Package ‘SmartPhos’

November 4, 2025

Type Package

Title A phosphoproteomics data analysis package with an interactive ShinyApp

Version 1.1.0

Description To facilitate and streamline phosphoproteomics data analysis, we developed SmartPhos, an R package for the pre-processing, quality control, and exploratory analysis of phosphoproteomics data generated by MaxQuant and Spectronaut. The package can be used either through the R command line or through an interactive ShinyApp called SmartPhos Explorer. The package contains methods such as normalization and normalization correction, transformation, imputation, batch effect correction, PCA, heatmap, differential expression, time-series clustering, gene set enrichment analysis, and kinase activity inference.

License GPL-3

biocViews Visualization, ShinyApps, GUI, QualityControl, Proteomics, DifferentialExpression, Normalization, Preprocessing, GeneSetEnrichment, Clustering, GeneExpression, MassSpectrometry, BatchEffect

BugReports https://github.com/Bioconductor/SmartPhos/issues

Imports MultiAssayExperiment, SummarizedExperiment, data.table, shiny, shinythemes, shinyjs, shinyBS, shinyWidgets, parallel, DT, tools, stats, ggplot2, plotly, ggbeeswarm, pheatmap, grid, XML, MsCoreUtils, imputeLCMD, missForest, limma, proDA, decoupleR, piano, BiocParallel, doParallel, doRNG, e1071, magrittr, matrixStats, rlang, stringr, tibble, dplyr, tidyr, Biobase, vsn, factoextra, cowplot

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.3

Depends R (>= 4.4.0)

Suggests knitr, BiocStyle, PhosR, testthat

Config/testthat/edition 3

URL https://lu-group-ukhd.github.io/SmartPhos/

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

git_branch devel

git_last_commit 9abaf07

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-03

Author Shubham Agrawal [aut, cre] (ORCID: <https://orcid.org/0009-0005-2630-9342>), Junyan Lu [aut] (ORCID: <https://orcid.org/0000-0002-9211-0746>)

Maintainer Shubham Agrawal <shubhamagrawal2706@gmail.com>

Contents

addZeroTime . . . 3
calcKinaseScore . . . 4
checkRatioMat . . . 6
clusterEnrich . . . 6
clusterTS . . . 8
dda_example . . . 9
dia_example . . . 10
enrichDifferential . . . 10
generateInputTable . . . 12
generateInputTable_DIA . . . 13
getDecouplerNetwork . . . 14
getOneSymbol . . . 14
getRatioMatrix . . . 15
Homo_sapien_kinase_substrate_network . . . 16
intensityBoxPlot . . . 16
makeSmartPhosDirectory . . . 17
medianNorm . . . 18
mscale . . . 19
Mus_musculus_kinase_substrate_network . . . 20
normByFullProteome . . . 21
performCombinedNormalization . . . 22
performDifferentialExp . . . 23
plotAdjustmentResults . . . 25
plotHeatmap . . . 26
plotIntensity . . . 27
plotKinaseDE . . . 28
plotKinaseTimeSeries . . . 29
plotLogRatio . . . 30
plotMissing . . . 30
plotPCA . . . 31
plotTimeSeries . . . 32

plotVolcano 34
preprocessPhos 35
preprocessProteome 36
readExperiment 37
readExperimentDIA 39
readOnePhos 40
readOnePhosDIA 41
readOneProteom 42
readOneProteomDIA 43
readPhosphoExperiment 43
readPhosphoExperimentDIA 44
readProteomeExperiment 45
readProteomeExperimentDIA 47
runFisher 48
runGSEAforPhospho 49
runPhosphoAdjustment 50
runSmartPhos 51
splineFilter 51
swissProt 53

Index **54**

`addZeroTime` *Add Zero Timepoint Data to Treatment Subset*

Description

`addZeroTime` adds a zero timepoint to a specific treatment's data subset.

Usage

```
addZeroTime(data, condition, treat, zeroTreat, timeRange)
```

Arguments

`data`	A `SummarizedExperiment` object containing the experimental data.
`condition`	`Character` string corresponds to one of the columns from the colData of SE object.
`treat`	`Character` string specifying the treatment to which zero timepoint should be added.
`zeroTreat`	`Character` string specifying the treatment representing the zero timepoint.
`timeRange`	`Character` vector specifying the timepoints to include for the treatment.

Details

The function performs the following steps:

1. Subsets the data for the specified treatment and time range.
2. Subsets the data for the zero timepoint of the specified zero treatment.
3. Combines the assays from the treatment and zero timepoint subsets.
4. Updates the column data to reflect the combined treatment.
5. Returns a `SummarizedExperiment` object with the combined data.

Value

A `SummarizedExperiment` object with the zero timepoint added to the specified treatment's data.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
# Call the function
addZeroTime(se, condition = "treatment", treat = "EGF",
zeroTreat = "1stCrtl", timeRange = c("20min","40min", "6h"))
```

`calcKinaseScore` *Calculate Kinase Activity Scores using* `decoupleR`

Description

`calcKinaseScore` calculates kinase activity scores based on input data and a specified network of regulatory relationships (`decoupler network`).

Usage

```
calcKinaseScore(
  resTab,
  decoupler_network,
  corrThreshold = 0.9,
  statType = c("stat", "log2FC"),
  nPerm = 100
)
```

Arguments

- `resTab` A `data frame` containing the input data with columns site, stat, and log2FC.
- `decoupler_network` A `data frame` representing the decoupleR network with columns source and target.
- `corrThreshold` A `numeric` value specifying the correlation threshold for filtering correlated regulons. Default is 0.9.
- `statType` A `character` string specifying the type of statistic to use. Options are "stat" or "log2FC". Default is "stat".
- `nPerm` A `numeric` value specifying the number of permutations for the null distribution. Default is 100.

Details

The function performs the following steps:

1. Removes duplicate rows based on the site column.
2. Filters the data to include only those sites present in the target column of the `decoupler network`.
3. Prepares the input table based on the specified statType.
4. Intersects the input table with the `decoupler network` to find common regulons.
5. Checks for correlated regulons and filters out those exceeding the correlation threshold.
6. Calculates kinase activity using a weighted mean approach.
7. Processes the results to handle NA values and formats the output.

Value

A `data frame` with kinase activity scores, including columns for ‘source‘, ‘score‘, and ‘p_value‘.

Examples

```
resTab <- data.frame(
site = c("EGFR_Y1172", "EGFR_Y1197", "EGFR_S1166", "ROCK2_S1374",
"WASL_Y256", "GAB1_Y259", "ADD1_S586", "EPHA2_Y772", "PRKDC_T2638",
"PRKDC_T2609", "PRKDC_S2612"),
stat = c(-10.038770, -5.945562, 5.773384, -7.303834, 5.585326, 5.971104,
5.199119, -5.169500, 5.130228, 5.407387, 4.493933),
log2FC = c(-2.6113343, -2.4858615, 1.0056629, -1.1561780, 1.6421145,
2.0296634, 1.3766283, -0.8531656, 1.0742881, 1.0042942, 1.0608129)
)
decoupler_network <- data.frame(
source = c(rep("ABL1", 5), rep("CDK2", 6)),
mor = c(rep(1, 11)),
target = c("EGFR_Y1172", "EGFR_Y1197", "EGFR_S1166", "ROCK2_S1374",
"WASL_Y256", "GAB1_Y259", "ADD1_S586", "EPHA2_Y772", "PRKDC_T2638",
"PRKDC_T2609", "PRKDC_S2612"),
likelihood = c(rep(1, 11))
```

```
)
# Call the function
calcKinaseScore(resTab, decoupler_network)
```

`checkRatioMat` *Check the PP/FP ratio matrix and remove feature that do not meet requirements*

Description

`checkRatioMat` checks the ratio `matrix` for samples that do not have sufficient overlap of phosphopeptides between enriched (PP) and unenriched (FP) samples.

Usage

```
checkRatioMat(ratioMat, minOverlap = 3)
```

Arguments

`ratioMat`	A numeric `matrix` representing the ratio of phosphoproteome data to full proteome data.
`minOverlap`	A `numeric` specifying the minimum number of overlapping peptides required between samples. Default is 3.

Value

A `character` vector of sample names that do not meet the overlap criteria.

`clusterEnrich` *Perform Cluster Enrichment Analysis*

Description

`clusterEnrich` performs enrichment analysis on gene clusters, using Fisher's Exact Test to determine the significance of enrichment for each cluster.

Usage

