Package 'PhosR'

March 26, 2024

Type Package

Title A set of methods and tools for comprehensive analysis of phosphoproteomics data

Version 1.12.0

Description PhosR is a package for the comprenhensive analysis of phosphoproteomic data. There are two major components to PhosR: processing and downstream analysis. PhosR consists of various processing tools for phosphoproteomics data including filtering, imputation, normalisation, and functional analysis for inferring active kinases and signalling pathways.

License GPL-3 + file LICENSE

Depends R (>= 4.2.0)

Imports ruv, e1071, dendextend, limma, pcaMethods, stats, RColorBrewer, circlize, dplyr, igraph, pheatmap, preprocessCore, tidyr, rlang, graphics, grDevices, utils, SummarizedExperiment, methods, S4Vectors, BiocGenerics, ggplot2, GGally, ggdendro, ggpubr, network, reshape2, ggtext, stringi

Encoding UTF-8

LazyData false

RoxygenNote 7.2.2

VignetteBuilder knitr

biocViews Software, ResearchField, Proteomics

Suggests testthat, knitr, rgl, sna, ClueR, directPA, rmarkdown, org.Rn.eg.db, org.Mm.eg.db, reactome.db, annotate, BiocStyle, stringr, calibrate

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

git_branch RELEASE_3_18

git_last_commit dfab942

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-03-25

Author Pengyi Yang [aut],
Taiyun Kim [aut, cre],
Hani Jieun Kim [aut]

Maintainer Taiyun Kim <taiyun.kim91@gmail.com>

R topics documented:

`createFrequencyMat` *Create frequency matrix*

Description

Create frequency matrix

Usage

```
createFrequencyMat(substrates.seq)
```

Arguments

`substrates.seq`	A substrate sequence

Value

A frequency matrix of amino acid from substrates.seq.

Examples

```
data("phospho_L6_ratio_pe")

# We will create a frequency matrix of Tfg S198 phosphosite.
idx = which(grepl("TFG\\;S198\\;", rownames(phospho.L6.ratio.pe)))
substrate.seq = Sequence(phospho.L6.ratio.pe)[idx]
freq.mat = createFrequencyMat(substrate.seq)
```

`frequencyScoring` *Frequency scoring*

Description

Frequency scoring

Usage

```
frequencyScoring(sequence.list, frequency.mat)
```

Arguments

`sequence.list`	A vector list of sequences
`frequency.mat`	A matrix output from ‘createFrequencyMat‘

Value

A vector of frequency score

Examples

```
data('phospho_L6_ratio_pe')
data('KinaseMotifs')

# Extracting first 10 sequences for demonstration purpose
seqs = Sequence(phospho.L6.ratio.pe)
seqs = seqs[seq(10)]

# extracting flanking sequences
seqWin = mapply(function(x) {
    mid <- (nchar(x)+1)/2
    substr(x, start=(mid-7), stop=(mid+7))
}, seqs)

# The first 10 for demonstration purpose
phospho.L6.ratio = SummarizedExperiment::assay(phospho.L6.ratio.pe,
    "Quantification")[seq(10),]

# minimum number of sequences used for compiling motif for each kinase.
numMotif=5

motif.mouse.list.filtered <-
    motif.mouse.list[which(motif.mouse.list$NumInputSeq >= numMotif)]

# scoring all phosphosites against all motifs
motifScoreMatrix <-
    matrix(NA, nrow=nrow(phospho.L6.ratio),
        ncol=length(motif.mouse.list.filtered))
```

```
rownames(motifScoreMatrix) <- rownames(phospho.L6.ratio)
colnames(motifScoreMatrix) <- names(motif.mouse.list.filtered)

# Scoring phosphosites against kinase motifs
for(i in seq_len(length(motif.mouse.list.filtered))) {
    motifScoreMatrix[,i] <-
        frequencyScoring(seqWin, motif.mouse.list.filtered[[i]])
    cat(paste(i, '.', sep=''))
}
```

---

getSPS *Generate set of stable phosphoporylated sites*

Description

Generate set of stable phosphoporylated sites

Usage

```
getSPS(phosData, assays, conds, num)
```

Arguments

phosData	a list of users' PhosphoExperiment objects from which generate SPSs
assays	an assay to use for each dataset in phosData
conds	a list of vector contains the conditions labels for each sample in the phosphoExperiment objects
num	the number of identified SPSs, by default is 100

Value

A vectors of stably phosphorylated sites

Examples

```
library(stringr)

data("phospho_L6_ratio_pe")
data("phospho.liver.Ins.TC.ratio.RUV.pe")
data("phospho.cells.Ins.pe")

ppe1 <- phospho.L6.ratio.pe
ppe2 <- phospho.liver.Ins.TC.ratio.RUV.pe
ppe3 <- phospho.cells.Ins.pe
grp3 = gsub('_[0-9]{1}', '', colnames(ppe3))

cond.list <- list(grp1 = gsub("_.+", "", colnames(ppe1)),
```

```
                grp2 = stringr::str_sub(colnames(ppe2), end=-5),
                grp3 = grp3)

ppe3 <- selectGrps(ppe3, grps = grp3, 0.5, n=1)
ppe3 <- tImpute(ppe3)

# convert matrix to ratio
FL83B.ratio <- SummarizedExperiment::assay(ppe3,"imputed")[, seq(12)] -
    rowMeans(
        SummarizedExperiment::assay(ppe3,"imputed")[,grep("FL83B_Control",
        colnames(ppe3))])
Hepa.ratio <- SummarizedExperiment::assay(ppe3,"imputed")[, seq(13,24,1)] -
    rowMeans(
        SummarizedExperiment::assay(ppe3, "imputed")[,grep("Hepa1.6_Control",
        colnames(ppe3))])
SummarizedExperiment::assay(ppe3, "Quantification") <-
    cbind(FL83B.ratio, Hepa.ratio)

ppe.list <- list(ppe1, ppe2, ppe3)

inhouse_SPSs <- getSPS(ppe.list, conds = cond.list)
```

---

hSEGs *A list of Stably Expressed Genes (SEGs)*

Description

A list of stably expressed genes (SEGs) in mouse and human identified from a collection of single-cell RNA-sequencing data. See Lin et al., Evaluating stably expressed genes in single cells, GigaScience, 8(9):giz106, https://doi.org/10.1093/gigascience/giz106 for more details

Usage

```
data(SEGs)
```

Format

An object of class `character` of length 1076.

