Package 'santaR'

January 24, 2018

Title Short Asynchronous Time-Series Analysis

Version 1.0

Date 2018-01-22

URL https://github.com/adwolfer

Description A graphical and automated pipeline for the analysis
of short time-series in R ('santaR'). This approach is designed to accommodate asynchronous time sampling (i.e. different time points for different individuals),
inter-individual variability, noisy measurements and large numbers of variables.
Based on a smoothing splines functional model, 'santaR' is able to detect variables highlighting significantly different temporal trajectories between study groups.
Designed initially for metabolic phenotyping, 'santaR' is also suited for other Systems Biology disciplines. Command line and graphical analysis (via a 'shiny' application) enable fast and parallel automated analysis and reporting, intuitive visualisation and comprehensive plotting options for non-specialist users.

Depends R (>= 3.4.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports plyr (>= 1.8.4), foreach (>= 1.4.4), doParallel (>= 1.0.11),
pcaMethods (>= 1.70.0), ggplot2 (>= 2.2.1), gridExtra (>= 2.3),
reshape2 (>= 1.4.3), iterators (>= 1.0.9), shiny (>= 1.0.5),
shinythemes (>= 1.1.1)

biocViews pcaMethods

Suggests knitr, rmarkdown, pander, R.rsp

VignetteBuilder knitr, R.rsp

RoxygenNote 6.0.1

NeedsCompilation no

Author Arnaud Wolfer [aut, cre],
Timothy Ebbels [ctb],
Joe Cheng [ctb] (Shiny javascript custom-input control)

Maintainer Arnaud Wolfer <adwolfer@gmail.com>

Repository CRAN

Date/Publication 2018-01-24 18:27:08 UTC

R topics documented:

acuteInflammation . . . 2
AICc_smooth_spline . . . 3
AIC_smooth_spline . . . 3
BIC_smooth_spline . . . 4
get_eigen_DF . . . 4
get_eigen_DFoverlay_list . . . 5
get_eigen_spline . . . 7
get_eigen_spline_matrix . . . 9
get_grouping . . . 10
get_ind_time_matrix . . . 11
get_param_evolution . . . 12
loglik_smooth_spline . . . 14
plot_nbTP_histogram . . . 14
plot_param_evolution . . . 15
santaR . . . 16
santaR_auto_fit . . . 17
santaR_auto_summary . . . 19
santaR_CBand . . . 21
santaR_fit . . . 22
santaR_plot . . . 24
santaR_pvalue_dist . . . 26
santaR_pvalue_dist_within . . . 27
santaR_pvalue_fit . . . 28
santaR_pvalue_fit_within . . . 29
santaR_start_GUI . . . 31
scaling_mean . . . 32
scaling_UV . . . 33

Index **34**

`acuteInflammation` *Measurement of 22 inflammatory mediators across time*

Description

A dataset containing the concentrations of 22 mediators of inflammation over an episode of acute inflammation. The mediators have been measured at 7 time-points on 8 subjects, concentration values have been unit-variance scaled for each variable.

Usage

```
acuteInflammation
```

Format

List of 2 data frames of 56 rows each, containing the 22 measured variables (data) and the corresponding sampling metadata (meta):

data: var mediator concentration, unit-variance scaled

meta: time time of the measurement, in hour

meta: ind subject ID for the measurement

meta: group group membership of the subject/measurement

AICc_smooth_spline *Calculate the Akaike Information Criterion Corrected for small observation numbers for a smooth.spline*

Description

Calculate the Akaike Information Criterion Corrected for small observation numbers (*AICc*) for a fitted `smooth.spline`. The smaller the AICc, the better the spline fit.

Usage

```
AICc_smooth_spline(fittedSmoothSpline)
```

Arguments

`fittedSmoothSpline`
: A fitted `smooth.spline`

Value

The AICc value.

AIC_smooth_spline *Calculate the Akaike Information Criterion for a smooth.spline*

Description

Calculate the Akaike Information Criterion (*AIC*) for a fitted `smooth.spline`. The smaller the AIC, the better the spline fit.

Usage

```
AIC_smooth_spline(fittedSmoothSpline)
```

Arguments

`fittedSmoothSpline`
: A fitted `smooth.spline`

Value

The AIC value.

`BIC_smooth_spline` *Calculate the Bayesian Information Criterion for a smooth.spline*

Description

Calculate the Bayesian Information Criterion (*BIC*) for a fitted `smooth.spline`. The smaller the BIC, the better the spline fit.

Usage

```
BIC_smooth_spline(fittedSmoothSpline)
```

Arguments

`fittedSmoothSpline`
: A fitted `smooth.spline`

Value

The BIC value.

`get_eigen_DF` *Compute the optimal df and weighted-df using 5 spline fitting metric*

Description

Compute the optimal degree of freedom (*df*) and weighted degree of freedom (*wdf*) using 5 fitting metrics (**CV***: Cross-Validation,* **GCV***: Generalised Cross-Validation,* **AIC***: Akaike Information Criterion,* **BIC***: Bayesian Information Criterion,* **AICc***: Akaike Information Criterion Corrected for small sample size*) over all eigenSplines generated by `get_eigen_spline`. The degree of freedom (*df*) is obtained by averaging the optimal *df* across each eigenSpline. The weighted degree of freedom (*wdf*) is obtained by weighting the optimal *df* in each eigenSpline by the percentage of variance explained by each eigenSpline, before summing the optimal *df*s (variance sums to 100%).

Usage

```
get_eigen_DF(eigen)
```

Arguments

eigen	A list of eigenSpline parameters as generated by get_eigen_spline, containing eigen$matrix, eigen$variance, eigen$model and eigen$countTP.

Value

A list: answer$df a vector of optimum *df* by CV, GCV, AIC, BIC, AICc. answer$wdf a vector of weighted optimum *df* by CV, GCV, AIC, BIC, AICc.

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DFoverlay_list, get_eigen_spline, get_param_evolution, plot_nbTP_histogram, plot_param_evolution

Examples

```
## 8 subjects, 8 time-points, 3 variables
inputData <- acuteInflammation$data[,1:3]
ind       <- acuteInflammation$meta$ind
time      <- acuteInflammation$meta$time
eigen     <- get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV",
                              method="nipals", verbose=TRUE, centering=TRUE, ncores=0)
# nipals calculated PCA
# Importance of component(s):
#                    PC1    PC2     PC3      PC4    PC5      PC6
# R2            0.8924 0.0848 0.01055 0.006084 0.0038 0.002362
# Cumulative R2 0.8924 0.9772 0.98775 0.993838 0.9976 1.000000
get_eigen_DF(eigen)
# $df
#       CV      GCV      AIC      BIC     AICc
# 3.362581 4.255487 3.031260 2.919159 2.172547
# $wdf
#       CV      GCV      AIC      BIC     AICc
# 2.293130 2.085212 6.675608 6.671545 4.467724
```

---

get_eigen_DFoverlay_list

Plot for each eigenSpline the automatically fitted spline, splines for all df and a spline at a chosen df

---

Description

Plot for each eigenSpline the automatically fitted spline (red), splines for all possible *df* (grey) and a spline at a manually chosen *df* (blue).

