

# Package ‘maSigPro’

October 9, 2015

**Type** Package

**Title** Significant Gene Expression Profile Differences in Time Course  
Microarray Data

**Version** 1.40.0

**Author** Ana Conesa <aconesa@cipf.es>, Maria Jose Nueda <mj.nueda@ua.es>

**Maintainer** Maria Jose Nueda <mj.nueda@ua.es>

## Description

maSigPro is a regression based approach to find genes for which there are significant gene expression profile differences between experimental groups in time course microarray experiments.

**Depends** R (>= 2.3.1), stats, Biobase, MASS

**Imports** Biobase, graphics, grDevices, limma, Mfuzz, stats, utils, MASS

**URL** <http://bioinfo.cipf.es/>

**LazyLoad** yes

**License** GPL (>= 2)

**biocViews** Microarray, DifferentialExpression, TimeCourse

**NeedsCompilation** no

## R topics documented:

average.rows . . . . .	2
data.abiotic . . . . .	3
edesign.abiotic . . . . .	5
edesign.OD . . . . .	6
edesignCT . . . . .	6
edesignDR . . . . .	7
get.siggenes . . . . .	8
i.rank . . . . .	11
make.design.matrix . . . . .	12
maSigPro . . . . .	13
maSigProUsersGuide . . . . .	18
NBdata . . . . .	19

NBdesign . . . . .	20
p.vector . . . . .	20
PlotGroups . . . . .	23
PlotProfiles . . . . .	25
position . . . . .	27
reg.coeffs . . . . .	28
see.genes . . . . .	29
stepback . . . . .	32
stepfor . . . . .	34
suma2Venn . . . . .	35
T.fit . . . . .	36
two.ways.stepback . . . . .	39
two.ways.stepfor . . . . .	41

<b>Index</b>	<b>43</b>
--------------	-----------

---

average.rows	<i>Average rows by match and index</i>
--------------	--

---

## Description

average.rows matches rownames of a matrix to a match vector and performs averaging of the rows by the index provided by an index vector.

## Usage

```
average.rows(x, index, match, r = 0.7)
```

## Arguments

x	a matrix
index	index vector indicating how rows must be averaged
match	match vector for indexing rows
r	minimal correlation value between rows to compute average

## Details

rows will be averaged only if the pearson correlation coefficient between all rows of each given index is greater than r. If not, that group of rows is discarded in the result matrix.

## Value

a matrix of averaged rows

## Author(s)

Ana Conesa, aconesa@cipf.es

## Examples

```
## create data matrix for row averaging
x <- matrix(rnorm(30), nrow = 6, ncol = 5)
rownames(x) <- paste("ID", c(1, 2, 11, 12, 19, 20), sep = "")
i <- paste("g", rep(c(1:10), each = 2), sep = "") # index vector
m <- paste("ID", c(1:20), sep = "") # match vector
average.rows(x, i, m, r = 0)
```

---

data.abiotic

*Gene expression data potato abiotic stress*

---

## Description

data.abiotic contains gene expression of a time course microarray experiment where potato plants were submitted to 3 different abiotic stresses.

## Usage

```
data(data.abiotic)
```

## Format

A data frame with 1000 observations on the following 36 variables.

Control\_3H\_1 a numeric vector  
Control\_3H\_2 a numeric vector  
Control\_3H\_3 a numeric vector  
Control\_9H\_1 a numeric vector  
Control\_9H\_2 a numeric vector  
Control\_9H\_3 a numeric vector  
Control\_27H\_1 a numeric vector  
Control\_27H\_2 a numeric vector  
Control\_27H\_3 a numeric vector  
Cold\_3H\_1 a numeric vector  
Cold\_3H\_2 a numeric vector  
Cold\_3H\_3 a numeric vector  
Cold\_9H\_1 a numeric vector  
Cold\_9H\_2 a numeric vector  
Cold\_9H\_3 a numeric vector  
Cold\_27H\_1 a numeric vector  
Cold\_27H\_2 a numeric vector