KinaseFamily *KinaseFamily*

Description

A summary table of kinase family

Usage

```
data(KinaseFamily)
```

Format

An object of class `matrix` (inherits from `array`) with 425 rows and 6 columns.

kinaseSubstrateHeatmap *Kinase-substrate annotation prioritisation heatmap*

Description

Kinase-substrate annotation prioritisation heatmap

Usage

```
kinaseSubstrateHeatmap(
  phosScoringMatrices,
  top = 3,
  printPlot = NULL,
  filePath = "./kinaseSubstrateHeatmap.pdf",
  width = 10,
  height = 10
)
```

Arguments

phosScoringMatrices	a matrix returned from kinaseSubstrateScore.
top	the number of top ranked phosphosites for each kinase to be included in the heatmap. Default is 1.
printPlot	indicate whether the plot should be saved as a PDF in the specified directory. Default is NULL, otherwise specify TRUE.
filePath	path name to save the plot as a PDF file. Default saves in the working directory.
width	width of PDF.
height	height of PDF.

Value

a pheatmap object.

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

rownames(L6.phos.std) <- paste0(GeneSymbol(ppe), ";", Residue(ppe),
    Site(ppe), ";")[idx]

L6.phos.seq <- Sequence(ppe)[idx]

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
    L6.phos.seq, numMotif = 5, numSub = 1)

kinaseSubstrateHeatmap(L6.matrices)
kinaseSubstrateHeatmap(L6.matrices, printPlot=TRUE)
```

---

kinaseSubstratePred *kinaseSubstratePred*

Description

A machine learning approach for predicting specific kinase for a given substrate. This prediction framework utilise adaptive sampling.

Usage

```
kinaseSubstratePred(
    phosScoringMatrices,
    ensembleSize = 10,
    top = 50,
    cs = 0.8,
    inclusion = 20,
    iter = 5,
    verbose = TRUE
)
```

Arguments

phosScoringMatrices	An output of kinaseSubstrateScore.
ensembleSize	An ensemble size.
top	a number to select top kinase substrates.
cs	Score threshold.
inclusion	A minimal number of substrates required for a kinase to be selected.
iter	A number of iterations for adaSampling.
verbose	Default to TRUE to show messages during the progress. All messages will be suppressed if set to FALSE

Value

Kinase prediction matrix

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
```

```
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

rownames(L6.phos.std) <- paste0(GeneSymbol(ppe), ";", Residue(ppe),
    Site(ppe), ";")[idx]

L6.phos.seq <- Sequence(ppe)[idx]

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
    L6.phos.seq, numMotif = 5, numSub = 1)
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)
```

`kinaseSubstrateProfile` *Kinase substrate profiling*

Description

This function generates substrate profiles for kinases that have one or more substrates quantified in the phosphoproteome data.

Usage

```
kinaseSubstrateProfile(substrate.list, mat)
```

Arguments

`substrate.list`	a list of kinases with each element containing an array of substrates.
`mat`	a matrix with rows correspond to phosphosites and columns correspond to samples.

Value

Kinase profile list.

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
```

```
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

ks.profile.list <- kinaseSubstrateProfile(PhosphoSite.mouse, L6.phos.std)
```

`kinaseSubstrateScore` *Kinase substrate scoring*

Description

This function generates substrate scores for kinases that pass filtering based on both motifs and dynamic profiles

Usage

```
kinaseSubstrateScore(
  substrate.list,
  mat,
  seqs,
  numMotif = 5,
  numSub = 1,
  species = "mouse",
  verbose = TRUE
)
```

Arguments

`substrate.list`	A list of kinases with each element containing an array of substrates.
`mat`	A matrix with rows correspond to phosphosites and columns correspond to samples.
`seqs`	An array containing aa sequences surrounding each of all phosphosites. Each sequence has length of 15 (-7, p, +7).
`numMotif`	Minimum number of sequences used for compiling motif for each kinase. Default is 5.
`numSub`	Minimum number of phosphosites used for compiling phosphorylation profile for each kinase. Default is 1.

species	Motif list species to be used. Currently there are mouse (default), human and rat.
verbose	Default to TRUE to show messages during the progress. All messages will be suppressed if set to FALSE

Value

A list of 4 elements. motifScoreMatrix, profileScoreMatrix, combinedScoreMatrix, ksActivityMatrix (kinase activity matrix) and their weights.

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

rownames(L6.phos.std) <- paste0(GeneSymbol(ppe), ";", Residue(ppe),
    Site(ppe), ";")[idx]

L6.phos.seq <- Sequence(ppe)[idx]

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
    L6.phos.seq, numMotif = 5, numSub = 1)
```

---

matANOVA *ANOVA test*

Description

Performs an ANOVA test and returns its adjusted p-value

Usage

```
matANOVA(mat, grps)
```

Arguments

mat	An p by n matrix where p is the number of phosphosites and n is the number of samples
grps	A vector of length n, with group or time point information of the samples

Value

A vector of multiple testing adjusted p-values

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")
ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.pe = RUVphospho(phospho.L6.ratio.pe,
                                 M = design, k = 3,ctl = ctl)
phosphoL6 = SummarizedExperiment::assay(phospho.L6.ratio.pe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
```

---

meanAbundance *Obtain average expression from replicates*

Description

Obtain average expression from replicates

Usage

```
meanAbundance(mat, grps)
```

Arguments

mat	a matrix with rows correspond to phosphosites and columns correspond to samples.
grps	a string specifying the grouping (replciates).

Value

a matrix with mean expression from replicates

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")
ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.pe = RUVphospho(phospho.L6.ratio.pe,
                                 M = design, k = 3,ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(phospho.L6.ratio.pe, "normalised")
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
```

---

medianScaling *Median centering and scaling*

Description

Median centering and scaling of an input numeric matrix

Usage

```
medianScaling(mat, scale = FALSE, grps = NULL, reorder = FALSE, assay = NULL)
```

Arguments

mat	a matrix with rows correspond to phosphosites and columns correspond to samples.
scale	a boolean flag indicating whether to scale the samples.
grps	a string or factor specifying the grouping (replciates).
reorder	To reorder the columns by group (grps). By default (reorder=FALSE), original column order is maintained.
assay	an assay to be selected if mat is a PhosphoExperiment object.

Value

A median scaled matrix

Examples

```
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(phospho.cells.Ins.filtered,
            0.5,
            grps)[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,seq(5)] <- ptImpute(
    phospho.cells.Ins.impute[,seq(6,10)],
    phospho.cells.Ins.impute[,seq(5)], percent1 = 0.6,
    percent2 = 0, paired = FALSE)

phospho.cells.Ins.ms <-
    medianScaling(phospho.cells.Ins.impute, scale = FALSE)
```

---

minmax *Minmax scaling*

Description

Perform a minmax standardisation to scale data into 0 to 1 range

Usage

```
minmax(mat)
```

Arguments

mat	a matrix with rows correspond to phosphosites and columns correspond to condition

Value

Minmax standardised matrix

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

ks.profile.list <- kinaseSubstrateProfile(PhosphoSite.mouse, L6.phos.std)

data(KinaseMotifs)

numMotif = 5
numSub = 1

motif.mouse.list.filtered <-
    motif.mouse.list[which(motif.mouse.list$NumInputSeq >= numMotif)]
ks.profile.list.filtered <-
    ks.profile.list[which(ks.profile.list$NumSub >= numSub)]