```
clusterEnrich(
  clusterTab,
  se,
  inputSet,
  reference = NULL,
  ptm = FALSE,
  adj = "BH",
  filterP = 0.05,
  ifFDR = FALSE
)
```

Arguments

clusterTab	A data frame containing cluster information, where each row corresponds to a gene and its assigned cluster.
se	A SummarizedExperiment object containing gene expression data and metadata.
inputSet	A list or data frame of gene sets to be used for enrichment analysis.
reference	A character vector of reference genes. If NULL, it will be extracted from se object. Default is NULL.
ptm	Logical. If TRUE, the function will perform enrichment analysis on post-translational modification (PTM) gene sets. Default is FALSE.
adj	Character. The method for adjusting p-values. Default is "BH".
filterP	Numeric. The p-value threshold for filtering significant results. Default is 0.05.
ifFDR	Logical. If TRUE, the function will use FDR-adjusted p-values for significance filtering. Default is FALSE.

Details

The function first retrieves or computes the reference set of genes or PTM sites. It then performs enrichment analysis for each cluster using the runFisher function. The results are filtered based on the p-value threshold and adjusted for multiple testing if ifFDR is TRUE. The function generates a dot plot where the size and color of the points represent the significance of enrichment.

Value

A list containing two elements:

- ‘table‘: A data frame with enrichment results for each cluster and pathway.
- ‘plot‘: A ggplot2 object showing the significance of enrichment for each pathway across clusters.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
seProcess <- preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
result <- addZeroTime(seProcess, condition = "treatment", treat = "EGF",
zeroTreat = "1stCrtl", timeRange = c("20min","40min", "6h"))
# Get the numeric matrix
exprMat <- SummarizedExperiment::assay(result)
# Call the clustering function
clust <- clusterTS(x = exprMat, k = 3)
genesetPath <- appDir <- system.file("shiny-app/geneset",
package = "SmartPhos")
inGMT <- piano::loadGSC(paste0(genesetPath,
"/Cancer_Hallmark.gmt"),type="gmt")
# Call the function
clusterEnrich(clust$cluster, seProcess, inGMT)
```

`clusterTS` *Perform Clustering on Time-Series Data*

Description

`clusterTS` performs clustering on time-series data and generates plots for visualization.

Usage

```
clusterTS(x, k = 5, pCut = NULL, twoCondition = FALSE)
```

Arguments

`x`	A numeric `matrix` with rows as features and columns as time points.
`k`	A `numeric` value specifying the number of clusters. Default is 5.
`pCut`	A `numeric` value specifying the probability cutoff for cluster membership. Default is `NULL`.
`twoCondition`	A `logical` value indicating if the data contains two conditions. Default is `FALSE`.

Details

The function performs the following steps:

1. Sets a seed for reproducibility.
2. Removes rows with missing values.

3. Performs clustering using fuzzy C-means.
4. Filters clusters based on the probability cutoff if provided.
5. Generates plots for visualizing clustering results.

Value

A list containing:

`cluster`	A `tibble` with clustering information for each feature.
`plot`	A `ggplot2` object for visualizing the clustering results.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
seProcess <- preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
result <- addZeroTime(seProcess, condition = "treatment", treat = "EGF",
zeroTreat = "1stCrtl", timeRange = c("20min","40min", "6h"))
# Get the numeric matrix
exprMat <- assay(result)
# Call the function
clusterTS(x = exprMat, k = 3)
```

`dda_example` *dda_example*

Description

A sample of Data Dependent Acquisition (DDA) mass spectrometry data from Max Quant.

Usage

```
data(dda_example)
```

Format

a class S4 object of MultiAssayExperiment

Value

A `MultiAssayExperiment` object containing a sample of DDA mass spectrometry data from Max Quant.

Examples

```
data(dda_example)
```

`dia_example` *dia_example*

Description

A sample of Data Independent Acquisition (DIA) mass spectrometry data from Spectronaut.

Usage

```
data(dia_example)
```

Format

a class S4 object of MultiAssayExperiment

Value

A `MultiAssayExperiment` object containing a sample of DIA mass spectrometry data from Spectronaut.

Examples

```
data(dia_example)
```

`enrichDifferential` *Perform Enrichment analysis on differentially expressed genes or phospho-sites*

Description

`enrichDifferential` performs enrichment analysis on differentially expressed genes and phospho-sites for either pathway or phospho-specific enrichment, depending on the input parameters. It supports multiple statistical methods such as PAGE and GSEA for pathway enrichment and a Kolmogorov-Smirnov approach for phospho-enrichment.

Usage

```
enrichDifferential(
  dea,
  type = c("Pathway enrichment", "Phospho-signature enrichment"),
  gsaMethod = c("PAGE", "GSEA"),
  geneSet,
  ptmSet,
  statType = c("stat", "log2FC"),
  nPerm = 100,
  sigLevel = 0.05,
  ifFDR = FALSE
)
```

Arguments

`dea`	A `data frame` containing the differential expression analysis results. It should include columns like ‘pvalue‘, ‘Gene‘ (or ‘site‘), ‘stat‘, and ‘log2FC‘.
`type`	A `character` string indicating the type of enrichment. Options are ‘"Pathway enrichment"‘ or ‘"Phospho-signature enrichment"‘.
`gsaMethod`	A `character` string specifying the gene set analysis method for pathway enrichment. Options are ‘"PAGE"‘ or ‘"GSEA"‘.
`geneSet`	A gene set collection to use for pathway enrichment.
`ptmSet`	A post-translational modification (PTM) set database for phospho-enrichment analysis.
`statType`	A `character` string specifying the statistic type to use. Options are ‘"stat"‘ or ‘"log2FC"‘.
`nPerm`	A `numeric` specifying the number of permutations for GSEA. Default is 100.
`sigLevel`	A `numeric` value representing the significance threshold for filtering results. Ddefault is 0.05.
`ifFDR`	A `logical` value indicating whether to filter results using FDR-adjusted p-values. Default is ‘FALSE‘.

Details

The ‘enrichDifferential‘ function performs either pathway enrichment or phospho-enrichment analysis based on the ‘type‘ parameter. For pathway enrichment, it uses either the PAGE or GSEA method with a provided gene set collection. For phospho-enrichment, it uses a Kolmogorov-Smirnov test with a PTM set database. Results can be filtered by significance level and optionally adjusted for FDR.

Value

A data frame containing the results of the enrichment analysis, including columns such as the gene set name, statistical significance, and adjusted p-values.

Examples

```
library(SummarizedExperiment)
library(piano)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
# Preprocess the proteome assay
result <- preprocessPhos(se, normalize = TRUE)
# Call the function to perform differential expression analyis
dea <- performDifferentialExp(se = result, assay = "Intensity",
method = "limma", reference = "1stCrtl", target = "EGF",
condition = "treatment")
# Load the gene set
genesetPath <- appDir <- system.file("shiny-app/geneset",
package = "SmartPhos")
inGMT <- loadGSC(paste0(genesetPath,"/Cancer_Hallmark.gmt"),type="gmt")
# Call the function
resTab <- enrichDifferential(dea = dea$resDE, type = "Pathway enrichment",
gsaMethod = "PAGE", geneSet = inGMT, statType = "stat", nPerm = 200,
sigLevel = 0.05, ifFDR = FALSE)
```

---

generateInputTable *Generate Input Table for Proteomic and Phosphoproteomic Analysis*

Description

`generateInputTable` generates an input table for proteomic and phosphoproteomic analysis by reading files from a specified folder.

Usage

```
generateInputTable(rawFolder, batchAsFolder = FALSE)
```

Arguments

`rawFolder`	A `character` string specifying the path to the folder containing the raw files.
`batchAsFolder`	A `logical` value indicating whether to treat subdirectories as separate batches. Default is `FALSE`.

Details

The function performs the following steps:

- Optionally treats subdirectories as separate batches.
- Reads the summary file containing experimental information.

- Generates unique experimental IDs based on batch and sample names.
- Processes file paths for full proteome and phosphoproteome data.
- Creates a combined input table with file names, sample names, search types, batches, and IDs.

Value

A `data.frame` with columns fileName, sample, searchType, batch, and id that can be used as input for further analysis.

`generateInputTable_DIA` *Generate Input Table for DIA Analysis*

Description

`generateInputTable_DIA` generates an input table for DIA analysis by reading files from a specified folder.

Usage

```
generateInputTable_DIA(rawFolder)
```

Arguments

`rawFolder`	A `character` string specifying the path to the folder containing the raw files.

Details

The function performs the following steps:

- Reads the summary file containing experimental information.
- Generates unique experimental IDs based on sample type, treatment, timepoint, and replicate.
- Processes file paths for full proteome and phosphoproteome data.
- Creates a combined input table with file names, search types, and IDs.

Value

A `data.frame` with columns fileName, searchType, and id that can be used as input for further analysis.

getDecouplerNetwork *Load Kinase-Substrate Interaction Network*

Description

getDecouplerNetwork loads the kinase-substrate interaction network for a specified species from pre-defined files.

Usage

```
getDecouplerNetwork(speciesRef = c("Homo sapiens", "Mus musculus"))
```

Arguments

speciesRef	A character string specifying the species. Supported values are "Homo sapiens" and "Mus musculus". Default is "Homo sapiens".

Value

A data frame containing the kinase-substrate interaction network for the specified species.

Examples

```
# Load the human kinase-substrate interaction network
getDecouplerNetwork("Homo sapiens")

