# Package 'moanin'

November 6, 2024

**Title** An R Package for Time Course RNASeq Data Analysis

**Version** 1.15.0

**Description** Simple and efficient workflow for time-course gene expression data, built on publictly available open-source projects hosted on CRAN and bioconductor. moanin provides helper functions for all the steps required for analysing time-course data using functional data analysis: (1) functional modeling of the timecourse data; (2) differential expression analysis; (3) clustering; (4) downstream analysis.

**Depends** R (>= 4.0), SummarizedExperiment, topGO, stats

**Imports** S4Vectors, MASS (>= 1.0.0), limma, viridis, edgeR, graphics, methods, grDevices, reshape2, NMI, zoo, ClusterR, splines, matrixStats

**Suggests** testthat (>= 1.0.0), timecoursedata, knitr, rmarkdown, markdown, covr, BiocStyle

**VignetteBuilder** knitr

**License** BSD 3-clause License + file LICENSE

**Encoding** UTF-8

**LazyData** false

**RoxygenNote** 7.1.1

**biocViews** TimeCourse, GeneExpression, RNASeq, Microarray, DifferentialExpression, Clustering

**git_url** https://git.bioconductor.org/packages/moanin

**git_branch** devel

**git_last_commit** 24c12c7

**git_last_commit_date** 2024-10-29

**Repository** Bioconductor 3.21

**Date/Publication** 2024-11-05

**Author** Elizabeth Purdom [aut] (ORCID: <https://orcid.org/0000-0001-9455-7990>), Nelle Varoquaux [aut, cre] (ORCID: <https://orcid.org/0000-0002-8748-6546>)

**Maintainer** Nelle Varoquaux <nelle.varoquaux@gmail.com>

# Contents



---

`consensus_matrix` *Compute consensus matrix from labels*

---

## Description

Compute consensus matrix from labels

## Usage

```
consensus_matrix(labels, scale = TRUE)
```

## Arguments

| | |
|---|---|
| `labels` | a matrix with each column corresponding to a the results of a single clustering routine. Each column should give the cluster assignment FIXME: What is the required format of entries?? |
| `scale` | boolean, optional, default: TRUE. Whether to rescale the resulting consensus matrix so that entries correspond to proportions. |

## Value

a symmetric matrix of size NxN, where N is the number of rows of the input matrix `labels`. Each i,j entry of the matrix corresponds the number of times the two rows were in the same cluster across the clusterings (`scale=FALSE`) or the proportion of clustering that the two rows are in the same cluster (`scale=TRUE`).

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData,meta=testMeta)
#small function to run splines_kmeans on subsample of 50 genes
subsampleCluster<-function(){
   ind<-sample(1:nrow(moanin),size=50,replace=FALSE)
   km<-splines_kmeans(moanin[ind,],n_clusters=3)
   assign<-splines_kmeans_predict(moanin,km,
      method="distance")
  }
kmClusters=replicate(10,subsampleCluster())
cm<-consensus_matrix(kmClusters)
heatmap(cm)
```

---

`create_meta_prediction` *Create prediction meta data from splines model*

---

## Description

Create prediction meta data from splines model

## Usage

```
create_meta_prediction(moanin_model, num_timepoints = 100)
```

## Arguments

| | |
|---|---|
| `num_timepoints` | integer, optional, default: 100. Number of timepoints to use for the prediction metadata |

## Value

moanin_model

DE_timecourse *Run spline models and test for DE of contrasts.*

## Description

Run spline models and test for DE of contrasts.

## Usage

```
## S4 method for signature 'Moanin'
DE_timecourse(
  object,
  contrasts,
  center = FALSE,
  statistic = c("ftest", "lrt"),
  use_voom_weights = TRUE
)
```

## Arguments

| | |
|---|---|
| object | An object of class Moanin, an object containing all related information for time course data and the splines model that will be used (if applicable). See create_moanin_model for more details. |
| contrasts | Contrasts, either provided as a vector of strings, or a matrix of contrasts coefficients obtained using makeContrasts from the package limma. If given as a character string, will be passed to makeContrasts to be converted into such a matrix. |
| center | boolean, whether to center the data matrix |
| statistic | Which test statistic to use, a likelihood ratio statistic or a F-test. |
| use_voom_weights | boolean, optional, default: TRUE. Whether to use voom weights. See details. |

## Details

The implementation of the spline fit and the calculation of p-values was based on code from edge, and expanded to enable handling of comparisons of groups via contrasts. The code assumes that the Moanin object was created via either a formula or a basis where a different spline was fit for each group_variable and thus the contrasts are comparisons of those spline fits. If the Moanin object was created via user-provided basis matrix or formula, then the user should take a great deal of caution in using this code, as the degrees of freedom for the tests of significance cannot be verified to be correct.

If use_voom_weights=TRUE, then before fitting splines to each gene, voom weights are calculated from assay(object):

```
y <- edgeR::DGEList(counts=assay(object))
y <- edgeR::calcNormFactors(y, method="upperquartile")
v <- limma::voom(y, design, plot=FALSE)
weights <- v$weights
```

The design matrix for the voom weights is based on the formula `~Group + Timepoint +0` where Group and Timepoint are replaced with the user-defined values where appropriate. These weights are given to the `lm.fit` which fits the spline coefficients. This workflow assumes that the input to the `Moanin` object were counts.

If the user set `log_transform=TRUE` in the creation of the `Moanin` object, the splines will be fit to the log of the input data, and not directly to the input data. This is independent of whether the user chooses `use_voom_weights`.

## Value

A `data.frame` with two columns for each of the contrasts given in `contrasts`, corresponding to the raw p-value of the contrast for that gene (`_pval`) and the adjusted p-value (`_qval`). The adjusted p-values are FDR-adjusted based on the Benjamini-Hochberg method, as implemented in `p.adjust`. The adjustment is done across all p-values for all contrasts calculated.

## See Also

`makeContrasts`, `create_moanin_model`, `DE_timepoints`, `edge`

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta)
deTimecourse=DE_timecourse(moanin,
   contrasts="K-C", use_voom_weights=FALSE)
head(deTimecourse)
```

