Package ‘SpaceTrooper’

February 21, 2026

Type Package

Title SpaceTrooper performs Quality Control analysis of Image-Based spatial

Version 1.0.1

Description SpaceTrooper performs Quality Control analysis using data driven GLM models of Image-Based spatial data, providing exploration plots, QC metrics computation, outlier detection.
It implements a GLM strategy for the detection of low quality cells in imaging-based spatial data (Transcriptomics and Proteomics).
It additionally implements several plots for the visualization of imaging based polygons through the ggplot2 package.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.4.0), SpatialExperiment

Imports DropletUtils, S4Vectors, SummarizedExperiment, arrow, data.table, dplyr, e1071, ggplot2, ggpubr, robustbase, scater, scuttle, sf, sfheaders, cowplot, glmnet, rhdf5, methods, rlang, SpatialExperimentIO

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

biocViews Software, Transcriptomics, GeneExpression, QualityControl, Spatial, SingleCell, DataImport, ImmunoOncology

RoxygenNote 7.3.3

VignetteBuilder knitr

BugReports https://github.com/drighelli/SpaceTrooper/issues

URL https://github.com/drighelli/SpaceTrooper

Config/testthat/edition 3

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

git_branch RELEASE_3_22

git_last_commit 098bd1e

git_last_commit_date 2025-12-12

Repository Bioconductor 3.22

Date/Publication 2026-02-20

Author Dario Righelli [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1504-3583>),
Benedetta Banzi [aut],
Oriana Romano [ctb],
Matteo Merchionni [ctb],
Mattia Forcato [ctb],
Silvio Bicciato [aut],
Davide Risso [ctb]

Maintainer Dario Righelli <dario.righelli@gmail.com>

Contents

`.addPolygonsToCD` *.addPolygonsToCD*

Description

This function enriches a DataFrame (e.g., from colData) with matching polygon geometries.

Usage

```
.addPolygonsToCD(cd, polygons, polygonsCol = "polygons")
```

Arguments

`cd`	A DataFrame containing at least ‘fov‘ and ‘cellID‘ columns.
`polygons`	An sf object with matching ‘fov‘ and ‘cellID‘ columns.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Value

A DataFrame identical to ‘cd‘, but row-subset to cells present in ‘polygons‘ and with a new ‘polygons‘ list-column of sf geometries.

`.centroid_image_theme` *.centroid_image_theme*

Description

internal function to setup the theme for the centroid plot background

Usage

```
.centroid_image_theme(backBorder = NA)
```

Arguments

`backBorder`	color for the borders of the background (default=NA)

Value

a ggplot2 theme object

`.checkFovPositionVersion` *.checkFovPositionVersion*

Description

Check and Standardize FOV Position Column Names

This internal utility function standardizes column names of a data frame containing Field of View (FOV) positional information. It modifies column names to ensure compatibility with expected naming conventions, including support for older formats.

Specifically, it: - Renames any column containing "FOV" to "fov" - Converts columns with coordinates matching "X", "Y", or "Z" to lowercase - Replaces suffix "_px" with "_global_px" for coordinate pixel columns - If the input contains ‘x_mm‘ and ‘y_mm‘ columns, the function computes corresponding ‘x_global_px‘ and ‘y_global_px‘ values by converting from millimeters to pixels using a fixed resolution factor (0.12028 mm/pixel).

Usage

```
.checkFovPositionVersion(spe)
```

Arguments

`spe`	A ‘SpatialExperiment‘ containing FOV position information in the metadata to be standardized.

Value

A ‘SpatialExperiment‘ with updated and standardized column names for the metadata ‘fov_position‘ ‘data.frame‘.

`.checkPolygonsValidity`

.checkPolygonsValidity

Description

checks validity on a geometry of ‘sf‘ object. It removes multipolygons when ‘keepMultiPol‘ is ‘FALSE‘

Usage

```
.checkPolygonsValidity(
  sf,
  geometry = NULL,
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

`sf`	An ‘sf‘ class object containing the spatial data.
`geometry`	character for the geometry to check validity, if ‘NULL‘ it checks the active geometry (default is ‘NULL‘)
`keepMultiPol`	logical for keeping/removing moltipolygons, if any (default is ‘TRUE‘, so keeping the multipolygons)
`verbose`	logical to print verbose output (default is ‘FALSE‘)

Details

In case geometry is NULL validity is checked on the active geometry, otherwise it is checked on the passed geometry without changing the active geometry of the sf object. In case of not valid polygons, these are removed. If keeMultiPol is FALSE, possible detected multipolygons are removed.

Value

An ‘sf‘ object with valid geometries, possibly with multipolygons removed.

`.computeBorderDistanceCosMx`

.computeBorderDistanceCosMx

Description

Calculates the minimum distance of each cell to the field-of-view border and adds it to ‘colData‘.

Usage

```
.computeBorderDistanceCosMx(
  spe,
  xwindim = metadata(spe)$fov_dim[["xdim"]],
  ywindim = metadata(spe)$fov_dim[["ydim"]]
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with CosMx data.
`xwindim`	Width of FOV in x (default from ‘metadata(spe)$fov_dim‘).
`ywindim`	Height of FOV in y (default from ‘metadata(spe)$fov_dim‘).

Value

A ‘SpatialExperiment‘ object with ‘dist_border‘ columns in ‘colData‘.

`.createPolygons` *.createPolygons*

Description

This internal function creates polygons from a data.frame or similar object.

Usage

```
.createPolygons(
  spat_obj,
  x = NULL,
  y = NULL,
  polygon_id = NULL,
  geometry = "Geometry"
)
```

Arguments

`spat_obj`	A data frame or similar object containing spatial data.
`x`	A character vector specifying the x-coordinates.
`y`	A character vector specifying the y-coordinates.
`polygon_id`	A character string specifying the polygon ID.

Value

An ‘sf‘ object containing the created polygons.

`.dark_theme` *.dark_theme*

Description

internal function to setup the black background theme for the First Filter plot

Usage

```
.dark_theme(fillColor = "black", foreColor = "white")
```

Arguments

`fillColor`	color to fill the element_rect (default is "black")
`foreColor`	color for all the other elements (default is "white")

Value

a ggplot2 theme object

`.fov_image_theme` *.fov_image_theme*

Description

internal function to setup the theme for the fov background on the whole image

Usage

```
.fov_image_theme(backColor = "black", backBorder = NA, titleCol = "white")
```

Arguments

`backColor`	not used
`backBorder`	color for the borders of the background (default=NA)
`titleCol`	character indicating the color of the title

Value

a ggplot2 theme object

.getActiveGeometryName *.getActiveGeometryName*

Description

.getActiveGeometryName

Usage

```
.getActiveGeometryName(sf)
```

Arguments

`sf`	an sf object

Value

character with the name of the active geometry

Examples

```
example(readPolygonsCosmx)
.getActiveGeometryName(polygons)
```

.light_theme *.light_theme*

Description

internal function to setup the white background theme for the First Filter plot

Usage

```
.light_theme(fillColor = "white", foreColor = "black")
```

Value

a ggplot2 theme object

.negative_image_theme *.negative_image_theme*

Description

internal function to setup the theme for the negative background for negative plots

Usage

```
.negative_image_theme(fillColor = "black", foreColor = "white")
```

Arguments

`fillColor`	color to fill the element_rect (default is "black")
`foreColor`	color for all the other elements (default is "white")

Value

a ggplot2 theme object

.renameGeometry *.renameGeometry*

Description

renames the ‘from‘ to ‘to‘ geometry of the ‘sf‘ object. If ‘activate‘ is ‘TRUE‘ it set as the active geometry the new geometry name. Default behaviour is to check if the renamed geometry is already active and leave it as active with the new name.