# Load the mouse kinase-substrate interaction network
getDecouplerNetwork("Mus musculus")
```

getOneSymbol *Extract the Last Gene Symbol from a Semicolon-Separated List*

Description

getOneSymbol extracts the last gene symbol from a semicolon-separated list of gene symbols.

Usage

```
getOneSymbol(Gene)
```

Arguments

Gene	A character vector where each element is a semicolon-separated list of gene symbols.

Details

This function processes a character vector where each element consists of gene symbols separated by semicolons. It splits each element by semicolons and extracts the last gene symbol from the resulting list. The output is a character vector of these last gene symbols.

Value

A `character` vector containing the last gene symbol from each element of the input vector.

`getRatioMatrix` *Get Ratio Matrix of Phosphoproteome Data*

Description

`getRatioMatrix` calculates the ratio matrix of phosphoproteome data from a `MultiAssayExperiment` object.

Usage

```
getRatioMatrix(maeData, normalization = FALSE, getAdjustedPP = FALSE)
```

Arguments

`maeData`	A `MultiAssayExperiment` object containing phosphoproteome and full proteome data.
`normalization`	A `logical` value indicating whether to perform normalization. Default is `FALSE`.
`getAdjustedPP`	A `logical` value indicating whether to use adjusted phosphoproteome data. Default is `FALSE`.

Value

A numeric `matrix` representing the ratio of intensity of PP (phosphoproteome) data to FP (full proteome) data.

Examples

```
# Load multiAssayExperiment object
data("dia_example")
# Call the function
getRatioMatrix(dia_example, normalization = TRUE)
```

Homo_sapien_kinase_substrate_network *Homo_sapien_kinase_substrate_network*

Description

A prior knowledge database about the known kinase-phosphosite interactions for Homo sapiens

Usage

```
data(Homo_sapien_kinase_substrate_network)
```

Format

a data.frame object

Value

A `data frame` containing the information about the known kinase-phosphosite interactions for Homo sapiens.

Examples

```
data(Homo_sapien_kinase_substrate_network)
```

intensityBoxPlot *Plot Boxplot of Intensity Data*

Description

`intensityBoxPlot` creates a boxplot for the Intensity data of a given gene or feature, with optional subject-specific lines.

Usage

```
intensityBoxPlot(se, id, symbol)
```

Arguments

`se`	A `SummarizedExperiment` object containing the data.
`id`	`Character`. The identifier of the gene or feature to plot.
`symbol`	`Character`. The symbol or name of the gene or feature to use as the plot title.

Details

This function generates a boxplot for the intensity data of a specified gene or feature from a `SummarizedExperiment` (SE) object. The plot shows the distribution of normalized intensities across different groups specified in the `comparison` column of the SE object.

The function can handle both grouped data and repeated measures: - If the SE object does not contain a `subjectID` column, the function plots a standard boxplot grouped by the `comparison` column. - If the SE object contains a `subjectID` column, the function adds lines connecting the points for each subject across the groups, providing a visual indication of subject-specific changes.

The `boxplot` is customized with various aesthetic elements, such as box width, transparency, point size, axis labels, and title formatting.

Value

A `ggplot2` object representing the boxplot of the intensity data.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Proteome"]]
colData(se) <- colData(dda_example)
# Preprocess the proteome assay
result <- preprocessProteome(se, normalize = TRUE)
# Call the function to perform differential expression analyis
de <- performDifferentialExp(se = result, assay = "Intensity",
method = "limma", reference = "1stCrtl", target = "EGF",
condition = "treatment")
# Plot the box plot for the given id and symbol
intensityBoxPlot(de$seSub, "p99", "PPP6C")
```

---

makeSmartPhosDirectory

Create SmartPhos Directory Structure

Description

`makeSmartPhosDirectory` creates a directory for the SmartPhos shiny app, and copies the necessary Shiny app files into the newly created directory.

Usage

```
makeSmartPhosDirectory(path)
```

Arguments

`path`	A `character` string specifying the directory path where the SmartPhos folder should be created.

Details

The function first creates the main directory at the specified path and a subdirectory named ‘"save"‘ for storing `MultiAssayExperiment` object. It then locates the Shiny application files from the SmartPhos package and copies them into the new directory.

Value

None (invisible NULL). The function creates the necessary directories and copies files.

Examples

```
makeSmartPhosDirectory("shinyApp")
```

`medianNorm` *Normalize a Matrix Using Median or Mean*

Description

`medianNorm` normalizes the columns of a matrix by either the median or the mean.

Usage

```
medianNorm(x, method = "median")
```

Arguments

`x`	A `numeric` matrix to be normalized.
`method`	A `character` string specifying the normalization method. Options are "median" or "mean". Default is "median".

Details

The function performs the following steps:

1. If the `method` is "median", it calculates the median of each column and adjusts by the overall median of these medians.
2. If the `method` is "mean", it calculates the mean of each column and adjusts by the overall mean of these means.
3. It constructs a matrix of these adjusted values and subtracts it from the original matrix to normalize the columns.

Value

A `numeric` `matrix` with normalized columns.

Examples

```
# Example usage:
x <- matrix(rnorm(20), nrow=5, ncol=4)
medianNorm(x, method = "median")
```

---

mscale — *Scale and Center a Matrix*

Description

`mscale` scales and centers each row of a matrix, with options for using mean or median, standard deviation or mean absolute deviation, and censoring extreme values.

Usage

```
mscale(x, center = TRUE, scale = TRUE, censor = NULL, useMad = FALSE)
```

Arguments

`x`	A `numeric` matrix where rows are features and columns are samples.
`center`	`Logical`. If `TRUE`, the rows are centered by subtracting the mean or median. Default is `TRUE`.
`scale`	`Logical`. If `TRUE`, the rows are scaled by dividing by the standard deviation or mean absolute deviation. Default is `TRUE`.
`censor`	A `numeric` vector of length one or two for censoring the scaled values. If length one, values are censored symmetrically at positive and negative values. If length two, the first value is the lower limit and the second value is the upper limit. Default is `NULL`.
`useMad`	`Logical`. If `TRUE`, the mean absolute deviation is used for scaling instead of the standard deviation. Default is `FALSE`.

Details

The function allows for flexible scaling and centering of the rows of a `matrix`:

- If both `center` and `scale` are `TRUE`, rows are centered and scaled.
- If only `center` is `TRUE`, rows are centered but not scaled.
- If only `scale` is `TRUE`, rows are scaled but not centered.
- If neither `center` nor `scale` is `TRUE`, the original `matrix` is returned.

The function can also censor extreme values, either symmetrically or asymmetrically, based on the `censor` parameter.

Value

A scaled and centered numeric `matrix` with the same dimensions as the input `matrix` ‘x‘.

Examples

```
# Create a sample matrix (3 rows by 5 columns)
sample_matrix <- matrix(c(1:15), nrow = 3, byrow = TRUE)

# Scale and center the matrix using the default settings
mscale(sample_matrix, center = TRUE, scale = TRUE)

# Only center the matrix without scaling
mscale(sample_matrix, center = TRUE, scale = FALSE)

# Only scale the matrix without centering
mscale(sample_matrix, center = FALSE, scale = TRUE)
```

`Mus_musculus_kinase_substrate_network` *Mus_musculus_kinase_substrate_network*

Description

A prior knowledge database about the known kinase-phosphosite interactions for Mus musculus

Usage

```
data(Mus_musculus_kinase_substrate_network)
```

Format

a data.frame object

Value

A `data frame` containing the information about the known kinase-phosphosite interactions for Mus musculus.

Examples

```
data(Mus_musculus_kinase_substrate_network)
```

normByFullProteome *Normalize Phosphoproteome by Full Proteome*

Description

normByFullProteome normalizes the phosphoproteome data by the corresponding full proteome data in a MultiAssayExperiment object. The "Phosphoproteome" assay in the MultiAssayExperiment will be replaced by the ratio.

Usage

```
normByFullProteome(mae, replace = TRUE)
```

Arguments

mae	A MultiAssayExperiment object containing both phosphoproteome and proteome assays.
replace	Logical, whether to replace the existing phosphoproteome assay in the MultiAssayExperiment object. If replace = FALSE, a new assay, phosphoRatio, will be created. Default is TRUE.

Details

The function performs the following steps:

- Checks if both phosphoproteome and proteome assays are present in the MultiAssayExperiment object.
- Extracts the phosphoproteome and proteome assays along with the sample annotations.
- Matches the samples between the phosphoproteome and proteome assays.
- Normalizes the phosphoproteome data by dividing it by the corresponding proteome data.
- Replaces the phosphoproteome assay in the MultiAssayExperiment object or adds the normalized data as a new assay, depending on the replace parameter.

Value

A MultiAssayExperiment object with the normalized phosphoproteome data.

Examples

```
# load mae object
data("dia_example")
# call the function
normByFullProteome(dia_example)
```

performCombinedNormalization

Perform Combined Normalization on MultiAssayExperiment Data

Description

`performCombinedNormalization` performs combined normalization on proteome and phosphoproteome data from a `MultiAssayExperiment` object.