# scoring all phosphosites against all motifs
motifScoreMatrix <-
    matrix(NA, nrow=nrow(L6.phos.std),
    ncol=length(motif.mouse.list.filtered))
rownames(motifScoreMatrix) <- rownames(L6.phos.std)
colnames(motifScoreMatrix) <- names(motif.mouse.list.filtered)
```

```
L6.phos.seq <- Sequence(ppe)[idx]

# extracting flanking sequences
seqWin = mapply(function(x) {
    mid <- (nchar(x)+1)/2
    substr(x, start=(mid-7), stop=(mid+7))
}, L6.phos.seq)


print('Scoring phosphosites against kinase motifs:')
for(i in seq_len(length(motif.mouse.list.filtered))) {
    motifScoreMatrix[,i] <-
        frequencyScoring(seqWin, motif.mouse.list.filtered[[i]])
        cat(paste(i, '.', sep=''))
}
motifScoreMatrix <- minmax(motifScoreMatrix)
```

`mIntersect` *Multi-intersection, union*

Description

A recusive loop for intersecting multiple sets.

Usage

```
mIntersect(x, y, ...)
mUnion(x, y, ...)
```

Arguments

`x, y, ...`	objects to find intersection/union.

Value

An intersection/union of input parameters

Examples

```
data('phospho_liverInsTC_RUV_sample')
data('phospho_L6_ratio')

site1 <- gsub('~[STY]', ';',
            sapply(strsplit(rownames(phospho.L6.ratio), ';'),
                    function(x){paste(toupper(x[2]), x[3], sep=';')}))
site2 <- rownames(phospho.liver.Ins.TC.ratio.RUV)

# step 2: rank by fold changes
```

```
treatment.grps = split(seq(ncol(phospho.L6.ratio)),
    gsub('_exp\\d+', '', colnames(phospho.L6.ratio)))
tmp <- do.call(
    cbind,
    lapply(treatment.grps, function(i){
        rowMeans(phospho.L6.ratio[,i])
    })
)
site1 <- t(sapply(split(data.frame(tmp), site1), colMeans))[,-1]

treatment.grps = split(
    seq(ncol(phospho.liver.Ins.TC.ratio.RUV)),
    gsub('(Intensity\\.)(.*)(\\_Bio\\d+)', '\\2',
        colnames(phospho.liver.Ins.TC.ratio.RUV)
    )
)
tmp <- do.call(
    cbind,
    lapply(
        treatment.grps,
        function(i){
            rowMeans(phospho.liver.Ins.TC.ratio.RUV[,i])
        }
    )
)
site2 <- t(sapply(split(data.frame(tmp), site2), colMeans))

o <- mIntersect(site1, site2)
```

`motif.human.list` *List of human kinase motifs*

Description

A list of human kinase motifs and their sequence probability matrix.

Usage

```
data(KinaseMotifs)
```

Format

An object of class `list` of length 380.

motif.mouse.list *List of mouse kinase motifs*

Description

A list of mouse kinase motifs and their sequence probability matrix.

Usage

```
data(KinaseMotifs)
```

Format

An object of class `list` of length 250.

motif.rat.list *List of rat kinase motifs*

Description

A list of rat kinase motifs and their sequence probability matrix.

Usage

```
data(KinaseMotifs)
```

Format

An object of class `list` of length 159.

mSEGs *A list of Stably Expressed Genes (SEGs)*

Description

A list of stably expressed genes (SEGs) in mouse and human identified from a collection of single-cell RNA-sequencing data. See Lin et al., Evaluating stably expressed genes in single cells, GigaScience, 8(9):giz106, https://doi.org/10.1093/gigascience/giz106 for more details

Usage

```
data(SEGs)
```

Format

An object of class `character` of length 916.

pathwayOverrepresent *phosphosite/Gene set over-representation analysis*

Description

This function performes phosphosite (or gene) set over-representation analysis using Fisher's exact test.

Usage

```
pathwayOverrepresent(geneSet, annotation, universe, alter = "greater")
```

Arguments

geneSet	an array of gene or phosphosite IDs (IDs are gene symbols etc that match to your pathway annotation list).
annotation	a list of pathways with each element containing an array of gene or phosphosite IDs.
universe	the universe/backgrond of all genes or phosphosites in your profiled dataset.
alter	test for enrichment ('greater', default), depletion ('less'), or 'two.sided'.

Value

A matrix of pathways and their associated substrates and p-values.

Examples

```
library(limma)
library(org.Rn.eg.db)
library(reactome.db)
library(annotate)


data('phospho_L6_ratio_pe')
data('SPSs')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")
```

```
# fit linear model for each phosphosite
f <- grps
X <- model.matrix(~ f - 1)
fit <- lmFit(phosphoL6, X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)

DE1.RUV <- c(sum(table.AICAR[,'adj.P.Val'] < 0.05),
    sum(table.Ins[,'adj.P.Val'] < 0.05),
    sum(table.AICARIns[,'adj.P.Val'] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)

DE2.RUV <- c(sum(table.AICARInsVSIns[,'adj.P.Val'] < 0.05),
    sum(table.AICARInsVSAICAR[,'adj.P.Val'] < 0.05))

o <- rownames(table.AICARInsVSIns)
Tc <- cbind(table.Ins[o,'logFC'], table.AICAR[o,'logFC'],
            table.AICARIns[o,'logFC'])
rownames(Tc) = gsub('(.*)(;[A-Z])([0-9]+)(;)', '\\1;\\3;', o)
colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')

# summary phosphosite-level information to proteins for performing downstream
# gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub(';.+', '', rownames(Tc)),
    stat=apply(abs(Tc), 1, max), by = 'max')
geneSet <- names(sort(Tc.gene[,1],
                    decreasing = TRUE))[seq(round(nrow(Tc.gene) * 0.1))]
#lapply(PhosphoSite.rat, function(x){gsub(';[STY]', ';', x)})


# Preparing Reactome annotation for our pathways analysis
pathways = as.list(reactomePATHID2EXTID)

path_names = as.list(reactomePATHID2NAME)
name_id = match(names(pathways), names(path_names))
names(pathways) = unlist(path_names)[name_id]

pathways = pathways[which(grepl("Rattus norvegicus", names(pathways),
    ignore.case = TRUE))]

pathways = lapply(pathways, function(path) {
    gene_name = unname(getSYMBOL(path, data = "org.Rn.eg"))
    toupper(unique(gene_name))
```

```
})


# 1D gene-centric pathway analysis
path1 <- pathwayOverrepresent(geneSet, annotation=pathways,
    universe = rownames(Tc.gene), alter = 'greater')
```

---

pathwayRankBasedEnrichment

Phosphosite/Gene set enrichment analysis

Description

This function performes phosphosite (or gene) set enrichment analysis using Wilcoxon Rank Sum test.

Usage

