Package 'CytoMDS'

November 22, 2024

Title Low Dimensions projection of cytometry samples

Version 1.3.2

Description This package implements a low dimensional visualization of a set of cytometry samples, in order to visually assess the 'distances' between them. This, in turn, can greatly help the user to identify quality issues like batch effects or outlier samples, and/or check the presence of potential sample clusters that might align with the exeprimental design. The CytoMDS algorithm combines, on the one hand, the concept of Earth Mover's Distance (EMD), a.k.a. Wasserstein metric and, on the other hand, the Multi Dimensional Scaling (MDS) algorithm for the low dimensional projection. Also, the package provides some diagnostic tools for both checking the quality of the MDS projection, as well as tools to help with the interpretation of the axes of the projection.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

BugReports https://github.com/UCLouvain-CBIO/CytoMDS/issues

URL https://uclouvain-cbio.github.io/CytoMDS

biocViews FlowCytometry, QualityControl, DimensionReduction, MultidimensionalScaling, Software, Visualization

Collate 'CytoMDS-package.R' 'stats.R' 'ggplots.R' 'MDS-class.R'

Depends R (>= 4.4)

Imports methods, stats, rlang, pracma, withr, flowCore, reshape2, ggplot2, ggrepel, ggforce, patchwork, transport, smacof, BiocParallel, CytoPipeline

Suggests testthat (>= 3.0.0), vdiffr, diffviewer, knitr, rmarkdown, BiocStyle, HDCytoData

VignetteBuilder knitr

Config/testthat/edition 3

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

git_branch devel

git_last_commit 692d917

git_last_commit_date 2024-11-19

Repository Bioconductor 3.21

Date/Publication 2024-11-22

Author Philippe Hauchamps [aut, cre] (ORCID: <https://orcid.org/0000-0003-2865-1852>), Laurent Gatto [aut] (ORCID: <https://orcid.org/0000-0002-1520-2268>), Dan Lin [ctb]

Maintainer Philippe Hauchamps <philippe.hauchamps@uclouvain.be>

Contents

`CytoMDS-package` *CytoMDS: Low Dimensions projection of cytometry samples*

Description

This package implements a low dimensional visualization of a set of cytometry samples, in order to visually assess the 'distances' between them. This, in turn, can greatly help the user to identify quality issues like batch effects or outlier samples, and/or check the presence of potential sample clusters that might align with the exeprimental design. The CytoMDS algorithm combines, on the one hand, the concept of Earth Mover's Distance (EMD), a.k.a. Wasserstein metric and, on the other hand, the Multi Dimensional Scaling (MDS) algorithm for the low dimensional projection. Also, the package provides some diagnostic tools for both checking the quality of the MDS projection, as well as tools to help with the interpretation of the axes of the projection.

Author(s)

Maintainer: Philippe Hauchamps <philippe.hauchamps@uclouvain.be> (ORCID)

Authors:

- Laurent Gatto <laurent.gatto@uclouvain.be> (ORCID)

Other contributors:

- Dan Lin <dan.8.lin@gsk.com> [contributor]

See Also

Useful links:

- https://uclouvain-cbio.github.io/CytoMDS
- Report bugs at https://github.com/UCLouvain-CBIO/CytoMDS/issues

---

`channelSummaryStats` *Summary statistics per channel computation*

Description

Computation of summary statistic for selected channels, for all flowFrames of a flowSet, or for all expression matrices of a list. This method provides three different input modes:

- the user provides directly a flowCore::flowSet loaded in memory (RAM)
- the user provides directly a list of expression matrices of which the column names are the channel/marker names
- the user provides (1.) a number of samples `nSamples`; (2.) an ad-hoc function that takes as input an index between 1 and `nSamples`, and codes the method to load the corresponding expression matrix in memory;

Usage

```
channelSummaryStats(
  x,
  loadExprMatrixFUN = NULL,
  loadExprMatrixFUNArgs = NULL,
  channels = NULL,
  statFUNs = stats::median,
  verbose = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  BPOPTIONS = BiocParallel::bpoptions(packages = c("flowCore"))
)
```

