Package 'GatingTree'

November 17, 2024

Version 0.1.1

Title Pathfinding Analysis of Group-Specific Effects in Cytometry Data

Description
This package offers a comprehensive toolkit for analyzing high-dimensional cytometric data, including flow and mass cytometry. It uniquely employs the novel GatingTree method, which enables comprehensive identification of group-specific effects across experimental groups without relying on dimensional reduction or clustering, enhancing the reproducibility of analyses. The toolkit includes a range of helper functions for data import, transformation, and effective visualization of GatingTree outputs, providing all essential tools for robust data analysis.

Depends R (>= 4.2.0)

Imports utils, stats, graphics, grDevices, methods, rlang, ggplot2, gridExtra, data.tree, dunn.test, DiagrammeR

Suggests knitr, rmarkdown, KernSmooth, flowCore, HDCytoData

License Apache License 2.0

URL https://github.com/MonoTockyLab/GatingTree, https://MonoTockyLab.github.io/GatingTree

BugReports https://github.com/MonoTockyLab/GatingTree/issues

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Contents

addChildNode 2
collect_history 3
collect_leaf_enrichment 4
collect_markers 5
convertToDataTree 5
convert_to_diagrammer 6
createChildNode 7
CreateFlowObject 9
createGatingTreeObject 10
DefineNegatives 12
export_negative_gate_def 13
extractNodes 13

findNodeByPath 14
flowObject 15
FlowObject-class 15
GatingTreeToDF 16
getNode 17
import_negative_gate_def 18
LogData 18
logSingleData 19
logTransformData 20
moderate_log_transform 20
NormAF 21
PlotDefineNegatives 22
PlotDeltaEnrichment 23
PlotDeltaEnrichmentPrunedTree 24
PlotFlow2D 25
PlotNodeScatterPlot 26
PlotNormAF 26
PruneGatingTree 27
recursiveAddChildNode 28
SampleDef 30
showSampleDef 31

Index **32**

`addChildNode` *Add a Child Node to a Gating Tree*

Description

This function adds a new child node to a specified location in a gating tree.

Usage

```
addChildNode(rootNode, childNode, path)
```

Arguments

`rootNode`	The root node or any node acting as a root in a sub-tree.
`childNode`	The child node to be added.
`path`	The path where the child node should be added.

Value

The modified node with the new child added.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
rootNode <- createGatingTreeObject(...) # Setup initial node
childNode <- createChildNode(...) # Create a child node
path <- "rootNode"
addChildNode(rootNode, childNode, path)

## End(Not run)
```

---

collect_history *Collect History from Gating Tree*

Description

This function recursively collects history from all nodes in a gating tree, aggregating any historical data stored at each node.

Usage

```
collect_history(tree)
```

Arguments

tree	A tree object representing the root or any subtree in a gating tree.

Value

A list containing all historical data from the tree.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
# Assuming `tree` is a root node of a gating tree with history data:
historyData <- collect_history(tree)

## End(Not run)
```

---

collect_leaf_enrichment

Collect Enrichment Data from Leaf Nodes

Description

This function traverses a gating tree to collect enrichment data specifically from leaf nodes, where no further subdivisions occur.

Usage

```
collect_leaf_enrichment(tree)
```

Arguments

tree	A tree object that may be a root node or any subtree in a gating tree.

Value

A list of enrichment data collected from the leaf nodes of the tree.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
# Assuming `tree` is a root node of a gating tree with enrichment data in leaf nodes:
leafEnrichments <- collect_leaf_enrichment(tree)

## End(Not run)
```

collect_markers *Collect Markers from a Gating Tree*

Description

This function collects all markers used in a gating tree.

Usage

```
collect_markers(tree)
```

Arguments

tree	The gating tree structure.

Value

A vector of unique markers used in the tree.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
collect_markers(tree)

## End(Not run)
```

convertToDataTree *Convert Gating Tree Node to Data Tree Node*

Description

This function converts a node from a gating tree into a data tree node format, suitable for further processing or visualization.

Usage

```
convertToDataTree(node, pathName = "rootNode")
```

Arguments

node	A node from a gating tree, usually containing statistical information and child nodes.
pathName	A string representing the node path, defaults to "rootNode".

Value

A data tree node object structured for hierarchical data representations.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
# Assuming `node` is an object from a gating analysis tree:
convertedNode <- convertToDataTree(node)

## End(Not run)
```

---

convert_to_diagrammer *Convert Gating Tree to DiagrammeR Graph*

Description

This function takes a gating tree object and constructs a graphical representation using the DiagrammeR package. It visually represents the enrichment, entropy, and optionally the average proportion of nodes in the gating tree.

Usage

