Package 'flowGate'

November 12, 2026

Type Package

Title Interactive Cytometry Gating in R

Version 1.12.1

Description flowGate adds an interactive Shiny app to allow manual GUI-based gating of flow cytometry data in R. Using flowGate, you can draw 1D and 2D span/rectangle gates, quadrant gates, and polygon gates on flow cytometry data by interactively drawing the gates on a plot of your data, rather than by specifying gate coordinates. This package is especially geared toward wet-lab cytometerists looking to take advantage of R for cytometry analysis, without necessarily having a lot of R experience.

License MIT + file LICENSE

Encoding UTF-8

LazyData false

Imports shiny (>= 1.5.0), BiocManager (>= 1.30.10), flowCore (>= 2.0.1), dplyr (>= 1.0.0), ggplot2 (>= 3.3.2), rlang (>= 0.4.7), purrr, tibble, methods

Depends flowWorkspace (>= 4.0.6), ggcyto (>= 1.16.0), R (>= 4.2)

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, stringr, tidyverse, testthat

VignetteBuilder knitr

biocViews Software, WorkflowStep, FlowCytometry, Preprocessing, ImmunoOncology, DataImport

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

git_branch RELEASE_3_23

git_last_commit fb67327

git_last_commit_date 2026-07-02

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Andrew Wight [aut, cre], Harvey Cantor [aut, ldr]

Maintainer Andrew Wight <andrew.wight10@gmail.com>

Contents

gs_apply_gating_strategy 2
gs_gate_interactive 3
gs_gate_transform_interactive 5

Index **7**

`gs_apply_gating_strategy` *Sequentially apply a manual gating strategy to a GatingSet or list*

Description

This function allows for a "semi-automatic" approach to interactive manual gating of flow cytometry data. It leverages the purrr package to let you easily define a gating strategy and then apply it sequentially to a GatingSet. This will call gs_gate_interactive() once for each line in your gating strategy, applying it to your GatingSet as soon as you draw each prompted gate.

Usage

```
gs_apply_gating_strategy(gs, gating_strategy, ...)
```

Arguments

`gs`	A GatingSet or list of GatingSets.
`gating_strategy`	A tibble-formatted gating strategy (see examples below)
`...`	Other parameters to pass to gs_gate_interactive(). Note that only constant parameters should be supplied here—anything that varies should be included in the gating_strategy tibble.

Details

The gating strategy should be a tibble, with each column name corresponding to one parameter from gs_gate_interactive. Any parameters not specified in this tibble will either use their defaults from gs_gate_interactive or can be specified directly in the function call to gs_apply_gating_strategy. Typically, this gating strategy will have a column for 'filterId', 'dims', 'subset', and 'coords', but techinicaly only filterId is required. See examples below for an easy way to construct this strategy using tribble().

Value

the GatingSet or list of GatingSets with the gates in gating_strategy applied as specified.

Examples

```
fs <- flowCore::read.flowSet(
  path = system.file("extdata", package = "flowGate"), pattern = ".FCS$")

gs <- flowWorkspace::GatingSet(fs)

# Note - this is a very rudamentary GatingSet for example purposes only.
# Please see the vignette accompanying this package or the flowWorkspace
# documentation # for a complete look at creating a GatingSet.

gating_strategy <- tibble::tribble(
~filterId,      ~dims,                       ~subset,        ~coords,
"Lymphocytes", list("FSC-H", "SSC-H"),       "root",         list(c(0, 3e5), c(0, 3e5)),
"CD45 CD15",   list("CD45 PE", "CD15 FITC"), "Lymphocytes",  list(c(0, 3e5), c(0, 2e5)),
)


if(interactive()){
gs_apply_gating_strategy(gs,
gating_strategy = gating_strategy,
bins = 512) # note that extra args for gs_gate_interactive can be supplied.
}
```

---

gs_gate_interactive *Interactive Manual Gating*

Description

gs_gate_interactive opens a new graphical window where you can draw rectangle, polygon, 1-D span, or 2-D quadrant gates that will be applied to an entire GateSet (see the flowWorkspace package for complete information about GateSets).

Usage

```
gs_gate_interactive(
  gs,
  filterId,
  sample = 1,
  dims = list("FSC-A", "SSC-A"),
  subset = "root",
  regate = FALSE,
  overlayGates = NULL
)
```

Arguments

gs	The GateSet that will be gated on.

