Package 'CompensAID'

May 18, 2026

Type Package

Title Automated detection tool for spillover errors

Version 1.1.0

Description The CompensAID is an automated quality control tool, which determines for each marker combination in the FCS file, whether there a potential presence of reference errors. Such reference errors, which represent themselves in the form of skewed populations, are detected by integrating the Secondary Stain Index (SSI) score. Marker combinations with an SSI < 1 are flagged by CompensAID.

URL https://github.com/Olsman/CompensAID

BugReports https://github.com/Olsman/CompensAID/issues

Depends R (>= 4.1.0)

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Imports checkmate, dplyr, flowCore, flowDensity, ggcyto, ggplot2 (>= 3.5.2), methods, ParallelLogger, reshape2, rlang, stats, tibble, tidyr, utils

biocViews FlowCytometry, QualityControl, Preprocessing

Suggests knitr, rmarkdown, testthat (>= 3.0.0), BiocStyle

VignetteBuilder knitr

Config/testthat/edition 3

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

git_branch devel

git_last_commit ba6a084

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-18

Author Rosan Olsman [aut, cre] (ORCID: <https://orcid.org/0000-0001-5070-0476>), Sarah Bonte [aut] (ORCID: <https://orcid.org/0000-0002-0637-4893>), Mattias Hofmans [aut] (ORCID: <https://orcid.org/0000-0003-2061-7617>), Malicorne Buysse [aut], Katrien Van der Borght [aut] (ORCID:

<https://orcid.org/0000-0003-1635-4359>),
Yvan Saeys [aut] (ORCID: <https://orcid.org/0000-0002-0415-1506>),
Vincent van der Velden [aut] (ORCID:
<https://orcid.org/0000-0001-9457-3763>),
Sofie Van Gassen [aut] (ORCID: <https://orcid.org/0000-0002-7119-5330>)

Maintainer Rosan Olsman <r.olsman@erasmusmc.nl>

Contents

`.AdjustSegments` *Adjust (merged) segment information*

Description

Adjust (merged) segment information

Usage

```
.AdjustSegments(si.input, mp, ms)
```

Arguments

`si.input`	(dataFrame): DataFrame containing the SSI info.
`mp`	(character): Name of the primary marker.
`ms`	(character): Name of the secondary marker.

Value

Dataframe with updated SSI info

.CalculateSSI *Calculate the Secondary Stain Index for each marker combination*

Description

Calculate the Secondary Stain Index for each marker combination

Usage

```
.CalculateSSI(si.input, primary.channel, secondary.channel, segment)
```

Arguments

`si.input`	(dataFrame): DataFrame containing the SSI info.
`primary.channel`	(character): Name of the primary marker.
`secondary.channel`	(character): Name of the secondary marker.
`segment`	(numeric): Numerical value of the segment

Value

Numeric value of Secondary Stain Index score

.CleanSegments *Clean adjust segment information*

Description

Clean adjust segment information

Usage

```
.CleanSegments(si.input, mp, ms)
```

Arguments

`si.input`	(dataFrame): DataFrame containing the SSI info.
`mp`	(character): Name of the primary marker.
`ms`	(character): Name of the secondary marker.

Value

(dataFrame) Returns a dataframe with an update SSI information dataFrame.

.DensityGating *Perform density-based cut-off detection*

Description

Perform density-based cut-off detection

Usage

```
.DensityGating(og, cp.value)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
cp.value	(numerical): Numerical value determining the preliminary center.

Value

(dataFrame) Returns a dataFrame with the output of the density-based cut-off detection

.DetermineCenter *Rougly estimate the 'center' of a marker*

Description

Rougly estimate the 'center' of a marker

Usage

```
.DetermineCenter(og)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.

Value

(list) Returns a list containing the estimated center and additional margin

.EmptyMatrix *Obtain empty SSI matrix*

Description

Obtain empty SSI matrix

Usage

```
.EmptyMatrix(og)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.

Value

(matrix) Returns an empty SSI matrix

.EmptyMatrixInfo *Obtain empty SSI dataFrame*

Description

Obtain empty SSI dataFrame

Usage

```
.EmptyMatrixInfo(
  og = ff,
  rv.input = range.value,
  mc.input = mc,
  co.input = co,
  sd.input = separation.distance
)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
rv.input	(numerical): Numerical value of the number of segments in the primary positive population.
mc.input	(dataFrame): dataFrame containing all possible marker combinations.
co.input	(dataFrame) dataFrame with the output of the density-based cut-off detection.
sd.input	(numerical): Numerical value determining the distance between the primary negative and positive population.