Usage

```
performCombinedNormalization(maeData)
```

Arguments

`maeData`	A `MultiAssayExperiment` object containing proteome and phosphoproteome data.

Details

The function performs the following steps:

1. Extracts the count matrices for Full Proteome (FP) samples.
2. Combines the proteome and phosphoproteome data into a single matrix.
3. Removes rows with all NA values.
4. Performs median normalization and log2 transformation on the combined matrix.

Value

A numeric `matrix` with normalized and log2-transformed data.

Examples

```
# Load multiAssayExperiment object
data("dia_example")
# Call the function
performCombinedNormalization(dia_example)
```

performDifferentialExp

Perform Differential Expression Analysis

Description

performDifferentialExp performs differential expression analysis on a given SummarizedExperiment object using either the limma or ProDA method.

Usage

```
performDifferentialExp(
  se,
  assay,
  method = c("limma", "ProDA"),
  condition = NULL,
  reference,
  target,
  refTime = NULL,
  targetTime = NULL,
  pairedTtest = FALSE
)
```

Arguments

se	A SummarizedExperiment object containing the data.
assay	A character string specifying the assay to use for the analysis.
method	A character string specifying the method to use for differential expression analysis ("limma" or "ProDA"). Default is "limma".
condition	A character string specifying the condition column in colData(se). Default is NULL.
reference	A character string or vector specifying the reference group.
target	A character string or vector specifying the target group.
refTime	A character string or vector specifying the reference time points. Default is NULL.
targetTime	A character string or vector specifying the target time points. Default is NULL.
pairedTtest	A logical value specifying to perform paired t-test or not. Default is FALSE.

Details

This function is designed to facilitate differential expression analysis on a SummarizedExperiment (SE) object. The function allows users to specify various parameters to tailor the analysis to their specific experimental setup.

The main steps of the function are as follows:

1. Sample Selection: Based on the provided `condition`, `reference`, and `target` arguments, the function identifies the relevant samples for the analysis. If time points (`refTime` and `targetTime`) are provided, it further refines the sample selection.

2. Subsetting the SE Object: The SE object is subsetted to include only the selected samples. A new column `comparison` is added to the `colData`, indicating whether each sample belongs to the reference or target group.

3. Design Matrix Construction: The function constructs a design matrix for the differential expression analysis. If the SE object contains a `subjectID` column, this is included in the design to account for repeated measures or paired samples.

4. Differential Expression Analysis: Depending on the specified `method`, the function performs the differential expression analysis using either the `limma` or `ProDA` package: - `Limma`: The function fits a linear model to the expression data and applies empirical Bayes moderation to the standard errors. The results are then extracted and formatted. - `ProDA`: The function fits a probabilistic dropout model to the expression data and tests for differential expression. The results are then extracted and formatted.

5. Result Formatting: The differential expression results are merged with the metadata from the SE object, and the resulting table is formatted into a tibble. The table includes columns for log2 fold change (log2FC), test statistic (stat), p-value (pvalue), adjusted p-value (padj), and gene/feature ID (ID).

The function returns a `list` containing the formatted differential expression results and the subsetted SE object. This allows users to further explore or visualize the results as needed.

Value

A list containing:

`resDE`	A `tibble` with the differential expression results.
`seSub`	A `SummarizedExperiment` object subset to the samples used in the analysis.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Proteome"]]
colData(se) <- colData(dda_example)
# Preprocess the proteome assay
result <- preprocessProteome(se, normalize = TRUE)
# Call the function to perform differential expression analyis
performDifferentialExp(se = result, assay = "Intensity", method = "limma",
reference = "1stCrtl", target = "EGF", condition = "treatment")
```

plotAdjustmentResults *Plot Adjustment Results*

Description

`plotAdjustmentResults` generates plots to visualize the results of phosphoproteome adjustment.

Usage

```
plotAdjustmentResults(maeData, normalization = FALSE)
```

Arguments

`maeData`	A `MultiAssayExperiment` object containing phosphoproteome and full proteome data.
`normalization`	A `logical` value indicating whether normalization was performed. Default is `FALSE`.

Details

The function performs the following steps:

1. Checks if the adjustment factor is present in the sample annotation.
2. Calculates the ratio matrix before and after adjustment.
3. Creates a trend line plot for features present in all samples.
4. Creates box plots of the PP/FP ratios and phosphorylation intensities before and after adjustment.

Value

A list containing:

`ratioTrendPlot`	A `ggplot2` object showing the line plot of PP/FP ratios for features present in all samples.
`ratioBoxplot`	A `ggplot2` object showing the box plot of PP/FP ratios before and after adjustment.
`ppBoxplot`	A `ggplot2` object showing the box plot of phosphorylation intensities in PP samples before and after adjustment.

plotHeatmap *Plot Heatmap of Intensity assay*

Description

plotHeatmap generates a heatmap for intensity assay for different conditions, including top variants, differentially expressed genes, and selected time series clusters.

Usage

```
plotHeatmap(
 type = c("Top variant", "Differentially expressed", "Selected time series cluster"),
  se,
  data = NULL,
  top = 100,
  cutCol = 1,
  cutRow = 1,
  clustCol = TRUE,
  clustRow = TRUE,
  annotationCol = NULL,
  title = NULL
)
```

Arguments

type	A character string specifying the type of heatmap to plot. Options are "Top variant", "Differentially expressed", and "Selected time series cluster".
se	A SummarizedExperiment object containing the imputed intensity assay.
data	An optional data frame containing additional data for "Differentially expressed" and "Selected time series cluster" types. Default is NULL.
top	A numeric value specifying the number of top variants to plot. Default is 100.
cutCol	A numeric value specifying the number of clusters for columns. Default is 1.
cutRow	A numeric value specifying the number of clusters for rows. Default is 1.
clustCol	A logical value indicating whether to cluster columns. Default is TRUE.
clustRow	A logical value indicating whether to cluster rows. Default is TRUE.
annotationCol	A character vector specifying the columns in the metadata to use for annotation. Default is NULL.
title	A character string specifying the title of the heatmap. Default is NULL.

Details

This function creates a heatmap using the Intensity assay from a SummarizedExperiment object. The heatmap can show the top variants based on standard deviation, differentially expressed genes, or selected time series clusters. Row normalization is performed, and the heatmap can include annotations based on specified metadata columns.

Value

A pheatmap object showing the heatmap of Intensity data.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
# Generate the imputed assay
result <- preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
# Plot heatmap for top variant
plotHeatmap(type = "Top variant", top = 10, se = result, cutCol = 2)
```

---

plotIntensity *Plot Intensity Boxplots*

Description

plotIntensity generates boxplots of assay intensities for each sample in a SummarizedExperiment object. Optionally, the boxplots can be colored based on a specified metadata column. The function handles missing values by filtering them out before plotting.

Usage

```
plotIntensity(se, colorByCol = "none")
```

Arguments

se	A SummarizedExperiment object containing the assay data and metadata.
colorByCol	A character string specifying the metadata column to use for coloring the boxplots. Default is "none".

Value

A ggplot2 object showing boxplots of intensities for each sample.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
```

```
# Preprocess the phosphoproteome assay
result <- preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
# Call the plotting function
plotIntensity(result, colorByCol = "replicate")
```

`plotKinaseDE` *Plot Kinase score for Differential Expression data*

Description

‘plotKinaseDE‘ generates a `bar plot` of the top kinases associated with the differentially expressed genes based on their scores.

Usage

```
plotKinaseDE(scoreTab, nTop = 10, pCut = 0.05)
```

Arguments

`scoreTab`	A `data frame` containing kinase scores with columns source, score, and p_value.
`nTop`	A `numeric` value specifying the number of top kinases to plot for each direction. Default is 10.
`pCut`	A `numeric` value specifying the p-value cutoff for significance. Default is 0.05.

Details

The function performs the following steps:

1. Adds a column for significance based on the p-value cutoff.
2. Adds a column for the sign of the score.
3. Filters out kinases with a score of 0.
4. Selects the top `nTop` kinases by absolute score for each sign of the score.
5. Creates a bar plot with the selected kinases.

Value

A `ggplot2` object representing the bar plot of kinase score.

Examples

```
# Example usage:
scoreTab <- data.frame(
 source = c("Kinase1", "Kinase2", "Kinase3", "Kinase4"),
 score = c(2.3, -1.5, 0, 3.1),
 p_value = c(0.01, 0.2, 0.05, 0.03)
)
plotKinaseDE(scoreTab, nTop = 3, pCut = 0.05)
```

plotKinaseTimeSeries *Plot Kinase Activity Time Series*

Description

`plotKinaseTimeSeries` creates a heatmap to visualize the result of kinase activity inference for time-series clustering, with significant activity changes marked.

Usage

