(c(0, 1), each=50),
               CD4=rnorm(100, 100, 10),
               CD8=rnorm(100, 100, 10)
)

# Run getDensityGates to obtain the gates
gates <- getDensityGates(intens_dat, marker="CD4", subset_col="CD3_pos", bin_n=40)

# Tag on the 0/1 on intens_dat
intens_dat_2 <- getGatedDat(intens_dat, cutoffs=gates, subset_col="CD3_pos")

# intens_dat_2 now has the cd4_pos tagged on
head(intens_dat_2)
```

---

getPerc *Calculate the percentage of positive cells for specific subpopulations*

Description

Expects data input same as the output from `get_gated_dat` with indicator columns of specific naming convention (see below).

Usage

```
getPerc(
  intens_dat,
  num_marker,
  denom_marker,
  expand_num = FALSE,
  expand_denom = FALSE,
  keep_indicators = TRUE
)
```

Arguments

`intens_dat`	dataframe of gated data with indicator columns per marker of interest (specify in `num_marker` and `denom_marker`) with naming convention `marker_pos` per marker with values of 0 to indicate negative-, 1 to indicate positive-expressing

num_marker — string for the marker(s) to specify the numerator for subpopulations of interest
See expand_num argument and examples for how to specify

denom_marker — string for the marker(s) to specify the denominator for subpopulations of interest
See expand_denom argument and examples for how to specify.

expand_num — logical, only accepts TRUE or FALSE with default of FALSE
if expand_num=TRUE, currently only considers up to pairs of markers specified in num_marker in the numerator of subpopulation calculations (e.g., CD4+ & CD8- of CD3+)
if expand_num=FALSE, only considers each marker specified in num_marker individually in the numerator of subpopulation calculations (e.g., CD4+ of CD3+)

expand_denom — logical, only accepts TRUE or FALSE with default of FALSE
if expand_denom=TRUE, currently considers up to 1 marker from the num_marker and the unique combinations of denom_marker to generate list of subpopulations
e.g., if denom_marker=c("CD8"), num_marker=c("LAG3", "KI67"), and expand_denom=TRUE, the subpopulations will include:
1. LAG3+ of CD8+, LAG3- of CD8+, LAG3+ of CD8-, LAG3- of CD8-,
2. KI67+ of CD8+, KI67- of CD8+, KI67+ of CD8-, KI67- of CD8-,
3. KI67+ of (LAG3+ & CD8+), KI67- of (LAG3+ & CD8+), KI67+ of (LAG3+ & CD8-), KI67- of (LAG3+ & CD8-)...etc.,
4. LAG3+ of (KI67+ & CD8+), LAG3- of (KI67+ & CD8+), LAG3+ of (KI67+ & CD8-), LAG3- of (KI67+ & CD8-)...etc.,
if expand_denom=FALSE, only generates the list of subpopulations based on unique combinations of the denom_marker (e.g., denom_marker=c("CD4") and expand_denom=FALSE only considers subpopulations with denominator CD4+ and CD4- whereas denom_marker=c("CD4", "CD8" and expand_denom=FALSE will consider subpopulations with denominators (CD4- & CD8-), (CD4+ & CD8-), (CD4- & CD8+) and (CD4+ & CD8+))

keep_indicators — logical, only accepts TRUE or FALSE with default of TRUE
if keep_indicators=TRUE, will return indicator columns of 0/1 to specify which markers are considered in the numerator and denominators of the subpopulations.
Naming convention for the numerator cols are <marker>_POS and for denominator cols are <marker>_POS_D.
For both sets of columns, 0 indicates considered the negative cells, 1 indicates considered the positive cells and NA_real_ indicates not in consideration for the subpopulation.
This is useful for matching to percentage data with potentially different naming conventions to avoid not having exact string matches for the same subpopulation
Take note that the order also matters when matching strings: "CD4+ & CD8- of CD3+" is different from "CD8- & CD4+ of CD3+"

Details

The subpopulations are defined as (num marker(s)) out of (denom marker(s)) where num denotes numerator, and denom denotes denominator (these shorthands are used in the function arguments)

Value

tibble containing the percentage of cells where

- rows correspond to each subpopulation specified in the subpopulation,

- n_num indicates the number of cells that satisifies the numerator conditions,
- n_denom indicates the number of cells that satisifies the denominator conditions,
- perc=n_num divided by n_denom unless n_denom=0, then perc=NA_real_

Examples

```
library(dplyr)

# Create a fake dataset
set.seed(100)
intens_dat <- tibble::tibble(
               CD3_pos=rep(c(0, 1), each=50),
               CD4=rnorm(100, 100, 10),
               CD8=rnorm(100, 100, 10)
)

# Run getDensityGates to obtain the gates
gates <- getDensityGates(intens_dat, marker="CD4", subset_col="CD3_pos", bin_n=40)

# Tag on the 0/1 on intens_dat
intens_dat_2 <- getGatedDat(intens_dat, cutoffs=gates, subset_col="CD3_pos")

# Get percentage for CD4 based on gating
getPerc(intens_dat_2, num_marker=c("CD4"), denom_marker="CD3")
```

Index