---

# DE_timepoints *Fit weekly differential expression analysis*

---

## Description

Fit weekly differential expression analysis

Creates pairwise contrasts for all timepoints

## Usage

```
## S4 method for signature 'Moanin'
DE_timepoints(object, contrasts, add_factors = NULL, use_voom_weights = TRUE)

## S4 method for signature 'Moanin'
create_timepoints_contrasts(
```

```
  object,
  group1,
  group2 = NULL,
  type = c("per_timepoint_group_diff", "per_group_timepoint_diff",
    "group_and_timepoint_diff"),
  timepoints = sort(unique(time_variable(object))),
  timepoints_before = head(sort(timepoints), -1),
  timepoints_after = tail(sort(timepoints), -1),
  format = c("vector", "data.frame")
)
```

## Arguments

| | |
|---|---|
| object | An object of class Moanin, an object containing all related information for time course data and the splines model that will be used (if applicable). See create_moanin_model for more details. |
| contrasts | Contrasts, either provided as a vector of strings, or a matrix of contrasts coefficients obtained using makeContrasts from the package limma. If given as a character string, will be passed to makeContrasts to be converted into such a matrix. |
| add_factors | A character vector of additional variables to add to the design. See details. |
| use_voom_weights | boolean, optional, default: TRUE. Whether to use voom weights. See details. |
| group1 | First group to consider in making contrasts, character value that must match a value of the grouping variable contained in moanin_model. |
| group2 | Second group to consider in making contrasts, character value that must match a value of the grouping variable contained in moanin_model, unless type=="per_group_timepoint_diff", in which case should be NULL (only group1 is used in comparison) |
| type | the type of contrasts that should be created. See details. |
| timepoints | vector of timepoints to compare. Must be contained in the time_variable of the moanin object. |
| timepoints_before | for type equal to "per_group_timepoint_diff" or, "group_and_timepoint_diff", the set of timepoints to compare, see details. By default, taken from the timepoints variable. |
| timepoints_after | for type equal to "per_group_timepoint_diff" or, "group_and_timepoint_diff", the set of timepoints to compare, see details. By default, taken from the timepoints variable. |
| format | the choice of "vector" (the default) for create_timepoints_contrasts returns just the character vector of contrasts. If instead format="data.frame" then a data.frame is return that identifies the timepoint and group comparisons involved in each contrast. If this is the desired output, then the input to DE_timepoints should be the column corresponding to the contrast. See examples. |

## Details

By default the formula fitted for each gene is

```
~ Group*Timepoint +0
```

If the user gives values to add_factors, then the vector of character values given in add_factors will be *added* to the default formula. So that add_factors="Replicate" will change the formula to

```
~ Group*Timepoint +0 + Replicate
```

This allows for a small amount of additional complexity to control for other variables. Users should work directly with limma for more complex models.

If use_voom_weights=TRUE, the data is given directly to limma via assay(object). The specific series of calls is:

```
y <- edgeR::DGEList(counts=assay(object))
y <- edgeR::calcNormFactors(y, method="upperquartile")
v <- limma::voom(y, design, plot=FALSE)
v <- limma::lmFit(v)
```

If the user set log_transform=TRUE in the creation of the Moanin object, this will not have an impact in the analysis if use_voom_weights=TRUE. Only if use_voom_weights=FALSE will this matter, in which case the log of the input data will be given to a regular call to limma:

```
y<-get_log_data(object)
v <- limma::lmFit(y, design)
```

create_timepoints_contrasts creates the needed contrasts for comparing groups or timepoints in the format needed for DE_timepoints (i.e. makeContrasts), to which the contrasts are ultimately passed. The time points and groups are determined by the levels of the grouping_variable and the values of time_variable in the moanin_object provided by the user.

Three different types of contrasts are created:

- "per_timepoint_group_diff"Contrasts that compare the groups within a timepoint
- "per_group_timepoint_diff"Contrasts that compare two timepoints within a group
- "group_and_timepoint_diff"Contrasts that compare the difference between two timepoints between two levels of the group_variable of the Moanin object. These are contrasts in the form (TP i - TP (i-1))[Group1] - (TP i - TP (i-1))[Group2].

## Value

create_timepoints_contrasts: a character vector with each element of the vector corresponding to a contrast to be compared.