Usage

```
.renameGeometry(sf, from, to, activate = FALSE)
```

Arguments

`sf`	an sf object with the ‘from‘ geometry
`from`	character indicating the name of the geometry to change
`to`	character indicating the new name of the geometry
`activate`	logical indicating if the renamed geometry has to be activated

Value

an sf object

Examples

```
example(readPolygonsCosmx)
.renameGeometry(polygons, "global", "global1")
```

`.setActiveGeometry` *.setActiveGeometry*

Description

.setActiveGeometry

Usage

```
.setActiveGeometry(sf, name)
```

Arguments

`sf`	an sf object
`name`	character for the geometry to activate

Value

an sf object

Examples

```
example(readPolygonsCosmx)
.setActiveGeometry(polygons, "local")
```

`addPolygonsToSPE` *addPolygonsToSPE*

Description

This function adds polygon data to a ‘SpatialExperiment‘ object.

Usage

```
addPolygonsToSPE(spe, polygons, polygonsCol = "polygons")
```

Arguments

`spe`	A ‘SpatialExperiment‘ object to which polygons will be added.
`polygons`	An ‘sf‘ object containing the polygon data.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Value

The ‘SpatialExperiment‘ object with polygons added to the ‘colData‘.

Examples

```
example(readCosmxSPE)
polygons <- readPolygonsCosmx(metadata(spe)$polygons)
spe <- addPolygonsToSPE(spe, polygons)
spe$polygons
```

`checkOutliers` *checkOutliers*

Description

Checks if computed outliers meet the minimum numerical requirement, being at least 0.1 If the requirement is not met, the variable is removed from the formula.

Usage

```
checkOutliers(spe, verbose = FALSE)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial omics data.
`verbose`	Logical. If ‘TRUE‘, prints how many outliers were found for each metric.

Details

The function checks if computed outliers for each metric meet the minimum number to get the metric included in the QC score formula. If verbose is TRUE, it also prints how many outliers were found for each metric.

Value

The ‘SpatialExperiment‘ object with added QCScore metric variables in the ‘metadata‘.

Examples

```
example(computeOutliersQCScore)
spe <- checkOutliers(spe, verbose = TRUE)
metadata(spe)$formula_variables
```

`computeAreaFromPolygons` *computeAreaFromPolygons*

Description

This function computes the area from polygon data.

Usage

```
computeAreaFromPolygons(polygons)
```

Arguments

`polygons`	An ‘sf‘ object containing polygon data.

Value

A ‘numeric‘ vector with the area information.

Examples

```
example(readPolygonsMerfish)
area <- computeAreaFromPolygons(polygons)
area
```

`computeAspectRatioFromPolygons` *computeAspectRatioFromPolygons*

Description

This function computes the aspect ratio (width / height) from polygon data.

Usage

```
computeAspectRatioFromPolygons(polygons)
```

Arguments

`polygons`	An ‘sf‘ object containing polygon data.

Value

A ‘numeric‘ vector with the aspect ratio information.

Examples

```
example(readPolygonsMerfish)
ar <- computeAspectRatioFromPolygons(polygons)
ar
```

`computeCenterFromPolygons` *computeCenterFromPolygons*

Description

This function computes the center coordinates on x and y axis from polygon data and adds it to the ‘colData‘. It is necessary only for Merfish.

Usage

```
computeCenterFromPolygons(polygons, coldata)
```

Arguments

`polygons`	An ‘sf‘ object containing polygon data.
`coldata`	A ‘DataFrame‘ containing the ‘colData‘ to which center coordinates information will be added.

Value

A ‘DataFrame‘ with the added center information.

Examples

```
example(readPolygonsMerfish)
coldata <- computeCenterFromPolygons(polygons, colData(spe))
colData(spe) <- coldata
```

`computeLambda` *computeLambda*

Description

Compute Optimal Ridge Regularization Parameter λ via Cross-Validation

`computeLambda` performs ridge (L2) logistic regression with cross-validation to identify the optimal regularization parameter λ for a binary response.

Usage

```
computeLambda(trainDF, modelFormula)
```

Arguments

`trainDF`	‘data.frame‘ A data frame for training that must include: Predictor columns: All columns referenced in the formula returned by ‘getModelFormula()‘. ‘qs-core_train‘ A binary (0/1) response vector to be modeled.
`modelFormula`	‘character‘ A character string representing the model formula ‘~ log2CountArea + ...‘, as returned by ‘getModelFormula()‘.

Details

Internally, the function: Constructs the design matrix via `model.matrix()`, Runs ridge logistic regression cross-validation using ‘cv.glmnet‘ with ‘alpha = 0‘, Extracts and returns ‘ridge_cv$lambda.min‘.

Value

‘numeric‘ The value of λ (i.e., ‘lambda.min‘) from ‘cv.glmnet‘ that minimizes the cross-validation error.

Examples

```
example(computeTrainDF)
modform <- getModelFormula(metadata(spe)$formula_variables)
best_lambda <- computeLambda(df_train, modform)
print(best_lambda)
```

`computeMissingMetricsMerfish`

computeMissingMetricsMerfish

Description

‘computeMissingMetricsMerfish()‘ takes cell metadata and boundary polygons, calculates per-cell area and aspect-ratio, and optionally appends the raw polygon geometries.

Usage

```
computeMissingMetricsMerfish(
  polFile,
  coldata,
  boundariesType = c("parquet", "HDF5"),
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

`polFile`	path to the polygon file
`coldata`	‘DataFrame‘ or ‘data.frame‘ Cell metadata with at least a ‘cell_id‘ column.
`boundariesType`	‘character(1)‘ One of ‘"HDF5"‘ or ‘"parquet"‘—passed on to ‘readPolygonsMerfish()‘.
`keepPolygons`	‘logical(1)‘ If ‘TRUE‘, cbinds the raw polygon ‘sf‘ columns onto ‘coldata‘.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Value

A ‘DataFrame‘ (or ‘data.frame‘) with: - all columns of ‘coldata‘ - ‘um_area‘: area of each cell’s polygon - ‘AspectRatio‘: width/height aspect ratio - (optionally) the polygon geometries

Examples

```
example(readMerfishSPE)
cd <- computeMissingMetricsMerfish(metadata(spe)$polygons, colData(spe),
    boundariesType="parquet")
colData(spe) <- cd
cd
```

---

computeMissingMetricsXenium *computeMissingMetricsXenium*

Description

Compute Missing Metrics for Xenium Data

This function computes missing metrics, such as the aspect ratio, from polygon data in a Xenium dataset and optionally appends the polygon data to the resulting ‘colData‘.

Usage

```
computeMissingMetricsXenium(
  polFile,
  colData,
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

`polFile`	A character string specifying the file path to the polygon data.
`colData`	A ‘DataFrame‘ containing the ‘colData‘ for the Xenium dataset.
`keepPolygons`	A logical value indicating whether to keep the polygon data in the resulting ‘colData‘. Default is ‘FALSE‘.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Details

The function reads the polygon data from the specified file, computes the aspect ratio for each polygon, and merges these metrics with the provided ‘colData‘. Optionally, the polygon data can be kept in the returned ‘colData‘.

Value

A ‘DataFrame‘ containing the updated ‘colData‘ with computed metrics. If ‘keepPolygons‘ is ‘TRUE‘, the polygon data is also included.

Examples

```
example(readXeniumSPE)
colData(spe) <- computeMissingMetricsXenium(metadata(spe)$polygons,
    colData(spe), keepPolygons=TRUE)
```

`computeOutliersQCScore` *computeOutliersQCScore*

Description

Compute outlier cells for each metric that can be used in QC score formula for SpatialExperiment.

This function calculates outlier cells for each variable specified in ‘metricList‘ for a ‘SpatialExperiment‘. Log2CountArea must be present in the ‘colData‘ of the ‘SpatialExperiment‘ object as a minimum requirement. The user can choose which metrics to include among the following: Area_um, log2Ctrl_total_ratio, log2AspectRatio. For Xenium and Merfish datasets, log2AspectRatio is automatically removed from the formula.

Usage

```
computeOutliersQCScore(
  spe,
 metricList = c("log2CountArea", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio")
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial omics data.
`metricList`	A character vector specifying the metrics to include in the QC score formula. Default is ‘c("log2CountArea", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio")‘.