Arguments

x	can be: • a flowCore::flowSet • a list of expression matrices (Double matrix with named columns) • the number of samples (integer >=1)
loadExprMatrixFUN	the function used to translate an integer index into an expression matrix. In other words, the function should code how to load the indexth expression matrix into memory. IMPORTANT: the expression matrix index should be the first function argument and should be named exprMatrixIndex.
loadExprMatrixFUNArgs	(optional) a named list containing additional input parameters of loadExprMatrixFUN()
channels	which channels needs to be included: • if it is a character vector, it can refer to either the channel names, or the marker names • if it is a numeric vector, it refers to the indices of channels in fs • if NULL, all scatter and fluorescent channels of fs #' will be selected.
statFUNs	a list (possibly of length one) of functions to call to calculate the statistics, or a simple function. This list can be named, in that case, these names will be transfered to the returned list.
verbose	if TRUE, output a message after each single statistics calculation
BPPARAM	sets the BPPARAM back-end to be used for the computation. If not provided, will use BiocParallel::SerialParam() (no task parallelization)
BPOPTIONS	sets the BPOPTIONS to be passed to bplapply() function. Note that if you use a SnowParams back-end, you need to specify all the packages that need to be loaded for the different CytoProcessingStep to work properly (visibility of functions). As a minimum, the flowCore package needs to be loaded. (hence the default BPOPTIONS = bpoptions(packages = c("flowCore")))

Value

a list of named statistic matrices. In each stat matrix, the columns are the channel statistics for all flowFrames of the flowSet. Exception: if only one stat function (and not a list) is passed in statFUNs, the return value is simplified to the stat matrix itself.

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
```

```
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

channelsOrMarkers <- c("FSC-A", "SSC-A", "BV785 - CD3")

# calculate mean for each 4 selected channels, for each 2 samples

channelMeans <- channelSummaryStats(
    OMIP021Trans,
    channels = channelsOrMarkers,
    statFUNs = mean)

# calculate median AND std deviation
# for each 4 selected channels, for each 2 samples

channelMedians <- channelSummaryStats(
    OMIP021Trans,
    channels = channelsOrMarkers,
    statFUNs = list("median" = stats::median,
                    "std.dev" = stats::sd))
```

`computeMetricMDS` *metric MDS projection of sample*

Description

Multi-dimensional scaling projection of samples, using a distance matrix as an input. The MDS algorithm is not the classical MDS (`cmdscale` alike, aka Torgerson's algorithm), but is the SMACOF algorithm for metric distances that are not necessarily euclidean. After having obtained the projections on the `nDim` dimensions, we always apply svd decomposition to visualize as first axes the ones that contain the most variance of the projected dataset in `nDim` dimensions. Instead of being provided directly by the user, the `nDim` parameter can otherwise be found iteratively by finding the minimum `nDim` parameter that allows the projection to reach a target pseudo RSquare. If this is the case, the `maxDim` parameter is used to avoid looking for too big projection spaces.

Usage

```
computeMetricMDS(
  pwDist,
  nDim = NULL,
  seed = NULL,
  targetPseudoRSq = 0.95,
  maxDim = 128,
```

```
  ...
)
```

Arguments

pwDist	(nSamples rows, nSamples columns), previously calculated pairwise distances between samples, must be provided as a full symmetric square matrix, with 0. diagonal
nDim	number of dimensions of projection, as input to SMACOF algorithm if not provided, will be found iteratively using targetPseudoRSq
seed	seed to be set when launching SMACOF algorithm (e.g. when init is set to "random" but not only)
targetPseudoRSq	target pseudo RSquare to be reached (only used when nDim is set to NULL)
maxDim	in case nDim is found iteratively, maximum number of dimensions the search procedure is allowed to explore
...	additional parameters passed to SMACOF algorithm

Value

an object of S4 class MDS

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

# As there are only 2 samples in OMIP021Samples dataset,
# we create artificial samples that are random combinations of both samples

ffList <- c(
    flowCore::flowSet_to_list(OMIP021Trans),
    lapply(3:5,
           FUN = function(i) {
               aggregateAndSample(
                   OMIP021Trans,
```

```
                    seed = 10*i,
                    nTotalEvents = 5000)[,1:22]
            }))

fsNames <- c("Donor1", "Donor2", paste0("Agg",1:3))
names(ffList) <- fsNames

fsAll <- as(ffList,"flowSet")

flowCore::pData(fsAll)$type <- factor(c("real", "real", rep("synthetic", 3)))
flowCore::pData(fsAll)$grpId <- factor(c("D1", "D2", rep("Agg", 3)))

# calculate all pairwise distances

pwDist <- pairwiseEMDDist(fsAll,
                              channels = c("FSC-A", "SSC-A"),
                              verbose = FALSE)

# compute Metric MDS object with explicit number of dimensions
mdsObj <- computeMetricMDS(pwDist, nDim = 4, seed = 0)

dim <- nDim(mdsObj) # should be 4

#' # compute Metric MDS object by reaching a target pseudo RSquare
mdsObj2 <- computeMetricMDS(pwDist, seed = 0, targetPseudoRSq = 0.999)
```