## See Also

makeContrasts

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta)
# compare groups within each timepoint
contrasts <- create_timepoints_contrasts(moanin,"C", "K",
   type="per_timepoint_group_diff")
head(contrasts)
deTimepoints=DE_timepoints(moanin,
    contrasts=contrasts, use_voom_weights=FALSE)
head(deTimepoints)
# Control for replicate variable:
deTimepoints=DE_timepoints(moanin,
    contrasts=contrasts, add_factors="Replicate",
    use_voom_weights=FALSE)
head(deTimepoints)

# compare adjacent timepoints within each group
contrastsDiff <- create_timepoints_contrasts(moanin,"C",
   type="per_group_timepoint_diff")
deDiffTimepoints=DE_timepoints(moanin,
    contrasts=contrastsDiff,
    use_voom_weights=FALSE)
# provide the sets of timepoints to compare:
contrastsDiff2<-create_timepoints_contrasts(moanin,"C",
   timepoints_before=c(72,120),timepoints_after=c(168,168),
   type="per_group_timepoint_diff")
deDiffTimepoints2=DE_timepoints(moanin,
    contrasts=contrastsDiff2,
    use_voom_weights=FALSE)

# Compare selected timepoints across groups.
# This time we also return format="data.frame" which helps us keep track of
# the meaning of each contrast.
contrastsGroupDiff<-create_timepoints_contrasts(moanin,"C", "K",
   timepoints_before=c(72,120),timepoints_after=c(168,168),
   type="group_and_timepoint_diff",format="data.frame")
head(contrastsGroupDiff)
deGroupDiffTimepoints=DE_timepoints(moanin,
    contrasts=contrastsGroupDiff$contrasts,
    use_voom_weights=FALSE)
```

---

`discont_basis` *Provides set of basis functions on either side of a time point, allowing for a discontinuity in the fitted functions*

---

## Description

Provides set of basis functions on either side of a time point, allowing for a discontinuity in the fitted functions

## Usage

```
discont_basis(
  timepoints,
  discont_point,
  knots = NULL,
  dfPre = NULL,
  dfPost = dfPre,
  degree = 3,
  intercept = TRUE,
  type = c("ns", "bs")
)
```

## Arguments

| | |
|---|---|
| timepoints | vector of numeric timepoints for which the splines basis will be evaluated |
| discont_point | a single numeric value that represents where the discontinuity should be |
| knots | passed to ns or bs. If not NULL, should give knots on either side of discon_point as single vector – they will be separated in the call to discon_point |
| dfPre | the df for the basis functions defined before the discontinuity point |
| dfPost | the df for the basis functions defined after the discontinuity point |
| degree | passed to bs (if applicable) |
| intercept | Whether to include an intercept (vector of all 1s) for each side of the discontinuity. Note this is different than the argument intercept of either bs or ns, which is set to FALSE. |
| type | either "ns" or "bs" indicating which splines basis function should be used. |

## Examples

```
x<-seq(0,10,length=100)
basis<-discont_basis(x,discont_point=3, dfPre=3, dfPost=4, intercept=TRUE)
# Plot of the basis functions
par(mfrow=c(3,3))
for(i in 1:ncol(basis)){
   plot(x,basis[,i],type="l")
   abline(v=3,lty=2)
}
# Use it in a moanin_model object instead of ns/bs:
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta,
    spline_formula=~Group:discont_basis(Timepoint,dfPre=3,
        dfPost=3,discont=20,intercept=TRUE)+0,
    degrees_of_freedom=6)
```

`estimate_log_fold_change` *Estimates log fold change*

## Description

Estimates log fold change

## Usage

```
## S4 method for signature 'Moanin'
estimate_log_fold_change(
  object,
  contrasts,
 method = c("timecourse", "sum", "max", "timely", "abs_sum", "abs_squared_sum", "min")
)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `Moanin`, an object containing all related information for time course data and the splines model that will be used (if applicable). See `create_moanin_model` for more details. |
| `contrasts` | The contrasts to consider |
| `method` | method for calculating the log-fold change. See details. |

## Details

The following methods exist for calculating the log-fold change between conditions over time (default is "timecourse"):

- `timely`The log-fold change for each individual timepoint ($lfc(t)$)
- `timecourse`The average absolute per-week fold-change, multiplied by the sign of the average per-week fold-change.
- `sum`Sum of per-week log fold change, over all timepoints
- `max`Max of per-week log fold change, over all timepoints
- `abs_sum`Sum of the absolute value of the per-week log fold change, over all timepoints
- `abs_squared_sum`Sum of the square value of the per-week log fold change, over all timepoint
- `min`Min of per-week log fold change, over all timepoints

If the user set `log_transform=TRUE` in the creation of the `Moanin` object, the data will be log transformed before calculating the fold-change.

## Value

A data.frame giving the estimated log-fold change for each gene (row). For all methods except for "timely", the data frame will consist of one column for each value of the argument `contrasts`. For "timely" there will be one column for each timepoint and contrast combination.

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData,meta=testMeta)
estsTimely <- estimate_log_fold_change(moanin,
   contrasts=c("K-C"), method="timely")
head(estsTimely)
estsTimecourse <- estimate_log_fold_change(moanin,
   contrasts=c("K-C"),method="timecourse")
head(estsTimecourse)
```

---

`exampleData` *Small data set for running examples*

---

## Description

Small data set for running examples

## Format

Three objects are loaded, a data frame of expression of 500 genes by 84 samples (`testData`), a data frame with meta information on those 84 samples (`testMeta`), and a data frame giving the GOID of the genes in `testData`.

## Details

This data is a subset of the full time course data available as `shoemaker2015` and is only provided for the purpose of running examples, and not for biological meaning. Users should refer to the full data set.

The rownames of `testData` are RefSeq.

## Examples

```
#code used to create data:
## Not run:
library(timecoursedata)
data(shoemaker2015)
testData<-shoemaker2015$data[1:500,]
whSamples<-which(shoemaker2015$meta$Group %in% c("C","K","M"))
testData<-testData[,whSamples]
testMeta<-droplevels(shoemaker2015$meta[whSamples,])
library(biomaRt)
ensembl = useMart("ensembl")
ensembl = useMart("ensembl")
ensembl = useDataset("mmusculus_gene_ensembl", mart=ensembl)
testGenesGO = getBM(attributes=c("go_id", "refseq_mrna"),
              values=rownames(testData),
              filters="refseq_mrna",
              mart=ensembl)
save(list=c("testData","testMeta","testGenesGO"),file="data/exampleData.rda")
```