Details

The function computes outliers for each specified metric after automatically choosing the appropriate method according to the skewness of the distribution. Internally the function:

1. Calls `.checkSkw()` to choose the proper outlier detection method according to the variable skewness,
2. Calls `computeSpatialOutlier()` on each included metric to get fences,
3. Labels cells as “LOW”/“HIGH” outliers or “NO”

Value

The ‘SpatialExperiment‘ object with added outlier variables in ‘colData‘ and the temporary QC-Score metric variables that in the ‘metadata‘.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
spe <- computeOutliersQCScore(spe)
table(spe$log2CountArea_outlier_train)
```

computeQCScore *computeQCScore*

Description

Compute QC score and automatically define weights for QC score through glm training. This function computes QC score with a formula that is defined based on the metrics specified in metric_list and on the number of available outliers for each metric.

Usage

```
computeQCScore(spe, bestLambda = NULL, verbose = FALSE)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial transcriptomics data.
`bestLambda`	the best lambda typically computed using ‘computeLambda‘.
`verbose`	logical for having a verbose output. Default is FALSE.

Details

For CosMx datasets, also CosMx Protein, the QC Score formula is defined as follows:

QC score ~ count density - aspect ratio - control-total ratio

count density is total counts-to-area ratio, aspect ratio represents FOV border effect typical of CosMx datasets and control-total ratio is the aspecific signal. For each couple of variables interaction terms are computed.

For Xenium and Merscope datasets, QC score cannot depend on aspect ratio as no FOV border effect was captured through this metric.

Inclusion of metrics in the formula depends also on the number of available outliers. If the number of outliers for each metric is < 0.1 entire dataset, the metric will be excluded from the QC score formula.

To automatically define the formula coefficient weights, model training is performed through ridge regression.

Value

The ‘SpatialExperiment‘ object with added QC score in ‘colData‘.

Examples

```
example(spatialPerCellQC)
set.seed(1998)
spe <- computeQCScore(spe)
summary(spe$QC_score)
```

`computeQCScoreFlags` *computeQCScoreFlags*

Description

Compute flagged cells based on a manually chosen threshold on quality score

This function Compute flagged cells based on a manually chosen threshold on quality score stored in ‘SpatialExperiment‘ object.

Usage

```
computeQCScoreFlags(spe, qsThreshold = 0.5, useQSQuantiles = FALSE)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial transcriptomics data.
`qsThreshold`	Numeric threshold or quantile for quality score. Default ‘0.5‘.
`useQSQuantiles`	Logical; if ‘TRUE‘, treat ‘qsThreshold‘ as a percentile.

Value

The ‘SpatialExperiment‘ object with added filter flags in ‘colData‘.

Examples

```
example(computeQCScore)
spe <- computeQCScoreFlags(spe)
table(spe$low_qcscore)
# if fixed filters are defined we have an additional column
spe <- computeThresholdFlags(spe)
spe <- computeQCScoreFlags(spe)
table(spe$low_threshold_qcscore)
```

`computeSpatialOutlier` *computeSpatialOutlier*

Description

Computes outliers based on the Area (in micron) of the experiment. It gives the possibility to choose between the medcouple (mc method argument) and the MADs (scuttle method argument).

Usage

```
computeSpatialOutlier(
  spe,
  computeBy = NULL,
  method = c("mc", "scuttle", "both"),
  mcDoScale = FALSE,
  scuttleType = c("both", "lower", "higher")
)
```

Arguments

`spe`	a SpatialExperiment object with target_counts, area in micron and log2 of the aspect ratio in the ‘colData‘.
`computeBy`	character indicating a ‘colData‘ column name on which compute the outlier.
`method`	one of ‘mc‘, ‘scuttle‘, ‘both‘. Use ‘mc‘ for medcouple, ‘scuttle‘ for median absolute deviations as computed in ‘scuttle‘, ‘both‘ for computing both of them.
`mcDoScale`	logical indicating if the values to compute the medcouple for the outlier detection should be scaled (default is FALSE, as suggested by the original Medcouple authors.). See mc for further readings.
`scuttleType`	One of ‘"both"‘, ‘"lower"‘, ‘"higher"‘ for scuttle method.

Details

The medcouple method is a measure for the skeweness of univariate distribution as described in Hubert M. et al. (2008). In particular, the computed medcouple value must be in a range between -0.6 and 0.6 to computed adjusted boxplots and perform the outlier detection. For median absolute deviations (MADs) method we just wrap the isOutlier function in the scuttle package. Please see McCarthy DJ et al (2017) for further details.

Value

a SpatialExperiment object with additional column(s) (named as the column name indicated in ‘column_by‘ followed by the outlier_sc/mc nomenclature) with the outlier detection as ‘outlier.filter‘ logical class object. This allows to store the thresholds as attributes of the column. use attr(,"thresholds") to retrieve them.

Examples

```
example(spatialPerCellQC)
spe <- computeSpatialOutlier(spe, computeBy="log2CountArea", method="both")
table(spe$log2CountArea_outlier_mc)
table(spe$log2CountArea_outlier_sc)
```

`computeThresholdFlags` *computeThresholdFlags*

Description

Compute Flagged cells using fixed thresholds for SpatialExperiment.

This function calculates flagged cells only for total counts and control on total probe counts ratio using fixed thresholds for a ‘SpatialExperiment‘ object.

Usage

```
computeThresholdFlags(spe, totalThreshold = 0, ctrlTotRatioThreshold = 0.1)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial transcriptomics data.
`totalThreshold`	A numeric value for the threshold of total counts to identify cells with low counts. Default is ‘0‘.
`ctrlTotRatioThreshold`	A numeric value for the threshold of control-to-total ratio to flag cells over a certain threshold. Default is ‘0.1‘.

Details

The function flags cells basing on zero counts and control-to-total ratio to identify junk cells. It also combines these flags into a single filter flag.

Value

The ‘SpatialExperiment‘ object with added filter flags in ‘colData‘.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
spe <- computeThresholdFlags(spe)
table(spe$threshold_flags)
```

`computeTrainDF` *computeTrainDF*

Description

Build a Balanced Training Data Frame from a SpatialExperiment

`computeTrainDF` takes a `SpatialExperiment` object and assembles a balanced training set of “good” vs “bad” cells for subsequent model fitting.

Usage

```
computeTrainDF(colData, formulaVars, tech, verbose = FALSE)
```

Arguments

`colData`	A per-cell metadata table. Typically ‘as.data.frame(colData(spe))‘. Must include at least: ‘cell_id‘, raw metric columns named in ‘formulaVars‘ (e.g. ‘log2CountArea‘, ‘Area_um‘, ‘log2Ctrl_total_ratio‘, optionally ‘log2AspectRatio‘), and the corresponding outlier-label columns referenced by ‘formulaVars‘.
`formulaVars`	A named character vector mapping variable name to its outlier label column name, e.g. ‘c(log2CountArea="log2CountArea_outlier_train", ...)‘.
`tech`	Character string with the acquisition technology. Used to enable CosMx-specific handling for ‘log2AspectRatio‘. Expected values include ‘"Nanostring_CosMx"‘ or ‘"Nanostring_CosMx_Protein"‘.
`verbose`	‘logical(1)‘ (default `FALSE`) If `TRUE`, prints the number of “bad” and “good” cells selected.

Details

The function builds a training set using the variables specified in the ‘metadata‘ of the ‘SpatialExperiment‘ object.

Value

A `data.frame` with one row per cell, including: `qcscore_train` (0/1) indicating “bad” vs “good”, relevant `colData` columns used for modeling. Deduplicates and down-samples “good” cells to match the number of “bad” cells.

Examples