EMDDist *Calculate Earth Mover's distance between two samples*

Description

Calculate Earth Mover's distance between two samples

Usage

```
EMDDist(
  x1,
  x2,
  channels = NULL,
  binSize = 0.05,
  minRange = -10,
  maxRange = 10,
  returnAll = FALSE
)
```

Arguments

x1	can be either a flowCore::flowFrame, or an expression matrix
x2	can be either a flowCore::flowFrame, or an expression matrix
channels	which channels (integer index(ices) or character(s)): • if it is a character vector, it can refer to either the channel names, or the marker names if x1 and x2 have been provided as flowCore::flowFrame • if it is a numeric vector, it refers to the indexes of channels in x1 • if NULL : if x1 and x2 are provided as flowCore::flowFrames, all scatter and fluorescent channels of x1 will be selected; if x1 and x2 are provided as expression matrices, all colnames of x1 will be selected.
binSize	size of equal bins to approximate the marginal distributions.
minRange	minimum value taken when approximating the marginal distributions
maxRange	maximum value taken when approximating the marginal distributions
returnAll	If TRUE, distributions and marginal distribution distances are returned as well. Default = FALSE.

Value

the Earth Mover's distance between x1 and x2, which is calculated by summing up all EMD approximates for the marginal distributions of each channel

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

# distance with itself (all channels at once)
# => should return 0
dist0 <- EMDDist(
    x1 = OMIP021Trans[[1]],
    x2 = OMIP021Trans[[1]])

# returning only distance, 2 channels
dist1 <- EMDDist(
```

```
    x1 = OMIP021Trans[[1]],
    x2 = OMIP021Trans[[2]],
    channels = c("FSC-A", "SSC-A"))

# using only one channel, passed by marker name
dist2 <- EMDDist(x1 = OMIP021Trans[[1]],
                 x2 = OMIP021Trans[[2]],
                 channels = c("BV785 - CD3"))

# using only one channel, passed by index
dist3 <- EMDDist(x1 = OMIP021Trans[[1]],
                 x2 = OMIP021Trans[[2]],
                 channels = 10)

dist2 == dist3
```

`ggplotMarginalDensities` *Plot of channel intensity marginal densities*

Description

`ggplotMarginalDensities` uses ggplot2 to draw plots of marginal densities of selected channels of a flowSet. If the flowSet contains several flowFrames, all events are concatenated together. By default, a pseudo Rsquare projection quality indicator, and the number of dimensions of the MDS projection are provided in sub-title

Usage