```
## End(Not run)
```

---

`expand_contrast` *Internal Validation Checks*

---

## Description

Will check that the contrasts provided are indeed contrasts. contrasts are expected to be either a vector of string or a matrix containing the contrasts coefficients.

## Usage

```
expand_contrast(moanin_model, contrast_vector)

check_data_meta(data, object)

check_is_2d(X)

is_contrasts(contrasts, moanin_model)
```

## Details

If a vector of string is provided, the function will call limma::makeContrast in order to obtain the contrasts coefficients.

If a contrasts matrix is provided, it will perform a number of checks on the contrasts matrix to make sure it contains the number of rows expected, and that each contrast indeed sums to 0.

## Value

Does not return anything. Only hits errors if there are problems.

`is_contrasts` returns the contrasts, with any corrections.

---

`expression_filtering` *Utility function to filter out low-expressed genes*

---

## Description

Utility function to filter out low-expressed genes

## Usage

```
expression_filtering(counts, min_counts = 20, min_samples = 3)
```

### Arguments

| | |
|---|---|
| counts | n by p matrix containing the count data. |
| min_counts | integer, corresponding to the minimum number of counts for the gene to be considered expressed in a sample. Default is 20. |
| min_samples | integer, corresponding to the minimum of samples for a gene to be expressed to be included in downstream analyses. |

### Value

The filtered counts matrix

---

## find_enriched_go_terms

*Find enriched GO terms*

---

### Description

Find enriched GO terms

Create the Gene to GO Term mapping

### Usage

```
find_enriched_go_terms(
  assignments,
  gene_id_to_go,
  ontology = "BP",
  weighted = FALSE,
  node_size = 10
)

create_go_term_mapping(genes, gene_col = "refseq_mrna")
```

### Arguments

| | |
|---|---|
| assignments | boolean named vector determining the gene subset to be tested for enrichment of GO terms. The names of the vector should be the gene names. Elements with TRUE will consist of the gene cluster. |
| gene_id_to_go | List giving the Gene ID to GO object required for topGO (see topGOdata-class). create_go_term_mapping can construct such a list from a data-frame. |
| ontology | string, optional, default: BP. specficies which ontology to use (passed to ontology argument in creating a new topGOdata object). Can be 'BP', 'CC', or 'NF'. See topGOdata-class. |
| weighted | boolean, optional, default: FALSE. Whether to use the weighted algorithm or not in runTest. |

| | |
|---|---|
| node_size | integer, optional, default: 10. Consider only GO terms with node_size number of genes, passed to nodeSize argument of topGOdata-class |
| genes | dataframe, with two required columns. The first gives the gene names, with column name by the argument gene_col. The other column must be named "go_id" and give the genes GO id. Genes will have multiple GO id that they map to, and each go mapping of a gene is a separate row. Thus genes will be in multiple rows of the input. |
| gene_col | the name of the column of the genes data frame that contains the correct gene reference. By default, is "refseq_mrna". |

## Details

find_enriched_go_terms is a wrapper for running a GO enrichment analysis via the package topGO. This function creates a topGOdata-class object, runs the function runTest to test for enrichment using the statistic="fisher" option, and then runs GenTable. This function then does some post-processing of the results, returning only GO terms that satisfy:

1. BH adjusted p-values less than 0.05 using p.adjust
2. GO terms are *enriched*, i.e. the number of genes from the GO term found in the subset is greater than expected

## Value

Returns results in the format of GenTable.

create_go_term_mapping returns a list giving the gene to GO id in the format required by topGOdata-class.

## See Also

create_go_term_mapping, find_enriched_pathway, GenTable, runTest, topGOdata-class, p.adjust

## Examples

```
data(exampleData)
head(testGenesGO) #gives the mapping of genes to GO
geneId2Go <- create_go_term_mapping(testGenesGO)
#create fake assignment of genes to group based on TRUE/FALSE values
inGroup=rep(FALSE,nrow(testData))
inGroup[1:10]=TRUE
names(inGroup) <- row.names(testData)
find_enriched_go_terms(inGroup, geneId2Go)
```

## fit_predict_splines *Get fitted values for splines for each gene*

### Description

Get fitted values for splines for each gene

### Usage

```
fit_predict_splines(data, moanin_model, meta_prediction = NULL)
```

### Arguments

| | |
|---|---|
| data | a matrix with data (required doesn't pull from moanin_model) |
| meta_prediction | |

### Value

a matrix of the fitted y values, with dimensions the same as data

## fit_splines *Fit splines to each gene of data matrix, used by DE_timecourse*

### Description

Fit splines to each gene of data matrix, used by DE_timecourse

### Usage

```
fit_splines(design_matrix, data, weights = NULL)
```

### Arguments

| | |
|---|---|
| design_matrix | design matrix (containing evaluated basis matrix, but potentially other variables) to fit |
| data | a matrix of data to fix splines to |
| weights | A matrix of weights, of the same dimension as data. |

### Details

Needed to allow for fitting weights, in which case for loop over every row/gene. Otherwise, just a call to lm.fit.

### Value

matrix of the coefficients for each basis function, each row of the matrix containing the coefficients for the corresponding gene in data.

Moanin-class *Class Moanin*

## Description

Moanin is a class that extends SummarizedExperiment and is used to store the additional spline basis and meta data for timecourse analysis.

In addition to the slots of the SummarizedExperiment class, the Moanin object has the additional slots described in the Slots section.

There are several methods implemented for this class. The most important methods have their own help page. Simple helper methods are described in the Methods section below. For a comprehensive list of methods specific to this class see the Reference Manual.

The constructor create_moanin_model creates an object of the class Moanin.

## Usage

```
## S4 method for signature 'DataFrame'
create_moanin_model(data, meta, ...)