```
pathwayRankBasedEnrichment(geneStats, annotation, alter = "greater")
```

Arguments

geneStats	an array of statistics (e.g. log2 FC) of all quantified genes or phosphosite with names of the array as gene or phosphosite IDs.
annotation	a list of pathways with each element containing an array of gene IDs.
alter	test for enrichment ('greater', default), depletion ('less'), or 'two.sided'.

Value

A matrix of pathways and their associated substrates and p-values.

Examples

```
library(limma)

library(org.Rn.eg.db)
library(reactome.db)
library(annotate)

data('phospho_L6_ratio_pe')
data('SPSs')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
```

```
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# fit linear model for each phosphosite
f <- grps
X <- model.matrix(~ f - 1)
fit <- lmFit(phosphoL6, X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)

DE1.RUV <- c(sum(table.AICAR[,'adj.P.Val'] < 0.05),
    sum(table.Ins[,'adj.P.Val'] < 0.05),
    sum(table.AICARIns[,'adj.P.Val'] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)

DE2.RUV <- c(sum(table.AICARInsVSIns[,'adj.P.Val'] < 0.05),
    sum(table.AICARInsVSAICAR[,'adj.P.Val'] < 0.05))

o <- rownames(table.AICARInsVSIns)
Tc <- cbind(table.Ins[o,'logFC'], table.AICAR[o,'logFC'],
            table.AICARIns[o,'logFC'])
rownames(Tc) = gsub('(.*)(;[A-Z])([0-9]+)(;)', '\\1;\\3;', o)
colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')

# summary phosphosite-level information to proteins for performing downstream
#  gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub(';.+', '', rownames(Tc)),
    stat=apply(abs(Tc), 1, max), by = 'max')

# Preparing Reactome annotation for our pathways analysis
pathways = as.list(reactomePATHID2EXTID)

path_names = as.list(reactomePATHID2NAME)
name_id = match(names(pathways), names(path_names))
names(pathways) = unlist(path_names)[name_id]

pathways = pathways[which(grepl("Rattus norvegicus", names(pathways),
    ignore.case = TRUE))]

pathways = lapply(pathways, function(path) {
```

```
    gene_name = unname(getSYMBOL(path, data = "org.Rn.eg"))
    toupper(unique(gene_name))
})

# 1D gene-centric pathway analysis
path2 <- pathwayRankBasedEnrichment(Tc.gene[,1],
                                    annotation=pathways,
                                    alter = 'greater')
```

---

phosCollapse *Summarising phosphosites to proteins*

Description

Summarising phosphosite-level information to proteins for performing downstream gene-centric analyses.

Usage

```
phosCollapse(mat, id, stat, by='min')
```

Arguments

mat	a matrix with rows correspond to phosphosites and columns correspond to samples.
id	an array indicating the groupping of phosphosites etc.
stat	an array containing statistics of phosphosite such as phosphorylation levels.
by	how to summarise phosphosites using their statistics. Either by 'min' (default), 'max', or 'mid'.

Value

A matrix summarised to protein level

Examples

```
library(limma)

data('phospho_L6_ratio_pe')
data('SPSs')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
```

```
                ";", sep = "")

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.pe = RUVphospho(phospho.L6.ratio.pe,
                                 M = design, k = 3, ctl = ctl)

# fit linear model for each phosphosite
f <- grps
X <- model.matrix(~ f - 1)
fit <- lmFit(SummarizedExperiment::assay(phospho.L6.ratio.pe, "normalised"), X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)

DE1.RUV <- c(sum(table.AICAR[,'adj.P.Val'] < 0.05),
    sum(table.Ins[,'adj.P.Val'] < 0.05),
    sum(table.AICARIns[,'adj.P.Val'] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)

DE2.RUV <- c(sum(table.AICARInsVSIns[,'adj.P.Val'] < 0.05),
    sum(table.AICARInsVSAICAR[,'adj.P.Val'] < 0.05))

o <- rownames(table.AICARInsVSIns)
Tc <- cbind(table.Ins[o,'logFC'], table.AICAR[o,'logFC'],
            table.AICARIns[o,'logFC'])
rownames(Tc) = gsub('(.*)(;[A-Z])([0-9]+)(;)', '\\1;\\3;', o)
colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')

# summary phosphosite-level information to proteins for performing downstream
# gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub(';.+', '', rownames(Tc)),
    stat=apply(abs(Tc), 1, max), by = 'max')
```

---

phospho.cells.Ins *phospho.cells.Ins*

Description

A subset of phosphoproteomics dataset generated by Humphrey et al., [doi:10.1038/nbt.3327] from two mouse liver cell lines (Hepa1.6 and FL38B) that were treated with either PBS (mock) or insulin.

A phosphoproteome Object containing a subset of phosphoproteomics dataset generated by Humphrey et al., [doi:10.1038/nbt.3327] from two mouse liver cell lines (Hepa1.6 and FL38B) that were treated with either PBS (mock) or insulin.

Usage

```
data(phospho.cells.Ins.sample)

data(phospho.cells.Ins.pe)
```

Format

An object of class `matrix` (inherits from `array`) with 49617 rows and 24 columns.

An object of class `matrix` (inherits from `array`) with 49617 rows and 24 columns.

Source

doi: 10.1038/nbt.3327 (PXD001792)

doi: 10.1038/nbt.3327 (PXD001792)

References

Humphrey et al., 2015, doi: 10.1038/nbt.3327

Humphrey et al., 2015, doi: 10.1038/nbt.3327

---

phospho.L6.ratio *phospho.L6.ratio*

Description

An L6 myotube phosphoproteome dataset (accession number: PXD019127).

Usage

```
data(phospho_L6_ratio)
```

Format

An object of class `matrix` (inherits from `array`) with 6660 rows and 12 columns.

Source

PRIDE accesion number: PXD001792

phospho.L6.ratio.pe *phospho_L6_ratio_pe*

Description

L6 myotube phosphoproteome dataset (accession number: PXD019127).

Usage

```
data(phospho_L6_ratio_pe)
```

Format

An PhosphoExperiment object

Source

PRIDE accesion number: PXD001792

phospho.liver.Ins.TC.ratio.RUV *phospho_liverInsTC_RUV_sample*

Description

A subset of phosphoproteomics dataset integrated from two time-course datasets of early and intermediate insulin signalling in mouse liver upon insulin stimulation.

Usage

```
data(phospho_liverInsTC_RUV_sample)
```

Format

An object of class `matrix` (inherits from `array`) with 5000 rows and 90 columns.

Source

PRIDE accesion number: PXD001792

References

Humphrey et al., 2015

phospho.liver.Ins.TC.ratio.RUV.pe

phospho.liver.Ins.TC.ratio.RUV.pe

Description

A subset of phosphoproteomics dataset integrated from two time-course datasets of early and intermediate insulin signalling in mouse liver upon insulin stimulation.

Usage