```
example(spatialPerCellQC)
spe <- computeOutliersQCScore(spe)
spe <- checkOutliers(spe)
df_train <- computeTrainDF(colData(spe), metadata(spe)$formula_variables,
    metadata(spe)$technology)
table(df_train$qcscore_train)
```

`createPaletteFromColData` *createPaletteFromColData*

Description

Create a Palette from colData in a SpatialExperiment Object

This function generates a palette mapping based on specified columns in the ‘colData‘ of a ‘SpatialExperiment‘ object.

Usage

```
createPaletteFromColData(spe, paletteNames, paletteColors)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with spatial transcriptomics data.
`paletteNames`	A character string specifying the column in ‘colData(spe)‘ to be used for the names in the palette.
`paletteColors`	A character string specifying the column in ‘colData(spe)‘ to be used for the colors in the palette.

Details

The function creates a new palette based on the unique combinations of values in the specified ‘paletteNames‘ and ‘paletteColors‘ columns in ‘colData(spe)‘.

Value

A character vector representing the palette mapping, where each element is a string in the format ‘"name=color"‘.

dot-addFovFromTx *.addFovFromTx*

Description

Add FOV information from transcript file to cell metadata.

This function retrieves FOV information from transcript file and appends the data to the resulting ‘colData‘.

Usage

```
.addFovFromTx(txFile, colData)
```

Arguments

`txFile`	‘character(1)‘ path to a Xenium Output tx file.
`colData`	A ‘DataFrame‘ containing the ‘colData‘ for the Xenium dataset.

Details

The function reads the transcript file then groups it by cell_id and merges the FOV information to the cell metadata in ‘colData‘. Only parquet file is supported for this operation

Value

A ‘DataFrame‘ containing the updated ‘colData‘ with FOV information.

dot-checkSkw *.checkSkw*

Description

Check skewness of metrics to choose outlier detection method.

Usage

```
.checkSkw(
  cd,
  metricList = c("log2CountArea", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio")
)
```

Arguments

`cd`	colData of ‘SpatialExperiment‘ object.
`metricList`	A character vector specifying the metrics to include in the QC score formula. Defaults are "log2CountArea", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio".

Value

A vector containing the list of chosen outlier detection method for each metric.

`dot-computeCosmxProteinTrainSet`
.computeCosmxProteinTrainSet

Description

Internal: Build Training Set for CosMx-Protein Splits a SpatialExperiment into “bad” vs “good” cells based on outliers in aspect ratio near tissue border or low count area.

Usage

```
.computeCosmxProteinTrainSet(spe)
```

Arguments

`spe`	`SpatialExperiment`

Value

A list with elements `bad` and `good`, each a data.frame with `qscore_train` and (for “good”) an `is_a_bad_boy` flag.

`dot-computeCosmxTrainSet`
.computeCosmxTrainSet

Description

Internal: Build Training Set for CosMx Splits a SpatialExperiment into “bad” vs “good” cells based on outliers in aspect ratio near tissue border or low count area.

Usage

```
.computeCosmxTrainSet(spe)
```

Arguments

`spe`	`SpatialExperiment`

Value

A list with elements `bad` and `good`, each a data.frame with `qscore_train` and (for “good”) an `is_a_bad_boy` flag.

`dot-computeXenMerTrainSet` *.computeXenMerTrainSet*

Description

Internal: Build Training Set for Xenium & MERFISH Splits a SpatialExperiment into “bad” vs “good” cells based on pre-computed outlier labels on log2CountArea.

Usage

```
.computeXenMerTrainSet(spe)
```

Arguments

`spe`	`SpatialExperiment`

Value

A list with elements `bad` and `good`, each a data.frame with `qscore_train` and (for “good”) an `is_a_bad_boy` flag.

`firstFlagPalette` *firstFlagPalette*

Description

neon color palette for firstFlagPlot

Usage

```
firstFlagPalette
```

Format

An object of class `character` of length 6.

Value

a palette for firstFlagPlot

getFencesOutlier *getFencesOutlier*

Description

Retrieve Threshold (Fence) Values from a SpatialExperiment Object

This function extracts the threshold values, also known as fences, from a specified column in the ‘colData‘ of a ‘SpatialExperiment‘ object.

Usage

```
getFencesOutlier(
  spe,
  fencesOf,
  highLow = c("both", "lower", "higher"),
  decimalRound = NULL
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`fencesOf`	A character string specifying the name of the column in ‘colData(spe)‘ from which to extract the fence values. This column should contain an ‘outlier.filter‘ object (see ‘computeSpatialOutlier“‘).
`highLow`	character indicating which fence to get if "higher", "lower" or "both" (default is "both").
`decimalRound`	An optional integer specifying the number of decimal places to which the fence values should be rounded. If ‘NULL‘, no rounding is applied. Default is ‘NULL‘.

Value

A numeric vector containing the lower and upper threshold values extracted from the specified column.

Examples

```
example(computeSpatialOutlier)
getFencesOutlier(spe, fencesOf="log2CountArea_outlier_mc")
```

`getModelFormula` *getModelFormula*

Description

Returns the right-hand side of a model formula string based on formula variables found in the ‘metadata‘ of a ‘SpatialExperiment‘ object.

Usage

```
getModelFormula(formulaVars, verbose = FALSE)
```

Arguments

`formulaVars`	A named character vector mapping variable names (e.g. ‘"log2CountArea"‘, ‘"Area_um"‘, etc.) to their corresponding outlier label columns, typically from ‘metadata(spe)$formula_variables‘.
`verbose`	Logical. If ‘TRUE‘, prints the final formula used for QC score

Value

‘character‘ A one-sided formula as a string (e.g. "~ log2CountArea + ...").

Examples

```
example(checkOutliers)
getModelFormula(metadata(spe)$formula_variables)
```

`plotCellsFovs` *plotCellsFovs*

Description

Plot cell centroids in FoVs Creates a scatter plot with cell centroids arranged in their FoVs as an overlapping grid.

Usage

```
plotCellsFovs(
  spe,
  sampleId = unique(spe$sample_id),
  pointCol = "firebrick",
  numbersCol = "black",
  alphaNumbers = 0.8,
  fovDim = metadata(spe)$fov_dim
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with ‘fov‘ in ‘colData‘.
`sampleId`	Character string identifying which sample to plot. Default: ‘unique(spe$sample_id)‘.
`pointCol`	Color for the cell centroids. Default: ‘"firebrick"‘.
`numbersCol`	Color for the FoV labels. Default: ‘"black"‘.
`alphaNumbers`	Numeric transparency for FoV labels. Default: ‘0.8‘.
`fovDim`	numeric with two named dimensions xdim, ydim. (Default is metadata(spe)$fov_dim)

Value

A ‘ggplot‘ object showing cell centroids and FoV boundaries.

Examples

```
example(readCosmxSPE)
g <- plotCellsFovs(spe)
print(g)
```

`plotCentroids` *plotCentroids*

Description

Plot Spatial Coordinates for a SpatialExperiment Object This function generates a ggplot of spatial coordinates from a ‘SpatialExperiment‘ object, optionally coloring the points by a specified column in ‘colData‘.