```
convert_to_diagrammer(
  root,
  size_factor = 1,
  average_proportion = FALSE,
  all_labels = TRUE
)
```

Arguments

root	The root node of the gating tree object, which must have properties like 'isRoot', 'name', 'CurrentEnrichment', 'CurrentEntropy', and optionally 'AverageProportion'.
size_factor	A multiplier for node sizes in the graph, allowing customization based on enrichment or average proportion values.
average_proportion	A logical flag indicating whether to use the average proportion of nodes to adjust node sizes and color gradient based on enrichment values.
all_labels	Logical. If TRUE, all parameters are included in each node within the output graph.

Details

The function recursively traverses the gating tree, starting from the root, adding nodes to the graph with labels that include relevant metrics. Node size and color are determined by either the enrichment or entropy values, depending on the 'average_proportion' flag.

Value

Returns a DiagrammeR graph object representing the gating tree with nodes colored and sized according to specified metrics.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
  # Assuming 'root' is your gating tree root node
  graph <- convert_to_diagrammer(root, size_factor = 1, average_proportion = TRUE)
  # To view the graph
  DiagrammeR::render_graph(graph)

## End(Not run)
```

---

createChildNode *Create a Child Node in a Gating Tree*

Description

Constructs a child node for a gating tree based on the specified gating marker and its state, along with related gating statistics and history.

Usage

```
createChildNode(
  marker,
  current_marker_state,
  indices,
  available_markers,
  current_entropy,
  current_enrichment,
  average_proportion,
  entropy_scores,
  enrichment_scores,
  history,
  isPositive = TRUE,
  depth,
  usedmarkers,
  path
)
```

Arguments

marker	Character string of the marker name.
current_marker_state	Current state of all markers at the node.
indices	Indices of the data used in this node.
available_markers	Markers available for further gating.
current_entropy	Current entropy score of the node.
current_enrichment	Current enrichment score of the node.
average_proportion	Average proportion of cells within the node.
entropy_scores	Entropy scores dataframe.
enrichment_scores	Enrichment scores dataframe.
history	List object containing the history of previous steps.
isPositive	Logical, indicating if the marker state is positive.
depth	Integer, the depth of the node in the tree.
usedmarkers	Vector of markers already used in previous nodes.
path	The path from the root to the current node.

Value

An object of class 'GatingTreeNode'.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(),

collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
marker = "CD4"
state = c(1, 0, 2)
indices = 1:100
markers = c("CD4", "CD8")
entropy = 0.5
enrichment = 0.7
scores = data.frame(score=runif(3))
history = list()
depth = 1
usedmarkers = c("CD4")
path = "rootNode"
createChildNode(marker, state, indices, markers, entropy, enrichment,
scores, scores, history, TRUE, depth, usedmarkers, path)

## End(Not run)
```

CreateFlowObject *Create FlowObject by importing flow cytometric data*

Description

This function constructs a new FlowObject using either data imported from CSV files or data provided directly as arguments.

Usage

```
CreateFlowObject(
  path = ".",
  select = TRUE,
  sample_file = NULL,
  Data = NULL,
  sampledef = NULL,
  experiment_name = NULL
)
```

Arguments

path	Path to CSV files.
select	Whether to select files via an interactive window.
sample_file	When select is FALSE, a vector of sample file names.
Data	A data frame containing flow cytometric expression data. If provided, file reading is skipped.

sampledef	An object of class SampleDef or a list containing sample definitions. If provided, sampledef population is skipped.
experiment_name	Name of the experiment.

Value

A FlowObject

See Also

Other Initialization: SampleDef

Examples

```
## Not run:
x  <-  CreateFlowObject()

## End(Not run)
```

---

createGatingTreeObject

Create a Gating Tree Object

Description

Initializes and populates a gating tree based on user-defined criteria, handling decision-making through multiple layers of a gating hierarchy. This function integrates gating rules, applies negative gating definitions, and assesses cell population statistics to manage the flow cytometry data analysis process.

Usage

```
createGatingTreeObject(
  x,
  select_markers = FALSE,
  graphics = FALSE,
  markers = NULL,
  maxDepth = 3,
  min_cell_num = 25,
  expr_group = NULL,
  ctrl_group = NULL,
  verbose = TRUE
)
```

Arguments

x	An object, expected to be of class 'FlowObject', containing the initial data and parameters for gating.
select_markers	Logical; if TRUE, allows the user to select markers interactively.
graphics	Logical; if TRUE, enables graphical selection of markers.

markers	Optional; a vector of markers to be included if not choosing interactively.
maxDepth	Integer; the maximum depth of the gating tree, controlling how many levels of decision nodes can be created.
min_cell_num	The minimal number of cells allowed in nodes.
expr_group	Optional; a character to specify the experimental group. If NULL, this is determined interactively.
ctrl_group	Optional; a character to specify the control group. If NULL, this is determined interactively.
verbose	Logical indicating whether to print progress messages and outputs. Default is TRUE.

Details

The function first checks for the type of the input object to ensure it matches expected classes. It then extracts necessary data and parameters from the object, such as negative gating thresholds and sample definitions. Depending on the options, it may allow interactive selection of markers. The function constructs a hierarchical tree where each node represents a gating decision based on statistical calculations like entropy and enrichment, which are used to determine the next steps in the gating process or to terminate the process.

The gating process involves: - Merging data with sample definitions. - Calculating initial gating statistics. - Recursively creating child nodes based on gating outcomes and thresholds. - Dynamically managing markers and gating paths based on user-defined depth and available data.

Value

Modifies the input object by adding a 'GatingTreeObject' that contains the entire structure of gating decisions and nodes.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_names(), getNode(), prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
# Assuming 'x' is properly instantiated and contains necessary gating setups:
x <- createGatingTreeObject(x, select_markers = TRUE, graphics = FALSE, maxDepth = 3)

## End(Not run)
```

`DefineNegatives` *Interactively determine gates to define negative (and positive) regions for each marker expression*

Description

This function allows interactive setting of thresholds on plots to define negative and positive gates for markers. Users can choose to plot data as 2D scatter plots or as density plots, and may utilize pseudocolor to enhance visualization.