Value

(dataFrame) Returns an empty SSI dataFrame

.EventRequirement *Assess event requirement*

Description

Assess event requirement

Usage

```
.EventRequirement(negative, positive, events.value)
```

Arguments

negative	(matrix): All events that fall within the negative primary population.
positive	(matrix): All events that fall within the positive primary population.
events.value	(numerical): Numerical value defining the minimum requirement of events.

Value

(boolean) Returns TRUE/FALSE depending on the required number of events.

.GetClosestCenter *Identify value closest to visual estimation*

Description

Identify value closest to visual estimation

Usage

```
.GetClosestCenter(row, closest)
```

Arguments

row	(numerical): Numerical values of all density-based cut-off detection values.
closest	(numerical): Numerical value determining the preliminary center.

Value

(numerical) Returns the value closest to the visual center of the plot.

.GetClosestLimit *Detect if smaller peaks inclusion improved the density-based cut-off detection.*

Description

Detect if smaller peaks inclusion improved the density-based cut-off detection.

Usage

```
.GetClosestLimit(old.limit, new.limit, center.plot)
```

Arguments

old.limit	(numerical): Numerical value of the limit detected under default settings.
new.limit	(numerical): Numerical value of the limit detected under adjusted settings
center.plot	(numerical): Numerical value of the estimated center.

Value

(numerical) Returns the limit that fits the data best.

.GetMarkerCombinations *Obtain all possible marker combinations*

Description

Obtain all possible marker combinations

Usage

```
.GetMarkerCombinations(og)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.

Value

(dataFrame) Returns a dataFrame containing all possible marker combinations

.GetPopulations *Obtain (segmented) populations*

Description

Obtain (segmented) populations

Usage

```
.GetPopulations(og, primary, secondary, co.input, sd.input)
```

Arguments

og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
primary	(character): Name of the primary marker.
secondary	(character): Name of the secondary marker.
co.input	(dataFrame) dataFrame with the output of the density-based cut-off detection.
sd.input	(numerical): Numerical value determining the distance between the primary negative and positive population.

Value

(list) Returns a list with the negative population of the primary and secondary marker, and positive population of the primary marker.

.GetSegment *Calculate segment information*

Description

Calculate segment information

Usage

```
.GetSegment(
  population,
  primary.channel,
  secondary.channel,
  segment,
  range.input
)
```

Arguments

population	(matrix): Matrix of the population that is segmented.
primary.channel	(character): Name of the primary channel.
secondary.channel	(character): Name of the secondary channel.
segment	(numerical) Segment number for which the information is obtained.
range.input	(numerical): Numerical value defining the width of each segment.

.MergeSegments *Combine segments that do not meet the requirements*

Description

Combine segments that do not meet the requirements

Usage

```
.MergeSegments(segment.event.count, ev.input)
```

Arguments

segment.event.count	(numerical): Number of events per segment
ev.input	(numerical): Minimum required number of events.

Value

(list) Returns a list with which segments need to be merged.

.UpdateMatrixInfo *Calculate segment information*

Description

Calculate segment information

Usage

```
.UpdateMatrixInfo(
  si.input,
  range.input = NULL,
  primary,
  secondary,
  output,
  rv.input,
  population = NULL
)
```

Arguments

si.input	(dataFrame): dataFrame containing SSI info.
range.input	(numerical): Width of each segment.
primary	(character): Name of the primary marker.
secondary	(character) Name of the secondary marker.
output	(character): Will be "PASS" or "No positive/negative population" depending on the required events.
rv.input	(numerical): Number of segments.
population	(matrix): Matrix with the population for which the information is obtained.

Value

(dataFrame) Returns a dataframe with an update SSI information dataFrame.

.UpdateSegments *Combine segments that do not meet the requirements*

Description

Combine segments that do not meet the requirements

Usage

```
.UpdateSegments(si.input, primary, secondary, ev.input, rv.input)
```

Arguments

si.input	(dataFrame): dataFrame containing SSI info.
primary	(character): Name of the primary marker.
secondary	(character) Name of the secondary marker.
ev.input	(numerical): Minimum required number of events.
rv.input	(numerical): Number of segments.

Value

(dataFrame) Returns a dataframe with an update SSI information dataFrame.

.UpdateSSI *Update Secondary Stain Index*

Description

Update Secondary Stain Index

Usage