Usage

```
plotCentroids(
  spe,
  colourBy = NULL,
  colourLog = FALSE,
  sampleId = unique(spe$sample_id),
  isNegativeProbe = FALSE,
  palette = NULL,
  pointCol = "darkmagenta",
  size = 0.05,
  alpha = 0.8,
  aspectRatio = 1
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`colourBy`	An optional character string specifying the column in ‘colData(spe)‘ to use for coloring the points. If ‘NULL‘, all points will be colored the same.
`colourLog`	Logical to log-transform the data to enhance visualization (Default is FALSE).

`sampleId`	A character string specifying the sample identifier to be used as the plot title. (Default is the unique sample ID from ‘spe‘)
`isNegativeProbe`	A logical value indicating whether to apply a custom color gradient for negative probe data. (Default is ‘FALSE‘)
`palette`	A vector of colors to be used as a custom palette. For categorical data, this should be a vector of colors with the same length as the number of levels in ‘colourBy‘. For continuous data, this should be a vector of colors used to create a gradient.
`pointCol`	A character string specifying the color of the points when ‘colourBy‘ is ‘NULL‘. (Default is ‘"darkmagenta"‘)
`size`	A numeric value specifying the size of the points. (Default is ‘0.05‘)
`alpha`	A numeric value specifying the transparency level of the points. (Default is ‘0.2‘)
`aspectRatio`	A numeric value specifying the aspect ratio of the plot. (Default is ‘1‘)

Value

A ‘ggplot‘ object representing the spatial coordinates plot of polygon centroids.

Examples

```
example(readCosmxSPE)
g <- plotCentroids(spe, colourBy="Mean.DAPI")
print(g)
```

`plotMetricHist` *plotMetricHist*

Description

Plot a Histogram for a Given Metric in a SpatialExperiment Object

This function generates a histogram for a specified metric in a ‘SpatialExperiment‘ object.

Usage

```
plotMetricHist(
  spe,
  metric,
  fillColor = "#c0c8cf",
  useFences = NULL,
  fencesColors = c(lower = "purple4", higher = "tomato"),
  bins = 30,
  binWidth = NULL
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`metric`	A character string specifying the name of the metric (column in ‘colData(spe)‘) to plot.
`fillColor`	A character string specifying the fill color of the histogram bars. (Default is ‘"#69b3a2"‘)
`useFences`	A character string specifying the name of the column in ‘colData(spe)‘ that contains the fence thresholds (typically from an outlier filter). If ‘NULL‘, no fences will be plotted. (Default is ‘NULL‘)
`fencesColors`	A named character vector specifying the colors to use for the lower and higher fences. The names should be ‘"lower"‘ and ‘"higher"‘ . (Default is ‘c("lower"="purple4", "higher"="tomato")‘)
`bins`	An integer specifying the number of bins to use in the histogram. (Default is ‘30‘)
`binWidth`	A numeric value specifying the width of the bins. If ‘NULL‘, the bin width will be automatically determined based on the ‘bins‘ parameter. (Default is ‘NULL‘)

Value

A ‘ggplot‘ object representing the histogram of the specified metric.

Examples

```
example(readCosmxSPE)
g <- plotMetricHist(spe, metric="Mean.DAPI")
print(g)
```

`plotPolygons` *plotPolygons*

Description

Plot polygons from a ‘SpatialExperiment‘ object using ggplot2.

Usage

```
plotPolygons(
  spe,
  colourBy = "darkgrey",
  colourLog = FALSE,
  polyColumn = "polygons.global",
  sampleId = unique(spe$sample_id),
  bgColor = "white",
  fillAlpha = 1,
  palette = NULL,
  borderCol = NA,
  borderAlpha = 1,
  borderLineWidth = 0.1,
  drawBorders = TRUE
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with polygon data as an ‘sf‘ object.
`colourBy`	A column in ‘colData(spe)‘ for coloring the polygons or a string color in colors(). (Default is "darkgrey")
`colourLog`	Logical to log-transform the data to enhance visualization (Default is FALSE).
`polyColumn`	character for the name of the column where the polygons sf are stored (default is "polygons.global")
`sampleId`	Sample ID for plot title. Default is the unique sample ID.
`bgColor`	character indicating color for the background (default is "white")
`fillAlpha`	Transparency level for polygon fill. Default is ‘1‘.
`palette`	Colors to use if ‘colourBy‘ is a factor. Default is ‘NULL‘.
`borderCol`	Color of polygon borders. Default is ‘"black"‘.
`borderAlpha`	Transparency level for borders. Default is ‘1‘.
`borderLineWidth`	Width of polygon borders. Default is ‘0.1‘.
`drawBorders`	Logical; whether to draw borders. Default is ‘TRUE‘.

Value

A ‘ggplot‘ object representing the polygon plot of the spatial data.

Examples

```
example(readAndAddPolygonsToSPE)
plotPolygons(spe, colourBy="Mean.DAPI")
```

`plotQScoreTerms` *plotQScoreTerms*

Description

Plots the individual terms that combine into the quality score formula, allowing assessment of each term’s impact on the final score.

Usage

```
plotQScoreTerms(
  spe,
  sampleId = unique(spe$sample_id),
  size = 0.05,
  alpha = 0.8,
  aspectRatio = 1,
  custom = FALSE
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object with ‘quality_score‘ and term columns in ‘colData‘.
`sampleId`	Character string for plot title. Must match values in the ‘fov‘ column of ‘colData(spe)‘. Default: ‘unique(spe$sample_id)‘.
`size`	Numeric point size for the scatter plots. Default: ‘0.05‘.
`alpha`	Numeric transparency for the scatter plots. Default: ‘0.2‘.
`aspectRatio`	Numeric aspect ratio of the plots. Default: ‘1‘.
`custom`	Logical; if ‘TRUE‘, use custom polygon-derived metrics.

Value

A combined plot (via ‘cowplot::plot_grid‘) showing spatial maps of each QS term.

Examples

```
example(readAndAddPolygonsToSPE)
example(spatialPerCellQC)
p <- plotQScoreTerms(spe)
print(p)
```

`plotZoomFovsMap` *plotZoomFovsMap*

Description

Plot Zoomed-in FOVs with Map and Polygons

This function generates a plot that shows a map of all fields of view (FOVs) within a ‘SpatialExperiment‘ object, alongside a zoomed-in view of the specified FOVs with an overlay of polygons and optional coloring.

Usage

```
plotZoomFovsMap(
  spe,
  fovs = NULL,
  mapPointCol = "darkmagenta",
  mapNumbersCol = "black",
  mapAlphaNumbers = 0.8,
  title = NULL,
  ...
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`fovs`	A character vector specifying the FOVs to be zoomed in and plotted. Must match values in the ‘fov‘ column of ‘colData(spe)‘.

`mapPointCol`	A character string specifying the color of the points in the map. Default is ‘"darkmagenta"‘.
`mapNumbersCol`	A character string specifying the color of the numbers on the map. Default is ‘"black"‘.
`mapAlphaNumbers`	A numeric value specifying the transparency of the numbers on the map. Default is ‘0.8‘.
`title`	An optional character string specifying the title of the final plot. If ‘NULL‘, no title is added. Default is ‘NULL‘.
`...`	Additional arguments passed to ‘plotPolygons‘.

Details

The function first filters the ‘SpatialExperiment‘ object to the specified FOVs, generates a plot of the cells for the entire map, then creates a detailed polygon plot of the selected FOVs, and finally combines these into a single side-by-side visualization. If ‘title‘ is not ‘NULL‘, it adds a title to the combined plot.

Value

A combined plot showing a map of all FOVs with zoomed-in views of the specified FOVs and their associated polygons.

Examples

```
example(readAndAddPolygonsToSPE)
plotZoomFovsMap(spe, fovs=16, title="FOV 16")
```

`qcFlagPlots` *qcFlagPlots*

Description

Plots the flagged cells identified with first filter, based on control count on total count ratio, area in um and DAPI signal.

This function generates a plot that shows selected (FOVs) within a ‘SpatialExperiment‘ object, with cells flagged in different colors over a light or dark layout chosen by the user.

Usage