Usage

```
DefineNegatives(
  x,
  select = TRUE,
  max_cells_displayed = 30000,
  y_axis_var = NULL,
  pseudocolour = TRUE
)
```

Arguments

`x`	A FlowObject containing the data.
`select`	Logical, whether to interactively select markers to be processed. If FALSE, all logged expression variables will be used. Default is TRUE.
`max_cells_displayed`	Maximum number of cells displayed in plots for determining a negative threshold.
`y_axis_var`	The variable to use for the y-axis in the interactive plot, or 'Density' to use a density plot. If NULL (default), the user will be prompted to select a variable from x@Data.
`pseudocolour`	Logical, whether to use pseudocolor based on density in scatter plots. Default is TRUE. The option FALSE will use monochrome.

Value

A modified FlowObject containing updated threshold values for autofluorescence in the selected variables in the slot QCdata. You can repeat DefineNegatives to renew or adjust some or all of the negative thresholds for variables.

See Also

Other Data Transformation: `LogData()`, `NormAF()`, `PlotDefineNegatives()`, `PlotNormAF()`

Examples

```
## Not run:
x <- DefineNegatives(x)

## End(Not run)
```

export_negative_gate_def *Export Negative Gate Definitions from FlowObject*

Description

This function exports the negative gate definitions from a FlowObject to a CSV file. It will stop if the negative gate definitions are not found within the FlowObject.

Usage

```
export_negative_gate_def(x, filename = "negative_gate_def.csv")
```

Arguments

x	A FlowObject containing negative gate definitions.
filename	The name of the file to which the negative gate definitions will be written.

Value

The FlowObject, unchanged, with the side effect of writing to a file.

Examples

```
## Not run:
export_negative_gate_def(x, filename = "negative_gate_def.csv")

## End(Not run)
```

extractNodes *Extract All Nodes from GatingTree*

Description

This function recursively extracts all nodes from the specified GatingTree object within a FlowObject, either the full GatingTree or the pruned version.

Usage

```
extractNodes(x, pruned = FALSE)
```

Arguments

x	A FlowObject containing a GatingTree.
pruned	Logical indicating whether to extract nodes from the pruned GatingTree. Defaults to FALSE.

Value

A data frame containing details of all nodes in the specified GatingTree.

Examples

```
## Not run:
  full_tree_nodes <- extractNodes(x)
  pruned_tree_nodes <- extractNodes(x, pruned = TRUE)

## End(Not run)
```

`findNodeByPath` *Find a Node by Path in a Gating Tree*

Description

Traverses a gating tree starting from a specified root node to find and return a node located at a given path. The path should be a sequence of node names indicating the traversal route from the root to the target node.

Usage

```
findNodeByPath(rootNode, path)
```

Arguments

`rootNode`	The root node of the gating tree from which to start the search.
`path`	A character vector representing the path to the desired node. Each element of the vector should correspond to a node name at each level of the tree.

Value

Returns the node at the specified path if found.

See Also

Other GatingTree: `GatingTreeToDF()`, `PlotDeltaEnrichment()`, `PlotDeltaEnrichmentPrunedTree()`, `PruneGatingTree()`, `addChildNode()`, `add_prune()`, `apply_gating_conditions()`, `baseline_entropy()`, `calculate_enrichment()`, `calculate_entropy()`, `collect_all_enrichment()`, `collect_all_entropy()`, `collect_history()`, `collect_leaf_enrichment()`, `collect_markers()`, `convertToDataTree()`, `convert_to_diagrammer()`, `count_nodes()`, `createChildNode()`, `createGatingTreeObject()`, `find_and_update_nodes()`, `gating_entropy()`, `general_node_rule()`, `generate_marker_names()`, `getNode()`, `prune_tree()`, `recursiveAddChildNode()`

Examples

```
## Not run:
rootNode <- createGatingTreeObject(...) # setup initial node, assumes function definition
path <- c("rootNode", "CD4pos", "CD8neg") # example path to find a specific node
tryCatch({
  targetNode <- findNodeByPath(rootNode, path)
  print(targetNode)
}, error = function(e) {
  cat("Error in findNodeByPath: ", e$message, "\n")
})

## End(Not run)
```

flowObject *Create a new FlowObject*

Description

This function constructs a new FlowObject using various inputs related to flow cytometry analysis, including data, sample definitions, metadata, and data processing configurations.