```
plotKinaseTimeSeries(scoreTab, pCut = 0.05, clusterName = "cluster1")
```

Arguments

`scoreTab`	A `data frame` containing kinase activity scores, p-values, and time points.
`pCut`	A `numeric` value specifying the p-value threshold for significance. Default is 0.05.
`clusterName`	A `character` string specifying the name of the cluster for the plot title. Default is "cluster1".

Details

The heatmap shows kinase activity scores over different time points. Significant activities (based on the specified p-value threshold) are marked with an asterisk (*). The color gradient represents the activity score, with blue indicating low activity, red indicating high activity, and white as the midpoint.

Value

A `ggplot2` object representing the heatmap of kinase activity score.

Examples

```
# Example usage:
scoreTab <- data.frame(
 timepoint = rep(c("0h", "1h", "2h"), each = 3),
 source = rep(c("KinaseA", "KinaseB", "KinaseC"), times = 3),
 score = runif(9, -2, 2),
 p_value = runif(9, 0, 0.1)
)
plotKinaseTimeSeries(scoreTab)
```

plotLogRatio — *Plot Log Ratio of PP/FP (Phosphoproteome to Full Proteome) intensities*

Description

`plotLogRatio` generates a boxplot of the log2 ratio of intensities of phosphoproteome to full proteome data from a `MultiAssayExperiment` object.

Usage

```
plotLogRatio(maeData, normalization = FALSE)
```

Arguments

`maeData`	A `MultiAssayExperiment` object containing phosphoproteome and full proteome data.
`normalization`	A `logical` value indicating whether to perform normalization. Default is `FALSE`.

Value

A `ggplot2` object representing the boxplot of the log2 ratios.

Examples

```
# Load multiAssayExperiment object
data("dia_example")
# Call the function
plotLogRatio(dia_example, normalization = TRUE)
```

plotMissing — *Plot Missing Data Completeness*

Description

`plotMissing` generates a bar plot showing the completeness (percentage of non-missing values) for each sample in a `SummarizedExperiment` object.

Usage

```
plotMissing(se)
```

Arguments

`se`	A `SummarizedExperiment` object containing the assay data.

Details

This function calculates the percentage of non-missing values for each sample in the provided `SummarizedExperiment` object. It then generates a bar plot where each bar represents a sample, and the height of the bar corresponds to the completeness (percentage of non-missing values) of that sample.

Value

A `ggplot2` object showing the percentage of completeness for each sample.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Phosphoproteome"]]
colData(se) <- colData(dda_example)
# Call the function
plotMissing(se)
```

---

plotPCA *Plot PCA*

Description

`plotPCA` generates a PCA plot using the results from a PCA analysis and a `SummarizedExperiment` object. The points on the plot can be colored and shaped based on metadata.

Usage

```
plotPCA(pca, se, xaxis = "PC1", yaxis = "PC2", color = "none", shape = "none")
```

Arguments

`pca`	A PCA result object, typically obtained from `prcomp`.
`se`	A `SummarizedExperiment` object containing the metadata.
`xaxis`	A `character` string specifying which principal component to use for the x-axis. Default is "PC1".
`yaxis`	A `character` string specifying which principal component to use for the y-axis. Default is "PC2".
`color`	A `character` string specifying the metadata column to use for coloring the points. Default is "none".
`shape`	A `character` string specifying the metadata column to use for shaping the points. Default is "none".

Details

This function creates a PCA plot using the scores from a PCA result object and metadata from a `SummarizedExperiment` object. The x-axis and y-axis can be customized to display different principal components, and the points can be optionally colored and shaped based on specified metadata columns.

Value

A `ggplot2` object showing the PCA plot.

Examples

```
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
SummarizedExperiment::colData(se) <- SummarizedExperiment::colData(
dia_example)
# Generate the imputed assay
result <- preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
# Perform PCA
pcaResult <- stats::prcomp(t(
SummarizedExperiment::assays(result)[["imputed"]]),
center = TRUE, scale. = TRUE)
# Plot PCA results
plotPCA(pca = pcaResult, se = result, color = "treatment")
```

`plotTimeSeries` *Plot Time Series Data for a gene or phospho site from SummarizedExperiment object*

Description

`plotTimeSeries` plots time series data for a given gene or phospho site from a given `SummarizedExperiment` object, allowing different types of plots such as expression, log fold change, or two-condition expression.

Usage

```
plotTimeSeries(
  se,
  type = c("expression", "logFC", "two-condition expression"),
  geneID,
  symbol,
  condition,
  treatment,
  refTreat,
```

```
  addZero = FALSE,
  zeroTreat = NULL,
  timerange
)
```

Arguments

se	A SummarizedExperiment object containing the data.
type	Character. The type of plot to generate. Options are " expression", "logFC", or "two-condition expression".
geneID	Character. The identifier of the gene or feature to plot.
symbol	Character. The symbol or name of the gene or feature to use as the plot title.
condition	Character. The condition corresponds to one of the columns from the colData of SE object.
treatment	Character. The treatment to use for filtering the data.
refTreat	Character. The reference treatment to compare against.
addZero	Logical, whether to add a zero time point to the data. Default is FALSE.
zeroTreat	Character. The treatment to use for adding the zero time point. Default is NULL.
timerange	Character vector.The range of time points to include in the plot.

Details

This function generates time series plots for a specified gene or feature from a SummarizedExperiment (SE) object. The type of plot can be one of the following: - "expression": Plots normalized expression levels over time. - "logFC": Plots log fold change (logFC) over time, comparing a treatment to a reference treatment. - "two-condition expression": Plots normalized expression levels over time for two conditions.

The function can add a zero time point if specified and handles data with and without subject-specific information. The plot includes points for each time point and a summary line representing the mean value.

The x-axis represents time, and the y-axis represents the selected metric (normalized expression or logFC). The plot is customized with various aesthetic elements, such as point size, line type, axis labels, and title formatting.

Value

A ggplot2 object representing the time series plot.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Proteome"]]
```

```
colData(se) <- colData(dda_example)
# Preprocess the proteome assay
result <- preprocessProteome(se, normalize = TRUE)
# Plot a specific gene experssion over time
timerange <- unique(se$timepoint)
plotTimeSeries(result, type = "expression", geneID = "p18",
symbol = "TMEM238", condition = "treatment", treatment = "EGF",
timerange = timerange)
```

`plotVolcano` *Plot Volcano Plot for Differential Expression Analysis*

Description

`plotVolcano` generates a volcano plot to visualize differential expression results.

Usage

```
plotVolcano(tableDE, pFilter = 0.05, fcFilter = 0.5)
```

Arguments

`tableDE`	A `data frame` containing differential expression results with columns 'ID', 'log2FC', 'pvalue', and 'Gene'.
`pFilter`	A `numeric` value specifying the p-value threshold for significance. Default is 0.05.
`fcFilter`	A `numeric` value specifying the log2 fold-change threshold for significance. Default is 0.5.

Details

This function creates a volcano plot where differentially expressed genes are categorized as 'Up', 'Down', or 'Not Sig' based on the provided p-value and log2 fold-change thresholds. Points on the plot are color-coded to indicate their expression status.

Value

A `ggplot2` object representing the volcano plot.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Proteome"]]
colData(se) <- colData(dda_example)
```

```
# Preprocess the proteome assay
result <- preprocessProteome(se, normalize = TRUE)
# Call the function to perform differential expression analyis
de <- performDifferentialExp(se = result, assay = "Intensity",
method = "limma", reference = "1stCrtl", target = "EGF",
condition = "treatment")
# Plot the volcano plot from the result
plotVolcano(de$resDE)
```

---

preprocessPhos *Preprocess Phosphoproteome Data*

Description

preprocessPhos preprocesses phosphoproteome data stored in a SummarizedExperiment object by performing filtering, transformation, normalization, imputation, and batch effect removal.

Usage

```
preprocessPhos(
  seData,
  filterList = NULL,
  missCut = 50,
  transform = c("log2", "vst", "none"),
  normalize = FALSE,
  getFP = FALSE,
  removeOutlier = NULL,
  assayName = NULL,
  batch = NULL,
  scaleFactorTab = NULL,
  impute = c("none", "QRILC", "MLE", "bpca", "missForest", "MinDet"),
  verbose = FALSE
)
```

Arguments

seData	A SummarizedExperiment object containing phosphoproteome data.
filterList	A list of filters to apply on the samples. Default is NULL.
missCut	Numeric value specifying the missing value cutoff percentage for filtering features. Default is 50.
transform	Character string specifying the transformation method ("log2", "vst", "none"). Default is "log2".
normalize	Logical value indicating whether to normalize the data. Default is FALSE.
getFP	Logical value indicating whether to retrieve FP samples. Default is FALSE.
removeOutlier	Character vector of samples to be removed as outliers. Default is NULL.

assayName	Character string specifying the assay name in the SummarizedExperiment object. Default is NULL.
batch	Character vector specifying batch effects to remove. Default is NULL.
scaleFactorTab	Data frame containing scale factors for normalization. Default is NULL.
impute	Character string specifying the imputation method ("QRILC", "MLE", "bpca", "missForest", "MinDet", "none"). Default is "none".
verbose	Logical value indicating whether to print detailed information. Default is FALSE.

