PeacoQC

Annelies Emmaneel

24 October 2023

Package

PeacoQC 1.12.0

Contents

1 Introduction

The PeacoQC package provides quality control functions that will check for monotonic increasing channels, that will remove outliers and unstable events introduced due to e.g. clogs, speed changes etc. during the measurement of your sample. It also provides the functionality of visualising the quality control result of only one sample and the visualisation of the results of multiple samples in one experiment.

2 Installing PeacoQC

```
# install.packages("devtools")
# devtools::install_github("https://github.com/saeyslab/PeacoQC")

library(PeacoQC)
```

2.1 WARNING

Please be aware of the fact that this vignette will create a directory in your current working directory. Since this package is meant to be run on multiple files and the figures are directly created in this directory. Please check the folders corresponding to the output_directory variable when called for in the chuncks of code.

3 Standard pre-processing and quality control pipeline

The following pipeline is recommended to use for pre-processing your data. The pipeline starts with a flowframe (flow cytometry or mass cytometry data) and will first remove margins, then it will compensate and transform while ending with the PeacoQC quality control. This will give back a list with the cleaned flowframe and the different parameter settings, it will also save the flowframe in a new fcs file and it will make a plot of the quality control for this sample.

Note that the remove margins functionality and compensation is not necessary for mass cytometry data as explained in the code below this example.

```
# Specify flowframe path and read your flowframe
fileName <- system.file("extdata", "111.fcs", package="PeacoQC")
ff <- flowCore::read.FCS(fileName)


# Determine channels on which quality control should be done
channels <- c(1, 3, 5:14, 18, 21)


# Remove margins
# Make sure you do this before any compensation since the internal parameters
# change and they are neccessary for the RemoveMargins function.
# If this is not possible, you can specify the internal parameters in the
```

```
# channel_specifications parameter.

ff <- RemoveMargins(ff=ff, channels=channels, output="frame")


# Compensate and transform the data
ff <- flowCore::compensate(ff, flowCore::keyword(ff)$SPILL)
ff <- flowCore::transform(ff,
    flowCore::estimateLogicle(ff, colnames(flowCore::keyword(ff)$SPILL)))


# Run PeacoQC and save the cleaned flowframe as an fcs file and plot the results
# of this quality control step.
peacoqc_res <- PeacoQC(
    ff=ff,
    channels=channels,
    determine_good_cells="all",
    save_fcs=TRUE,
    plot=TRUE,
    output_directory = "PeacoQC_Example1")
#> Starting quality control analysis for 111.fcs
#> Calculating peaks
#> MAD analysis removed 11.63% of the measurements
#> The algorithm removed 11.63% of the measurements
#> Saving fcs file
#> Plotting the results


# Filtered flowframe is stored in peacoqc_res$FinalFF and can be used for
# further analysis.
ff <- peacoqc_res$FinalFF
```

4 Mass cytometry data

If you want to clean mass cytometry data files you should alter some parameters. The parameter remove_zeros should be set to TRUE. The IT_limit will range between 0.6 and 0.65 since some channels will be more sparse than flow cytometry results. The time_units parameter in the plot function should be also be altered since mass cytometry data is typically measures for a longer time than the flow cytometry data. You should play a bit with the parameter until you don't see the picketfencing effect anymore in the event rate plot (Top left in the overview).

Note that this chunck of code will not be excecuted and that no results will appear in your working directory. This is an example of how to work with mass cytometry data.

```
# # Example of how the code could look for mass cytometry data

ff <- flowCore::read.FCS(file)

# You don't have to remove margin events or compensate the data but you
```

```
# should transform it
channels <- c(3, 5, 6:53)

ff <- transform(ff,transformList(colnames(ff)[channels],
                                 arcsinhTransform(a = 0, b = 1/5, c = 0)))

# Make sure the parameters are set correctly and that the remove_zeros variable
# is set to TRUE.
peacoqc_results <- PeacoQC(ff,
    channels=channels,
    IT_limit=0.6,
    remove_zeros=TRUE,
    time_units=50000)
```

5 Large dataset

If you have a large dataset and you want to run your preprocessing pipeline for multiple files, it is recommended to run it first on a couple of files to tweak your parameters. If your dataset inlcudes channels that have a sparse pattern (e.g. in mass data), the IT_limit should probably be increased in order to be more strict. If it seems that the MAD parameter removes too much, you can also increase this to make PeacoQC less strict.