## S4 method for signature 'data.frame'
create_moanin_model(data, ...)

## S4 method for signature 'matrix'
create_moanin_model(data, meta, ...)

## S4 method for signature 'SummarizedExperiment'
create_moanin_model(
  data,
  spline_formula = NULL,
  basis_matrix = NULL,
  group_variable_name = "Group",
  time_variable_name = "Timepoint",
  degrees_of_freedom = NULL,
  log_transform = FALSE,
  drop_levels = TRUE
)
```

## Arguments

| | |
|---|---|
| data | The input data. Can be a SummarizedExperiment class, or matrix/data.frame. If the input data is a matrix or data.frame, then the user must also provide input to the meta argument, which will be transformed into colData of the resulting Moanin object |
| meta | Meta data on the samples (columns) of the data argument. Must be given f input data is a matrix or data.frame. If input is SummarizedExperiment, this argument is ignored. |

| | |
|---|---|
| ... | arguments passed from methods to the SummarizedExperiment method. |
| spline_formula | formula object, optional, default: NUlL. Used to construct splines from the data in meta. See details. |
| basis_matrix | matrix, optional, default: NULL. A basis matrix, where each row corresponds to the evaluation of a sample on the basis function (thus one column for each basis function). |
| group_variable_name | A character value giving the column that corresponds to the grouping variable to test for DE. By default "Group" |
| time_variable_name | A character value giving the column that corresponds to the time variable. By default "Timepoint". |
| degrees_of_freedom | int, optional. Number of degrees of freedom to use if neither the basis_matrix nor the spline_formula is provided. If not provided by the user, internally will be set to 4 |
| log_transform | whether the data should be log-transformed by certain methods (see splines_kmeans) |
| drop_levels | Logical, whether to perform droplevels on the grouping variable (i.e. remove empty levels) |

## Details

If neither spline_formula nor basis_matrix is given, then by default, the function will create a basis matrix based on the formula:

```
spline_formula = ~Group:ns(Timepoint, df=4) + Group +
 0
```

Note that the meta data will have levels dropped (via droplevels).

Input to data that is given as a class DataFrame or data.frame will be converted to class matrix. The reason for this is that use of a data.frame creates errors in taking duplicate rows/columns of SummarizedExperiment, as in bootstrapping. Users who absolutely want the object to hold a object that is not a matrix can construct a SummarizedExperiment object (which will not convert the input into a matrix), and use this as input to create_moanin_model.

## Value

An object of class Moanin

## Slots

time_variable_name character value giving the column in colData that defines the time variable (must be of class numeric)

group_variable_name character value giving the column in colData that defines the grouping variable (must be of class factor)

basis_matrix A basis matrix, where each row corresponds to the evaluation of a sample on the basis function (thus one column for each basis function).

`spline_formula` a formula. The formula used in creating the basis matrix

`degrees_of_freedom` a numeric integer. Number of degrees of freedom used in creating basis matrix. If NULL, degrees of freedom is not known (usually if user provided basis without degrees of freedom)

`log_transform` logical, whether to log-transform the data for certain methods

## Examples

```
# Load some data
data(exampleData)

# Use the default options
moanin = create_moanin_model(data=testData,meta=testMeta)
moanin

# Change the number of degrees of freedom
moanin = create_moanin_model(data=testData,meta=testMeta,
   degrees_of_freedom=6)
moanin
```

---

`Moanin-methods` *Helper methods for the Moanin class*

---

## Description

This is a collection of helper methods for the Moanin class.

## Usage

```
## S4 method for signature 'Moanin'
group_variable_name(object)

## S4 replacement method for signature 'Moanin'
group_variable_name(object) <- value

## S4 method for signature 'Moanin'
time_by_group_variable(object)

## S4 method for signature 'Moanin'
group_variable(object)

## S4 replacement method for signature 'Moanin'
group_variable(object) <- value

## S4 method for signature 'Moanin'
time_variable_name(object)
```

```
## S4 replacement method for signature 'Moanin'
time_variable_name(object) <- value

## S4 method for signature 'Moanin'
time_variable(object)

## S4 replacement method for signature 'Moanin'
time_variable(object) <- value

## S4 method for signature 'Moanin'
degrees_of_freedom(object)

## S4 method for signature 'Moanin'
basis_matrix(object)

## S4 method for signature 'Moanin'
spline_formula(object)

## S4 method for signature 'Moanin'
show(object)

## S4 method for signature 'Moanin,ANY,character,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Moanin,ANY,logical,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Moanin,ANY,numeric,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Moanin'
log_transform(object)

## S4 method for signature 'Moanin'
get_log_data(object)
```

## Arguments

| | |
|---|---|
| object | An object of class Moanin, an object containing all related information for time course data and the splines model that will be used (if applicable). See create_moanin_model for more details. |
| value | replacement value |
| x | Moanin object |
| i, j | A vector of logical or integer subscripts, indicating the rows and columns to be subsetted for i and j, respectively. |
| ... | arguments passed to subsetting |
| drop | A logical scalar that is ignored. |

## Details

Note that when subsetting the data, the dendrogram information and the co-clustering matrix are lost.

## Value

`group_variable_name` and `time_variable_name` return the name of the column containing the variable. `group_variable` and `time_variable` return the actual variable.

## Examples