```
.UpdateSSI(si.input, rv.input, primary, secondary, population)
```

Arguments

si.input	(dataFrame): DataFrame containing the SSI info.
rv.input	(numerical): Numerical value for the number of segments.
primary	(character): Name of the primary marker
secondary	(character): Name of the secondary marker
population	(list): List with the population matrices

Value

(numerical) Returns the Secondary Stain Index score.

.WithinLimit *Detect if smaller peaks need included for density-based cut-off detection.*

Description

Detect if smaller peaks need included for density-based cut-off detection.

Usage

```
.WithinLimit(
  population,
  og,
  primary,
  secondary,
  min = 10,
  max = 90,
  si.input,
  sd.input,
  co.input,
  cp.value
)
```

Arguments

population	(list) List with the negative population of the primary and secondary marker, and positive population of the primary marker.
og	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
primary	(character): Name of the primary channel.
secondary	(character): Name of the secondary channel.
min	(numerical): Minimum percentage of events required.
max	(numerical): Maximum percentage of events required.
si.input	(dataFrame): dataFrame containing SSI info.
sd.input	(numerical): Numerical value determining the distance between the primary negative and positive population.
co.input	(dataFrame) dataFrame with the output of the density-based cut-off detection.

Value

(list) Returns a list with the adjusted cutoffs and update secondary stain index information dataFrame.

CompensAID *CompensAID*

Description

Run CompensAID to assess the potential presence of reference errors within the FCS file. The outcome is a list which contains an SSI matrix. The matrix quickly visualizes the marker combinations for each marker combinations. Additional information regarding the number of events per segments, which segments were merged, and the corresponding SSI values can be found in the Info dataframe.

Usage

```
CompensAID(ff, segment.value = 4, events.value = 50)
```

Arguments

ff	(FlowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
segment.value	(numerical): Numerical value of the number of segments in the primary positive population.
events.value	(numerical): Numerical value of the minimum number of events per population/segment.

Value

(list) Returns a list containing the full SSI output and SSI matrix

Examples

```
# Import FCS file
file <- flowCore::read.FCS(system.file("extdata", "68983.fcs", package = "CompensAID"))

# Run CompensAID tool
compensAID.res <- CompensAID(ff = file)
```

`PlotDotSSI` *Plot dot plot of Secondary Stain Index scores*

Description

This function plots the marker combinations of interest. Additionally, the gating, segmentation, and SSI values can be visualized by setting showScore to TRUE.

Usage

```
PlotDotSSI(output, og, primary, secondary, showScores = TRUE)
```

Arguments

`output`	(matrix): Matrix containing the SSI output from the CompensAID tool.
`og`	(flowFrame): FlowFrame containing the expression matrix, channel names, and marker names.
`primary`	(character): Primary marker (x-axis)
`secondary`	(character): Seconday marker (y-axis)
`showScores`	(boolean): Boolean variable determining if scores and gating should be visualized.

Value

(ggplot2) Returns figure containing a dot plot of the CompensAID output.

See Also

`CompensAID`

Examples

```
# Import FCS file
file <- flowCore::read.FCS(system.file("extdata", "68983.fcs", package = "CompensAID"))

# Run compensAID tool
compensAID.res <- CompensAID(ff = file)

# Marker names
primary.marker <- "CD19"
secondary.marker <- "CD3"

# Plot matrix
```

```
figure <- PlotDotSSI(output = compensAID.res,
                     og = file,
                     primary = primary.marker,
                     secondary = secondary.marker,
                     showScores = TRUE)
plot(figure)
```

PlotMatrix *Plot Secondary Stain Index matrix*

Description

This function plot the SSI matrix, quickly allowing the identification of potential reference errors if a SSI < -1 is obtained.

Usage

```
PlotMatrix(output)
```

Arguments

output	(matrix): Matrix containing the SSI output.

Value

(ggplot2) Returns figure containing a matrix of the total compensAID output.

See Also

CompensAID

Examples

```
# Import FCS file
file <- flowCore::read.FCS(system.file("extdata", "68983.fcs", package = "CompensAID"))

# Run compensAID tool
compensAID.res <- CompensAID(ff = file)

# Plot matrix
figure <- PlotMatrix(output = compensAID.res)
plot(figure)
```

Index