Usage

```
get_eigen_DFoverlay_list(eigen, manualDf = 5, nPC = NA, step = NA,
  showPt = TRUE, autofit = TRUE)
```

Arguments

eigen	A list of eigenSpline parameters as generated by get_eigen_spline, containing eigen$matrix, eigen$variance, eigen$model and eigen$countTP.
manualDf	(int) A manually selected *df*. Default is 5.
nPC	(int) The first *n* eigenSplines to plot. Default is NA, plot all eigenSplines.
step	(float) The *df* increment employed to plot splines over the range of *df*.
showPt	(bool) If True the eigenSpline data points are plotted. Default is TRUE.
autofit	(bool) If True the automatically fitted splines *(using CV)* are plotted. Default is TRUE.

Value

A list of ggplot2 plotObjects, one plot per eigenSpline. All results can be plotted using do.call(grid.arrange, returnedl

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DF, get_eigen_spline, get_param_evolution, plot_nbTP_histogram, plot_param_evolution

Examples

```
## 8 subjects, 4 time-points, 3 variables
inputData  <- acuteInflammation$data[0:32,1:3]
ind        <- acuteInflammation$meta$ind[0:32]
time       <- acuteInflammation$meta$time[0:32]
eigen      <- get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV",
                               method="nipals", verbose=TRUE, centering=TRUE, ncores=0)
paramSpace <- get_param_evolution(eigen, step=1)
plotList   <- get_eigen_DFoverlay_list(eigen,manualDf=3,step=0.5,showPt=TRUE,autofit=TRUE)
plotList[1]
# do.call(grid.arrange, plotList)
```

get_eigen_spline *Compute eigenSplines across a dataset*

Description

Compute "eigenSplines" across a dataset to discover the best *df* for spline fitting.

Steps::

- UV Scale the data.
- Turn each VAR in (IND x TIME) and group all VAR in (IND+VAR x TIME) using `get_eigen_spline_matrix`.
- Compute "eigen.splines" on the transposed table (TIME x IND+VAR).
- Returns eigen$matrix = PCprojection x TIME and eigen$variance = variance explained for each PC.

Usage

```
get_eigen_spline(inputData, ind, time, nPC = NA, scaling = "scaling_UV",
  method = "nipals", verbose = TRUE, centering = TRUE, ncores = 0)
```

Arguments

`inputData`	Matrix of measurements with observations as rows and variables as columns.
`ind`	Vector of subject identifier (individual) corresponding to each measurement.
`time`	Vector of time corresponding to each measurement.
`nPC`	(int) Number of Principal Components to compute, if none given (`nPC=NA`) compute all PC (usually number TP-1 as there is 1PC less than the smallest dimension).
`scaling`	`"scaling_UV"` or `"scaling_mean"` scaling across all samples for each variable. Default `"scaling_UV"`. Note: scaling takes place outside of the pcaMethods call, therefore `$model` will indicate "Data was NOT scaled before running PCA".
`method`	PCA method `"svd"` doesn't accept missing value. `"nipals"` can handle missing values. Default `"nipals"`.
`verbose`	If `TRUE` print the PCA summary. Default `TRUE`.
`centering`	If `TRUE` centering for PCA, needed to remove baseline levels of each pc (often PC1). Default `TRUE`.
`ncores`	(int) Number of cores to use for parallelisation of the grouping of all splines. Default 0 for no parallelisation.

Value

A list eigen: `eigen$matrix` `data.frame` of eigenSplines values with PCprojection as row and TIME as column. `eigen$variance` Vector of variance explained for each PC. `eigen$model` resulting pcaMethods model. `eigen$countTP` Matrix of number of measurements for each unique timepoint (as row).

Comments::

- CENTERING: Centering converts all the values to fluctuations around zero instead of around the mean of the variable measurements. Hereby, it adjusts for differences in the offset between high and low intensity variables. It is therefore used to focus on the fluctuating part of the data, and leaves only the relevant variation (being the variation between the observations) for analysis.
- SCALING: Scaling methods are data pretreatment approaches that divide each variable by a factor -the scaling factor- which is different for each variable. They aim to adjust for the differences in fold differences between the various variables by converting the data into differences in values relative to the scaling factor. This often results in the inflation of small values, which can have an undesirable side effect as the influence of the measurement error -that is usually relatively large for small values- is increased as well.
- UNIT VARIANCE SCALING: UV or Autoscaling, is commonly applied and uses the standard deviation as the scaling factor. After autoscaling, all variables have a standard deviation of one and therefore the data is analysed on the basis of correlations instead of covariances, as is the case with centering.
- BEFORE PCA, centering must be applied on the matrix that will be submitted to PCA to remove "baseline" levels.

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DFoverlay_list, get_eigen_DF, get_param_evolution, plot_nbTP_histogram, plot_param_evolution

Examples

```
## 7 measurements, 3 subjects, 4 unique time-points, 2 variables
inputData <- matrix(c(1,2,3,4,5,6,7,8,9 ,10,11,12,13,14,15,16,17,18), ncol=2)
ind  <- c('ind_1','ind_1','ind_1','ind_2','ind_2','ind_2','ind_3','ind_3','ind_3')
time <- c(0,5,10,0,10,15,5,10,15)
get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV", method="nipals",
                 verbose=TRUE, centering=TRUE, ncores=0)
# nipals calculated PCA
# Importance of component(s):
#                   PC1    PC2     PC3
# R2            0.7113 0.2190 0.05261
# Cumulative R2 0.7113 0.9303 0.98287
# total time: 0.12 secs
# $matrix
#              0          5        10         15
# PC1 -1.7075707 -0.7066426 0.7075708  1.7066425
# PC2 -0.3415271  0.9669724 1.0944005 -0.4297013
# PC3 -0.1764657 -0.5129981 0.5110671  0.1987611
#
# $variance
# [1] 0.71126702 0.21899068 0.05260949
#
# $model
# nipals calculated PCA
```

```
# Importance of component(s):
#                   PC1    PC2     PC3
# R2            0.7113 0.2190 0.05261
# Cumulative R2 0.7113 0.9303 0.98287
# 6  Variables
# 4  Samples
# 6  NAs ( 25 %)
# 3  Calculated component(s)
# Data was mean centered before running PCA
# Data was NOT scaled before running PCA
# Scores structure:
# [1] 4 3
# Loadings structure:
# [1] 6 3
#
# $countTP
#   [,1]
# 3    6
```

`get_eigen_spline_matrix` *Generate a Ind x Time + Var data.frame concatenating all variables from input variable*

Description

Generate Ind x Time `data.frame` for each variable using `get_ind_time_matrix` and then concatenate all variables rowise. Resulting `data.frame` contrain Time as columns and Individuals and Variables as rows. Pairs of Individual and Timepoint without a measurement are left as NA. If ncore!=0 the function is parallelised, however the parallelisation overhead cost is high if not required.

Usage

```
get_eigen_spline_matrix(inputData, ind, time, ncores = 0)
```

Arguments

`inputData`	`data.frame` of measurements with observations as rows and variables as columns
`ind`	Vector of subject identifier (individual) corresponding to each measurement
`time`	Vector of time corresponding to each measurement
`ncores`	(int) Number of cores to use for parallelisation. Default 0 for no parallelisation.

Value

`data.frame` of measurements for each IND x TIME + VAR. Rows are unique Individual IDs per variable, and columns unique measurement Time. Pairs of (IND,TIME+VAR) without a measurement are left as NA.