```
# Load some data
data(exampleData)
moanin = create_moanin_model(data=testData,meta=testMeta)
group_variable_name(moanin)
time_variable_name(moanin)
```

---

# perWeek_barplot *Creates barplot of results of per-timepoint comparison*

---

## Description

Creates barplot of results of per-timepoint comparison

## Usage

```
perWeek_barplot(
  de_results,
  type = c("qval", "pval"),
  labels = NULL,
  threshold = 0.05,
  xlab = "Timepoint",
  ylab = "Number of DE genes",
  main = "",
  ...
)
```

## Arguments

| | |
|---|---|
| `de_results` | results from `DE_timepoints` |
| `type` | type of p-value to count ("qval" or "pval") |
| `labels` | labels to give each bar |
| `threshold` | cutoff for counting gene as DE |
| `xlab` | x-axis label |
| `ylab` | y-axis label |
| `main` | title of plot |
| `...` | arguments passed to `barplot` |

## Details

`create_timepoints_contrasts` creates the needed contrasts for comparing two groups for every timepoint in the format needed for `DE_timepoints` (i.e. `makeContrasts`, to which the contrasts are ultimately passed). The time points are determined by the meta data in the `moanin_object` provided by the user.

## Value

This is a plotting function, and returns (invisibly) the results of `barplot`

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta)
contrasts <- create_timepoints_contrasts(moanin, "C", "K")
deTimepoints <- DE_timepoints(moanin,
    contrasts=contrasts, use_voom_weights=FALSE)
perWeek_barplot(deTimepoints)
```

---

# plot_cdf_consensus *Evaluate the consensus between sets of clusterings*

---

## Description

Methods for evaluating the consensus between sets of clusterings, usually in the context of subsetting of the data or different numbers of clusters.

## Usage

```
plot_cdf_consensus(labels)

get_auc_similarity_scores(labels, method = c("consensus", "nmi"))

plot_model_explorer(labels, colors = rainbow(length(labels)))
```

## Arguments

| | |
|---|---|
| `labels` | a list. Each element of the list is a matrix that gives the results of a clustering routine in each column (see `consensus_matrix`). Usually each column would be the result of running the clustering on a subsample or bootstrap resample of the data. |
| `method` | method for calculation of similarity for the AUC measure, one of "consensus" or "nmi". See details. |
| `colors` | a vector of colors, of length equal to the length of `labels` |

## Details

For each element of the list `labels`, `plot_cdf_consensus` calculates the consensus between the clusterings in the matrix, i.e. the number of times that pairs of rows are in the same cluster for different clusterings (columns) of the matrix using the `consensus_matrix` function. Then the set of values (the N(N-1) values in the upper triangle of the matrix), are converted into a cdf function and plotted.

For each set of clusterings given by `labels` (i.e. for each matrix `M` which is an element of the list `labels`) `get_auc_similarity_scores` calculates a pairwise measure of similarity between the columns of `M`. These pairwise scores are plotted against their rank, and the final AUC measure is the area under this curve.

For method "consensus", the pairwise measure is given by calculating the consensus matrix using `consensus_matrix` with `scale=FALSE`. The consensus matrix is divided by the max of `M`.

For method "nmi", the pairwise value is the NMI value between each pair of columns of the matrix of clusterings using the `NMI` function.

## Value

`plot_cdf_consensus` invisibily returns list of the upper triangle values, with the list of same length as that of `labels`.

`get_auc_similarity_scores` returns a vector, equal to length of the list `labels`, giving the AUC value for each element of `labels`.

This function is a plotting function does not return anything

## See Also

`consensus_matrix`, `NMI`, `plot_cdf_consensus`

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData,meta=testMeta)
#small function to run splines_kmeans on subsample of 50 genes
subsampleCluster<-function(){
   ind<-sample(1:nrow(moanin),size=50)
   km<-splines_kmeans(moanin[ind,],n_clusters=3)
   assign<-splines_kmeans_score_and_label(moanin, km,
       proportion_genes_to_label=1.0)$label
}
kmClusters1=replicate(10,subsampleCluster())
kmClusters2=replicate(10,subsampleCluster())
# Note, because of the small number of replicates (10),
# these plots are not representative of what to expect.
out<-plot_cdf_consensus(labels=list(kmClusters1,kmClusters2))
get_auc_similarity_scores(list(kmClusters1,kmClusters2))
plot_model_explorer(list(kmClusters1,kmClusters2))
```

plot_splines_data *Plotting splines*

## Description

Plotting splines

## Usage

```
## S4 method for signature 'Moanin,matrix'
plot_splines_data(
  object,
  data,
  colors = NULL,
  smooth = FALSE,
  legend = TRUE,
  legendArgs = NULL,
  subset_conditions = NULL,
  subset_data = NULL,
  simpleY = TRUE,
  centroid = NULL,
  scale_centroid = c("toData", "toCentroid", "none"),
  mar = c(2.5, 2.5, 3, 1),
  mfrow = NULL,
  addToPlot = NULL,
  ylab = "",
  xaxis = TRUE,
  yaxis = TRUE,
  xlab = "Time",
  ...
)