```
ggplotMarginalDensities(
  x,
  sampleSubset,
  channels,
  pDataForColour,
  pDataForGroup,
  nEventInSubsample = Inf,
  seed = NULL,
  transList
)
```

Arguments

`x`	a `flowCore::flowSet` (or a single `flowCore::flowFrame`)
`sampleSubset`	(optional) a logical vector, of size `nrow(pData)`, which is by construction the nb of samples, indicating which samples to keep in the plot. Typically it is obtained through the evaluation of a logical condition on `pData` rows.
`channels`	(optional)

pDataForColour	(optional) which phenoData(fs) variable will be used as colour aesthetic. Should be a character.
pDataForGroup	(optional) which phenoData(fs) variable will be used as group aesthetic. Should be a character.
nEventInSubsample	how many event to take (per flowFrame of the flowSet).
seed	if not null, used in subsampling.
transList	a flowCore::transformList that will be applied before plotting.

Value

a ggplot object

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)


# As there are only 2 samples in OMIP021Samples dataset,
# we create artificial samples that are random combinations of both samples

ffList <- c(
    flowCore::flowSet_to_list(OMIP021Trans),
    lapply(3:5,
           FUN = function(i) {
               aggregateAndSample(
                   OMIP021Trans,
                   seed = 10*i,
                   nTotalEvents = 5000)[,1:22]
           }))

fsNames <- c("Donor1", "Donor2", paste0("Agg",1:3))
names(ffList) <- fsNames

fsAll <- as(ffList,"flowSet")
```

```
flowCore::pData(fsAll)$grpId <- factor(c("D1", "D2", rep("Agg", 3)))
flowCore::pData(fsAll)$lbl <- paste0("S", 1:5)

# plot densities, all samples together
p <- ggplotMarginalDensities(fsAll)

# plot densities, per sample
p <- ggplotMarginalDensities(fsAll, pDataForGroup = "lbl")

# plot densities, per sample and coloured by group
p <- ggplotMarginalDensities(
    fsAll,
    pDataForGroup = "lbl",
    pDataForColour = "grpId")
```

---

ggplotSampleMDS *Plot of Metric MDS object*

Description

ggplotSampleMDS uses ggplot2 to provide plots of Metric MDS results. By default, a pseudo Rsquare projection quality indicator, and the number of dimensions of the MDS projection are provided in sub-title

Usage

```
ggplotSampleMDS(
  mdsObj,
  pData,
  sampleSubset,
  projectionAxes = c(1, 2),
  biplot = FALSE,
  biplotType = c("correlation", "regression"),
  extVariables,
  pDataForColour,
  pDataForShape,
  pDataForLabel,
  pDataForAdditionalLabelling,
  pointSize = 1,
  pointSizeReflectingStress = FALSE,
  title = "Multi Dimensional Scaling",
  displayPointLabels = TRUE,
  pointLabelSize = 3.88,
  repelPointLabels = TRUE,
  displayArrowLabels = TRUE,
  arrowLabelSize = 3.88,
  repelArrowLabels = FALSE,
```

```
  arrowThreshold = 0.8,
  flipXAxis = FALSE,
  flipYAxis = FALSE,
  displayPseudoRSq = TRUE,
  ...
)
```

Arguments

mdsObj	a MDS object, output of the computeMetricMDS() method.
pData	(optional) a data.frame providing user input sample data. These can be design of experiment variables, phenotype data per sample,... and will be used to highlight sample categories in the plot and/or for subsetting.
sampleSubset	(optional) a logical vector, of size nrow(pData), which is by construction the nb of samples, indicating which samples to keep in the plot. Typically it is obtained through the evaluation of a logical condition on pData rows.
projectionAxes	which two axes should be plotted (should be a numeric vector of length 2)
biplot	if TRUE, adds projection of external variables
biplotType	type of biplot used: • if "correlation", projection of external variables will be according to Pearson correlations w.r.t. projection axes (arrow x & y coordinates) • if "regression", a linear regression of external variables using the 2 projection axes as explanatory variables is performed, and the projection of external variables will be according to regression coefficients (arrow direction) and R square of regression (arrow size)
extVariables	are used to generate a biplot these are the external variables that will be used in the biplot. They should be provided as a matrix with named columns corresponding to the variables. The number of rows should be the same as the number of samples. The matrix might contain some NA's, in that case only complete rows will be used to calculate biplot arrows.
pDataForColour	(optional) which pData variable will be used as colour aesthetic. Should be a character.
pDataForShape	(optional) which pData variable will be used as shape aesthetic. Should be a character.
pDataForLabel	(optional) which pData variable will be used as point labels in the plot. Should be a character. If missing, point labels will be set equal to point names defined in MDS object (if not NULL, otherwise no labels will be set).
pDataForAdditionalLabelling	(optional) which pData variable(s) will be add to the ggplot mapping, as to make them available for *plotly* tooltipping. Should be an array of character of maximum length 3. Note this works only if biplot=FALSE, as biplots contain circle and arrows that are currently not supported under ggplotly.
pointSize	size of all points on the plots - only when pointSizeReflectingStress is FALSE.