filterId	String that gives the name of the new gate. Must be unique (can specify parent gates to aid in this).
sample	Numeric specifying which of the GatingHierarchy objects (i.e. which FCS file/flow sample) that make up the GateSet do you want to use to draw the gate? Note that the gate you draw will be applied to all GatingHierarchy objects in the GateSet. Defaults to the first GatingHierarchy object in the GateSet.
dims	A list of strings, length-1 or length-2, that specifies the x- and y- parameters that you will be gating on. Giving a length-1 list will result in a histogram, while a length-2 list will result in a dot-plot. Giving a length-3 or longer list will result in only the first two dimensions being used, and will generate a warning to say as much. Defaults to forward scatter ("FSC-A") and side scatter ("SSC-A").
subset	String that gives the name of the parent gate that you want to sample from. For example, if you wanted to gate all live cells out of a previously drawn "lymphocytes" gate, you would specify "lymphocytes" here. Defaults to "root" (ungated).
regate	A boolean specifying whether all gates with a name matching `filterId` should first be deleted before being re-drawn. Attempting to draw a gate with a non-unique `filterId` without specifying `regate = TRUE` will result in an error. Defaults to `FALSE`
overlayGates	List of strings giving the `filterId`s of other gates to draw on the example plot when gating. Useful for drawing multiple gates on the same population (for example, after specifying a marker-low population, you can overlay the marker-low gate to aid in drawing a marker-high gate). Defaults to `NULL` (no overlaid gates).

Value

A list of the interactively-specified parameters, including the drawn gate's coordinates, plot bins, and any flowjo biex coefs used to calculate those transforms.

Examples

```
path_to_fcs <- system.file("extdata", package = "flowGate")
fs <- read.flowSet(path = path_to_fcs,
                   pattern = ".FCS$",
                   full.names = TRUE)
gs <- GatingSet(fs)

if(interactive()) { # only run in interactive sessions
gs_gate_interactive(gs,
                    filterId = "Lymphocytes",
                    dims = list("FSC-H", "SSC-H"))
}

# returns gs with the same "Lymphocytes" gate on FSC-H and SSC-H applied to
# the root node (all events) of each sample in the GateSet.

if(interactive()) {
gs_gate_interactive(gs,
                    filterId = "Live cells",
```

```
                    dims = "Viability",
                    subset = "Lymphocytes")
}

# returns gs with a "Live cells" gate drawn on all cells included in the
# parent "Lymphocytes" gate. This gate would be based on a histogram of a
# marker called Viability, using the first GatingHierarchy sample as an
# example.

if(interactive()){
gs_gate_interactive(gs,
                    filterId = "Live cells",
                    dims = list("Viability", "SSC-A"),
                    subset = "Lymphocytes",
                    regate = TRUE)
}

# first deletes the "Live cells" gate drawn above, then adds a new "Live
# cells" gate to the set, this time based on a dot plot of Viability by
# side-scatter.

if(interactive()){
gs_gate_interactive(gs,
                    filterId = "Dead cells",
                    dims = list("Viability", "SSC-A"),
                    subset = "Lymphocytes",
                    overlayGates = "Live cells")
}

# returns gs with a "Dead cells" gate drawn on the same example graph that
# was used to draw the "Live cells" gate above. Overlays the "Live cells"
# gate on top of this graph to aid in drawing the "Dead cells" gate.
```

`gs_gate_transform_interactive` *Interactively adjust a gate from a GatingSet*

Description

CAUTION: Experimental Function. Still probably has bugs.

Usage

```
gs_gate_transform_interactive(
  gs,
  node,
  sample = 1,
  dims = list("FSC-A", "SSC-A"),
  overlayGates = NULL
)
```

Arguments

gs	The GatingSet containing the gate you want to adjust
node	String specifying the (unambiguous) name of the node to adjust
sample	Numeric specifying which sample in the GatingSet to use for example purposes. Note that the adjusted gate will be applied to ALL samples, not just this one.
dims	List of characters specifying channel names or marker names to plot on x and y axis. Defaults to list("FSC-A", "SSC-A") mostly just to make the format clear.
overlayGates	(optional) string or character vector specifying names of gates to draw on the plot but NOT adjust, for ease of adjusting a gate in the vicinity of other gates. Leave NULL to not overlay any gates.

Details

Call gs_gate_transform_interactive to open a small Shiny app to allow for manual, interactive adjustments to gates. Currently only supports rectangleGates and polygonGates.

Value

NULL, but silently deletes the old gate, adds the new one, and recomputes the GatingSet.

Examples

```
path_to_fcs <- system.file("extdata", package = "flowGate")
fs <- read.flowSet(path = path_to_fcs,
                   pattern = ".FCS$",
                   full.names = TRUE)
gs <- GatingSet(fs)

if(interactive()) { # only run in interactive sessions
gs_gate_interactive(gs,
                    filterId = "Lymphocytes",
                    dims = list("FSC-H", "SSC-H"))

# Adds a lymphocytes gate to the GatingSet (exactly as in gs_gate_interactive)

gs_gate_transform_interactive(gs,
                              filterId = "Lymphocytes",
                              dims = list("FSC-H", "SSC-H"))
}

# Opens a window to adjust the gate manually
```

Index

gs_apply_gating_strategy, 2
gs_gate_interactive, 3
gs_gate_transform_interactive, 5