Usage

```
flowObject(
  Data = data.frame(),
  sampledef = new("SampleDef"),
  metadata = list(),
  prep = list(),
  Clustering = list(),
  Gating = list(),
  QCdata = list(),
  Transformation = list()
)
```

Arguments

Data	A data frame containing flow cytometric expression data.
sampledef	An object of class SampleDef, specifying sample definitions.
metadata	A list containing annotation data for the flow cytometry experiment.
prep	A list of character string vectors defining the samples and controls.
Clustering	A list containing cell identities and clustering data.
Gating	A list containing outputs of gating, including GatingTree objects.
QCdata	A list containing the quality control information for the flow cytometry data.
Transformation	A list containing settings for data transformation, including the log data settings.

Value

An object of class FlowObject, which encapsulates all provided data and settings for flow cytometry analysis.

FlowObject-class *A class representing a FlowObject object*

Description

This class provides a representation of a FlowObject object.

Slots

`Data` A data frame containing flow cytometric expression data

`sampledef` A SampleDef object containing sample file definitions.

`metadata` A list containing annotation data

`prep` A list containing sample file definitions.

`Clustering` A list containing cell identities and clustering data

`Gating` A list containing gating information.

`QCdata` A list containing quality control information.

`Transformation` A list containing settings for data transformation, including the log data settings.

`GatingTreeToDF` *Convert Gating Tree Object to Data Frames*

Description

This function processes a Gating Tree object from a flow cytometry analysis package, extracting and calculating various statistics such as enrichment, entropy, and delta values for each gate. It returns the object with additional data frames attached that contain detailed analysis results.

Usage

```
GatingTreeToDF(x)
```

Arguments

`x`	FlowObject.

Details

The function traverses the gating tree recursively to collect enrichment and entropy values from each node. It calculates deltas of enrichment values as differences between consecutive gates. Additionally, it computes a maximum enrichment statistic for each path through the tree, sorting these values to identify the most significant gates. Interaction effects (IE) and a detailed breakdown of changes in enrichment values are also calculated and stored in the returned object.

Value

FlowObject with a GatingTreeDF.

See Also

Other GatingTree: `PlotDeltaEnrichment()`, `PlotDeltaEnrichmentPrunedTree()`, `PruneGatingTree()`, `addChildNode()`, `add_prune()`, `apply_gating_conditions()`, `baseline_entropy()`, `calculate_enrichment()`, `calculate_entropy()`, `collect_all_enrichment()`, `collect_all_entropy()`, `collect_history()`, `collect_leaf_enrichment()`, `collect_markers()`, `convertToDataTree()`, `convert_to_diagrammer()`, `count_nodes()`, `createChildNode()`, `createGatingTreeObject()`, `findNodeByPath()`, `find_and_update_nodes()`, `gating_entropy()`, `general_node_rule()`, `generate_marker_names()`, `getNode()`, `prune_tree()`, `recursiveAddChildNode()`

Examples

```
## Not run:
x <- GatingTreeToDF(x)

## End(Not run)
```

---

getNode *Retrieve a Node from a Gating Tree*

Description

This function traverses a gating tree structure to retrieve a node at a specified path.

Usage

```
getNode(gatingTreeObject, path)
```

Arguments

gatingTreeObject	The root object of the gating tree containing child nodes.
path	A vector of node names specifying the path to the desired node.

Value

The node object at the specified path.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_na prune_tree(), recursiveAddChildNode()

Examples

```
## Not run:
gatingTreeObject <- createGatingTreeObject(...) # assuming this function exists
path <- c("rootNode", "CD4pos")
getNode(gatingTreeObject, path)

## End(Not run)
```

import_negative_gate_def *Import Negative Gate Definitions into FlowObject*

Description

This function imports a specified negative gate definition data frame into a FlowObject. The data frame must contain exactly two columns: 'variable' and 'negative.gate'.

Usage

```
import_negative_gate_def(x, negative_gate_def)
```

Arguments

x	A FlowObject.
negative_gate_def	A data frame containing the negative gate definitions.

Value

The modified FlowObject with updated negative gate definitions.

Examples

```
## Not run:
x <- import_negative_gate_def(x, negative_gate_def = my_gate_definitions)

## End(Not run)
```

LogData *Log fluorescence data*

Description

Log fluorescence data

Usage

```
LogData(x, graphics = TRUE, variables = NULL, prompt = FALSE)
```

Arguments

x	A FlowObject containing non-logged, raw fluorescence data
graphics	Whether to use a graphic device to choose variables for log transformation.
variables	A character vector for defining the variables to be log-transformed.
prompt	Whether to invoke a prompt for asking optional questions.

Value

logged data. FlowObject will include the slot logdata_parameters in the slot Transformation, which includes a list object of logged channel names and parameter s used for log transformation: x_log = log10 (x - s + 1)

See Also

Other Data Transformation: `DefineNegatives()`, `NormAF()`, `PlotDefineNegatives()`, `PlotNormAF()`

Examples

```
## Not run:
x  <-  LogData(x) #x is a FlowObject

## End(Not run)
```

---

logSingleData *Logarithmically Transforms a Vector*

Description

Applies a logarithmic transformation to each element of a numeric vector. Elements greater than 1 are transformed using log10, while elements less than or equal to 1 are set to 0.