Value

A SummarizedExperiment object with preprocessed phosphoproteome data.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Phosphoproteome"]]
colData(se) <- colData(dia_example)
# Call the function
preprocessPhos(seData = se, normalize = TRUE, impute = "QRILC")
```

---

preprocessProteome *Preprocess Proteome Data*

Description

preprocessProteome preprocesses proteome data stored in a SummarizedExperiment object by performing filtering, transformation, normalization, imputation, and batch effect removal.

Usage

```
preprocessProteome(
  seData,
  filterList = NULL,
  missCut = 50,
  transform = c("log2", "vst", "none"),
  normalize = FALSE,
  getPP = FALSE,
  removeOutlier = NULL,
  impute = c("none", "QRILC", "MLE", "bpca", "missForest", "MinDet"),
  batch = NULL,
  verbose = FALSE,
  scaleFactorTab = NULL
)
```

Arguments

seData	A SummarizedExperiment object containing proteome data.
filterList	A list of filters to apply on the samples. Default is NULL.
missCut	Numeric value specifying the missing value cutoff percentage for filtering features. Default is 50.
transform	Character string specifying the transformation method ("log2", "vst", "none"). Default is "log2".
normalize	Logical value indicating whether to normalize the data. Default is FALSE.
getPP	Logical value indicating whether to retrieve PP samples. Default is FALSE.
removeOutlier	Character vector of samples to be removed as outliers. Default is NULL.
impute	Character string specifying the imputation method ("QRILC", "MLE", "bpca", "missForest", "MinDet", "none"). Default is "none".
batch	Character vector specifying batch effects to remove. Default is NULL.
verbose	Logical value indicating whether to print detailed information. Default is FALSE.
scaleFactorTab	Data frame containing scale factors for normalization. Default is NULL.

Value

A SummarizedExperiment object with preprocessed proteome data.

Examples

```
library(SummarizedExperiment)
# Load multiAssayExperiment object
data("dia_example")
# Get SummarizedExperiment object
se <- dia_example[["Proteome"]]
colData(se) <- colData(dia_example)
# Call the function
preprocessProteome(seData = se, normalize = TRUE, impute = "QRILC")
```

---

readExperiment *Read and Process the DDA experiment.*

Description

readExperiment reads and processes DDA (Data-Dependent Acquisition) phosphoproteomic and proteomic data from a given file table, and returns a MultiAssayExperiment object.

Usage

```
readExperiment(
  fileTable,
  localProbCut = 0.75,
  scoreDiffCut = 5,
  fdrCut = 0.1,
  scoreCut = 10,
  pepNumCut = 1,
  ifLFQ = TRUE,
  annotation_col = c(),
  verbose = FALSE
)
```

Arguments

`fileTable`	A `data.frame` containing information about the input files, including searchType, id, sample, and other annotations.
`localProbCut`	`Numeric`, local probability cutoff for filtering phosphoproteomic data. Default is 0.75.
`scoreDiffCut`	`Numeric`, score difference cutoff for filtering phosphoproteomic data. Default is 5.
`fdrCut`	`Numeric`, false discovery rate cutoff for filtering proteomic data. Default is 0.1.
`scoreCut`	`Numeric`, score cutoff for filtering proteomic data. Default is 10.
`pepNumCut`	`Numeric`, peptide number cutoff for filtering proteomic data. Default is 1.
`ifLFQ`	`Logical`, whether to use LFQ quantification for proteomic data. Default is `TRUE`.
`annotation_col`	A `character` vector specifying additional columns to be included in the sample annotation. Default is an empty vector.
`verbose`	`Logical` value indicating whether to print detailed information. Default is `FALSE`.

Details

The function performs the following steps:

- Reads and processes the phosphoproteomic data using the `readPhosphoExperiment` function.
- Reads and processes the proteomic data using the `readProteomeExperiment` function.
- Prepares the sample annotation table.
- Constructs and returns a `MultiAssayExperiment` object containing the processed data.

Value

A `MultiAssayExperiment` object containing the processed phosphoproteomic and proteomic data from a DDA experiment.

Examples

```
# Example usage:
file1 <- system.file("extdata", "phosDDA_1.xls", package = "SmartPhos")
file2 <- system.file("extdata", "proteomeDDA_1.xls", package = "SmartPhos")
# Create fileTable
fileTable <- data.frame(
  searchType = c("phosphoproteome", "proteome"),
  fileName = c(file1, file2),
  sample = c("Sample1", "sample1"),
  id = c("s1", "s2")
)
# Call the function
readExperiment(fileTable, localProbCut = 0.75, scoreDiffCut = 5,
fdrCut = 0.1, scoreCut = 10, pepNumCut = 1, ifLFQ = TRUE,
annotation_col = c("id"))
```

---

readExperimentDIA *Read and Process a DIA Experiment*

Description

`readExperimentDIA` reads and processes DIA (Data-Independent Acquisition) data for both phosphoproteome and proteome experiments, and constructs a `MultiAssayExperiment` object.

Usage

```
readExperimentDIA(
  fileTable,
  localProbCut = 0.75,
  annotation_col = c(),
  onlyReviewed = TRUE,
  normalizeByProtein = FALSE,
  verbose = FALSE
)
```

Arguments

`fileTable`	A `data frame` containing metadata about the files to be read. Must contain columns searchType, fileName, id, and optionally outputID.
`localProbCut`	`Numeric`, the local probability cutoff for phosphoproteomic data. Default is 0.75.
`annotation_col`	A `character` vector specifying the columns in fileTable to be used for sample annotation.
`onlyReviewed`	A `logical` value indicating whether to include only reviewed proteins. Default is `TRUE`.
`normalizeByProtein`	`Logical`, whether to normalize the data by protein. Default is `FALSE`.
`verbose`	`Logical` value indicating whether to print detailed information. Default is `FALSE`.

Details

The function performs the following steps:

- Reads and processes phosphoproteomic data using `readPhosphoExperimentDIA`.
- Reads and processes proteomic data using `readProteomeExperimentDIA`.
- Prepares sample annotations based on the provided fileTable and annotation_col.
- Constructs a `MultiAssayExperiment` object with the processed data and sample annotations.

The `readPhosphoExperimentDIA` and `readProteomeExperimentDIA` functions are used to read and filter the data for phosphoproteome and proteome experiments, respectively, and they must be available in the environment.

Value

A `MultiAssayExperiment` object containing the processed phosphoproteome and proteome data.

Examples

```
# Example usage:
file1 <- system.file("extdata", "phosDIA_1.xls", package = "SmartPhos")
file2 <- system.file("extdata", "proteomeDIA_1.xls", package = "SmartPhos")
# Create fileTable
fileTable <- data.frame(
  searchType = c("phosphoproteome", "proteome", "proteome"),
  fileName = c(file1, file2, file2),
  id = c("Sample_1", "sample1", "sample2"),
  outputID = c("s1", "s2", "s3")
)
# Call the function
readExperimentDIA(fileTable, localProbCut = 0.75, annotation_col = c("id"),
onlyReviewed = FALSE, normalizeByProtein = FALSE)
```

---

readOnePhos — *Read and Filter Phosphorylation Data for a Specific Sample*

Description

`readOnePhos` reads phosphorylation data from an input table, filters it based on localization probability, score difference, and intensity, and returns the filtered data for a specific sample.

Usage

```
readOnePhos(
  inputTab,
  sampleName,
  localProbCut = 0.75,
  scoreDiffCut = 5,
  multiMap
)
```

Arguments

`inputTab`	A `data.table` or `data.frame` containing phosphorylation data with columns for localization probability, score difference, and intensity for various samples.
`sampleName`	A `character` string specifying the sample name to filter data for.
`localProbCut`	A `numeric` value specifying the cutoff for localization probability. Default is 0.75.
`scoreDiffCut`	A `numeric` value specifying the cutoff for score difference. Default is 5.
`multiMap`	A `logical` value indicating whether to allow multiple mapping (not used in this function but could be relevant for further extensions).

Details

The function filters the input phosphorylation data based on three criteria: localization probability, score difference, and intensity. Only rows that meet or exceed the specified cutoffs for these criteria and have non-zero intensity are retained. The filtered data is then returned with a unique identifier for each row.

Value

A `data.frame` containing the filtered phosphorylation data for the specified sample, with columns for intensity, Uniprot ID, gene name, position within proteins, amino acid residue, and sequence window.

`readOnePhosDIA` *Read Phosphorylation Data for One Sample from DIA*

Description

`readOnePhosDIA` reads and processes phosphorylation data for a single sample from a DIA experiment, applying filters for localization probability and removing duplicates if specified.

Usage