`pointSizeReflectingStress`
: if TRUE, size of points will appear proportional to stress by point, i.e. the bigger the sample point appears, the less accurate its representation is (in terms of distances w.r.t. other points)

`title`
: title to give to the plot

`displayPointLabels`
: if TRUE, displays labels attached to points (see `pDataForLabels` for the setting of the label values)

`pointLabelSize`
: size of point labels (default: 3.88 as in `geom_text()`)

`repelPointLabels`
: if TRUE, uses `ggrepel::geom_text_repel()` instead of `ggplot2::geom_text()` (try to split the labels such that they do not overlap) for the points

`displayArrowLabels`
: if TRUE, displays arrows labels (only with biplot)

`arrowLabelSize`
: size of arrow labels (default: 3.88 as in `geom_text()`)

`repelArrowLabels`
: if TRUE, uses `ggrepel::geom_text_repel()` instead of `ggplot2::geom_text()` for the arrows (only with biplot)

`arrowThreshold`
: (only with biplot), arrows will be made barely visible if their length is (in absolute value) less than this threshold.

`flipXAxis`
: if TRUE, take the opposite of x values (provided as it might ease low dimensional projection comparisons)

`flipYAxis`
: if TRUE, take the opposite of y values (provided as it might ease low dimensional projection comparisons)

`displayPseudoRSq`
: if TRUE, display pseudo RSquare in subtitle, on top of nb of dimensions

`...`
: additional parameters passed to `ggrepel::geom_text_repel()` (if used)

Value

a ggplot object

See Also

ggplotSampleMDSWrapBiplots, ggplotSampleMDSShepard, computeMetricMDS

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
```

```
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

# As there are only 2 samples in OMIP021Samples dataset,
# we create artificial samples that are random combinations of both samples

ffList <- c(
    flowCore::flowSet_to_list(OMIP021Trans),
    lapply(3:5,
           FUN = function(i) {
               aggregateAndSample(
                   OMIP021Trans,
                   seed = 10*i,
                   nTotalEvents = 5000)[,1:22]
           }))

fsNames <- c("Donor1", "Donor2", paste0("Agg",1:3))
names(ffList) <- fsNames

fsAll <- as(ffList,"flowSet")

flowCore::pData(fsAll)$type <- factor(c("real", "real", rep("synthetic", 3)))
flowCore::pData(fsAll)$grpId <- factor(c("D1", "D2", rep("Agg", 3)))

# calculate all pairwise distances

pwDist <- pairwiseEMDDist(fsAll,
                          channels = c("FSC-A", "SSC-A"),
                          verbose = FALSE)

# compute Metric MDS object with explicit number of dimensions
mdsObj <- computeMetricMDS(pwDist, nDim = 4, seed = 0)

dim <- nDim(mdsObj) # should be 4

#' # compute Metric MDS object by reaching a target pseudo RSquare
mdsObj2 <- computeMetricMDS(pwDist, seed = 0, targetPseudoRSq = 0.999)



# plot mds projection on axes 1 and 2,
# use 'grpId' for colour, 'type' for shape, and no label

p_12 <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(1,2),
    pDataForColour = "grpId",
```

```
    pDataForShape = "type")

# plot mds projection on axes 3 and 4,
# use 'grpId' for colour, and 'name' as point label

p_34 <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(3,4),
    pDataForColour = "grpId",
    pDataForLabel = "name")

# plot mds projection on axes 1 and 2,
# use 'group' for colour, 'type' for shape, and 'name' as point label
# have sample point size reflecting 'stress'
# i.e. quality of projection w.r.t. distances to other points

p12_Stress <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(1,2),
    pDataForColour = "grpId",
    pDataForLabel = "name",
    pDataForShape = "type",
    pointSizeReflectingStress = TRUE)

# try to associate axes with median of each channel
# => use bi-plot

extVars <- channelSummaryStats(
    fsAll,
    channels = c("FSC-A", "SSC-A"),
    statFUNs = stats::median)


bp_12 <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(1,2),
    biplot = TRUE,
    extVariables = extVars,
    pDataForColour = "grpId",
    pDataForShape = "type",
    seed = 0)

bp_34 <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(3,4),
    biplot = TRUE,
    extVariables = extVars,
    pDataForColour = "grpId",
    pDataForLabel = "name",
```