```
qcFlagPlots(
  spe,
  fov = unique(spe$fov),
  theme = c("light", "dark"),
  custom = FALSE
)
```

Arguments

spe	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
fov	An integer or numeric vector specifying the FOVs to be plotted Must match values in the ‘fov‘ column of ‘colData(spe)‘.
theme	A character string among "light" or "dark".
custom	A boolean value. If TRUE, custom polygons derived metrics will be used.

Value

A panel with multiple plots showing flagged cells for different variables.

Examples

```
example(readAndAddPolygonsToSPE)
spe <- spatialPerCellQC(spe)
spe <- computeThresholdFlags(spe)
p <- qcFlagPlots(spe, fov=16, theme="dark")
print(p)
```

`readAndAddPolygonsToSPE` *readAndAddPolygonsToSPE*

Description

Read and Add Polygons to a SpatialExperiment Object

This function reads polygon boundary data based on the technology associated with the provided SpatialExperiment (SPE) object and adds the polygons to the SPE.

Usage

```
readAndAddPolygonsToSPE(
  spe,
  polygonsCol = "polygons",
  keepMultiPol = TRUE,
  boundariesType = c("csv", "HDF5", "parquet")
)
```

Arguments

spe	A `SpatialExperiment` object. The object should contain metadata with the field `"technology"`, specifying the technology used (e.g., "Nanostring_CosMx", "Vizgen_MERFISH", or "10X_Xenium").
polygonsCol	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).
keepMultiPol	Logical. If `TRUE`, multi-polygon features will be kept when reading the boundary data. Defaults to `TRUE`.
boundariesType	Character. Specifies the type of boundary file format to read. Options are `"HDF5"` or `"parquet"`. Defaults to `"HDF5"`.

Value

A SpatialExperiment object with the added polygon data.

Examples

```
example(readCosmxSPE)
spe <- readAndAddPolygonsToSPE(spe)
colData(spe)
```

---

readCosmxSPE *readCosmxSPE*

Description

Read and Construct a SpatialExperiment Object from CosMx Data

This function reads in data from Nanostring CosMx files and constructs a ‘SpatialExperiment‘ object, optionally including polygon data.

Usage

```
readCosmxSPE(
  dirName,
  sampleName = "sample01",
  coordNames = c("CenterX_global_px", "CenterY_global_px"),
  countMatFPattern = "exprMat_file.csv",
  metadataFPattern = "metadata_file.csv",
  polygonsFPattern = "polygons.csv",
  fovPosFPattern = "fov_positions_file.csv",
  fovdims = c(xdim = 4256, ydim = 4256)
)
```

Arguments

dirName	A character string specifying the directory containing the CosMx data files.
sampleName	A character string specifying the sample name. Default is ‘"sample01"‘.
coordNames	A character vector specifying the names of the spatial coordinate columns in the data. Default is ‘c("CenterX_global_px", "CenterY_global_px")‘.
countMatFPattern	A character string specifying the pattern to match the count matrix file. Default is ‘"exprMat_file.csv"‘.
metadataFPattern	A character string specifying the pattern to match the metadata file. Default is ‘"metadata_file.csv"‘.
polygonsFPattern	A character string specifying the pattern to match the polygons file. Default is ‘"polygons.csv"‘.
fovPosFPattern	A character string specifying the pattern to match the FOV positions file. Default is ‘"fov_positions_file.csv"‘.
fovdims	A named numeric vector specifying the dimensions of the FOV in pixels. Default is ‘c(xdim=4256, ydim=4256)‘.

Details

The function reads in the specified files for count matrices, metadata, and FOV positions, and constructs a ‘SpatialExperiment‘ object. Optionally, polygon data can be read and added to the object.

readCosmxProteinSPE is a wrapper of readCosmxSPE, it only changes the technology metadata in Nanostring_CosMx_Protein.

Value

A ‘SpatialExperiment‘ object containing the read CosMx data, including count matrices, metadata, and optionally polygons.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
cospath <- system.file(file.path("extdata", "CosMx_DBKero_Tiny"),
   package="SpaceTrooper")
spe <- readCosmxSPE(cospath, sampleName="DBKero_Tiny")
```

---

readh5polygons *readh5polygons*

Description

This function reads polygon data from an HDF5 file.

Usage

```
readh5polygons(polFile)
```

Arguments

polFile	A character string specifying the file path to the HDF5 polygon data.

Value

A list containing the polygon geometries and their associated cell_id

Author(s)

Lambda Moses

readMerfishSPE *readMerfishSPE*

Description

‘readMerfishSPE()‘ imports MERFISH/Merscore outputs (counts, metadata, and optionally cell boundary polygons) from a directory and builds a SpatialExperiment object.

Usage

```
readMerfishSPE(
  dirName,
  sampleName = "sample01",
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  boundariesType = c("HDF5", "parquet"),
  countmatFPattern = "cell_by_gene.csv",
  metadataFPattern = "cell_metadata.csv",
  polygonsFPattern = "cell_boundaries.parquet",
  coordNames = c("center_x", "center_y"),
  polygonsCol = "polygons"
)
```

Arguments

`dirName`	‘character(1)‘ Path to a folder containing MERFISH output files.
`sampleName`	‘character(1)‘ Identifier to assign to the ‘sample_id‘ field in the returned object. Default: ‘"sample01"‘.
`computeMissingMetrics`	‘logical(1)‘ If ‘TRUE‘, compute area and aspect-ratio metrics from the cell boundary polygons. Default: ‘TRUE‘.
`keepPolygons`	‘logical(1)‘ If ‘TRUE‘, attach raw polygon geometries as extra columns in ‘colData‘. Default: ‘FALSE‘.
`boundariesType`	‘character(1)‘ One of ‘"HDF5"‘ or ‘"parquet"‘. If ‘"HDF5"‘, uses a folder of HDF5 polygon files; if ‘"parquet"‘, reads a single Parquet file of boundaries.
`countmatFPattern`	‘character(1)‘ Regex passed to ‘list.files()‘ to find the count matrix CSV. Default: ‘"cell_by_gene.csv"‘.
`metadataFPattern`	‘character(1)‘ Pattern to find the cell metadata CSV. Default: ‘"cell_metadata.csv"‘.
`polygonsFPattern`	‘character(1)‘ Pattern to find the cell boundaries file. Default: ‘"cell_boundaries.parquet"‘.
`coordNames`	‘character(2)‘ Names of the columns in ‘colData‘ that store X/Y spatial coordinates. Default: ‘c("center_x", "center_y")‘.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Value

A ‘SpatialExperiment‘ object with: - ‘assays$counts‘: gene × cell count matrix - ‘colData‘: per-cell metadata (including computed metrics) - spatial coordinates named by ‘coordNames‘ - ‘metadata$polygons‘: path to the boundaries file - ‘metadata$technology‘: ‘"Vizgen_MERFISH"‘.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
path <- system.file("extdata", "Merfish_Tiny",
                    package = "SpaceTrooper")
spe <- readMerfishSPE(
  dirName = path,
  sampleName = "Patient2",
  keepPolygons = TRUE,
  boundariesType = "parquet"
)
spe
```

`readPolygons` *readPolygons*

Description

Read and Validate Polygons from a File

This function reads polygon data from a specified file, validates the polygons, and returns them as an ‘sf‘ object. It supports multiple file formats and can handle both global and local coordinates.

Usage

```
readPolygons(
  polygonsFile,
  type = c("csv", "parquet", "h5"),
  x = c("x_global_px", "vertex_x"),
  y = c("y_global_px", "vertex_y"),
  xloc = "x_local_px",
  yloc = "y_local_px",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

`polygonsFile`	A character string specifying the path to the polygon file.
`type`	A character string specifying the file type. Supported types are ‘"csv"‘, ‘"parquet"‘, and ‘"h5"‘. Default is ‘"csv"‘.
`x`	A character vector specifying the column names for the x-coordinates in the polygon data. Default is ‘c("x_global_px", "vertex_x")‘.