Cold\_27H\_3 a numeric vector  
Heat\_3H\_1 a numeric vector  
Heat\_3H\_2 a numeric vector  
Heat\_3H\_3 a numeric vector  
Heat\_9H\_1 a numeric vector  
Heat\_9H\_2 a numeric vector  
Heat\_9H\_3 a numeric vector  
Heat\_27H\_1 a numeric vector  
Heat\_27H\_2 a numeric vector  
Heat\_27H\_3 a numeric vector  
Salt\_3H\_1 a numeric vector  
Salt\_3H\_2 a numeric vector  
Salt\_3H\_3 a numeric vector  
Salt\_9H\_1 a numeric vector  
Salt\_9H\_2 a numeric vector  
Salt\_9H\_3 a numeric vector  
Salt\_27H\_1 a numeric vector  
Salt\_27H\_2 a numeric vector  
Salt\_27H\_3 a numeric vector

### **Details**

This data set is part of a larger experiment in which gene expression was monitored in both roots and leaves using a 11K cDNA potato chip. This example data set contains a random subset of 1000 genes of the leave study.

### **References**

Rensink WA, Iobst S, Hart A, Stegalkina S, Liu J, Buell CR. Gene expression profiling of potato responses to cold, heat, and salt stress. *Funct Integr Genomics*. 2005 Apr 22.

### **Examples**

```
data(data.abiotic)
```

---

`edesign.abiotic`*Experimental design potato abiotic stress*

---

## Description

edesign.abiotic contains experimental set up of a time course microarray experiment where potato plants were submitted to 3 different abiotic stresses.

## Usage

```
data(edesign.abiotic)
```

## Format

A matrix with 36 rows and 6 columns

rows [1:36] "Control 3h 1" "Control 3h 2" "Control 3h 3" "Control 9h 1" ...

columns [1:6] "Time" "Replicates" "Control" "Cold" "Heat" "Salt"

## Details

Arrays are given in rows and experiment descriptors are given in columns. Row names contain array names.

"Time" indicates the values that variable Time takes in each hybridization.

"Replicates" is an index indicating replicate hybridizations, i.e. hybridizations are numbered, giving replicates the same number.

"Control", "Cold", "Heat" and "Salt" columns indicate array assignment to experimental groups, coding with 1 and 0 whether each array belongs to that group or not.

## References

Rensink WA, Iobst S, Hart A, Stegalkina S, Liu J, Buell CR. Gene expression profiling of potato responses to cold, heat, and salt stress. *Funct Integr Genomics*. 2005 Apr 22.

## Examples

```
data(edesignCR)
```

---

 edesign.OD

*Experimental design with a measured independent variable*


---

### Description

edesign.OD contains the experimental design of a E.coli growth time course microarray experiment with a temperature shift treatment. The OD of each culture was measured and used in the experimental design as independent variable.

### Usage

```
data(edesign.OD)
```

### Format

A data frame with 52 rows and the following 4 variables.

OD a numeric vector. Indicates the OD value of the sampled culture

Replicate a numeric vector

37 a numeric vector. No temperature shift treatment

SHIFT a numeric vector. Temperature shift treatment

### Examples

```
data(edesign.OD)
## maybe str(edesign.OD) ; plot(edesign.OD) ...
```

---

 edesignCT

*Experimental design with a shared time*


---

### Description

edesignCT contains the experimental set up of a time course microarray experiment where there is a common starting point for the different experimental groups.

### Usage

```
data(edesignCT)
```

### Format

A matrix with 32 rows and 7 columns

rows [1:32] "Array1" "Array2" "Array3" "Array4" ...

columns [1:7] "Time" "Replicates" "Control" "Tissue1" "Tissue2" "Tissue3" "Tissue4"

**Details**

Arrays are given in rows and experiment descriptors are given in columns. Row names contain array names.

"Time" indicates the values that variable Time takes in each hybridization. There are 4 time points, which allows an up to 3 degree regression polynome.

"Replicates" is an index indicating replicate hybridizations, i.e. hybridizations are numbered, giving replicates the same number.

"Control", "Tissue1", "Tissue2", "Tissue3" and "Tissue4" columns indicate array assignment to experimental groups, coding with 1 and 0 whether each array belongs to that group or not.