Usage

```
logSingleData(x)
```

Arguments

x	A numeric vector to be transformed.

Value

A numeric vector with the transformed values.

Examples

```
logSingleData(c(10, 1, 0.5, 20))
```

logTransformData *Logarithmically Transforms Each Column of a Matrix*

Description

Transforms each column of a numeric matrix by applying a logarithmic transformation adjusted by a quantile and a safety margin.

Usage

```
logTransformData(data, quant_val = 0.001, safety_margin = 100)
```

Arguments

data	A numeric matrix whose columns are to be transformed.
quant_val	A quantile value used for the transformation adjustment.
safety_margin	A numeric value added for safety to ensure positive logarithm domain.

Value

A numeric matrix with each column transformed.

Examples

```
data <- matrix(c(1, 2, 3, 4, 5, 6), nrow=2)
logTransformData(data)
```

moderate_log_transform *Moderately Log Transforms a Vector with Adjustments*

Description

Applies a moderated logarithmic transformation to a numeric vector. Adjustments are made based on a specified quantile and a floor value. All transformed values are guaranteed to be in the domain suitable for log10.

Usage

```
moderate_log_transform(x, q, f)
```

Arguments

x	A numeric vector to be transformed.
q	The quantile adjustment value.
f	The minimum value of the adjustment floor, used to avoid negative or zero logarithm domain.

Value

A numeric vector with the logarithmically transformed values.

Examples

```
moderate_log_transform(c(1, 2, 3, 4, 5), 0.5, 0.1)
```

NormAF *Moderate Extreme Negative Outliers with Random Values within Noise*

Description

This function moderates extreme negative outliers in the specified variables of a FlowObject by replacing these values with random numbers. These random numbers are drawn from a normal distribution determined by the data lying within the 'negative_gate_def' thresholds in the FlowObject. It is typically used to handle outliers in flow cytometry data after defining negative gates using the DefineNegative function.

Usage

```
NormAF(x, var = NULL, output = "QC", plot = FALSE)
```

Arguments

x	A FlowObject that has already been processed using DefineNegative.
var	A character vector specifying the variables (markers) in the FlowObject for which the moderation of extreme negative values should be performed. If NULL, the user is prompted to select variables interactively.
output	The output directory name for output files
plot	Logical, whether to produce diagnotic plots.

Value

A modified FlowObject in which extreme negative values in the specified variables are replaced with random numbers based on the distribution of values within the defined negative gates. This modification is intended to reduce the impact of extreme outliers on subsequent analyses.

See Also

Other Data Transformation: DefineNegatives(), LogData(), PlotDefineNegatives(), PlotNormAF()

Examples

```
## Not run:
  # Assuming 'x' is a valid FlowObject with required preprocessing:
  x <- NormAF(x, var = c("marker1", "marker2"))

## End(Not run)
```

PlotDefineNegatives *Plot the threshold for for each marker expression as per determined by DefineNegatives*

Description

Plot the threshold for for each marker expression as per determined by DefineNegatives

Usage

```
PlotDefineNegatives(
  x,
  y_axis_var = NULL,
  output = FALSE,
  outputFile = "DefineNegativePlot.pdf",
  panel = NULL
)
```

Arguments

x	A FlowObject.
y_axis_var	The variable to use for the y-axis in the generated plots, or 'Density' to use a density plot.
output	If TRUE, plots are generated as a file. The default is FALSE and shows plots in X window.
outputFile	When output is TRUE, this defines the filename of the output file.
panel	The number of panels to be included. If NULL, all panels are included in the output plot.

Value

The same FlowObject is returned for safety.

See Also

Other Data Transformation: DefineNegatives(), LogData(), NormAF(), PlotNormAF()

Examples

```
## Not run:
PlotDefineNegatives(x)

## End(Not run)
```

`PlotDeltaEnrichment` *Plot Delta Enrichment or Interaction Effects for Each Marker*

Description

This function takes a complex object containing gating tree data and performs statistical tests to evaluate the significance of delta enrichment or interaction effects across markers. It plots the results as boxplots and returns the modified object with additional annotations based on the analysis.

Usage

```
PlotDeltaEnrichment(x)
```

Arguments

x	FlowObject.

Details

The function conducts Kruskal-Wallis tests to determine the significance of differences across markers, followed by Dunn's test for post-hoc analysis with Bonferroni correction if significant.