## S4 method for signature 'Moanin,numeric'
plot_splines_data(object, data, ...)

## S4 method for signature 'Moanin,data.frame'
plot_splines_data(object, data, ...)

## S4 method for signature 'Moanin,DataFrame'
plot_splines_data(object, data, ...)

## S4 method for signature 'Moanin,missing'
plot_splines_data(object, data, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class Moanin, an object containing all related information for time course data and the splines model that will be used (if applicable). See create_moanin_model |

| | |
|---|---|
| | for more details. |
| data | matrix containing the data to be plotted, where each row of the data provided will be plotted as a separate plot. If missing, will rely on data in assay(object) |
| colors | vector, optional, default NULL. Vector of colors |
| smooth | boolean, optional, default: FALSE. Whether to smooth the centroids or not. |
| legend | boolean whether to include a legend (default:TRUE) |
| legendArgs | list of arguments to be passed to legend command (if legend=TRUE) |
| subset_conditions | list if provided, only plots the subset of conditions provided. Else, plots all conditions |
| subset_data | list if provided, only plots the subset of data (ie, the rows) provided. Can be any valid vector for subsetting a matrix. See details. |
| simpleY | boolean, if true, will plot all genes on same y-axis and minimize the annotation of the y axis to only label the axis in the exterior plots (the x-axis is always assumed to be the same across all plots and will always be simplified) |
| centroid | numeric vector (or matrix of 1 row) with data to use to fit the splines. If NULL, the splines plotted will be from the data. |
| scale_centroid | determines whether the centroid data given in centroid should be rescaled to match that of the data ("toData"), or the data scaled to match that of centroid ("toCentroid"), or simply plotted as is ("none"). |
| mar | vector of margins to set the space around each plot (see par) |
| mfrow | a vector of integers of length 2 defining the grid of plots to be created (see par). If missing, the function will set a value. |
| addToPlot | A function that will be called after the plotting, allowing the user to add more to the plot. |
| ylab | label for the y-axis |
| xaxis | Logical, whether to add x-axis labels to plot (if FALSE can be manually created by user with call to addToPlot) |
| yaxis | Logical, whether to add y-axis labels to plot (if FALSE can be manually created by user with call to addToPlot) |
| xlab | label for the x-axis |
| ... | arguments to be passed to the individual plot commands (Will be sent to all plot commands) |

## Details

If data is NULL, the data plotted will be from assay(object), after log-transformation if log_transform(object)=TRUE.

If centroid is missing, then splines will be estimated (per group) for the the data in data – separately for each row of data. If centroid is provided, this data will be used to plot a spline function, and this same spline will be plotted for each row of data. This is useful, for example, in plotting cluster centroids over a series of genes.

If the user set log_transform=TRUE in the creation of the Moanin object, the data will be log transformed before plotting and calculating the spline fits.

## Value

This function creates a plot and does not return anything to the user.

## Examples

```
# First, load some data and create a moanin model
data(exampleData)
moanin <- create_moanin_model(data=testData,meta=testMeta,
   degrees_of_freedom=6)

# The moanin model contains all the information for plotting purposes. The
# plot_splines_data will automatically fit the splines from the
# information contained in the moanin model
genes <- c("NM_001042489", "NM_008725")
plot_splines_data(moanin, subset_data=genes,
mfrow=c(2, 2))
# By default, same axis for all genes. Can change with 'simpleY=FALSE'
plot_splines_data(moanin, subset_data=genes,
   smooth=TRUE, mfrow=c(2,2), simpleY=FALSE)

# The splines can also be smoothed
plot_splines_data(moanin, subset_data=genes,
   smooth=TRUE, mfrow=c(2, 2))
# You can provide different data (on same subjects),
# instead of data in moanin object
# (in which case moanin just provides grouping information)
plot_splines_data(moanin, data=1/assay(moanin), subset_data=genes,
   smooth=TRUE, mfrow=c(2, 2))

# You can also provide data to use for fitting splines to argument
# "centroid". This is helpful for overlaying centroids or predicted data
# Here we do a silly example, just to demonstrate syntax,
# where we use the data from the first gene as our centroid to fit a
# spline estimate, but plot data from genes 3-4
plot_splines_data(moanin, centroid=assay(moanin[1,]), subset_data=3:4,
   smooth=TRUE, mfrow=c(2,2))
```

---

# pvalues_fisher_method *Fisher's method to combine pvalues*

---

## Description

Combines all p-values per rows.

## Usage

```
pvalues_fisher_method(pvalues)
```

### Arguments

| | |
|---|---|
| pvalues | a matrix of pvalues, with columns corresponding to different tests or sources of p-values, and rows corresponding to the genes from which the p-values come. |

### Value

a vector of p-values, one for each row of pvalues, that is the result of Fisher's combined probability test applied to the p-values in that row.

### Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData,meta=testMeta)
contrasts <- create_timepoints_contrasts(moanin,"C", "K")
deTimepoints=DE_timepoints(moanin,
  contrasts=contrasts, use_voom_weights=FALSE)
fisherPval=pvalues_fisher_method(
  deTimepoints[,grep("pval",colnames(deTimepoints))])
head(fisherPval)
```

---

## rescale_values *Rescales rows of data to be between 0 and 1*

---

### Description

Rescales rows of data to be between 0 and 1

### Usage

```
## S4 method for signature 'Moanin'
rescale_values(object, data = NULL, use_group = FALSE)