You can then run all your files with the parameters you chose.

```
# Change IT_limit for one compensated and transformed file.
# (Higher=more strict, lower=less strict)
# The fcs file should not be saved since we are still optimising the parameters
fileName <- system.file("extdata", "111.fcs", package="PeacoQC")
ff <- flowCore::read.FCS(fileName)

# Determine channels on which quality control should be done
channels <- c(1, 3, 5:14, 18, 21)

# Remove margins
ff <- RemoveMargins(ff=ff, channels=channels, output="frame")

# Compensate and transform the data
ff <- flowCore::compensate(ff, flowCore::keyword(ff)$SPILL)
ff <- flowCore::transform(ff,
    flowCore::estimateLogicle(ff, colnames(flowCore::keyword(ff)$SPILL)))


# Run PeacoQC and save the cleaned flowframe as an fcs file and plot the results
# of this quality control step.
peacoqc_res <- PeacoQC(
    ff=ff,
    channels=channels,
    determine_good_cells="all",
    save_fcs=FALSE,
    plot=TRUE,
```

```
    output_directory = "PeacoQC_Example2",
    IT_limit = 0.65)
#> Starting quality control analysis for 111.fcs
#> Calculating peaks
#> MAD analysis removed 11.63% of the measurements
#> The algorithm removed 11.63% of the measurements
#> Plotting the results
```

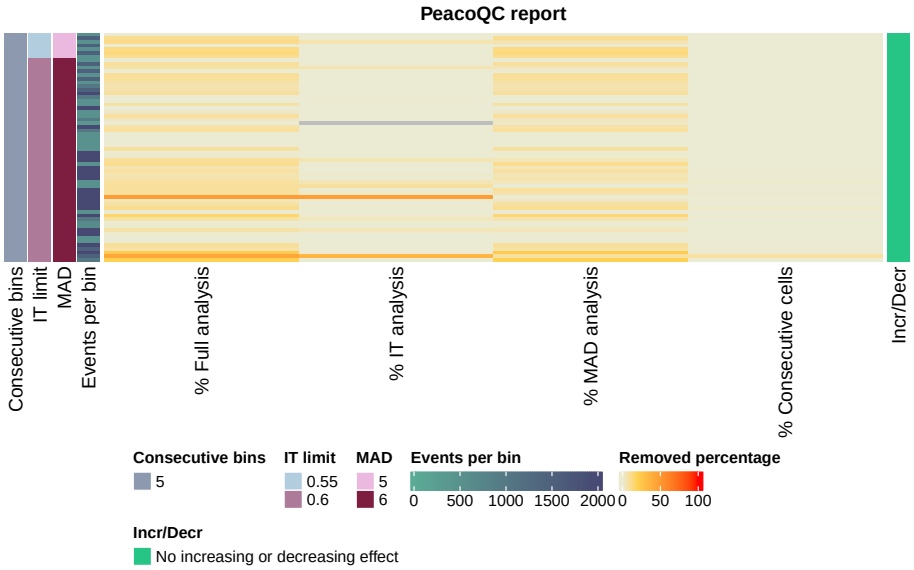
Note that the next chunck of code will not generate any results.

```
# You can also change the MAD parameter to a lower value
# (to make it more strict) or to a higher value (to make it less strict).
# Since the MAD analysis does not remove something, this is not neccesary now.

peacoqc_res <- PeacoQC(
    ff,
    channels,
    determine_good_cells="all",
    save_fcs=FALSE,
    plot=TRUE,
    MAD=8
)


# When the correct parameters are chosen you can run the different files in
# a for loop

for (file in files){
    ff <- flowCore::read.FCS(file)

    # Remove margins
    ff <- RemoveMargins(ff=ff, channels=channels, output="frame")

    # Compensate and transform the data
    ff <- flowCore::compensate(ff, flowCore::keyword(ff)$SPILL)
    ff <- flowCore::transform(ff,
                              flowCore::estimateLogicle(ff,
                              colnames(flowCore::keyword(ff)$SPILL)))
    peacoqc_res <- PeacoQC(
                            ff,
                            channels,
                            determine_good_cells="all",
                            IT_limit=0.6,
                            save_fcs=T,
                            plot=T)
    }
```