Examples

```
## Not run:
## 6 measurements, 3 subjects, 3 unique time-points, 2 variables
inputData <- matrix(c(1,2,3,4,5,6, 7,8,9,10,11,12), ncol=2)
ind  <- c('ind_1','ind_1','ind_1','ind_2','ind_2','ind_3')
time <- c(0,5,10,0,10,5)
get_eigen_spline_matrix(inputData, ind, time, ncores=0)
#     0   5  10
# 1   1   2   3
# 2   4  NA   5
# 3  NA   6  NA
# 4   7   8   9
# 5  10  NA  11
# 6  NA  12  NA

## End(Not run)
```

`get_grouping` *Generate a matrix of group membership for all individuals*

Description

Establish the group membership of individuals based on the metadata across all observations using the vector of subject identifier and the matching vector of group membership.

Usage

```
get_grouping(ind, group)
```

Arguments

`ind`	vector of subject identifier (individual) for each observation
`group`	vector of group membership for each observation

Value

`data.frame` with as rows each unique Individual ID and 2 columns (ind and group).

See Also

Other Analysis: `get_ind_time_matrix`, `santaR_CBand`, `santaR_auto_fit`, `santaR_auto_summary`, `santaR_fit`, `santaR_plot`, `santaR_pvalue_dist`, `santaR_pvalue_fit`, `santaR_start_GUI`

Examples

```
## 3 subjets in 2 groups
ind   <- c('ind_1','ind_1','ind_1','ind_2','ind_2','ind_3')
group <- c('g1','g1','g1','g2','g2','g1')
get_grouping(ind, group)
#     ind group
# 1 ind_1    g1
# 2 ind_2    g2
# 3 ind_3    g1

## 8 subjects in 2 groups
ind   <- acuteInflammation$meta$ind
group <- acuteInflammation$meta$group
get_grouping(ind, group)
#    ind   group
# 1 ind_1 Group1
# 2 ind_2 Group2
# 3 ind_3 Group1
# 4 ind_4 Group2
# 5 ind_5 Group1
# 6 ind_6 Group2
# 7 ind_7 Group1
# 8 ind_8 Group2
```

`get_ind_time_matrix` *Generate a Ind x Time DataFrame from input data*

Description

Convert input data with each measurement as a row, to a `data.frame` of measurements with Individual as rows and Time as columns. Pairs of Individual and Timepoint without a measurement are left as NA. The resulting `data.frame` is employed as input for `santaR_fit`.

Usage

```
get_ind_time_matrix(Yi, ind, time, orderVect)
```

Arguments

`Yi`	vector of measurements
`ind`	vector of subject identifier (individual) corresponding to each measurement
`time`	vector of time corresponding to each measurement
`orderVect`	if provided, a vector of unique time to be used to order the time columns (otherwise rely on `sort`)

Value

data.frame of measurements for each IND x TIME. Rows are unique Individual IDs and columns unique measurement Time. Pairs of (IND,TIME) without a measurement are left as NA.

See Also

Other Analysis: get_grouping, santaR_CBand, santaR_auto_fit, santaR_auto_summary, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_pvalue_fit, santaR_start_GUI

Examples

```
## 6 measurements, 3 subjects, 3 unique time-points
Yi   <- c(1,2,3,4,5,6)
ind  <- c('ind_1','ind_1','ind_1','ind_2','ind_2','ind_3')
time <- c(0,5,10,0,10,5)
get_ind_time_matrix(Yi, ind, time)
#        0  5 10
# ind_1  1  2  3
# ind_2  4 NA  5
# ind_3 NA  6 NA

## 56 measurements, 8 subjects, 7 unique time-points
Yi   <- acuteInflammation$data$var_1
ind  <- acuteInflammation$meta$ind
time <- acuteInflammation$meta$time
get_ind_time_matrix(Yi, ind, time)
```

---

get_param_evolution *Compute the value of different fitting metrics over all possible df for each eigenSpline*

Description

Compute the value of 5 fitting metrics (**CV***: Cross-Validation,* **GCV***: Generalised Cross-Validation,* **AIC***: Akaike Information Criterion,* **BIC***: Bayesian Information Criterion,* **AICc***: Akaike Information Criterion Corrected for small sample size*) over all possible *df* for each eigenSpline generated by get_eigen_spline. The resulting matrix of fitting parameter values can be plotted using plot_param_evolution.

Usage

```
get_param_evolution(eigen, step = 0.1)
```

Arguments

eigen	A list of eigenSpline parameters as generated by get_eigen_spline, containing eigen$matrix, eigen$variance, eigen$model and eigen$countTP.
step	(float) The *df* increment employed to cover the range of *df*. Default steps of 0.1

Value

A list of *n* matrices (*n* being the number or eigenSplines). Each matrix of fitting parameters has as rows different fitting metrics, as columns different *df* values.

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DFoverlay_list, get_eigen_DF, get_eigen_spline, plot_nbTP_histogram, plot_param_evolution

Examples

```
## 8 subjects, 4 time-points, 3 variables
inputData <- acuteInflammation$data[0:32,1:3]
ind       <- acuteInflammation$meta$ind[0:32]
time      <- acuteInflammation$meta$time[0:32]
eigen     <- get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV",
                              method="nipals", verbose=TRUE, centering=TRUE, ncores=0)
# nipals calculated PCA
# Importance of component(s):
#                   PC1     PC2      PC3
# R2            0.9272 0.06606 0.006756
# Cumulative R2 0.9272 0.99324 1.000000
# total time: 0.02 secs
get_param_evolution(eigen, step=1)
# [[1]]
#                                  2           3          4
# Penalised_residuals(CV)  103.55727   141.55548 267.197267
# Penalised_residuals(GCV)  90.84612   122.03917 198.953021
# AIC                      185.57835    67.02707   8.000000
# BIC                      184.35094    65.18611   5.545177
# AICc                     197.57835 95464.81688 -32.000000
#
# [[2]]
#                                   2            3          4
# Penalised_residuals(CV)   0.2257652 6.401150e-01   1.512174
# Penalised_residuals(GCV)  0.3034771 6.647154e-01   1.173309
# AIC                       4.6062841 6.331849e+00   8.000000
# BIC                       3.3788728 4.490887e+00   5.545177
# AICc                     16.6062865 9.540412e+04 -32.000000
#
# [[3]]
#                                   2            3          4
# Penalised_residuals(CV)   0.8338811 9.171538e-01   1.484069
# Penalised_residuals(GCV)  0.6607046 7.148925e-01   1.105211
# AIC                       5.3094592 6.354912e+00   8.000000
# BIC                       4.0820479 4.513949e+00   5.545177
# AICc                     17.3094616 9.540414e+04 -32.000000
```

loglik_smooth_spline *Calculate the penalised loglikelihood of a smooth.spline*

Description

Calculate the penalised loglikelihood of a `smooth.spline` using the integrated second derivative. The likelihood consists of 1) the (weighted) residuals sum of squares, 2) a penalty term (integrated second derivative = total curvature). The smaller the penalised loglikelihood, the better the fit as the residuals and penalty on roughness are minimised. Adapted from `aroma.light::likelihood.smooth.spline`.

Usage

```
loglik_smooth_spline(fittedSmoothSpline)
```

Arguments

`fittedSmoothSpline`	A fitted `smooth.spline`

Value

The penalised loglikelihood.

plot_nbTP_histogram *Plot an histogram of the number of time-trajectories with a given number of time-points*

Description

Histogram of the number of time-trajectories with a minimum number of time-points. When the number of time-points is inferior to the *df* selected, a spline cannot be fitted. The histogram highlights the number and percentage of time-trajectories that will be rejected for a given *df*.