## S4 method for signature '`NULL`'
rescale_values(object, data)

## S4 method for signature 'missing'
rescale_values(object, ...)
```

### Arguments

| | |
|---|---|
| object | a object of class Moanin, only needed if choose to rescale by grouping variable in the moanin object. If NULL, then data will be rescaled jointly across all observations. |
| data | The matrix to rescale by row. If NULL, and object is given, data will be taken as assay(object) Each row should correspond to a gene or a centroid, and columns to samples. |

use_group — If true, then the data will be rescaled such that, for each row, all values associated to each group (defined by grouping variable of `object`) is between 0 and 1. For example, if column

... — arguments passed to the matrix or Moanin method.

## Details

If the user set `log_transform=TRUE` in the creation of the `Moanin` object, the data will be log transformed before rescaling

## Value

rescaled y, such that for each row, the values are comprised between 0 and 1. Note that if `use_group=TRUE` and `object` is not NULL, the values associated to the columns of unique values of the grouping variable of `object` will be rescaled separately.

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta)
# Can rescale data in Moanin object
allData <- rescale_values(moanin)
# Or provide different data and/or rescale within grouping variable
smallData <- rescale_values(moanin, data=testData[1:10,], use_group=TRUE)
```

---

# splines_kmeans *Performs splines clustering using K-means*

---

## Description

Performs splines clustering using K-means

## Usage

```
## S4 method for signature 'Moanin'
splines_kmeans(
  object,
  n_clusters = 10,
  init = "kmeans++",
  n_init = 10,
  max_iter = 300,
  random_seed = .Random.seed[1],
  fit_splines = TRUE,
  rescale = TRUE
)
```

### Arguments

| | |
|---|---|
| object | An object of class Moanin, an object containing all related information for time course data and the splines model that will be used (if applicable). See create_moanin_model for more details. |
| n_clusters | int optional, default: 10 |
| init | ["kmeans++", "random", "optimal_init"] |
| n_init | int, optional, default: 10 Number of initialization to perform. |
| max_iter | int, optional, default: 300 Maximum number of iteration to perform |
| random_seed | int, optional, default: NULL. Passed to argument seed in KMeans_rcpp. If NULL (default), set to .Random.seed[1]. |
| fit_splines | boolean, optional, default: TRUE Whether to fit splines or not. |
| rescale | boolean, optional, default: TRUE Whether to rescale the data or not. |

### Details

If Moanin object's slot has log_transform=TRUE, then the data will be transformed by the function log(x+1) before applying splines and clustering.

### Value

A list in the format returned by KMeans_rcpp, with the following elements added or changed:

- centroids The centroids are rescaled so that they range from 0-1
- fit_splines Logical, the value of fit_splines given to the function
- rescale The value of rescale given to the function

### Examples

```
data(exampleData)
# Use the default options
moanin <- create_moanin_model(data=testData, meta=testMeta)
out <- splines_kmeans( moanin,n_clusters=5)
table(out$clusters)
```

---

## splines_kmeans_predict,Moanin-method

*Assign score and labels from raw data*

---

### Description

Assign score and labels from raw data

## Usage

```
## S4 method for signature 'Moanin'
splines_kmeans_predict(
  object,
  kmeans_clusters,
  data = NULL,
  method = c("distance", "goodnessOfFit"),
  ...
)

## S4 method for signature 'Moanin'
splines_kmeans_score_and_label(
  object,
  kmeans_clusters,
  data = NULL,
  proportion_genes_to_label = 0.5,
  max_score = NULL,
  previous_scores = NULL,
  rescale_separately = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | the Moanin object that contains the basis functions used in creating the clusters |
| `kmeans_clusters` | List returned by `splines_kmeans` |
| `data` | the data to predict. If not given, will use `assay(object)`. If given, the number of columns of `data` must match that of `object` |
| `method` | If "distance", predicts based on distance of data to kmeans centroids. If "goodnessOfFit", is a wrapper to `splines_kmeans_score_and_label`, assigning labels based on goodness of fit, including any filtering. |
| `...` | arguments passed to `splines_kmeans_score_and_label` |
| `proportion_genes_to_label` | float, optional, default: 0.5 Percentage of genes to label. If max_score is provided, will label genes that are either in the top ‘proportion_genes_to_label‘ or with a score below ‘max_score‘. |
| `max_score` | optional, default: Null When provided, will only label genes below that score. If NULL, ignore this option. |
| `previous_scores` | matrix of scores, optional. Allows user to give the matrix scores results from a previous run of `splines_kmeans_score_and_label`, and only redo the filtering (i.e. if want to change `proportion_genes_to_label` without rerunning the calculation of scores) |
| `rescale_separately` | logical, whether to score separately within grouping variable |

## Value

`splines_kmeans_predict` returns a vector giving the labels for the given data.

A list consisting of

- `labels`the label or cluster assigned to each gene based on the cluster with the best (i.e. lowest) score, with no label given to genes that do not have a score lower than a specified quantity
- `scores`the matrix of size n_cluster x n_genes, containing for each gene and each cluster, the goodness of fit score
- `score_cutoff`The required cutoff for a gene receiving an assignment

## Examples

```
data(exampleData)
moanin <- create_moanin_model(data=testData, meta=testMeta)
# Cluster on a subset of genes
kmClusters=splines_kmeans(moanin[1:50,],n_clusters=3)
# get scores on all genes
scores_and_labels <- splines_kmeans_score_and_label(object=moanin, kmClusters)
head(scores_and_labels$scores)
head(scores_and_labels$labels)
# should be same as above, only just the assignments
predictLabels1 <- splines_kmeans_predict(object=moanin, kmClusters,
     method="goodnessOfFit")
# Instead use distance to centroid:
predictLabels2 <- splines_kmeans_predict(object=moanin, kmClusters,
     method="distance")
```

# Index