Value

Returns the object 'x' with statistical values

See Also

Other GatingTree: `GatingTreeToDF()`, `PlotDeltaEnrichmentPrunedTree()`, `PruneGatingTree()`, `addChildNode()`, `add_prune()`, `apply_gating_conditions()`, `baseline_entropy()`, `calculate_enrichment()`, `calculate_entropy()`, `collect_all_enrichment()`, `collect_all_entropy()`, `collect_history()`, `collect_leaf_enrichment()`, `collect_markers()`, `convertToDataTree()`, `convert_to_diagrammer()`, `count_nodes()`, `createChildNode()`, `createGatingTreeObject()`, `findNodeByPath()`, `find_and_update_nodes()`, `gating_entropy()`, `general_node_rule()`, `generate_marker_names()`, `getNode()`, `prune_tree()`, `recursiveAddChildNode()`

Examples

```
## Not run:
  x <- PlotDeltaEnrichment(x)

## End(Not run)
```

PlotDeltaEnrichmentPrunedTree *Plot Delta Enrichment or Interaction Effects for Each Marker Using Pruned GatingTree*

Description

Plot Delta Enrichment or Interaction Effects for Each Marker Using Pruned GatingTree

Usage

```
PlotDeltaEnrichmentPrunedTree(x)
```

Arguments

x	FlowObject after applying PruneGatingTree.

Details

The function conducts Kruskal-Wallis tests to determine the significance of differences across markers, followed by Dunn's test for post-hoc analysis with Bonferroni correction if significant.

Value

Returns the object 'x' with statistical values

See Also

Other GatingTree: `GatingTreeToDF()`, `PlotDeltaEnrichment()`, `PruneGatingTree()`, `addChildNode()`, `add_prune()`, `apply_gating_conditions()`, `baseline_entropy()`, `calculate_enrichment()`, `calculate_entropy()`, `collect_all_enrichment()`, `collect_all_entropy()`, `collect_history()`, `collect_leaf_enrichment()`, `collect_markers()`, `convertToDataTree()`, `convert_to_diagrammer()`, `count_nodes()`, `createChildNode()`, `createGatingTreeObject()`, `findNodeByPath()`, `find_and_update_nodes()`, `gating_entropy()`, `general_node_rule()`, `generate_marker_names()`, `getNode()`, `prune_tree()`, `recursiveAddChildNode()`

Examples

```
## Not run:
  x <- PlotDeltaEnrichmentPrunedTree(x)

## End(Not run)
```

PlotFlow2D *Two-Dimensional Plot with Gating and Quadrant Statistics*

Description

This function plots a two-dimensional flow cytometry data highlighting the gated regions and calculates quadrant statistics. It handles data preprocessing to manage outliers by replacing extreme negative values with values from a normal distribution within specified gates. The function is ideal for analyzing and visualizing flow cytometry data, providing insights into the distribution of cell populations across specified markers.

Usage

```
PlotFlow2D(
  x,
  graphics = FALSE,
  output = "output",
  markers = NULL,
  states = NULL,
  gating = TRUE,
  split_group = TRUE,
  max_cells_displayed = 30000
)
```

Arguments

x	A FlowObject that has already been processed using DefineNegative and NormAF.
graphics	Logical, if TRUE, enables graphical selection of markers and gating thresholds via a GUI; otherwise, selections must be input manually. Defaults to FALSE.
output	The directory path where the output plots and data summaries will be saved.
markers	Optionally, a vector of marker names to be used for gating; if NULL, the function prompts for selection.
states	A vector indicating the gating state ('positive' or 'negative') for each marker; if NULL, the function prompts for selection.
gating	Logical, if TRUE, applies gating based on the markers and states provided; defaults to TRUE.
split_group	Logical, if TRUE, splits the data by 'group' variable within the dataset for separate analysis and plotting; defaults to TRUE.
max_cells_displayed	The maximum number of cells to display in the plots, which can help manage performance and clarity in visualizing dense datasets.

Value

Returns the same FlowObject with additional provenance data indicating the analysis steps performed.

Examples

```
## Not run:
  x <- PlotFlow2D(x)

## End(Not run)
```

`PlotNodeScatterPlot` *Plot Node Scatter Plot*

Description

This function creates a scatter plot visualization of node percentages based on the given path through a gating tree. It computes and displays statistical summaries including means and standard deviations by group, adds error bars to the plot, and marks significant differences with asterisks.

Usage

```
PlotNodeScatterPlot(x, path)
```

Arguments

`x`	A TockyObject or FlowObject with GatingTreeObject
`path`	A character vector specifying the path through the gating hierarchy.

Value

A ggplot object representing the node percentage scatter plot with error bars and optionally asterisks denoting statistical significance.

Examples

```
## Not run:
PlotNodeScatterPlot(x, c("path", "to", "node"))

## End(Not run)
```

`PlotNormAF` *Plot Effects of Autofluorescnce Modelling*

Description

This function moderates extreme negative outliers in the specified variables of a FlowObject by replacing these values with random numbers. These random numbers are drawn from a normal distribution determined by the data lying within the 'negative_gate_def' thresholds in the FlowObject. It is typically used to handle outliers in flow cytometry data after defining negative gates using the DefineNegative function.

Usage

```
PlotNormAF(x, graphics = FALSE, max_cells_displayed = 30000, output = "output")
```

Arguments

x	A FlowObject that has already been processed using DefineNegative and NormAF.
graphics	Logical. The default is FALSE, showing variable options in console.
max_cells_displayed	The number of cells displayed in plots for determining a negative therehold.
output	A character for the name of output directory.