```
    seed = 0)
```

`ggplotSampleMDSShepard` *Plot of Metric MDS object - Shepard diagram*

Description

`ggplotSampleMDSShepard` uses ggplot2 to provide plot of Metric MDS results. Shepard diagram provides a scatter plot of :

- on the x axis, the high dimensional pairwise distances between each sample pairs
- on the y axis, the corresponding pairwise distances in the obtained low dimensional projection

Usage

```
ggplotSampleMDSShepard(
  mdsObj,
  nDim,
  title = "Multi Dimensional Scaling - Shepard's diagram",
  pointSize = 0.5,
  lineWidth = 0.5,
  displayPseudoRSq = TRUE
)
```

Arguments

`mdsObj`	a MDS object, output of the `computeMetricMDS()` method.
`nDim`	(optional) number of dimensions to use when calculating Shepard's diagram and pseudoRSquare. If missing, it will be set equal to the number of projection dimensions as calculated in `mdsObj`
`title`	title to give to the plot
`pointSize`	point size in plot
`lineWidth`	line width in plot
`displayPseudoRSq`	if TRUE, display pseudo RSquare in subtitle, on top of nb of dimensions

Value

a ggplot object

See Also

ggplotSampleMDS, computeMetricMDS

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

ffList <- c(
    flowCore::flowSet_to_list(OMIP021Trans),
    lapply(3:5,
           FUN = function(i) {
               aggregateAndSample(
                   OMIP021Trans,
                   seed = 10*i,
                   nTotalEvents = 5000)[,1:22]
           }))

fsNames <- c("Donor1", "Donor2", paste0("Agg",1:3))
names(ffList) <- fsNames

fsAll <- as(ffList,"flowSet")

flowCore::pData(fsAll)$type <- factor(c("real", "real", rep("synthetic", 3)))
flowCore::pData(fsAll)$grpId <- factor(c("D1", "D2", rep("Agg", 3)))

# calculate all pairwise distances

pwDist <- pairwiseEMDDist(fsAll,
                          channels = c("FSC-A", "SSC-A"),
                          verbose = FALSE)

# compute Metric MDS object with explicit number of dimensions
mdsObj <- computeMetricMDS(pwDist, nDim = 4, seed = 0)

dim <- nDim(mdsObj) # should be 4

#' # compute Metric MDS object by reaching a target pseudo RSquare
mdsObj2 <- computeMetricMDS(pwDist, seed = 0, targetPseudoRSq = 0.999)

# Shepard diagrams
```

```
p2D <- ggplotSampleMDSShepard(
    mdsObj,
    nDim = 2,
    pointSize = 1,
    title = "Shepard with 2 dimensions")

p3D <- ggplotSampleMDSShepard(
    mdsObj,
    nDim = 3,
    title = "Shepard with 3 dimensions")
    #'
pDefD <- ggplotSampleMDSShepard(
    mdsObj,
    title = "Shepard with default nb of dimensions")
```

`ggplotSampleMDSWrapBiplots` *SampleMDS biplot wrapping*

Description

`ggplotSampleMDSWrapBiplots` calls `ggplotSampleMDS` repeatly to generate biplots with different sets of external variables and align them in a grid using the `patchwork` package, in a similar fashion as `ggplot2::facet_wrap()` does.

Usage

```
ggplotSampleMDSWrapBiplots(
  mdsObj,
  extVariableList,
  ncol = NULL,
  nrow = NULL,
  byrow = NULL,
  displayLegend = TRUE,
  ...
)
```

Arguments

`mdsObj`	a MDS object, output of the `computeMetricMDS()` method
`extVariableList`	should be a named list of external variable matrices Each element of the list should be a matrix with named columns corresponding to the variables. The number of rows should be the same as the number of samples.
`ncol`	passed to `patchwork::wrap_plots()`
`nrow`	passed to `patchwork::wrap_plots()`

byrow	passed to patchwork::wrap_plots()
displayLegend	if FALSE, will de-active the legend display
...	additional parameters passed to ggplotSampleMDS() (if used)