```
data(phospho.liver.Ins.TC.ratio.RUV.pe)
```

Format

A Phosphoproteome Object

Source

PRIDE accesion number: PXD001792

References

Humphrey et al., 2015

PhosphoExperiment-class

The PhosphoExperiment class

Description

The PhosphoExperiment class

Usage

```
PhosphoExperiment(
  ...,
  UniprotID = c(),
  GeneSymbol = c(),
  Site = c(),
  Residue = c(),
  Sequence = c(),
  Localisation = c()
)
```

Arguments

...	Arguments parsed, identical to those used to create SummarizedExperiment.
UniprotID	A character vector of Uniprot ID
GeneSymbol	A character vector of gene symbol
Site	A numeric vector of phosphorylation site
Residue	A character vector of site residue
Sequence	A character vector of sequences
Localisation	A localisation score.

Examples

```
data(phospho_L6_ratio)
quant <- as.matrix(phospho.L6.ratio)
uniprot <- as.character(sapply(strsplit(rownames(quant),";"),
    function(x) x[[2]]))
symbol <- as.character(sapply(strsplit(rownames(quant),";"),
    function(x) x[[2]]))
site <- as.numeric(gsub("[STY]","",sapply(strsplit(rownames(quant),";"),
    function(x) x[[3]])))
res <- as.character(gsub("[0-9]","",sapply(strsplit(rownames(quant),";"),
    function(x) x[[3]])))
seq <- as.character(sapply(strsplit(rownames(quant),";"),
    function(x) x[[4]]))
phosData <- PhosphoExperiment(assays = list(Quantification = quant),
    UniprotID = uniprot, Site = site, GeneSymbol = symbol, Residue = res,
    Sequence = seq)
```

PhosphoSite.human *PhosphoSitePlus annotations for human*

Description

The data object contains the annotations of kinases and their conrresponding substrates as phosphorylation sites in human. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage

```
data(PhosphoSitePlus)
```

Format

An object of class `list` of length 379.

Source

https://www.phosphosite.org

PhosphoSite.mouse *PhosphoSitePlus annotations for mouse*

Description

The data object contains the annotations of kinases and their conrresponding substrates as phosphorylation sites in mouse. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage

```
data(PhosphoSitePlus)
```

Format

An object of class `list` of length 260.

Source

https://www.phosphosite.org

PhosphoSite.rat *PhosphoSitePlus annotations for rat*

Description

The data object contains the annotations of kinases and their conrresponding substrates as phosphorylation sites in rat. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage

```
data(PhosphoSitePlus)
```

Format

An object of class `list` of length 158.

Source

https://www.phosphosite.org

plotKinaseNetwork *Plot kinase network*

Description

Plot kinase network

Usage

```
plotKinaseNetwork(KSR, predMatrix, threshold = 0.9, color,
type = NULL, verbose = FALSE)
```

Arguments

KSR	Kinase-substrate relationship scoring results
predMatrix	Output of kinaseSubstratePred function
threshold	Threshold used to select interconnected kinases for the expanded signalomes
color	A string specifying the color vector for nodes
type	A type (graph or chord) of plot. If NULL, network graph is plotted
verbose	Default to TRUE to show messages during the progress. All messages will be suppressed if set to FALSE

Value

a graphical plot

plotQC *A set of function for data QC plot*

Description

The ‘panel‘ parameter allows different type of visualisation for output object from PhosR. ‘panel = "all"‘ is used to create a 2*2 panel of plots including the following. ‘panel = "quantify"‘ is used to visualise percentage of quantification after imputataion. ‘panel = "dendrogram"‘ is used to visualise dendrogram (hierarchical clustering) of the input matrix. ‘panel = "abundance"‘ is used to visualise abundance level of samples from the input matrix. ‘panel = "pca"‘ is used to show PCA plot

Usage

```
plotQC(mat, grps, labels, panel =
c("quantify", "dendrogram", "abundance", "pca", "all"))
```

Arguments

mat	A p by n matrix, where p is the number of phosphosites and n is the number of samples.
grps	A vector of colours to be used in the plot. The length should be equal to the columns of the mat.
labels	A vector of sample names. Used the label points in PCA plot (panel=4)
panel	A type of plot to output. See description for details.

Value

A graphical plot

Examples

```
# Imputation
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(
        phospho.cells.Ins.filtered,
        0.5,
        grps)[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,seq_len(5)] <- ptImpute(
    phospho.cells.Ins.impute[,seq(6,10)],
    phospho.cells.Ins.impute[,seq(5)],
    percent1 = 0.6, percent2 = 0, paired = FALSE)

phospho.cells.Ins.ms <- medianScaling(phospho.cells.Ins.impute,
                                      scale = FALSE)

p1 = plotQC(phospho.cells.Ins.filtered,
        labels=colnames(phospho.cells.Ins.filtered),
        panel = "quantify", grps = grps)
p2 = plotQC(phospho.cells.Ins.ms,
        labels=colnames(phospho.cells.Ins.ms),
        panel = "quantify", grps = grps)
ggpubr::ggarrange(p1, p2, nrow = 1)

# Batch correction
data('phospho_L6_ratio_pe')
data('SPSs')

grps = gsub('_.+', '', rownames(
    SummarizedExperiment::colData(phospho.L6.ratio.pe))
)
```

```
# Cleaning phosphosite label
L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe),function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")
phospho.L6.ratio = t(sapply(split(data.frame(
    SummarizedExperiment::assay(phospho.L6.ratio.pe, "Quantification")),
    L6.sites),colMeans))
phospho.site.names = split(
    rownames(
        SummarizedExperiment::assay(phospho.L6.ratio.pe, "Quantification")
    ), L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
                                  ctl = ctl)

# plot after batch correction
p1 = plotQC(phospho.L6.ratio, panel = "dendrogram", grps=grps,
         labels = colnames(phospho.L6.ratio))
p2 = plotQC(phospho.L6.ratio.RUV, grps=grps,
        labels = colnames(phospho.L6.ratio),
        panel="dendrogram")
ggpubr::ggarrange(p1, p2, nrow = 1)

p1 = plotQC(phospho.L6.ratio, panel = "pca", grps=grps,
        labels = colnames(phospho.L6.ratio)) +
        ggplot2::ggtitle('Before Batch correction')
p2 = plotQC(phospho.L6.ratio.RUV, grps=grps,
        labels = colnames(phospho.L6.ratio),
        panel="pca") +
        ggplot2::ggtitle('After Batch correction')
ggpubr::ggarrange(p1, p2, nrow = 1)
```

`plotSignalomeMap` *Plot signalome map*

Description

Plot signalome map

Usage

```
plotSignalomeMap(signalomes, color)
```

Arguments

signalomes	output from ‘Signalomes‘ function
color	a string specifying the color vector for kinases

Value

a ggplot object

PPE-accessors *PhosphoExperiment object accessors*

Description

These are methods for getting for setting accessors of PhosphoExperiment object. This provides some convenience for users.

Usage

```
UniprotID(x, ...)

UniprotID(x) <- value

GeneSymbol(x, ...)