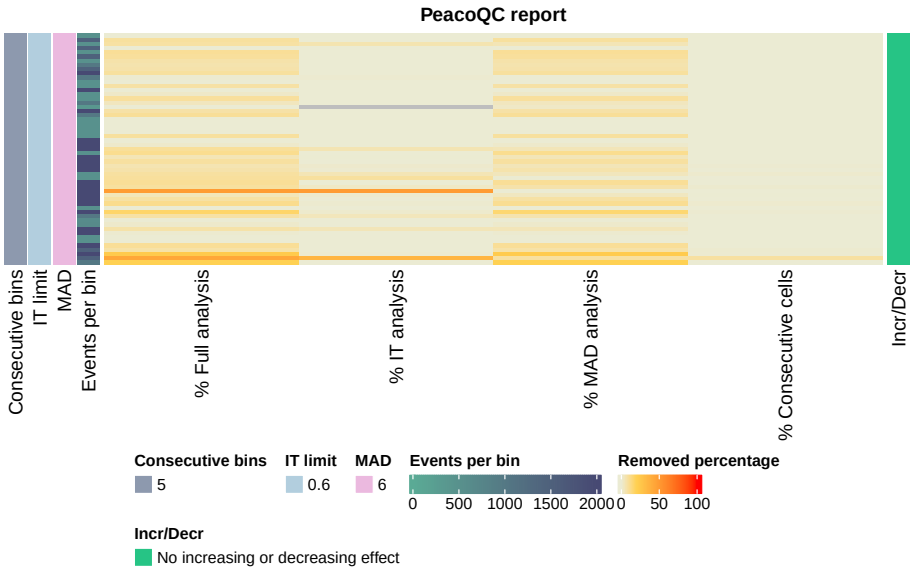
6 PeacoQCHeatmap

In order to get an overview on how much the quality control removed, the PeacoQCHeatmap allows for a visualised representation of the different conditions and the different percentages that were removed in different runs.

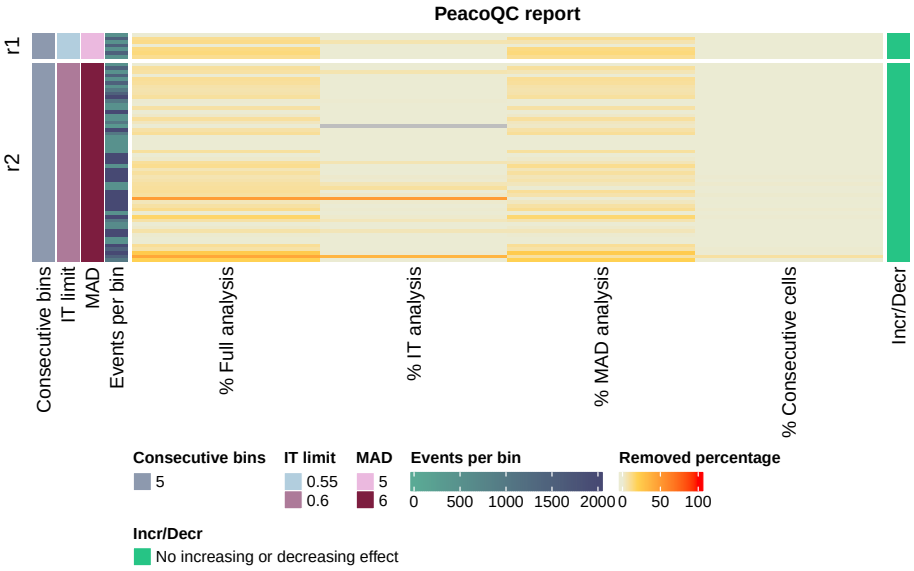
```
# Find the path to the report that was created by using the PeacoQC function
location <- system.file("extdata", "PeacoQC_report.txt", package="PeacoQC")

# Make heatmap overview of the quality control run
PeacoQCHeatmap(report_location=location, show_values = FALSE,
               show_row_names = FALSE)
```




```
# Make heatmap with only the runs of the last test
PeacoQCHeatmap(report_location=location, show_values = FALSE,
               latest_tests=TRUE, show_row_names = FALSE)
```




```
# Make heatmap with row annotation
PeacoQCHeatmap(report_location=location, show_values = FALSE,
               show_row_names = FALSE,
    row_split=c(rep("r1",7), rep("r2", 55)))
```



7 PlotPeacoQC without quality control

The PlotPeacoQC function can also be used to only display the peaks of the different channels without doing any quality control. It can even be used to only display the measurements.

These results will appear in the PeacoQC_plots folder.

```r
# Load in compensated and transformed flowframe

fileName <- system.file("extdata", "111_Comp_Trans.fcs", package="PeacoQC")
ff <- flowCore::read.FCS(fileName)

# Plot only the peaks (No quality control)
PlotPeacoQC(ff, channels, display_peaks=TRUE, prefix = "PeacoQC_peaks_")
#> Running PeacoQC to determine peaks
#> Starting quality control analysis for 111_Comp_Trans.fcs
#> Calculating peaks

# Plot only the dots of the file
PlotPeacoQC(ff, channels, display_peaks=FALSE, prefix = "PeacoQC_nopeaks_")
```