```
readOnePhosDIA(inputTab, sampleName, localProbCut = 0.75, removeDup = FALSE)
```

Arguments

`inputTab`	A `data.table` or `data.frame` containing phosphorylation data.
`sampleName`	A `character` string specifying the sample name.
`localProbCut`	A `numeric` value specifying the cutoff for localization probability. Default is 0.75.
`removeDup`	A `logical` value indicating whether to remove duplicate entries based on UniprotID and intensity. Default is `FALSE`.

Details

This function processes phosphorylation data for a single sample by filtering based on localization probability and non-zero intensity. It handles multiplicity by summarizing intensities and optionally removes duplicates. The resulting data is returned as a data.table with unique identifiers.

Value

A `data.table` containing the processed phosphorylation data for the specified sample.

`readOneProteom` *Read and Process One Proteomics Sample*

Description

`readOneProteom` reads and processes proteomics data for a single sample, applying filters for peptide count and optionally using LFQ quantification. It returns a `data.table` with useful columns and unique identifiers.

Usage

```
readOneProteom(inputTab, sampleName, pepNumCut = 1, ifLFQ = TRUE)
```

Arguments

`inputTab`	A `data.table` or `data.frame` containing the input data for the proteomics sample.
`sampleName`	A `character` string specifying the name of the sample to be processed.
`pepNumCut`	A `numeric` value specifying the minimum number of peptides required for a protein to be included. Default is 1.
`ifLFQ`	A `logical` value indicating whether to use LFQ quantification. Default is `TRUE`.

Details

This function processes proteomics data for a single sample by filtering based on the number of peptides and optionally using LFQ quantification. It ensures that unique identifiers are created for each protein, and removes rows with missing or zero quantification values.

Value

A `data.table` with the processed proteomics data, including columns for intensity, Uniprot ID, peptide counts, and gene names.

readOneProteomDIA *Read and Process a Single DIA Proteomics Sample*

Description

readOneProteomDIA reads and processes data from a single DIA proteomics sample, applying filtering and data transformation steps.

Usage

```
readOneProteomDIA(inputTab, sampleName)
```

Arguments

inputTab	A data.table or data.frame containing the input data.
sampleName	A character string specifying the sample name.

Details

This function processes DIA proteomics data for a single sample by filtering out rows with non-quantitative data, converting character values to numeric, and renaming columns for consistency. It also ensures that each protein group has a unique identifier.

Value

A data.table containing the processed data with columns for intensity, UniProt ID, and gene name.

readPhosphoExperiment *Read Phosphorylation Experiment Data*

Description

readPhosphoExperiment reads and processes phosphorylation experiment data from multiple files, filtering based on localization probability and score difference, and constructs a SummarizedExperiment object.

Usage

```
readPhosphoExperiment(fileTable, localProbCut = 0.75, scoreDiffCut = 5)
```

Arguments

`fileTable`	A `data.table` or `data.frame` containing information about the files, including columns for file names, sample names, and other relevant metadata. It must include a column named "searchType" with value "phosphoproteome" for relevant entries.
`localProbCut`	A `numeric` value specifying the cutoff for localization probability. Default is 0.75.
`scoreDiffCut`	A `numeric` value specifying the cutoff for score difference. Default is 5.

Details

This function reads phosphorylation data from multiple files as specified in fileTable, filters the data based on localization probability and score difference, and removes reverse and potential contaminant entries. It constructs an intensity matrix and annotation data, which are then used to create a `SummarizedExperiment` object.

Value

A `SummarizedExperiment` object containing the processed phosphorylation data.

Examples

```
file1 <- system.file("extdata", "phosDDA_1.xls", package = "SmartPhos")
file2 <- system.file("extdata", "proteomeDDA_1.xls", package = "SmartPhos")
# Create fileTable
fileTable <- data.frame(
   searchType = c("phosphoproteome", "proteome"),
   fileName = c(file1, file2),
   sample = c("Sample1", "sample1"),
   id = c("s1", "s2")
)
# Call the function
readPhosphoExperiment(fileTable, localProbCut = 0.75, scoreDiffCut = 5)
```

`readPhosphoExperimentDIA` *Read Phosphorylation Experiment Data from DIA*

Description

`readPhosphoExperimentDIA` reads and processes phosphorylation data from DIA experiments, applying filters for localization probability, and optionally including only reviewed proteins. It constructs a `SummarizedExperiment` object.

Usage

```
readPhosphoExperimentDIA(
  fileTable,
  localProbCut = 0.75,
  onlyReviewed = TRUE,
  showProgressBar = FALSE
)
```

Arguments

`fileTable`	A `data.table` or `data.frame` containing metadata about the files to read. Must include columns fileName, searchType, and optionally outputID.
`localProbCut`	A `numeric` value specifying the cutoff for localization probability. Default is 0.75.
`onlyReviewed`	A `logical` value indicating whether to include only reviewed proteins. Default is `TRUE`.
`showProgressBar`	A `logical` value indicating whether to show a progress bar. Default is `FALSE`.

Details

This function processes phosphorylation data from DIA experiments by filtering based on localization probability and non-zero intensity, handling multiplicity, and optionally including only reviewed proteins. The resulting data is returned as a `SummarizedExperiment` object with annotations and an intensity matrix.

Value

A `SummarizedExperiment` object containing the processed phosphorylation data.

Examples

```
file <- system.file("extdata", "phosDIA_1.xls", package = "SmartPhos")
fileTable <- data.frame(searchType = "phosphoproteome", fileName = file,
id = c("Sample_1"))
readPhosphoExperimentDIA(fileTable, localProbCut = 0.75,
onlyReviewed = FALSE, showProgressBar = FALSE)
```

---

readProteomeExperiment

Read and Process Proteomics Experiment Data

Description

`readProteomeExperiment` reads and processes proteomics data from multiple samples, applying various quality filters, and returns a `SummarizedExperiment` object.

Usage

```
readProteomeExperiment(
  fileTable,
  fdrCut = 0.1,
  scoreCut = 10,
  pepNumCut = 1,
  ifLFQ = TRUE
)
```

Arguments

`fileTable`	A `data.table` or `data.frame` containing the file information with columns for file names, sample names, and IDs.
`fdrCut`	A `numeric` value specifying the maximum false discovery rate (FDR) threshold. Default is 0.1.
`scoreCut`	A `numeric` value specifying the minimum score threshold. Default is 10.
`pepNumCut`	A `numeric` value specifying the minimum number of peptides required for a protein to be included. Default is 1.
`ifLFQ`	A `logical` value indicating whether to use LFQ quantification. Default is `TRUE`.

Details

This function processes proteomics data by filtering based on FDR, score, and peptide count, and optionally using LFQ quantification. It aggregates the data from multiple samples and constructs a `SummarizedExperiment` object.

Value

A `SummarizedExperiment` object containing the processed proteomics data.

Examples

```
file1 <- system.file("extdata", "phosDDA_1.xls", package = "SmartPhos")
file2 <- system.file("extdata", "proteomeDDA_1.xls", package = "SmartPhos")
# Create fileTable
fileTable <- data.frame(
   searchType = c("phosphoproteome", "proteome"),
   fileName = c(file1, file2),
   sample = c("Sample1", "sample1"),
   id = c("s1", "s2")
)
# Call the function
readProteomeExperiment(fileTable, fdrCut = 0.1, scoreCut = 10,
pepNumCut = 1, ifLFQ = TRUE)
```

readProteomeExperimentDIA

Read and Process a DIA Proteome Experiment

Description

`readProteomeExperimentDIA` reads and processes DIA (Data-Independent Acquisition) proteome data from multiple files and constructs a `SummarizedExperiment` object.

Usage

```
readProteomeExperimentDIA(fileTable, showProgressBar = FALSE)
```

Arguments

`fileTable` A `data frame` containing metadata about the files to be read. Must contain columns searchType, fileName, id, and optionally outputID.

`showProgressBar` `Logical`, whether to show a progress bar during processing. Default is `FALSE`.

Value

A `SummarizedExperiment` object containing the processed proteome data. #' @details The function performs the following steps:

- Filters the ‘fileTable‘ to include only rows where ‘searchType‘ is "proteome".
- For each file specified in ‘fileTable‘, reads the data using ‘data.table::fread‘.
- Removes rows where the ‘PG.ProteinGroups‘ column is NA or empty.
- Processes each sample in parallel using ‘BiocParallel::bplapply‘, applying the ‘readOneProteomDIA‘ function to filter and clean the data for each sample.
- Combines the processed data from all files.
- Constructs a matrix of intensities with rows corresponding to proteins and columns corresponding to samples.
- Constructs a ‘SummarizedExperiment‘ object with the intensity matrix and protein annotations.

The `readOneProteomDIA` function is used to read and filter the data for each individual sample, and it must be available in the environment.

Examples

```
file <- system.file("extdata", "proteomeDIA_1.xls", package = "SmartPhos")
fileTable <- data.frame(searchType = "proteome", fileName = file,
id = c("sample1", "sample2"))
readProteomeExperimentDIA(fileTable)
```

runFisher *Perform Fisher's Exact Test on Gene Sets*

Description

runFisher performs Fisher's Exact Test to determine the enrichment of a set of genes within reference gene sets.

Usage