Value

The same FlowObject is returned for safety.

See Also

Other Data Transformation: DefineNegatives(), LogData(), NormAF(), PlotDefineNegatives()

Examples

```
## Not run:
  x <- PlotNormAF(x)

## End(Not run)
```

PruneGatingTree *Prune a Gating Tree Based on Statistical Criteria*

Description

This function prunes a gating tree by applying various statistical thresholds to the nodes based on entropy, enrichment, and average proportion metrics. Nodes that do not meet the specified criteria are pruned from the tree. Additionally, p-values are adjusted for multiple comparisons.

Usage

```
PruneGatingTree(
  x,
  max_entropy = 0.9,
  min_enrichment = 0.1,
  min_average_proportion = 0.001,
  p_adjust_method = "BY",
  theta = 0
)
```

Arguments

x	An object, expected to be of class 'FlowObject', containing gating tree data and metadata.
max_entropy	Maximum allowable entropy for a node to remain in the gating tree.
min_enrichment	Minimum enrichment required for a node to remain in the gating tree.

`min_average_proportion`
: Minimum average proportion of cells required for a node to remain in the gating tree.

`p_adjust_method`
: A character string indicating the method to be used for adjusting p-values for multiple comparisons. Defaults to 'BY' (Benjamini-Yekutieli).

`theta`
: A numeric threshold added to the enrichment values for each node. This threshold is used to enforce a criterion where only nodes showing a steady increase in enrichment, greater than this threshold, can be considered for retention in the pruned gating tree. The default is zero.

Details

The function first identifies nodes that meet specified entropy and enrichment criteria, computes statistical metrics for these nodes, and then prunes the gating tree based on these metrics and average proportion criteria. Adjusted p-values are calculated to account for multiple testing.

Value

Returns the modified object 'x' with the gating tree pruned according to the specified parameters. The function also attaches the pruned gating tree and a data frame containing node statistics to the object.

See Also

Other GatingTree: `GatingTreeToDF()`, `PlotDeltaEnrichment()`, `PlotDeltaEnrichmentPrunedTree()`, `addChildNode()`, `add_prune()`, `apply_gating_conditions()`, `baseline_entropy()`, `calculate_enrichment()`, `calculate_entropy()`, `collect_all_enrichment()`, `collect_all_entropy()`, `collect_history()`, `collect_leaf_enrichment()`, `collect_markers()`, `convertToDataTree()`, `convert_to_diagrammer()`, `count_nodes()`, `createChildNode()`, `createGatingTreeObject()`, `findNodeByPath()`, `find_and_update_nodes()`, `gating_entropy()`, `general_node_rule()`, `generate_marker_names()`, `getNode()`, `prune_tree()`, `recursiveAddChildNode()`

Examples

```
## Not run:
  updated_object <- PruneGatingTree(x, min_enrichment = 0.5,max_entropy =0.5)

## End(Not run)
```

---

recursiveAddChildNode *Recursively Add Child Nodes in a Gating Tree*

Description

This function recursively expands a gating tree by adding child nodes according to gating rules. It is designed to iteratively apply gating decisions down to a specified depth or until no further subdivisions are applicable (terminal nodes).

Usage

```
recursiveAddChildNode(
  currentNode,
  root_data,
  sampledef,
  neg_gate,
  expr_group,
  ctrl_group,
  total_cell_per_file,
  maxDepth = 3,
  usedmarkers,
  min_cell_num = 25,
  depth = 1
)
```

Arguments

currentNode	The current node in the gating tree from which children will be generated.
root_data	A data frame containing the data set used for gating decisions.
sampledef	A data frame specifying sample definitions and group assignments.
neg_gate	A list containing thresholds for negative gating decisions.
expr_group	The name of the experimental group within 'sampledef'.
ctrl_group	The name of the control group within 'sampledef'.
total_cell_per_file	A data frame mapping file names to total cell counts, used for normalization.
maxDepth	The maximum depth to which the tree can expand.
usedmarkers	A vector of markers already used in the gating path up to the current node.
min_cell_num	The minimal number of cells allowed in nodes.
depth	Integer, the depth of the node in the tree.

Details

The function checks if the current node is terminated or if its depth equals the maximum allowed depth. If not, it applies gating rules to decide how to expand the tree by adding child nodes. These decisions are based on statistical measures such as enrichment and entropy, calculated for different marker states. If a child node results in a terminal condition ('Leaf'), the tree expansion stops at that node.

The function uses recursion to navigate and expand deeper levels of the tree, ensuring that all potential gating paths are explored up to the 'maxDepth' or until no further divisions are valid.

Value

The modified current node with potentially new child nodes added, reflecting the gating tree expansion.