Usage

```
plot_nbTP_histogram(eigen, dfCutOff = NA)
```

Arguments

`eigen`	A list of eigenSpline parameters as generated by `get_eigen_spline`, containing `eigen$matrix`, `eigen$variance`, `eigen$model` and `eigen$countTP`.
`dfCutOff`	(int) A number (a selected *df*) to highlight the portion of trajectories that would be rejected form the dataset (numberTP < *df*). Default is NA, with no cut-off plotted.

Value

A ggplot2 plotObject.

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DFoverlay_list, get_eigen_DF, get_eigen_spline, get_param_evolution, plot_param_evolution

Examples

```
## 8 subjects, 4 time-points, 3 variables, some missing values
inputData  <- acuteInflammation$data[0:32,1:3]
inputData  <- inputData[-1,]
inputData  <- inputData[-8,]
inputData  <- inputData[-30,]
inputData  <- inputData[-29,]
ind        <- acuteInflammation$meta$ind[0:32]
ind        <- ind[-1]
ind        <- ind[-8]
ind        <- ind[-30]
ind        <- ind[-29]
time       <- acuteInflammation$meta$time[0:32]
time       <- time[-1]
time       <- time[-8]
time       <- time[-30]
time       <- time[-29]
eigen      <- get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV",
                               method="nipals", verbose=TRUE, centering=TRUE, ncores=0)
plot_nbTP_histogram(eigen, dfCutOff=3)
```

---

plot_param_evolution *Plot the evolution of different fitting parameters across all possible df for each eigenSpline*

Description

Plot the evolution of 5 different fitting metrics (**CV***: Cross-Validation,* **GCV***: Generalised Cross-Validation,* **AIC***: Akaike Information Criterion,* **BIC***: Bayesian Information Criterion,* **AICc***: Akaike Information Criterion Corrected for small sample size*) over all possible *df* for each eigenSpline generated by get_param_evolution.

Usage

```
plot_param_evolution(paramSpace, scaled = FALSE)
```

Arguments

paramSpace	A list of *n* matrices (*n* being the number or eigenSplines) as generated by plot_param_evolution. Each matrix of fitting parameters has as rows different fitting metrics, as columns different *df* values.
scaled	(bool) If TRUE, the value of each eigenSpline fitting parameter are scaled between 0 and 1. Default is TRUE.

Value

A list of ggplot2 plotObjects, one plot per fitting parameters. All results can be plotted using `do.call(grid.arrange, returnedResult)`

See Also

Graphical implementation with santaR_start_GUI

Other DFsearch: get_eigen_DFoverlay_list, get_eigen_DF, get_eigen_spline, get_param_evolution, plot_nbTP_histogram

Examples

```
## 8 subjects, 4 time-points, 3 variables
inputData  <- acuteInflammation$data[0:32,1:3]
ind        <- acuteInflammation$meta$ind[0:32]
time       <- acuteInflammation$meta$time[0:32]
eigen      <- get_eigen_spline(inputData, ind, time, nPC=NA, scaling="scaling_UV",
                               method="nipals", verbose=TRUE, centering=TRUE, ncores=0)
paramSpace <- get_param_evolution(eigen, step=0.25)
plotList   <- plot_param_evolution(paramSpace, scaled=TRUE)
plotList[1]
#do.call(grid.arrange, plotList )
```

---

santaR *santaR: A package for Short AsyNchronous Time-series Analysis in R*

Description

santaR provides a graphical and automated pipeline for the analysis of short time-series studies. It enables the detection of significantly altered time trajectories between study groups, while being resilient to missing values and unsynchronised measurements.

Details

The main functions of **santaR** are santaR_start_GUI to start the graphical user interface, as well as santaR_auto_fit and santaR_auto_summary for automated command line analysis and reporting. Refer to the vignettes for graphical user interface and command line tutorials.

Author(s)

Maintainer: Arnaud Wolfer <adwolfer@gmail.com>

Other contributors:

- Timothy Ebbels <t.ebbels@imperial.ac.uk> [contributor]
- Joe Cheng <joe@rstudio.com> (Shiny javascript custom-input control) [contributor]

See Also

Useful links:

- https://github.com/adwolfer

`santaR_auto_fit` *Automate all steps of santaR fitting, Confidence bands estimation and p-values calculation for one or multiple variables*

Description

`santaR_auto_fit` encompasses all the analytical steps for the detection of significantly altered time trajectories (*input data preparation:* `get_ind_time_matrix`*, establishing group membership:* `get_grouping`*, spline modelling of individual and group time evolutions:* `santaR_fit`*, computation of group mean curve confidence bands:* `santaR_CBand`*, identification of significantly altered time trajectories:* `santaR_pvalue_dist` *and/or* `santaR_pvalue_fit`). As *santaR* is an univariate approach, multiple variables can be processed independently, which `santaR_auto_fit` can execute in parallel over multiple CPU cores.

Usage

```
santaR_auto_fit(inputData, ind, time, group = NA, df, ncores = 0,
  CBand = TRUE, pval.dist = TRUE, pval.fit = FALSE, nBoot = 1000,
  alpha = 0.05, nPerm = 1000, nStep = 5000, alphaPval = 0.05,
  forceParIndTimeMat = FALSE)
```

Arguments

`inputData`	`data.frame` of measurements with observations as rows and variables as columns.
`ind`	Vector of subject identifier (individual) corresponding to each measurement.
`time`	Vector of the time corresponding to each measurement.
`group`	NA or vector of group membership for each measurement. Default is NA for no groups.
`df`	(float) Degree of freedom to employ for fitting the individual and group mean `smooth.spline`.
`ncores`	(int) Number of cores to use for parallelisation. Default 0 for no parallelisation.
`CBand`	If TRUE calculate confidence bands for group mean curves. Default is TRUE.

`pval.dist` If TRUE calculate *p*-value based on inter-group mean curve distance. Default is TRUE.

`pval.fit` If TRUE calculate *p*-value based on group mean curve improvement in fit. Default is FALSE.

`nBoot` (int) Number of bootstrapping rounds for confidence band calculation. Default 1000.

`alpha` (float) Confidence *(0.05 for 95% Confidence Bands)*. Default 0.05.

`nPerm` (int) Number of permutations for *p*-value calculation. Default 1000.

`nStep` (int) Number of steps (granularity) employed for the calculation of the area between group mean curves (*p-value dist*). Default is 5000.

`alphaPval` (float) Confidence Interval on the permuted *p*-value *(0.05 for 95% Confidence Interval)*. Default 0.05.

`forceParIndTimeMat`
If TRUE parallelise the preparation of input data by `get_ind_time_matrix`. Default is FALSE.