**Examples**

```
data(edesignCT)
```

---

edesignDR

*Experimental design with different replicates*

---

**Description**

edesignDR contains experimental set up of a replicated time course microarray experiment where rats were submitted to 3 different dosis of a toxic compound. A control and an placebo treatments are also present in the experiment.

**Usage**

```
data(edesignDR)
```

**Format**

A matrix with 54 rows and 7 columns

rows [1:54] "Array1" "Array2" "Array3" "Array4" ...

columns [1:7] "Time" "Replicates" "Control" "Placebo" "Low" "Medium" "High"

**Details**

Arrays are given in rows and experiment descriptors are given in columns. Row names contain array names.

"Time" indicates the values that variable Time takes in each hybridization.

"Replicates" is an index indicating replicate hybridizations, i.e. hybridizations are numbered, giving replicates the same number.

"Control", "Placebo", "Low", "Medium" and "High" columns indicate array assignment to experimental groups, coding with 1 and 0 whether each array belongs to that group or not.

## References

Heijne, W.H.M.; Stierum, R.; Slijper, M.; van Bladeren P.J. and van Ommen B.(2003). Toxicogenomics of bromobenzene hepatotoxicity: a combined transcriptomics and proteomics approach. *Biochemical Pharmacology* 65 857-875.

## Examples

```
data(edesignDR)
```

---

get.siggenes	<i>Extract significant genes for sets of variables in time series gene expression experiments</i>
--------------	---

---

## Description

This function creates lists of significant genes for a set of variables whose significance value has been computed with the `T.fit` function.

## Usage

```
get.siggenes(tstep, rsq = 0.7, add.IDs = FALSE, IDs = NULL, matchID.col = 1,
             only.names = FALSE, vars = c("all", "each", "groups"),
             significant.intercept = "dummy",

             groups.vector = NULL, trat.repl.spots = "none",
             index = IDs[, (matchID.col + 1)], match = IDs[, matchID.col],
             r = 0.7)
```

## Arguments

tstep	a <code>T.fit</code> object
rsq	cut-off level at the R-squared value for the stepwise regression fit. Only genes with R-squared more than rsq are selected
add.IDs	logical indicating whether to include additional gene id's in the result
IDs	matrix containing additional gene id information (required when add.IDs is TRUE)
matchID.col	number of matching column in matrix IDs for adding genes ids
only.names	logical. If TRUE, expression values are omitted in the results
vars	variables for which to extract significant genes (see details)
significant.intercept	experimental groups for which significant intercept coefficients are considered (see details)
groups.vector	required when vars is "groups".



trat.repl.spots	treatment given to replicate spots. Possible values are "none" and "average"
index	argument of the <a href="#">average.rows</a> function to use when trat.repl.spots is "average"
match	argument of the <a href="#">average.rows</a> function to use when trat.repl.spots is "average"
r	minimum pearson correlation coefficient for replicated spots profiles to be averaged

## Details

There are 3 possible values for the vars argument:

"all": generates one single matrix or gene list with all significant genes.

"each": generates as many significant genes extractions as variables in the general regression model. Each extraction contains the significant genes for that variable.

"groups": generates a significant genes extraction for each experimental group.

The difference between "each" and "groups" is that in the first case the variables of the same group (e.g. "TreatmentA" and "time\*TreatmentA" ) will be extracted separately and in the second case jointly.

When add.IDs is TRUE, a matrix of gene ids must be provided as argument of IDs, the matchID.col column of which having same levels as in the row names of sig.profiles. The option only.names is TRUE will generate a vector of significant genes or a matrix when add.IDs is set also to TRUE.

When trat.repl.spots is "average", match and index vectors are required for the [average.rows](#) function. In gene expression data context, the index vector would contain geneIDs and indicate which spots are replicates. The match vector is used to match these genesIDs to rows in the significant genes matrix, and must have the same levels as the row names of sig.profiles.

The argument significant.intercept modulates the treatment for intercept coefficients to apply for selecting significant genes when vars equals "groups". There are three possible values: "none", no significant intercept (differences) are considered for significant gene selection, "dummy", includes genes with significant intercept differences between control and experimental groups, and "all" when both significant intercept coefficient for the control group and significant intercept differences are considered for selecting significant genes.

add.IDs = TRUE and trat.repl.spots = "average" are not compatible argument values. add.IDs = TRUE and only.names = TRUE are compatible argument values.