Value

a ggplot object

See Also

ggplotSampleMDS, ggplotSampleMDSShepard, computeMetricMDS

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

# As there are only 2 samples in OMIP021Samples dataset,
# we create artificial samples that are random combinations of both samples

ffList <- c(
    flowCore::flowSet_to_list(OMIP021Trans),
    lapply(3:5,
           FUN = function(i) {
               aggregateAndSample(
                   OMIP021Trans,
                   seed = 10*i,
                   nTotalEvents = 5000)[,1:22]
           }))

fsNames <- c("Donor1", "Donor2", paste0("Agg",1:3))
names(ffList) <- fsNames

fsAll <- as(ffList,"flowSet")

flowCore::pData(fsAll)$type <- factor(c("real", "real", rep("synthetic", 3)))
flowCore::pData(fsAll)$grpId <- factor(c("D1", "D2", rep("Agg", 3)))
```

```
# calculate all pairwise distances

pwDist <- pairwiseEMDDist(fsAll,
                          channels = c("FSC-A", "SSC-A"),
                          verbose = FALSE)

# compute Metric MDS object with explicit number of dimensions
mdsObj <- computeMetricMDS(pwDist, nDim = 4, seed = 0)

dim <- nDim(mdsObj) # should be 4

#' # compute Metric MDS object by reaching a target pseudo RSquare
mdsObj2 <- computeMetricMDS(pwDist, seed = 0, targetPseudoRSq = 0.999)

# plot mds projection on axes 1 and 2,
# use 'group' for colour, 'type' for shape, and no label

p_12 <- ggplotSampleMDS(
    mdsObj = mdsObj,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(1,2),
    pDataForColour = "grpId",
    pDataForShape = "type")

# try to associate axes with median or std deviation of each channel
# => use bi-plots

extVarList <- channelSummaryStats(
    fsAll,
    channels = c("FSC-A", "SSC-A"),
    statFUNs = c("median" = stats::median,
                 "std.dev" = stats::sd))

bpFull <- ggplotSampleMDSWrapBiplots(
    mdsObj = mdsObj,
    extVariableList = extVarList,
    pData = flowCore::pData(fsAll),
    projectionAxes = c(1,2),
    pDataForColour = "group",
    pDataForShape = "type",
    seed = 0)
```

`MDS-class` *MDS class*

Description

Class representing Multi Dimensional Scaling (MDS) projection.

returns the value of the stress criterion, minimized by the SMACOF algorithm.

returns a vector of nPoints dimension, containing the stress indicator per point. The `stress` minimization criterion can indeed be allocated per represented point. The more the stress of a particular point, the less accurate its distances w.r.t. the other points.

Usage

```
## S4 method for signature 'MDS'
show(object)

nDim(x)

nPoints(x)

pwDist(x)

projections(x)

projDist(x)

stress(x)

spp(x)

eigenVals(x)

pctvar(x)

RSq(x)

RSqVec(x)

GoF(x)

smacofRes(x)
```

Arguments

`object`	a `MDS` object
`x`	a `MDS` object

Value

nothing

Slots

`nDim numeric`, nb of dimensions of the projection

`pwDist` An object of class `dist` storing the triangular relevant part of the symmetric, zero diagonal pairwise distance matrix (nPoints * nPoints), BEFORE projection.

`proj` The projection matrix, resulting from MDS

`projDist` An object of class `dist` storing the triangular relevant part of the symmetric, zero diagonal pairwise distance matrix (nPoints * nPoints), AFTER projection.

`eigen` `numeric`, vector of `nDim` length, containing the eigen values of the PCA that is applied after the Smacof algorithm.

`pctvar` `numeric`, vector of `nDim` length, containing the percentage of explained variance per axis.

`RSq` `numeric`, vector of pseudo R square indicators, as a function of number of dimensions. `RSq[nDim]` is the global pseudo R square, as displayed on plots.

`GoF` `numeric`, vector of goodness of fit indicators, as a function of number of dimensions. `GoF[nDim]` is the global goodness of fit.

Note pseudo R square and goodness of fit indicators are essentially the same indicator, only the definition of total sum of squares differ:

- for pseudo RSq: TSS is calculated using the mean pairwise distance as minimum
- for goodness of fit: TSS is calculated using 0 as minimum

`smacofRes` an object of class 'smacofB' containing the algorithmic optimization results, for example stress and stress per point, as returned by `smacof::smacofSym()` method.

Examples