Details

Note:

- The calculation of confidence bands accounts for approximately a third of the time taken by `santaR_auto_fit`, while the identification of significantly altered time trajectories (either `santaR_pvalue_dist` or `santaR_pvalue_fit`) accounts for two third of the total time. The time taken by these steps increases linearly with the increase of their respective parameters: `nBoot` for confidence bands, `nPerm` and `nStep` for identification of significantly altered trajectories using `santaR_pvalue_dist`, `nPerm` for `santaR_pvalue_fit`. Default values of these parameters are optimised to balance the time taken with the precision of the value estimation; increasing `nPerm` can tighten the *p*-value confidence intervals.
- If the parallelisation is activated (`ncores>0`), the fit of spline models, the calculation of confidence bands on the group mean curves and the identification of altered trajectories are executed for multiple variables simultaneously. However the preparation of input data (`get_ind_time_matrix`) is not parallelised by default as the parallelisation overhead cost is superior to the time potentially gained for all but the most complex datasets. The parallelisation overhead (*instantiating worker nodes, duplicating and transferring inputs to the worker nodes, concatenating results*) typically equals around 2 seconds, while executing `get_ind_time_matrix` is usually a matter of millisecond for a single variable (*ex: 7 time-points, 24 individuals, 1 variable)*; the parallelisation overhead far exceeding the time needed to process all variables sequentially. If the number of individual trajectories (subjects), of time-points, or of variables is very large, `forceParIndTimeMat` enables the parallelisation of `get_ind_time_matrix`.

Value

A list of *SANTAObj* corresponding to each variable's analysis result.

See Also

Other AutoProcess: santaR_auto_summary, santaR_plot, santaR_start_GUI

Other Analysis: get_grouping, get_ind_time_matrix, santaR_CBand, santaR_auto_summary, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_pvalue_fit, santaR_start_GUI

Examples

```
## 2 variables, 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
inputData     <- acuteInflammation$data[,1:2]
ind           <- acuteInflammation$meta$ind
time          <- acuteInflammation$meta$time
group         <- acuteInflammation$meta$group
SANTAObjList  <- santaR_auto_fit(inputData, ind, time, group, df=5, ncores=0, CBand=TRUE,
                                 pval.dist=TRUE, nBoot=100, nPerm=100)
# Input data generated: 0.02 secs
# Spline fitted: 0.03 secs
# ConfBands done: 0.53 secs
# p-val dist done: 0.79 secs
# total time: 1.37 secs
length(SANTAObjList)
# [1] 2
names(SANTAObjList)
# [1] "var_1" "var_2"
```

---

santaR_auto_summary *Summarise, report and save the results of a santaR analysis*

Description

After multiple variables have been analysed using santaR_auto_fit, santaR_auto_summary helps identify significant results and summarise them in an interpretable fashion. Correction for multiple testing can be applied to generate Bonferroni *[1]*, Benjamini-Hochberg *[2]* or Benjamini-Yekutieli *[3]* corrected *p*-values. *P*-values can be saved to disk in .csv files. For a given significance cut-off (plotCutOff), the number of variables significantly altered is reported and plots are automatically saved to disk by increasing *p*-value. The aspect of the plots can be altered such as the representation of confidence bands (showConfBand) or the generation of a mean curve across all samples (showTotalMeanCurve) to help assess difference between groups when group sizes are unbalanced.

Usage

```
santaR_auto_summary(SANTAObjList, targetFolder = NA, summaryCSV = TRUE,
  CSVName = "summary", savePlot = TRUE, plotCutOff = 0.05,
  showTotalMeanCurve = TRUE, showConfBand = TRUE, legend = TRUE,
  fdrBH = TRUE, fdrBY = FALSE, fdrBonf = FALSE, CIpval = TRUE,
  plotAll = FALSE)
```

Arguments

SANTAObjList	A list of *SANTAObj* with *p*-values calculated, as generated by santaR_auto_fit.
targetFolder	(NA or str) NA or the path to a folder in which to save summary.xls and plots. If NA no outputs are saved to disk. If targetFolder does not exist, folders will be created. Default is NA.
summaryCSV	If TRUE save the (*corrected if applicable*) *p*-values to 'CSVName'_summary.csv, 'CSVName'_pvalue-all.csv, 'CSVName'_pvalue-dist.csv, 'CSVName'_pvalue-dist.csv (*default* summary_summary.csv,...). Default is TRUE.
CSVName	(string) Filename of the *csv* to save. Default is 'summary'.
savePlot	If TRUE save to targetFolder all variables with $p <$ plotCutOff ordered by *p*-values. Default is TRUE.
plotCutOff	(float) *P*-value cut-off value to save summary plots to disk. Default 0.05.
showTotalMeanCurve	If TRUE add the mean curve across all groups on the plots. Default is TRUE.
showConfBand	If TRUE plot the confidence band for each group. Default is TRUE.
legend	If TRUE add a legend to the plots. Default is TRUE.
fdrBH	If TRUE add the Benjamini-Hochberg corrected *p*-value to the output. Default is TRUE.
fdrBY	If TRUE add the Benjamini-Yekutieli corrected *p*-value to the output. Default is FALSE.
fdrBonf	If TRUE add the Bonferroni corrected *p*-value to the output. Default is FALSE.
CIpval	If TRUE add the upper and lower confidence interval on *p*-value to the output. Default is TRUE.
plotAll	If TRUE override the plotCutOff parameter and plot all variables. Default is FALSE.

Value

A list: result$pval.all data.frame of *p*-values, with all variables as rows and different *p*-value corrections as columns. result$pval.summary data.frame of number of variables with a *p*-value inferior to a cut-off. Different metric and p-value correction as rows, different cut-off (*Inf 0.05*, *Inf 0.01*, *Inf 0.001*) as columns.

References

[1] Bland, J. M. & Altman, D. G. *Multiple significance tests: the Bonferroni method*. British Medial Journal **310**, 170 (1995).

[2] Benjamini, Y. & Hochberg, Y. *Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing*. Journal of the Royal Statistical Society **57**, 1, 289-300 (1995).

[3] Benjamini, Y. & Yekutieli, D. *The control of the false discovery rate in multiple testing under depencency*. The Annals of Statistics **29**, 1165-1188 (2001).

See Also

Other AutoProcess: santaR_auto_fit, santaR_plot, santaR_start_GUI

Other Analysis: get_grouping, get_ind_time_matrix, santaR_CBand, santaR_auto_fit, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_pvalue_fit, santaR_start_GUI

Examples

```
## 2 variables, 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
inputData     <- acuteInflammation$data[,1:2]
ind           <- acuteInflammation$meta$ind
time          <- acuteInflammation$meta$time
group         <- acuteInflammation$meta$group
SANTAObjList  <- santaR_auto_fit(inputData, ind, time, group, df=5, ncores=0, CBand=TRUE,
                                 pval.dist=TRUE, nBoot=100, nPerm=100)
# Input data generated: 0.02 secs
# Spline fitted: 0.03 secs
# ConfBands done: 0.53 secs
# p-val dist done: 0.79 secs
# total time: 1.37 secs
result <- santaR_auto_summary(SANTAObjList)
print(result)
# $pval.all
#              dist dist_upper  dist_lower     curveCorr    dist_BH
# var_1 0.03960396 0.09783202 0.015439223 -0.2429725352 0.03960396
# var_2 0.00990099 0.05432519 0.001737742  0.0006572238 0.01980198
#
# $pval.summary
#       Test Inf 0.05 Inf 0.01 Inf 0.001
# 1    dist        2        1         0
# 2 dist_BH        2        0         0
```

---

santaR_CBand *Compute Group Mean Curve Confidence Bands*

Description

Generate bootstrapped group mean curve Confidence Bands, by resampling of individual curves with replacement. Returns a *SANTAObj* with added Confidence Bands.