`y`	A character vector specifying the column names for the y-coordinates in the polygon data. Default is ‘c("y_global_px", "vertex_y")‘.
`xloc`	A character string specifying the column name for the local x-coordinates. Default is ‘"x_local_px"‘.
`yloc`	A character string specifying the column name for the local y-coordinates. Default is ‘"y_local_px"‘.
`keepMultiPol`	A logical value indicating whether to keep multipolygons during validation. Default is ‘TRUE‘.
`verbose`	A logical value indicating whether to print additional information during processing. Default is ‘FALSE‘.

Details

The function reads polygon data from the specified file and formats. It validates the polygons and handles both global and local coordinates if provided. If the file type is ‘"h5"‘, the function currently does not handle the data, as this part of the code is not implemented.

Value

An ‘sf‘ object with the loaded and validated polygons.

Examples

```
example(readCosmxSPE)
polygons <- readPolygons(metadata(spe)$polygons)
polygons
```

`readPolygonsCosmx` *readPolygonsCosmx*

Description

This function reads polygon data specific to CosMx technology.

Usage

```
readPolygonsCosmx(
  polygonsFile,
  type = c("csv", "parquet"),
  x = "x_global_px",
  y = "y_global_px",
  xloc = "x_local_px",
  yloc = "y_local_px",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

`polygonsFile`	A character string specifying the file path to the polygon data.
`type`	A character string specifying the file type ("csv" or "parquet").
`x`	A character string specifying the x-coordinate column.
`y`	A character string specifying the y-coordinate column.
`xloc`	A character string specifying the local x-coordinate column.
`yloc`	A character string specifying the local y-coordinate column.
`keepMultiPol`	A logical value indicating whether to keep multipolygons.
`verbose`	A logical value indicating whether to print additional information.

Value

An ‘sf‘ object containing the CosMx polygon data.

Examples

```
example(readCosmxSPE)
polygons <- readPolygonsCosmx(metadata(spe)$polygons)
polygons
```

`readPolygonsMerfish` *readPolygonsMerfish*

Description

This function reads polygon data specific to MERFISH technology.

Usage

```
readPolygonsMerfish(
  polygons,
  type = c("parquet", "HDF5"),
  keepMultiPol = TRUE,
  hdf5pattern = "hdf5",
  zLev = 3L,
  zcolumn = "ZIndex",
  geometry = "Geometry",
  verbose = FALSE
)
```

Arguments

`polygons`	A character string specifying the folder containing the polygon data files in case of HDF5, or a path to a parquet file (see ‘type‘).
`type`	A character string specifying the file type("HDF5" or "parquet"). Default is parquet.
`keepMultiPol`	A logical value indicating whether to keep multipolygons.
`hdf5pattern`	A character string specifying the pattern to match HDF5 files.

`zLev`	An integer specifying the Z level to filter the data. Default is ‘3L‘.
`zcolumn`	A character string specifying the column name for the Z index.
`geometry`	A character string specifying the geometry column name.
`verbose`	A logical value indicating whether to print additional information.

Value

An ‘sf‘ object containing the MERFISH polygon data.

Examples

```
example(readMerfishSPE)
polygons <- readPolygonsMerfish(metadata(spe)$polygons, type="parquet")
polygons
```

`readPolygonsXenium` *readPolygonsXenium*

Description

This function reads polygon data specific to Xenium technology.

Usage

```
readPolygonsXenium(
  polygonsFile,
  type = c("parquet", "csv"),
  x = "vertex_x",
  y = "vertex_y",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

`polygonsFile`	A character string specifying the file path to the polygon data.
`type`	A character string specifying the file type ("parquet" or "csv"). Default is parquet.
`x`	A character string specifying the x-coordinate column.
`y`	A character string specifying the y-coordinate column.
`keepMultiPol`	A logical value indicating whether to keep multipolygons.
`verbose`	A logical value indicating whether to print additional information.

Value

An ‘sf‘ object containing the Xenium polygon data.

Examples

```
example(readXeniumSPE)
polygons <- readPolygonsXenium(metadata(spe)$polygons, type="parquet")
polygons
```

readXeniumSPE	*Load data from a 10x Genomics Xenium experiment*

Description

Creates a [‘SpatialExperiment‘] from an unzipped Xenium Output Bundle directory containing spatial gene expression data.

Usage

```
readXeniumSPE(
  dirName,
  sampleName = "sample01",
  type = c("HDF5", "sparse"),
  coordNames = c("x_centroid", "y_centroid"),
  boundariesType = c("parquet", "csv"),
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  countsFilePattern = "cell_feature_matrix",
  metadataFPattern = "cells.csv.gz",
  polygonsFPattern = "cell_boundaries",
  polygonsCol = "polygons",
  txPattern = "transcripts",
  addFOVs = FALSE
)
```

Arguments

`dirName`	‘character(1)‘ Path to a Xenium Output Bundle directory.
`sampleName`	‘character(1)‘ Sample identifier to assign to ‘sample_id‘. Default: ‘"sample01"‘.
`type`	‘character(1)‘ One of ‘"HDF5"‘ or ‘"sparse"‘; method to read the feature matrix.
`coordNames`	‘character(2)‘ Names of X/Y spatial coordinate columns. Default: ‘c("x_centroid", "y_centroid")‘.
`boundariesType`	‘character(1)‘ One of ‘"parquet"‘ or ‘"csv"‘; format of the polygon file.
`computeMissingMetrics`	‘logical(1)‘ If ‘TRUE‘, compute area and aspect-ratio from boundary polygons.
`keepPolygons`	‘logical(1)‘ If ‘TRUE‘, append raw polygon geometries to ‘colData‘.
`countsFilePattern`	‘character(1)‘ Pattern to locate the feature matrix file. Default: ‘"cell_feature_matrix"‘.
`metadataFPattern`	‘character(1)‘ Pattern to locate the cell metadata file. Default: ‘"cells"‘.
`polygonsFPattern`	‘character(1)‘ Pattern to locate the cell boundaries file. Default: ‘"cell_boundaries"‘.
`polygonsCol`	‘character(1)‘ Name of the polygons column to add to ‘colData‘. Default: ‘"polygons"‘.
`txPattern`	‘character(1)‘ Pattern (base filename, without extension) to locate the transcript file (usually a ‘.parquet‘ file) from which to extract Field-Of-View (FOV) information for each cell. Default: ‘"transcripts"‘.

`addFOVs`	‘logical(1)‘ If ‘TRUE‘, extract Field-Of-View (FOV) information from the transcript file (as located by ‘txPattern‘) and append it to cell metadata (‘colData‘). Default: ‘FALSE‘.