GeneSymbol(x) <- value

Site(x, ...)

Site(x) <- value

Residue(x, ...)

Residue(x) <- value

Sequence(x, ...)

Sequence(x) <- value

Localisation(x, ...)

Localisation(x) <- value

## S4 method for signature 'PhosphoExperiment'
UniprotID(x, withDimnames = TRUE)

## S4 method for signature 'PhosphoExperiment'
```

```
GeneSymbol(x, withDimnames = TRUE)

## S4 method for signature 'PhosphoExperiment'
Site(x, withDimnames = TRUE)

## S4 method for signature 'PhosphoExperiment'
Residue(x, withDimnames = TRUE)

## S4 method for signature 'PhosphoExperiment'
Sequence(x, withDimnames = TRUE)

## S4 method for signature 'PhosphoExperiment'
Localisation(x, withDimnames = TRUE)

## S4 replacement method for signature 'PhosphoExperiment'
UniprotID(x) <- value

## S4 replacement method for signature 'PhosphoExperiment'
GeneSymbol(x) <- value

## S4 replacement method for signature 'PhosphoExperiment'
Site(x) <- value

## S4 replacement method for signature 'PhosphoExperiment'
Residue(x) <- value

## S4 replacement method for signature 'PhosphoExperiment'
Sequence(x) <- value

## S4 replacement method for signature 'PhosphoExperiment'
Localisation(x) <- value
```

Arguments

x	A PhosphoExperiment object to be assigned to.
...	Ignored for accessors.
value	A vector of values to set to respective accessor. See section Available methods for more details.
withDimnames	A logical(1), indicating whether the names of the vector should be applied.

Available methods

In the following code snippets, ppe is a PhosphoExperiment object.

`UniprotID(ppe)`, `UniprotID(ppe) <- value`: Get or set a Uniprot ID, where `value` is a character vector

`GeneSymbol(ppe)`, `GeneSymbol(ppe) <- value`: Get or set a gene symbol , where `value` is a character vector

`Site(ppe)`, `Site(ppe) <- value`: Get or set a phosphorylation site, where `value` is a numeric vector

`Residue(ppe)`, `Residue(ppe) <- value`: Get or set a residue of phosphorylation site, where `value` is a character

`Sequence(ppe)`, `Sequence(ppe) <- value`: Get or set a sequence, where `value` is a character vector

`Localisation(ppe)`, `Localisation(ppe) <- localisation`: Get or set a localisation score, where `localisation` is a numeric vector

Author(s)

Taiyun Kim

Examples

```
example(PhosphoExperiment, echo = FALSE)

UniprotID(phosData) <- uniprot
head(UniprotID(phosData))

GeneSymbol(phosData) <- symbol
head(GeneSymbol(phosData))

Site(phosData) <- site
head(Site(phosData))

Residue(phosData) <- res
head(Residue(phosData))

Sequence(phosData) <- seq
head(Sequence(phosData))

Localisation(phosData) <- rnorm(nrow(phosData))
head(Localisation(phosData))
```

PPE-operate *PhosphoExperiment object subset, combine methods*

Description

These are methods for combining or subsetting for `PhosphoExperiment` object. This provides some convenience for users.

Usage

```
## S4 method for signature 'PhosphoExperiment,ANY,ANY,ANY'
x[i, j, drop = TRUE]

## S4 replacement method for signature 'PhosphoExperiment,ANY,ANY,ANY'
x[i, j, ...] <- value

## S4 method for signature 'PhosphoExperiment'
rbind(..., deparse.level = 1)

## S4 method for signature 'PhosphoExperiment'
cbind(..., deparse.level = 1)
```

Arguments

x	A PhosphoExperiment object
i	For [,PhosphoExperiment, [,PhosphoExperiment<-, i, j are subscripts that can act to subset the rows of x
j	For [,PhosphoExperiment, [,PhosphoExperiment<-, i, j are subscripts that can act to subset the columns of x
drop	A logical(1), ignored by these methods
...	In cbind or rbind, a PhosphoExperiment objects
value	An object of a class specified in the S4 method signature.
deparse.level	See ?base::cbind for a description of this argument.

Available methods

In the following code snippets, ppe1 and ppe2 is a PhosphoExperiment object with matching colData. ppe3 and ppe4 is a PhosphoExperiment object with matching rowData.

rbind(ppe1, ppe2): Combine row-wise

cbind(ppe3, ppe4): Combine column-wise

Author(s)

Taiyun Kim

See Also

method rbind, cbind from SummarizedExperiment object.

Examples

```
example(PhosphoExperiment, echo = FALSE)

n = ncol(phosData)
ppe1 = phosData[,seq(round(n/2))]
```

```
ppe2 = phosData[,-seq(round(n/2))]

ppe = cbind(ppe1, ppe2)
identical(ppe, phosData)

ppe[,seq(round(n/2))] = ppe1
identical(ppe, phosData)

p = nrow(phosData)
ppe1 = phosData[seq(round(p/2)),]
ppe2 = phosData[-seq(round(p/2)),]

ppe = rbind(ppe1, ppe2)
identical(ppe, phosData)

ppe[seq(round(p/2)),] = ppe1
identical(ppe, phosData)
```

`ptImpute` *Paired-tail (pt) based impute*

Description

Impute the missing values for mat2 using tail imputation approach if mat1 has more than percent1 (percentage) of quantified values and mat2 has less than percent2 (percentage) quantified values, and vice versa if paired is set to be true. That is if mat2 has percentage of quantified values more than percent1 and mat1 has percentage quantified values less than percent2.

Usage

```
ptImpute(
    mat1,
    mat2,
    percent1,
    percent2,
    m = 1.6,
    s = 0.6,
    paired = TRUE,
    verbose = TRUE,
    assay
)
```

Arguments

`mat1`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to replicates within treatment1.
`mat2`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to replicates within treatment2.

percent1	a percent indicating minimum quantified percentages required for considering for imputation.
percent2	a percent indicating minimum quantified percentages required for considering for imputation.
m	a numeric number of for controlling mean downshifting.
s	a numeric number of for controlling standard deviation of downshifted sampling values.
paired	a flag indicating whether to impute for both treatment1 and treatment2 (default) or treatment2 only (if paired=FALSE).
verbose	Default to TRUE to show messages during the progress. All messages will be suppressed if set to FALSE
assay	an assay to be selected if mat is a PhosphoExperiment object.

Value

An imputed matrix

Examples

```
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(
    phospho.cells.Ins.filtered,
    0.5,
    grps)[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,seq(6)] <-
    ptImpute(phospho.cells.Ins.impute[,seq(7,12)],
phospho.cells.Ins.impute[,seq(6)], percent1 = 0.6, percent2 = 0,
    paired = FALSE)