- Resampling whole data curves assumes less of the data than resampling of residuals.
- The resampled distribution is of same size as the original distribution (same number of individuals in each group as in the input data).
- The degree of freedom for the estimator is identical to the one employed for curve fitting in santaR_fit.

Usage

```
santaR_CBand(SANTAObj, nBoot = 1000, alpha = 0.05, subsampling = 250)
```

Arguments

SANTAObj	A fitted *SANTAObj* as generated by santaR_fit.
nBoot	(int) Number of bootstrapping rounds. Default 1000.
alpha	(float) Confidence *(0.05 for 95% Confidence Bands)*. Default 0.05.
subsampling	(int) Number of points to sample in the time range (for the estimator and Confidence Bands). Default is 250.

Value

A *SANTAObj* with added Confidence Bands for each group.

See Also

Other Analysis: get_grouping, get_ind_time_matrix, santaR_auto_fit, santaR_auto_summary, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_pvalue_fit, santaR_start_GUI

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAObj    <- santaR_CBand(SANTAObj, nBoot=100)
```

---

santaR_fit *Generate a SANTAObj for a variable*

Description

Generate a *SANTAObj* containing all the splines model for individual and group time evolutions. Once all the splines representing individual and group evolutions are fitted, all time-points are back-projected (projected) and employed in subsequent analysis in place of the input measurements (functional approach). A grouping can be provided to separate individuals and compare trajectories: any number of groups can be provided, but comparision of group trajectories can only be executed between 2 groups.

- Individual trajectories with less than 4 time-points are rejected due to constraints on smooth.spline fitting *(number of time-points < 4)*.

- Individual trajectories with less time-points than *df* are rejected due to constraints on `smooth.spline` fitting *(number of time-points < df)*.
- Rejected individual trajectories are not taken into account for mean curves calculations.

Usage

```
santaR_fit(inputMatrix, df, grouping = NA, verbose = TRUE)
```

Arguments

`inputMatrix`	`data.frame` of measurements for each IND x TIME as generated by `get_ind_time_matrix`. Rows are unique Individual IDs and columns unique measurement Time. Pairs of (IND,TIME) without a measurement are left as NA.
`df`	(float) Degree of freedom to employ for fitting the `smooth.spline`
`grouping`	NA or a `data.frame` with 2 columns (ind and group) listing as rows each unique Individual ID and the corresponding group membership, as generated by `get_grouping`. Default is NA for no groups.
`verbose`	(bool) If TRUE output the progress of fitting. Default is TRUE.

Value

A *SANTAObj* containing all the spline models with individual and group time evolutions, for further analysis.

Details:: The returned *SANTAObj* is structured as follow:

SANTAObj	santaR object for futher analysis
SANTAObj$properties$df	input degree of freedom
SANTAObj$properties$CBand$status	Confidence Bands for group mean curve calculated *(TRUE or FALSE)*
SANTAObj$properties$CBand$nBoot	parameter, number or bootstrap rounds for calculation of the group mean curve cor
SANTAObj$properties$CBand$alpha	parameter, confidence of the group mean curve band
SANTAObj$properties$pval.dist$status	*p*-value distance calculated *(TRUE or FALSE)*
SANTAObj$properties$pval.dist$nPerm	parameter, number of permutations for calculation of distance *p*-value
SANTAObj$properties$pval.dist$alpha	parameter, confidence on the bootstrapped *p*-value
SANTAObj$properties$pval.fit$status	*p*-value fitting calculated *(TRUE or FALSE)*
SANTAObj$properties$pval.fit$nPerm	parameter, number of permutations for calculation of fitting *p*-value
SANTAObj$properties$pval.fit$alpha	parameter, confidence on the bootstrapped *p*-value
SANTAObj$general$inputData	*inputMatrix*
SANTAObj$general$cleanData.in	only kept individuals INPUT values *(equivalent to inputMatrix - rejected)*
SANTAObj$general$cleanData.pred	only kept individuals PREDICTED values on Ind splines
SANTAObj$general$grouping	grouping vector given as input
SANTAObj$general$meanCurve	spline fit over all kept datapoint *(cleanData.pred)* | `smooth.spline` object
SANTAObj$general$pval.curveCorr	Pearson correlation coefficient between the two group curves, to detect highly corr
SANTAObj$general$pval.dist	*p*-value between groups based on distance between groupMeanCurves
SANTAObj$general$pval.dist.l	lower bound confidence interval on *p*-value
SANTAObj$general$pval.dist.u	upper bound confidence interval on *p*-value

SANTAObj$general$pval.fit	*p*-value between groups based on groupMeanCurves fitting
SANTAObj$general$pval.fit.l	lower bound confidence interval on *p*-value
SANTAObj$general$pval.fit.u	upper bound confidence interval on *p*-value
SANTAObj$groups	list of group information
SANTAObj$groups$rejectedInd	list of rejected individual *(#tp < 4 or df)* | data
SANTAObj$groups$curveInd	list of spline fit | `smooth.spline` object
SANTAObj$groups$groupMeanCurve	spline fit over groupData.pred | `smooth.spline` object
SANTAObj$groups$point.in	all group points INPUT values (x,y) [kept individuals]
SANTAObj$groups$point.pred	all group points PREDICTED values on Ind splines (x,y)
SANTAObj$groups$groupData.in	only individuals from this group INPUT value (IND x TIME)
SANTAObj$groups$groupData.pred	only individuals from this group PREDICTED values on Ind splines (x,y)

See Also

Other Analysis: `get_grouping`, `get_ind_time_matrix`, `santaR_CBand`, `santaR_auto_fit`, `santaR_auto_summary`, `santaR_plot`, `santaR_pvalue_dist`, `santaR_pvalue_fit`, `santaR_start_GUI`

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
Yi             <- acuteInflammation$data$var_1
ind            <- acuteInflammation$meta$ind
time           <- acuteInflammation$meta$time
group          <- acuteInflammation$meta$group
grouping       <- get_grouping(ind, group)
inputMatrix    <- get_ind_time_matrix(Yi, ind, time)
resultSANTAObj <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
```

`santaR_plot` *Plot a SANTAObj*

Description

Plot a *SANTAObj* generated by `santaR_fit`. Returns a `ggplot2` *plotObject* that can be further modified using `ggplot2` grammar.

Usage

```
santaR_plot(SANTAObj, title = "", legend = TRUE, showIndPoint = TRUE,
  showIndCurve = TRUE, showGroupMeanCurve = TRUE,
  showTotalMeanCurve = FALSE, showConfBand = TRUE, colorVect = NA,
  sampling = 250, xlab = "x", ylab = "y", shortInd = FALSE)
```

Arguments

SANTAObj	A fitted *SANTAObj* as generated by santaR_fit.
title	(str) A plot title. The default title is empty.
legend	(bool) If TRUE a legend panel is added to the right. Default is TRUE. *Note: the legend cannot be generated if only the Confidence Bands or the Total Mean Curve are plotted.*
showIndPoint	(bool) If TRUE plot each input measurements (in group color). Default is TRUE.
showIndCurve	(bool) If TRUE plot each individual's curve (in group color). Default is TRUE.
showGroupMeanCurve	(bool) If TRUE plot the mean curve for each group (in group color). Default is TRUE.
showTotalMeanCurve	(bool) If TRUE plot the mean curve across all measurements and groups (in grey). Default is FALSE.
showConfBand	If TRUE plot the confidence bands calculated with santaR_CBand.
colorVect	Vector of ggplot2 colors. The number of colors must match the number of groups *(ex:*colorVect=c("deepskyblue","red")*)*.
sampling	(int) Number of data points to use when plotting each spline (sub-sampling). Default is 250.
xlab	(str) x-axis label. Default is 'x'.
ylab	(str) y-axis label. Default is 'y'.
shortInd	if TRUE individual trajectories are only plotted on the range on which they are defined. Default is FALSE.