See Also

Other GatingTree: GatingTreeToDF(), PlotDeltaEnrichment(), PlotDeltaEnrichmentPrunedTree(), PruneGatingTree(), addChildNode(), add_prune(), apply_gating_conditions(), baseline_entropy(), calculate_enrichment(), calculate_entropy(), collect_all_enrichment(), collect_all_entropy(), collect_history(), collect_leaf_enrichment(), collect_markers(), convertToDataTree(), convert_to_diagrammer(), count_nodes(), createChildNode(), createGatingTreeObject(), findNodeByPath(), find_and_update_nodes(), gating_entropy(), general_node_rule(), generate_marker_na getNode(), prune_tree()

Examples

```
## Not run:
# Assuming currentNode and other required objects are predefined:
updatedNode <- recursiveAddChildNode(currentNode, root_data, sampledef,
neg_gate, expr_group, ctrl_group, total_cell_per_file, maxDepth = 3, usedmarkers)

## End(Not run)
```

SampleDef *SampleDef*

Description

A class to represent a sample definition object

This function updates a FlowObject with a SampleDef object created from a CSV file or a provided data frame.

Usage

```
SampleDef(
  x,
  sample_df = NULL,
  path = "sampledef",
  file = "sample.csv",
  group_column = "group",
  confirm = TRUE
)
```

Arguments

x	A FlowObject.
sample_df	Optional data frame containing the sample definitions with columns 'file' and 'group'.
path	A character string specifying the path to the directory containing the CSV file. Default is './sampledef'.
file	A character string specifying the name of the CSV file. Default is 'sample.csv'.
group_column	A character string specifying the column name for sample grouping in the CSV file.
confirm	If TRUE, prompts confirmation of editing the CSV file.

Value

A modified FlowObject containing a SampleDef object.

See Also

Other Initialization: `CreateFlowObject()`

Examples

```
## Not run:
x <- SampleDef(x)

## End(Not run)
```

`showSampleDef` *Show SampleDef*

Description

Show SampleDef

Usage

```
showSampleDef(x)
```

Arguments

`x`	A FlowObject.

Examples

```
## Not run:
showSampleDef(x)

## End(Not run)
```

Index

∗ **Data Transformation**
DefineNegatives, 12
LogData, 18
NormAF, 21
PlotDefineNegatives, 22
PlotNormAF, 26
∗ **GatingTree**
addChildNode, 2
collect_history, 3
collect_leaf_enrichment, 4
collect_markers, 5
convert_to_diagrammer, 6
convertToDataTree, 5
createChildNode, 7
createGatingTreeObject, 10
findNodeByPath, 14
GatingTreeToDF, 16
getNode, 17
PlotDeltaEnrichment, 23
PlotDeltaEnrichmentPrunedTree, 24
PruneGatingTree, 27
recursiveAddChildNode, 28
∗ **Initialization**
CreateFlowObject, 9
SampleDef, 30
∗ **classes**
FlowObject-class, 15
SampleDef, 30
∗ **flow cytometry analysis**
PlotFlow2D, 25

add_prune, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
addChildNode, 2, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
apply_gating_conditions, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*

baseline_entropy, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*

calculate_enrichment, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
calculate_entropy, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
collect_all_enrichment, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
collect_all_entropy, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
collect_history, *3*, 3, *4–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
collect_leaf_enrichment, *3*, 4, *5–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
collect_markers, *3, 4*, 5, *6, 7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
convert_to_diagrammer, *3–6*, 6, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
convertToDataTree, *3–5*, 5, *7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
count_nodes, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
createChildNode, *3–7*, 7, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
CreateFlowObject, 9, *31*
createGatingTreeObject, *3–7*, *9*, 10, *14*, *16, 17*, *23, 24*, *28*, *30*

DefineNegatives, 12, *19*, *21, 22*, *27*

export_negative_gate_def, 13
extractNodes, 13

find_and_update_nodes, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
findNodeByPath, *3–7*, *9*, *11*, 14, *16, 17*, *23, 24*, *28*, *30*
flowObject, 15
FlowObject-class, 15

gating_entropy, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
GatingTreeToDF, *3–8*, *11*, *14*, 16, *17*, *23, 24*, *28*, *30*
general_node_rule, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
generate_marker_names, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
getNode, *3–7*, *9*, *11*, *14*, *16*, 17, *23, 24*, *28*, *30*

import_negative_gate_def, 18

LogData, *12*, 18, *21, 22*, *27*
logSingleData, 19
logTransformData, 20

moderate_log_transform, 20

NormAF, *12*, *19*, 21, *22*, *27*

PlotDefineNegatives, *12*, *19*, *21*, 22, *27*
PlotDeltaEnrichment, *3–8*, *11*, *14*, *16, 17*, 23, *24*, *28*, *30*
PlotDeltaEnrichmentPrunedTree, *3–8*, *11*, *14*, *16, 17*, *23*, 24, *28*, *30*
PlotFlow2D, 25
PlotNodeScatterPlot, 26
PlotNormAF, *12*, *19*, *21, 22*, 26
prune_tree, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, *30*
PruneGatingTree, *3–8*, *11*, *14*, *16, 17*, *23, 24*, *27*, *30*

recursiveAddChildNode, *3–7*, *9*, *11*, *14*, *16, 17*, *23, 24*, *28*, 28

SampleDef, *10*, 30
showSampleDef, 31