# For PhosphoExperiment objects
# mat = PhosphoExperiment(
#     assay = phospho.cells.Ins.impute,
#     colData = S4Vectors::DataFrame(
#         groups = grps
#     )
# )
# SummarizedExperiment::assay(mat)[,seq(6)] <-
#     ptImpute(SummarizedExperiment::assay(mat)[,seq(7,12)],
#         SummarizedExperiment::assay(mat)[,seq(6)], percent1 = 0.6,
#         percent2 = 0, paired = FALSE)
```

RUVphospho *RUV for phosphoproteomics data normalisation*

Description

This is a wrapper implementation of RUVIII for phosphoproteomics data normalisation. This function will call tailImpute function to impute all the missing values (if there is any) in the phosphoproteomics data for applying RUVIII. It will then return the normalised values for quantified phosphosites and remove imputed values.

Usage

```
RUVphospho(
  mat,
  M,
  ctl,
  k = NULL,
  m = 1.6,
  s = 0.6,
  keepImpute = FALSE,
  assay = NULL,
  ...
)
```

Arguments

mat	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples.
M	is the design matrix as defined in RUVIII.
ctl	is the stable phosphosites (or negative controls as defined in RUVIII).
k	is the number of unwanted factors as defined in RUVIII.
m	a numeric number for controlling mean downshifting.
s	a numeric number for controlling standard deviation of downshifted sampling values.
keepImpute	a boolean to keep the missing value in the returned matrix.
assay	an assay to be selected if mat is a PhosphoExperiment object.
...	additional parameters that may be passed to RUVIII.

Value

A normalised matrix.

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(
    SummarizedExperiment::assay(phospho.L6.ratio.pe, "Quantification"),
    M = design, k = 3, ctl = ctl)
```

`scImpute` *Site- and condition-specific (sc) impute*

Description

Impute the missing values for a phosphosite across replicates within a single condition (or treatment) if there are n or more quantified values of that phosphosite in that condition.

Usage

```
scImpute(mat, percent, grps, assay)
```

Arguments

`mat`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to replicates within a condition.
`percent`	a percent from 0 to 1, specifying the percentage of quantified values in any treatment group.
`grps`	a string specifying the grouping (replciates).
`assay`	an assay to be selected if `mat` is a PhosphoExperiment object.

Value

An imputed matrix. If param `mat` is a PhosphoExperiment object, a PhosphoExperiment object will be returned.

Examples

```
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(phospho.cells.Ins.filtered,
    0.5,
    grps)[,colnames(phospho.cells.Ins.filtered)]

# for PhosphoExperiment Object
data('phospho.cells.Ins.pe')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins.pe))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins.pe, grps,
    0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(phospho.cells.Ins.filtered,
    0.5,
    grps)[,colnames(phospho.cells.Ins.filtered)]
```

`selectGrps` *Select by treatment groups (replicate block)*

Description

Select phosphosites that have been quantified in a given percentage of treatment groups (e.g. 0.75 as 3 out of 4 replicates) in n groups.

Usage

```
selectGrps(mat, grps, percent, n, assay)
```

Arguments

`mat`	a matrix (PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
`grps`	a string specifying the grouping (replicates).
`percent`	a percent from 0 to 1, specifying the percentage of quantified values in any treatment group.
`n`	an integer indicating n or more replicates pass the percentage filtering for a phosphosite to be included.
`assay`	an assay to be selected if `mat` is a PhosphoExperiment object.

Value

a filtered matrix (or a PhosphoExperiment Oject) with at least 'percent' quantification in one or more conditions. If an input `mat` is a SummarizedExperiment object, filtered SummarizedExperiment object will be returned.

Author(s)

Pengyi Yang, Taiyun Kim

Examples

```
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

# For PhosphoExperiment object
data('phospho.cells.Ins.pe')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins.pe))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins.pe, grps, 0.5, n=1)
```

`selectLocalisedSites` *Select phosphosites by localisation score*

Description

Select phosphosites with a localisation score higher than the pre-defined probability score (default score = 0.75)

Usage

```
selectLocalisedSites(mat, loc=NULL, prob = 0.75)
```

Arguments

`mat`	a matrix (or PhosphoExperiment object) with rows corresponding to phosphosites and columns corresponding to samples in replicates for different treatments.
`loc`	a vector of localisation scores
`prob`	a percent from 0 to 1, specifying the localisation probability of quantified values in across all samples for retaining a phosphosite for subsequent analysis.

Value

a filtered matrix

Examples

```
data('phospho.cells.Ins.pe')
ppe <- phospho.cells.Ins.pe
ppe_mat <- as.data.frame(SummarizedExperiment::assay(ppe))
# Before filtering
dim(ppe)
dim(ppe_mat)

# Generate arbitrary localisation probabilities for each phosphosite
set.seed(2020)
localisation_scores <- round(rnorm(nrow(ppe), 0.8, 0.05), 2)
table(localisation_scores >= 0.75)

# Filter
Localisation(ppe) <- localisation_scores
ppe_filtered <- selectLocalisedSites(ppe, prob=0.75)
ppe_mat_filtered <- selectLocalisedSites(ppe_mat, loc=localisation_scores,
     prob=0.75)

# After filtering
dim(ppe_filtered)
dim(ppe_mat_filtered)
```

`selectOverallPercent` *Select phosphosite by percentage of quantification*

Description

Select phosphosites that have been quantified in more than a given percentage of samples

Usage

```
selectOverallPercent(mat, percent, n, assay)
```

Arguments

`mat`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
`percent`	a percent from 0 to 1, specifying the percentage of quantified values in across all samples for retaining a phosphosite for subsequent analysis.
`n`	an integer indicating n or more quantified values required for retaining a phosphosite for subsequent analysis.
`assay`	an assay to be selected if `mat` is a PhosphoExperiment object.

Value

a filtered matrix

Examples

```
data('phospho.cells.Ins.sample')

phospho.cells.Ins.filtered <- selectOverallPercent(phospho.cells.Ins, 0.5)

# Before filtering
dim(phospho.cells.Ins)
# After filtering
dim(phospho.cells.Ins.filtered)
```

`selectTimes` *selectTimes*

Description

selectTimes

Usage

```
selectTimes(mat, timepoint, order, percent, w, assay)
```

Arguments

`mat`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
`timepoint`	a timepoint as factor with a length equal to the number of columns of mat.
`order`	a vector specifying the order of timepoints.
`percent`	a percent (decimal) from 0 to 1, to filter phosphosites with with missing value larger than percent per timepoint.
`w`	a timepoint window for selection of phosphosites to remove.
`assay`	an assay to be selected if `mat` is a PhosphoExperiment object.

Value

a filtered matrix. If param `mat` is a SummarizedExperiment object, a SummarizedExperiment object will be returned.