Value

A ggplot2 *plotObject*.

See Also

Other Analysis: get_grouping, get_ind_time_matrix, santaR_CBand, santaR_auto_fit, santaR_auto_summary, santaR_fit, santaR_pvalue_dist, santaR_pvalue_fit, santaR_start_GUI

Other AutoProcess: santaR_auto_fit, santaR_auto_summary, santaR_start_GUI

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAObj    <- santaR_CBand(SANTAObj, nBoot=100)
p           <- santaR_plot(SANTAObj, title='Example')
```

```
print(p)
```

`santaR_pvalue_dist` *Evaluate difference in group trajectories based on the comparison of distance between group mean curves*

Description

Evaluate the difference in group trajectories by executing a t-test based on the comparison of distance between group mean curves. Individual group membership is repeatedly randomly permuted to generate new random groups and group mean curves, then employed to compute a *Null* distribution of distance between goup mean curves. The distance between two group mean curves is defined as the area between both curves. The distance between the real group mean curves is then compared to this *Null* distribution and a *p*-value is computed.

- The Pearson correlation coefficient between the two group mean curves is calculated to detect highly correlated group shapes if required.
- The *p*-value is calculated as `(b+1)/(nPerm+1)` as to not report a *p*-value=0 (which would give problem with FDR correction) and reduce type I error.
- The *p*-value will vary depending on the random sampling. Therefore a confidence interval can be constructed using Wilson's interval which presents good properties for small number of trials and probabilities close to 0 or 1.

Usage

```
santaR_pvalue_dist(SANTAObj, nPerm = 1000, nStep = 5000, alpha = 0.05)
```

Arguments

`SANTAObj`	A fitted *SANTAObj* as generated by `santaR_fit`.
`nPerm`	(int) Number of permutations. Default 1000.
`nStep`	(int) Number of steps employed for the calculation of the area between group mean curves. Default is 5000.
`alpha`	(float) Confidence Interval on the permuted *p*-value *(0.05 for 95% Confidence Interval)*. Default 0.05.

Value

A *SANTAObj* with added *p*-value dist and confidence interval on the *p*-value.

See Also

Comparison with constant model with `santaR_pvalue_dist_within`

Other Analysis: `get_grouping`, `get_ind_time_matrix`, `santaR_CBand`, `santaR_auto_fit`, `santaR_auto_summary`, `santaR_fit`, `santaR_plot`, `santaR_pvalue_fit`, `santaR_start_GUI`

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAObj    <- santaR_pvalue_dist(SANTAObj, nPerm=100)
```

---

santaR_pvalue_dist_within

Evaluate difference between a group mean curve and a constant model

---

Description

Execute a t-test based on the comparison of distance between a group mean curve and a constant linear model. Generate *n* constant linear model. The *Null* distribution is generated by permuting the *n* group individuals and the *n* constant trajectories. The real distance (area) between the group trajectory and the flat trajectory is compared to the *Null* distribution of distances, similarly to santaR_pvalue_dist.

Usage

```
santaR_pvalue_dist_within(SANTAGroup, nPerm = 1000, nStep = 5000)
```

Arguments

SANTAGroup	A fitted group extracted from a *SANTAObj* generated by santaR_fit.
nPerm	(int) Number of permutations. Default 1000.
nStep	(int) Number of steps employed for the calculation of the area between group mean curves. Default is 5000.

Value

A *p-value*

See Also

Inter-group comparison with santaR_pvalue_dist

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAGroup  <- SANTAObj$groups[[2]]
#SANTAGroup <- SANTAObj$groups$Group2
santaR_pvalue_dist_within(SANTAGroup, nPerm=500)
# ~0.00990099
```

`santaR_pvalue_fit` *Evaluate difference in group trajectories based on the comparison of model fit (F-test) between one and two groups*

Description

Evaluate the difference in group trajectories by executing a t-test based on the comparison of improvement in model fit *(F-test)* between fitting one group mean curve to all individuals and fitting two group mean curves. This between-class differential evolution test, evaluates whether fitting 2 group curves decreases the residuals compared to a single group mean curve. The statistic employed is defined as a quantification of evidence for differential evolution, with the larger the statistic the more differentially evolving the variable appears to be. Individual group membership is repeatedly randomly permuted to generate new random groups and group mean curves, then employed to compute a *Null* distribution of the statistic (improvement in model fit from one to two groups). The improvement in model fit for the real group membership is then compared to this *Null* distribution *(of no group difference)* and a *p*-value is computed. Adapted from *Storey and al. 'Significance analysis of time course microarray experiments', PNAS, 2005 [1].*

- The *p*-value is calculated as `(b+1)/(nPerm+1)` as to not report a *p*-value=0 (which would give problem with FDR correction) and reduce type I error.
- The *p*-value will vary depending on the random sampling. Therefore a confidence interval can be constructed using Wilson's interval which presents good properties for small number of trials and probabilities close to 0 or 1.

Usage

```
santaR_pvalue_fit(SANTAObj, nPerm = 1000, alpha = 0.05)
```

Arguments

SANTAObj	A fitted *SANTAObj* as generated by santaR_fit.
nPerm	(int) Number of permutations. Default 1000.
alpha	(float) Confidence Interval on the permuted *p*-value *(0.05 for 95% Confidence Interval)*. Default 0.05.

Value

A *SANTAObj* with added *p*-value fit and confidence interval on the *p*-value.

References

[1] Storey, J. D., Xiao, W., Leek, J. T., Tompkins, R. G. & Davis, R. W. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences of the United States of America* **102**, 12837-42 (2005).

See Also

Comparison with constant model with santaR_pvalue_fit_within

Other Analysis: get_grouping, get_ind_time_matrix, santaR_CBand, santaR_auto_fit, santaR_auto_summary, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_start_GUI

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAObj    <- santaR_pvalue_fit(SANTAObj, nPerm=100)
```

---

santaR_pvalue_fit_within

Evaluate difference between a group mean curve and a constant model using the comparison of model fit (F-test)

Description

Execute a t-test based on the comparison of improvement of model fit from a single group mean curve to the fit of both a group mean curve and a constant linear model. This statistic identifies within-class differential evolution, and test whether the population average time curve is flat or not. *n* constant linear model are generated to match the *n* individual trajetories. The *Null* distribution is generated by permuting the *n* group individuals and the *n* constant trajectories. The real improvement in model fit for the real group membership versus flat trajectories is then compared to the *Null* distribution of model fit improvement, similarly to `santaR_pvalue_fit`. Adapted from *Storey and al. 'Significance analysis of time course microarray experiments', PNAS, 2005 [1].*

Usage

```
santaR_pvalue_fit_within(SANTAGroup, nPerm = 1000)
```

Arguments

`SANTAGroup`	A fitted group extracted from a *SANTAObj* generated by `santaR_fit`.
`nPerm`	(int) Number of permutations. Default 1000.