```
runFisher(genes, reference, inputSet, ptm = FALSE)
```

Arguments

genes	A character vector of genes of interest.
reference	A character vector of reference genes.
inputSet	A list containing gene set collections. If ptm is TRUE, this should be a data frame with specific columns.
ptm	Logical. If TRUE, perform the test on post-translational modification (PTM) gene sets. Default is FALSE.

Details

The function can operate in two modes: standard gene sets and PTM-specific gene sets. For PTM-specific gene sets, additional filtering and processing are performed.

Value

A data frame with the results of the Fisher's Exact Test, including the gene set name, the number of genes in the set, set size, p-value, adjusted p-value, and the genes in the set.

Examples

```
library(SummarizedExperiment)
library(piano)
# Load multiAssayExperiment object
data("dda_example")
# Get SummarizedExperiment object
se <- dda_example[["Proteome"]]
colData(se) <- colData(dda_example)
# Preprocess the proteome assay
result <- preprocessProteome(se, normalize = TRUE)
# Call the function to perform differential expression analyis
de <- performDifferentialExp(se = result, assay = "Intensity",
method = "limma", reference = "1stCrtl", target = "EGF",
condition = "treatment")
genesList <- unique(de$resDE$Gene)
referenceList <- unique(SummarizedExperiment::rowData(result)$Gene)
```

```
genesetPath <- appDir <- system.file("shiny-app/geneset",
package = "SmartPhos")
inGMT <- loadGSC(paste0(genesetPath,"/Cancer_Hallmark.gmt"),type="gmt")
# Run the function
runFisher(genes = genesList, reference = referenceList, inputSet = inGMT)
```

---

runGSEAforPhospho *Run GSEA for Phosphorylation Data*

Description

runGSEAforPhospho performs Gene Set Enrichment Analysis (GSEA) for phosphorylation data.

Usage

```
runGSEAforPhospho(
  geneStat,
  ptmSetDb,
  nPerm,
  weight = 1,
  correl.type = c("rank", "symm.rank", "z.score"),
  statistic = c("Kolmogorov-Smirnov", "area.under.RES"),
  min.overlap = 5
)
```

Arguments

geneStat	A data frame containing gene statistics, with gene names as row names and a column named 'stat' for the statistics.
ptmSetDb	A data frame of post-translational modification (PTM) signature sets.
nPerm	A numeric value specifying the number of permutations for the null distribution.
weight	A numeric value for the weight parameter in the GSEA algorithm. If weight == 0 then the test statistics do not matter. Default is 1.
correl.type	A character string specifying the correlation type. Options are "rank", "symm.rank", and "z.score". Default is "rank".
statistic	A character string specifying the statistic to be used. Options are "Kolmogorov-Smirnov" and "area.under.RES". Default is "Kolmogorov-Smirnov".
min.overlap	A numeric specifying the minimum overlap required between gene sets and the input data. Default is 5.

Details

This function runs GSEA on phosphorylation data to identify enriched PTM sets. It calculates enrichment scores and p-values for each set, normalizes the scores, and adjusts p-values for multiple testing.

Value

A `tibble` with enrichment scores and associated statistics for each PTM set.

---

runPhosphoAdjustment *Run Phospho Adjustment*

Description

`runPhosphoAdjustment` performs phospho adjustment on a `MultiAssayExperiment` object to normalize the phosphoproteome data.

Usage

```
runPhosphoAdjustment(
  maeData,
  normalization = FALSE,
  minOverlap = 3,
  completeness = 0,
  ncore = 1
)
```

Arguments

`maeData`	A `MultiAssayExperiment` object containing phosphoproteome and full proteome data.
`normalization`	A `logical` value indicating whether to perform normalization. Default is `FALSE`.
`minOverlap`	A `numeric` value specifying the minimum number of overlapping peptides required between samples. Default is 3.
`completeness`	A `numeric` value indicating the required completeness of data for features to be included. Default is 0.
`ncore`	A `numeric` value specifying the number of cores to use for parallel processing. Default is 1.

Details

The function performs the following steps:

1. Defines an optimization function to minimize the sum of squared differences between pairs of samples.
2. Calculates the ratio matrix of phosphoproteome to full proteome data.
3. Subsets features based on completeness criteria.
4. Performs a sanity check to identify and exclude problematic samples.
5. Sets initial values for the adjustment factor based on column medians.
6. Estimates the adjustment factor using parallel optimization.
7. Adjusts the phosphoproteome measurements using the estimated adjustment factor.

Value

A `MultiAssayExperiment` object with adjusted phosphoproteome data.

---

runSmartPhos *Launch the SmartPhos Shiny Application*

Description

`runSmartPhos` launches the SmartPhos `Shiny` application, which provides an interactive interface for analyzing phosphoproteomic data.

Usage

```
runSmartPhos()
```

Details

The `runSmartPhos` function locates the `Shiny` app directory within the `SmartPhos` package and launches the application. If the app directory cannot be found, the function will stop and prompt the user to re-install the `SmartPhos` package.

Value

The function does not return a value; it starts the Shiny application for `SmartPhos`.

Examples

```
# To run the SmartPhos Shiny application, simply call:
# runSmartPhos()
```

---

splineFilter *Filter Expression Matrix Using Spline Models*

Description

`splineFilter` filters an expression `matrix` based on spline models fitted to time-series data, optionally considering treatment and subject ID.

Usage

```
splineFilter(
  exprMat,
  subjectID = NULL,
  time,
  df,
  pCut = 0.5,
  ifFDR = FALSE,
  treatment = NULL,
  refTreatment = NULL
)
```

Arguments

`exprMat`	A `numeric` matrix of expression data, where rows are features and columns are samples.
`subjectID`	`Character`. An optional vector of subject IDs corresponding to columns in exprMat. Default is `NULL`.
`time`	A `numeric` vector representing the time points corresponding to columns in exprMat.
`df`	A `numeric` value specifying the degrees of freedom for the spline basis.
`pCut`	A `numeric` value for the p-value cutoff to filter significant features. Default is 0.05.
`ifFDR`	A `logical` value indicating if the false discovery rate (FDR) should be used for filtering. If `FALSE`, raw p-values are used. Default is `FALSE`.
`treatment`	`Character`. An optional vector of treatment labels corresponding to columns in exprMat. Default is `NULL`.
`refTreatment`	`Character`. An optional reference treatment label for the treatment vector. Default is `NULL`.

Details

The function performs the following steps:

1. Converts time points from minutes to hours if both units are present.
2. Removes rows with missing values from the expression `matrix`.
3. Constructs a design matrix for the spline model, optionally including subject IDs and treatments.
4. Fits a linear model using the design matrix and performs empirical Bayes moderation.
5. Extracts significant features based on the specified p-value or FDR cutoff.

Value

A filtered expression `matrix` containing only the features that meet the significance criteria.

swissProt *swissProt*

Description

This is a high-quality, manually curated protein sequence database which provides a high level of annotations (such as the description of the function of a protein, structure of its domains, post-translational modifications, variants, etc.), a minimal level of redundancy and high level of integration with other databases.

Usage

```
data(swissProt)
```

Format

an object of "tbl_df" (tidy table)

Value

A `data frame` or `tibble` containing high-level annotations for manually curated proteins.

Examples

```
data(swissProt)
```

Index

∗ **datasets**
- dda_example, 9
- dia_example, 10
- Homo_sapien_kinase_substrate_network, 16
- Mus_musculus_kinase_substrate_network, 20
- swissProt, 53

addZeroTime, 3

calcKinaseScore, 4
checkRatioMat, 6
clusterEnrich, 6
clusterTS, 8

dda_example, 9
dia_example, 10

enrichDifferential, 10

generateInputTable, 12
generateInputTable_DIA, 13
getDecouplerNetwork, 14
getOneSymbol, 14
getRatioMatrix, 15

Homo_sapien_kinase_substrate_network, 16

intensityBoxPlot, 16

makeSmartPhosDirectory, 17
medianNorm, 18
mscale, 19
Mus_musculus_kinase_substrate_network, 20

normByFullProteome, 21

performCombinedNormalization, 22
performDifferentialExp, 23
plotAdjustmentResults, 25
plotHeatmap, 26
plotIntensity, 27
plotKinaseDE, 28
plotKinaseTimeSeries, 29
plotLogRatio, 30
plotMissing, 30
plotPCA, 31
plotTimeSeries, 32
plotVolcano, 34
preprocessPhos, 35
preprocessProteome, 36

readExperiment, 37
readExperimentDIA, 39
readOnePhos, 40
readOnePhosDIA, 41
readOneProteom, 42
readOneProteomDIA, 43
readPhosphoExperiment, 43
readPhosphoExperimentDIA, 44
readProteomeExperiment, 45
readProteomeExperimentDIA, 47
runFisher, 48
runGSEAforPhospho, 49
runPhosphoAdjustment, 50
runSmartPhos, 51

splineFilter, 51
swissProt, 53