Examples

```
data("phospho_liverInsTC_RUV_sample")
timepoint = gsub("(.*)(\\d+[ms])(.*)", "\\2",
                 colnames(phospho.liver.Ins.TC.ratio.RUV))
timepoint[which(timepoint == "0m")] = "0s"
timepoint = factor(timepoint)
timepointOrder = c("0s", "5s", "1m", "2m", "3m", "4m", "6m")
```

```
# For demonstration purpose, we introduce missing value at 0s
table(timepoint)

phospho.liver.Ins.TC.sim = phospho.liver.Ins.TC.ratio.RUV
rmId = which(timepoint == "0s")

# We replace the values to NA for the first 26 (~60%) of the '0s' samples
# for the first 100 phosphosite as NA
phospho.liver.Ins.TC.sim[seq(100),rmId[seq(26)]] = NA

phospho.liver.Ins.TC.sim = selectTimes(phospho.liver.Ins.TC.sim,
                                       timepoint, timepointOrder, 0.5,
                                       w = length(table(timepoint)))

# For PhosphoExperiment objects
# mat = PhosR::PhosphoExperiment(
#     assay = phospho.liver.Ins.TC.sim,
#     colData = S4Vectors::DataFrame(
#         timepoint = timepoint
#     )
# )
# phospho.liver.Ins.TC.sim = selectTimes(mat, mat$timepoint, timepointOrder,
#       0.5, w = length(table(mat$timepoint)))

# Before filtering
dim(phospho.liver.Ins.TC.ratio.RUV)
# After filtering
dim(phospho.liver.Ins.TC.sim)
```

`Signalomes` *PhosR Signalomes*

Description

A function to generate signalomes

Usage

```
Signalomes(KSR, predMatrix, exprsMat, KOI, threskinaseNetwork=0.9,
signalomeCutoff=0.5, module_res = NULL, filter = FALSE, verbose = TRUE)
```

Arguments

`KSR`	kinase-substrate relationship scoring results
`predMatrix`	output of kinaseSubstratePred function
`exprsMat`	a matrix with rows corresponding to phosphosites and columns corresponding to samples

KOI — a character vector that contains kinases of interest for which expanded signalomes will be generated

threskinaseNetwork — threshold used to select interconnected kinases for the expanded signalomes

signalomeCutoff — threshold used to filter kinase-substrate relationships

module_res — parameter to select number of final modules

filter — parameter to filter modules with only few proteins

verbose — Default to TRUE to show messages during the progress. All messages will be suppressed if set to FALSE

Value

A list of 3 elements. Signalomes, proteinModules and kinaseSubstrates

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV

L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")
ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(
    SummarizedExperiment::assay(phospho.L6.ratio.pe, "Quantification"),
    M = design, k = 3, ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps=grps)
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
                           (rowSums(phosphoL6.mean > 0.5) > 0),, drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
idx <- match(rownames(L6.phos.std), rownames(phospho.L6.ratio.pe))
rownames(L6.phos.std) <- L6.sites[idx]

L6.phos.seq <- Sequence(phospho.L6.ratio.pe)[idx]
```

```
L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
                                    L6.phos.seq, numMotif = 5, numSub = 1)
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)

kinaseOI = c('PRKAA1', 'AKT1')

Signalomes_results <- Signalomes(KSR=L6.matrices,
                                 predMatrix=L6.predMat,
                                 exprsMat=L6.phos.std,
                                 KOI=kinaseOI)
```

`siteAnnotate` *Phosphosite annotation*

Description

This function plots the combined scores of each of all kinases for a given phosphosites

Usage

```
siteAnnotate(site, phosScoringMatrices, predMatrix)
```

Arguments

`site`	site the ID of a phosphosite
`phosScoringMatrices`	output from function kinaseSubstrateScore()
`predMatrix`	a prediction matrix from kinaseSubstratePred()

Value

A graphical plot

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')
data('PhosphoSitePlus')

ppe <- phospho.L6.ratio.pe
sites = paste(sapply(GeneSymbol(ppe), function(x)x),";",
    sapply(Residue(ppe), function(x)x),
    sapply(Site(ppe), function(x)x),
    ";", sep = "")
grps = gsub("_.+", "", colnames(ppe))
```

```
design = model.matrix(~ grps - 1)
ctl = which(sites %in% SPSs)
ppe = RUVphospho(ppe, M = design, k = 3, ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(ppe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
idx <- (aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0)
phosphoL6.reg <- phosphoL6[idx, ,drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)

rownames(L6.phos.std) <- paste0(GeneSymbol(ppe), ";", Residue(ppe),
    Site(ppe), ";")[idx]

L6.phos.seq <- Sequence(ppe)[idx]

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
    L6.phos.seq, numMotif = 5, numSub = 1)

set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)
dev.off()

# We will look at the phosphosite AAK1;S677 for demonstration purpose.
site = "AAK1;S677;"
siteAnnotate(site, L6.matrices, L6.predMat)
```

SPSs *A list of Stably Phosphorylated Sites (SPSs)*

Description

A list of stably phosphoryalted sites defined from a panel of phosphoproteomics datasets. For full list of the datasets used, please refer to our preprint for the full list.

Usage

```
data(SPSs)
```

Format

An object of class `character` of length 100.

`standardise` *Standardisation*

Description

Standardisation by z-score transformation.

Usage

```
standardise(mat)
```

Arguments

`mat`	a matrix (or a PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples.

Value

A standardised matrix

Examples

```
data('phospho_L6_ratio_pe')
data('SPSs')

grps = gsub('_.+', '', colnames(phospho.L6.ratio.pe))

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
L6.sites = paste(sapply(GeneSymbol(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";",
                 sapply(Residue(phospho.L6.ratio.pe), function(x)paste(x)),
                 sapply(Site(phospho.L6.ratio.pe), function(x)paste(x)),
                 ";", sep = "")
ctl = which(L6.sites %in% SPSs)
phospho.L6.ratio.pe = RUVphospho(phospho.L6.ratio.pe,
                                 M = design, k = 3,ctl = ctl)

phosphoL6 = SummarizedExperiment::assay(phospho.L6.ratio.pe, "normalised")

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = grps)
aov <- matANOVA(mat=phosphoL6, grps = grps)
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
                        (rowSums(phosphoL6.mean > 0.5) > 0),,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
```

`tImpute` *Tail-based impute*

Description

Tail-based imputation approach as implemented in Perseus.

Usage

```
tImpute(mat, m, s, assay)
```

Arguments

`mat`	a matrix (or PhosphoExperiment object) with rows correspond to phosphosites and columns correspond to samples.
`m`	a numeric number for controlling mean downshifting.
`s`	a numeric number for controlling standard deviation of downshifted sampling values.
`assay`	an assay to be selected if `mat` is a PhosphoExperiment object.

Value

An imputed matrix. If param `mat` is a SummarizedExperiment object, a SummarizedExperiment object will be returned.

Examples

```
data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <- tImpute(phospho.cells.Ins.filtered)

# For PhosphoExperiment Object
data('phospho.cells.Ins.pe')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins.pe))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins.pe, grps,
    0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <- tImpute(phospho.cells.Ins.filtered)
```

Index