Value

A *p-value*

References

[1] Storey, J. D., Xiao, W., Leek, J. T., Tompkins, R. G. & Davis, R. W. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences of the United States of America* **102**, 12837-42 (2005).

See Also

Inter-group comparison with `santaR_pvalue_fit`

Examples

```
## 56 measurements, 8 subjects, 7 unique time-points
## Default parameter values decreased to ensure an execution < 2 seconds
Yi          <- acuteInflammation$data$var_3
ind         <- acuteInflammation$meta$ind
time        <- acuteInflammation$meta$time
group       <- acuteInflammation$meta$group
grouping    <- get_grouping(ind, group)
inputMatrix <- get_ind_time_matrix(Yi, ind, time)
SANTAObj    <- santaR_fit(inputMatrix, df=5, grouping=grouping, verbose=TRUE)
SANTAGroup  <- SANTAObj$groups[[1]]
#SANTAGroup <- SANTAObj$groups$Group1
santaR_pvalue_fit_within(SANTAGroup, nPerm=500)
# ~0.6726747
```

santaR_start_GUI *santaR Graphical User Interface*

Description

santaR Graphical User Interface (GUI) implements all the functions for short asynchronous time-series analysis. To exit press ESC in the command line. Once started, the GUI presents 4 tabs corresponding to the main steps of analysis: *Import*, *DF search*, *Analysis* and *Export*.

- The *Import* tab manages input data in comma separated value (*csv*) format or as an *RData* file containing a SANTAObj previously generated with **santaR**. Once data is imported the *DF search* and *Analysis* tabs become available.
- *DF search* implements the tools for the selection of an optimal number of degrees of freedom (df).
- With the data imported and a pertinent df selected, *Analysis* regroups the interface to visualise and identify variables significantly altered over time. All options present in the command line version of **santaR** are available, with the added possibility to modify the class labelling of each subject (*group*). A plotting interface enables the interactive visualisation of the raw data points, individual trajectories, group mean curves and confidence bands for all variables, which subsequently can be saved. Finally, if inter-group differential trajectories have been characterised, all significance testing results (with correction for multiple testing) are presented in interactive tables.
- The *Export* tab manages the saving of results and automated reporting. Fitted data is saved as a SANTAObj, which contains all inputs and outputs, and can be downloaded as an *RData* file for future analysis, or reproduction of results. *csv* files containing significance testing results can also be generated and summary plot for each significantly altered variable saved for rapid evaluation.

santaR's command line procedure is the most efficient approach for very high number of variables due to the added level of automation. However the GUI can help understand the use of the methodology, select the best parameters on a subset of the data, or to visually explore the results.

Usage

```
santaR_start_GUI(browser = TRUE)
```

Arguments

browser	If TRUE open the graphical user interface in a web browser instead of a R window. Default is TRUE

Value

None, start GUI. To exit press ESC in the command line.

See Also

Other AutoProcess: santaR_auto_fit, santaR_auto_summary, santaR_plot

Other Analysis: get_grouping, get_ind_time_matrix, santaR_CBand, santaR_auto_fit, santaR_auto_summary, santaR_fit, santaR_plot, santaR_pvalue_dist, santaR_pvalue_fit

Examples

```
## Start graphical interface, press 'ESC' in the command line to stop.
santaR_start_GUI()
```

---

scaling_mean *Mean scaling of each column*

Description

Scale each variable (column) by the mean. Mean-scaling applied as (value - mean) / mean. As scaling_UV might give too much importance to flat trajectories due to the division by the standard deviation, by dividing by the mean, high intensity values will have a lower influence and the low intensity will be boosted.

Usage

```
scaling_mean(inputMat)
```

Arguments

inputMat	(Observation x Variable) data.frame of measurements, with observations as rows and different variables as columns.

Value

Matrix of measurements mean-scaled columnwise.

Examples

```
inputMat <- data.frame(matrix(c(1,4,7, 8,4,0, 3,6,9), nrow=3))
scaling_mean(inputMat)
#          X1 X2  X3
# [1,] -0.75  1 -0.5
# [2,]  0.00  0  0.0
# [3,]  0.75 -1  0.5
```

scaling_UV *Unit-Variance scaling of each column*

Description

Unit-Variance (UV) scale each variable (column). UV-scaling applied as (value - mean) / stdev. Unit-Variance Scaling or Autoscaling, is commonly applied and uses the standard deviation as the scaling factor. After autoscaling, all metabolites have a standard deviation of one and therefore the data is analyzed on the basis of correlations instead of covariances.

Usage

```
scaling_UV(inputMat)
```

Arguments

inputMat	(Observation x Variable) data.frame of measurements, with observations as rows and different variables as columns.

Value

Matrix of measurements UV-scaled columnwise.

Examples

```
inputMat <- data.frame(matrix(c(1,4,7, 8,4,0, 3,6,9), nrow=3))
scaling_UV(inputMat)
#       X1 X2 X3
# [1,] -1  1 -1
# [2,]  0  0  0
# [3,]  1 -1  1
```

Index

∗Topic **datasets**
acuteInflammation, 2
_PACKAGE *(*santaR*)*, 16

acuteInflammation, 2
AIC_smooth_spline, 3
AICc_smooth_spline, 3

BIC_smooth_spline, 4

get_eigen_DF, 4, *6*, *8*, *13*, *15, 16*
get_eigen_DFoverlay_list, *5*, 5, *8*, *13*, *15, 16*
get_eigen_spline, *4–6*, 7, *12–16*
get_eigen_spline_matrix, *7*, 9
get_grouping, 10, *12*, *17*, *19*, *21–26*, *29*, *32*
get_ind_time_matrix, *9, 10*, 11, *17–19*, *21–26*, *29*, *32*
get_param_evolution, *5, 6*, *8*, 12, *15, 16*

loglik_smooth_spline, 14

plot_nbTP_histogram, *5, 6*, *8*, *13*, 14, *16*
plot_param_evolution, *5, 6*, *8*, *12, 13*, *15*, 15, *16*

SANTAR *(*santaR*)*, 16
santaR, 16
santaR-package *(*santaR*)*, 16
santaR_auto_fit, *10*, *12*, *16*, 17, *19–22*, *24–26*, *29*, *32*
santaR_auto_summary, *10*, *12*, *16*, *19*, 19, *22*, *24–26*, *29*, *32*
santaR_CBand, *10*, *12*, *17*, *19*, *21*, 21, *24–26*, *29*, *32*
santaR_fit, *10–12*, *17*, *19*, *21, 22*, 22, *24–27*, *29, 30*, *32*
santaR_plot, *10*, *12*, *19*, *21, 22*, *24*, 24, *26*, *29*, *32*
santaR_pvalue_dist, *10*, *12*, *17–19*, *21, 22*, *24, 25*, 26, *27*, *29*, *32*
santaR_pvalue_dist_within, *26*, 27
santaR_pvalue_fit, *10*, *12*, *17–19*, *21, 22*, *24–26*, 28, *30*, *32*
santaR_pvalue_fit_within, *29*, 29
santaR_start_GUI, *5, 6*, *8*, *10*, *12, 13*, *15, 16*, *19*, *21, 22*, *24–26*, *29*, 31
scaling_mean, 32
scaling_UV, *32*, 33
smooth.spline, *3, 4*, *14*, *17*, *22–24*
sort, *11*