```
nHD <- 10
nLD <- 2
nPoints <- 20

# generate uniformly distributed points in 10 dimensions
points <- matrix(
    data = runif(n = nPoints * nHD),
    nrow = nPoints)

# calculate euclidian distances
pwDist  <- dist(points)

# compute Metric MDS object by reaching a target pseudo RSquare
mdsObj <- computeMetricMDS(pwDist, targetPseudoRSq = 0.95)

show(mdsObj)
```

`pairwiseEMDDist` *Pairwise Earth Mover's Distance calculation*

Description

Computation of all EMD between pairs of flowFrames belonging to a flowSet. This method provides three different input modes:

- the user provides directly a flowCore::flowSet loaded in memory (RAM).
- the user provides directly a list of expression matrices loaded in RAM, of which the column names are the channel/marker names
- the user provides (1.) a number of samples `nSamples`; (2.) an ad-hoc function that takes as input an index between 1 and `nSamples`, and codes the method to load the corresponding expression matrix in memory; Optional row and column ranges can be provided to limit the calculation to a specific rectangle of the matrix. These i.e. can be specified as a way to split heavy calculations of large distance matrices on several computation nodes.

Usage

```
pairwiseEMDDist(
  x,
  rowRange = c(1, nSamples),
  colRange = c(min(rowRange), nSamples),
  loadExprMatrixFUN = NULL,
  loadExprMatrixFUNArgs = NULL,
  channels = NULL,
  verbose = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  BPOPTIONS = BiocParallel::bpoptions(packages = c("flowCore")),
  binSize = 0.05,
  minRange = -10,
  maxRange = 10
)
```

Arguments

`x`	can be: • a flowCore::flowSet • a list of expression matrices (Double matrix with named columns) • the number of samples (integer >=1)
`rowRange`	the range of rows of the distance matrix to be calculated
`colRange`	the range of columns of the distance matrix to be calculated
`loadExprMatrixFUN`	the function used to translate an integer index into an expression matrix. In other words, the function should code how to load the `index`th expression matrix into memory. IMPORTANT: the expression matrix index should be the first function argument and should be named `exprMatrixIndex`.
`loadExprMatrixFUNArgs`	(optional) a named list containing additional input parameters of `loadExprMatrixFUN()`
`channels`	which channels (integer index(ices) or character(s)):

- if it is a character vector, it can refer to either the channel names, or the marker names
- if it is a numeric vector, it refers to the indexes of channels in `fs`
- if NULL all scatter and fluorescent channels of `fs` #' will be selected

`verbose`	if `TRUE`, output a message after each single distance calculation
`BPPARAM`	sets the `BPPARAM` back-end to be used for the computation. If not provided, will use `BiocParallel::SerialParam()` (no task parallelization)
`BPOPTIONS`	sets the BPOPTIONS to be passed to `bplapply()` function. Note that if you use a `SnowParams` back-end, you need to specify all the packages that need to be loaded for the different CytoProcessingStep to work properly (visibility of functions). As a minimum, the `flowCore` package needs to be loaded. (hence the default `BPOPTIONS = bpoptions(packages = c("flowCore"))`)
`binSize`	size of equal bins to approximate the marginal distributions.
`minRange`	minimum value taken when approximating the marginal distributions
`maxRange`	maximum value taken when approximating the marginal distributions

Value

a distance matrix of pairwise distances (full symmetric with 0. diagonal)

Examples

```
library(CytoPipeline)

data(OMIP021Samples)

# estimate scale transformations
# and transform the whole OMIP021Samples

transList <- estimateScaleTransforms(
    ff = OMIP021Samples[[1]],
    fluoMethod = "estimateLogicle",
    scatterMethod = "linearQuantile",
    scatterRefMarker = "BV785 - CD3")

OMIP021Trans <- CytoPipeline::applyScaleTransforms(
    OMIP021Samples,
    transList)

# calculate pairwise distances using only FSC-A & SSC-A channels
pwDist <- pairwiseEMDDist(
    x = OMIP021Trans,
    channels = c("FSC-A", "SSC-A"))
```

Index