Details

readXeniumSPE

Expects the unzipped bundle to contain an ‘outs/‘ folder with: - ‘cell_feature_matrix.h5‘ or ‘cell_feature_matrix/‘ - ‘cells.csv.gz‘

Value

A [‘SpatialExperiment‘] object with assays, ‘colData‘, spatial coordinates, and ‘metadata$polygons‘ & ‘metadata$technology‘.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
xepath <- system.file(
  "extdata", "Xenium_small", package = "SpaceTrooper"
)
(spe <- readXeniumSPE(
  dirName = xepath,
  keepPolygons = TRUE
))
```

`spatialPerCellQC` *spatialPerCellQC*

Description

Computes quality-control metrics for each cell and adds them to ‘colData‘.

Usage

```
spatialPerCellQC(
  spe,
  micronConvFact = 0.12,
  rmZeros = TRUE,
 negProbList = c("NegPrb", "Negative", "SystemControl", "Ms IgG1", "Rb IgG", "BLANK_",
    "NegControlProbe", "NegControlCodeword", "UnassignedCodeword", "Blank"),
  use_altexps = NULL
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial data.
`micronConvFact`	Numeric factor to convert pixels to microns. Default ‘0.12‘.
`rmZeros`	logical for removing zero counts cells (default is TRUE).
`negProbList`	Character vector of patterns to identify negative probes. Defaults include: Nanostring CosMx: ‘"NegPrb"‘, ‘"Negative"‘, ‘"SystemControl"‘ Xenium: ‘"NegControlProbe"‘, ‘"NegControlCodeword"‘, ‘"UnassignedCodeword"‘ MERFISH: ‘"Blank"‘
`use_altexps`	logical for ‘use_altexps‘ in ‘scuttle‘ package. If TRUE uses the altexps for computing some metrics on it. Useful for interoperability with ‘SpatialExperimentIO‘. (See addPerCellQC for additional details).

Details

Calculates sums and detected counts for control and target probes, computes ratio and count-area metrics, converts coords to microns for CosMx, and drops zero-count cells.

Value

A ‘SpatialExperiment‘ object with added QC metrics in ‘colData‘.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
```

`trainModel` *trainModel*

Description

Fit a Ridge Logistic Regression Model

`trainModel` fits an L2-regularized (ridge) logistic regression using **glmnet**, given a design matrix and a training data frame.

Usage

```
trainModel(modelMatrix, trainDF)
```

Arguments

`modelMatrix`	a matrix describing the model variables, tipically created with ‘getModelFormula‘ and ‘model.matrix‘ functions.
`trainDF`	‘data.frame‘ A data frame containing at least the response column ‘qscore_train‘, coded as 0/1.

Value

A `glmnet` model object fitted with `family="binomial"`, `alpha=0` (ridge), and a sequence of λ values.

Examples

```
example(computeTrainDF)
model_formula <- getModelFormula(metadata(spe)$formula_variables)
model_matrix <- model.matrix(as.formula(model_formula), data=df_train)
fit <- trainModel(model_matrix, df_train)
coef(fit, s = 0.01)
```

`updateCosmxProteinSPE` *updateCosmxProteinSPE*

Description

Update a SpatialExperiment object corresponding to Nanostring CosMx Protein data by adding metadata identifying the technology and optionally passing through file-location parameters.

Usage

```
updateCosmxProteinSPE(
  spe,
  dirName,
  sampleName = "sample01",
  coordNames = c("CenterX_global_px", "CenterY_global_px"),
  countMatFPattern = "exprMat_file.csv",
  metadataFPattern = "metadata_file.csv",
  polygonsFPattern = "polygons.csv",
  fovPosFPattern = "fov_positions_file.csv",
  fovdims = c(xdim = 4256, ydim = 4256)
)
```

Arguments

`spe`	SpatialExperiment object.
`dirName`	Directory containing CosMx Protein data files.
`sampleName`	Character sample ID. Default `"sample01"`.
`coordNames`	Character vector of length two indicating coordinate column names in the per-cell metadata. Default `c("CenterX_global_px","CenterY_global_px")`.
`countMatFPattern`	Character pattern for the counts matrix file.
`metadataFPattern`	Character pattern for the single-cell metadata file.
`polygonsFPattern`	Character pattern for the polygon file(s).
`fovPosFPattern`	Character pattern for the FOV positions file.
`fovdims`	Named numeric vector with FOV size in pixels.

Details

This function sets `metadata(spe)$technology <- "Nanostring_CosMx_Protein"`. It does not modify other assay or metadata components.

Value

SpatialExperiment object with updated technology metadata.

See Also

readCosmxProteinSPE, readCosmxSPE

Examples

```
protfolder <- system.file( "extdata", "S01_prot", package="SpaceTrooper")
spe <- SpatialExperimentIO::readCosmxSXE(dirName=protfolder,
    addParquetPaths=FALSE)
spe <- updateCosmxProteinSPE(spe, protfolder, sampleName="cosmx_prots")
```

`updateCosmxSPE` *updateCosmxSPE*

Description

Update a SpatialExperiment object derived from CosMx data by adding polygons, FOV dimensions, standardized column names, and metadata.

Usage

```
updateCosmxSPE(
  spe,
  dirName,
  sampleName = "sample01",
  polygonsFPattern = "polygons.csv",
  fovdims = c(xdim = 4256, ydim = 4256)
)
```

Arguments

`spe`	SpatialExperiment object.
`dirName`	Directory containing CosMx output files (e.g., polygon CSVs).
`sampleName`	Character scalar, sample identifier stored in `colData(spe)$sample_id`. Default `"sample01"`.
`polygonsFPattern`	Character, pattern used by `list.files()` to locate polygon files. Default `"polygons.csv"`.
`fovdims`	Named numeric vector with entries `xdim` and `ydim` representing the FOV dimensions in pixels.

Details

The function standardizes CosMx SPE structure by: - creating unique cell names of the form `f<fov>_c<cell_ID>`; - ensuring consistent cell identifiers and sample metadata; - recording FOV dimensions, polygon paths, and technology type in `metadata(spe)`.

Value

A SpatialExperiment object with updated metadata and column names.

See Also

readCosmxSPE, readCosmxProteinSPE

Examples

```
cospath <- system.file(file.path("extdata", "CosMx_DBKero_Tiny"),
    package="SpaceTrooper")
spe <- SpatialExperimentIO::readCosmxSXE(dirName=cospath,
    addParquetPaths=FALSE)
spe <- updateCosmxSPE(spe, dirName=cospath, sampleName="DBKero_Tiny")
```

`updateXeniumSPE` *updateXeniumSPE*

Description

Update a SpatialExperiment created from 10x Genomics Xenium outputs by wiring polygons/boundaries, computing optional QC metrics, and standardizing metadata and column names. This is a thin wrapper that delegates to the internal helper '.setupXeniumSPE()'.

Usage

```
updateXeniumSPE(
  spe,
  dirName,
  sampleName = "sample01",
  boundariesType = c("parquet", "csv"),
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  polygonsFPattern = "cell_boundaries",
  polygonsCol = "polygons",
  txPattern = "transcripts",
  addFOVs = FALSE
)
```

Arguments

`spe`	SpatialExperiment object to update.
`dirName`	Directory containing Xenium outputs.
`sampleName`	Sample identifier to store (default `"sample01"`).
`boundariesType`	One of `c("parquet","csv")` indicating the source format of cell boundaries.
`computeMissingMetrics`	Logical; if `TRUE`, compute metrics that are not already present from transcripts/polygons.
`keepPolygons`	Logical; if `TRUE`, keep polygons in the resulting object (e.g., in metadata or colData, depending on implementation).

`polygonsFPattern`	Character pattern used to locate polygon files when boundaries are provided as CSVs (default `"polygons.csv"`).
`polygonsCol`	Name of the geometry/column storing polygons when reading from parquet (default `"polygons"`).
`txPattern`	Pattern (file/glob) for transcript-level files (default `"transcripts"`).
`addFOVs`	Logical; if `TRUE`, derive and attach FOV identifiers from transcript resources.

Details

This function performs input checks and then calls ‘.setupXeniumSPE()‘, which does the heavy lifting (I/O, renaming, metadata updates, metrics).

Value

Updated SpatialExperiment object.

Examples

```
xepath <- system.file("extdata", "Xenium_small", package="SpaceTrooper")
(spe <- SpatialExperimentIO::readXeniumSXE(dirName=xepath))
spe <- updateXeniumSPE(spe, dirName=xepath, boundariesType="parquet",
    computeMissingMetrics=TRUE, keepPolygons=TRUE)
```

Index