## Value

summary	a vector or matrix listing significant genes for the variables given by the function parameters
sig.genes	a list with detailed information on the significant genes found for the variables given by the function parameters. Each element of the list is also a list containing: <ul style="list-style-type: none"> <li>sig.profiles: expression values of significant genes</li> <li>coefficients: regression coefficients of the adjusted models</li> <li>groups.coeffs: regression coefficients of the implicit models of each experimental group</li> </ul>

sig.pvalues: p-values of the regression coefficients for significant genes  
 g: number of genes  
 ...: arguments passed by previous functions

### Author(s)

Ana Conesa, aconesa@cipf.es; Maria Jose Nueda, mj.nueda@ua.es

### References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2006. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments. *Bioinformatics* 22, 1096-1102

### Examples

```
#### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{
  tc.dat <- NULL
  for (i in 1:n) {
    Ct1 <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ct1 group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ct1, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## Create 270 flat profiles
flat <- tc.GENE(n = 270, r = 3)
## Create 10 genes with profile differences between Ct1 and Tr1 groups
twodiff <- tc.GENE(n = 10, r = 3, b2 = 0.5, c2 = 1.3)
## Create 10 genes with profile differences between Ct1, Tr2, and Tr3 groups
threediff <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
## Create 10 genes with profile differences between Ct1 and Tr2 and different variance
vardiff <- tc.GENE(n = 10, r = 3, a3 = 0.7, b3 = 1, c3 = 1.2, var32 = 0.03, var33 = 0.03)
## Create dataset
tc.DATA <- rbind(flat, twodiff, threediff, vardiff)
```

```

rownames(tc.DATA) <- paste("feature", c(1:300), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")
tc.DATA [sample(c(1:(300*36)), 300)] <- NA # introduce missing values

#### CREATE EXPERIMENTAL DESIGN
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")

tc.p <- p.vector(tc.DATA, design = make.design.matrix(edesign), Q = 0.01)
tc.tstep <- T.fit(data = tc.p , alfa = 0.05)

## This will obtain significant genes per experimental group
## which have a regression model Rsquared > 0.9
tc.sigs <- get.siggenes (tc.tstep, rsq = 0.9, vars = "groups")

## This will obtain all significant genes regardless the Rsquared value.
## Replicated genes are averaged.
IDs <- rbind(paste("feature", c(1:300), sep = ""),
            rep(paste("gene", c(1:150), sep = ""), each = 2))
tc.sigs.ALL <- get.siggenes (tc.tstep, rsq = 0, vars = "all", IDs = IDs)
tc.sigs.groups <- get.siggenes (tc.tstep, rsq = 0, vars = "groups", significant.intercept="dummy")

```

---

i.rank

*Ranks a vector to index*


---

### Description

Ranks the values in a vector to successive values. Ties are given the same value.

### Usage

```
i.rank(x)
```

### Arguments

x                    vector

### Value

Vector of ranked values

### Author(s)

Ana Conesa, aconesa@cipf.es

**See Also**[rank,order](#)**Examples**

```
i.rank(c(1, 1, 1, 3, 3, 5, 7, 7, 7))
```

---

make.design.matrix	<i>Make a design matrix for regression fit of time series gene expression experiments</i>
--------------------	---

---

**Description**

make.design.matrix creates the design matrix of dummies for fitting time series micorarray gene expression experiments.

**Usage**

```
make.design.matrix(edesign, degree = 2, time.col = 1,
                   repl.col = 2, group.cols = c(3:ncol(edesign)))
```

**Arguments**

edesign	matrix describing experimental design. Rows must be arrays and columns experiment descriptors
degree	the degree of the regression fit polynome. degree = 1 returns linear regression, degree = 2 returns quadratic regression, etc
time.col	column number in edesign containing time values. Default is first column
repl.col	column number in edesign containing coding for replicate arrays. Default is second column
group.cols	column numbers in edesign indicating the coding for each experimental group (treatment, tissue, ...). See details

**Details**

rownames of edesign object should contain the arrays naming (i.e. array1, array2, ...). colnames of edesign must contain the names of experiment descriptors(i.e. "Time", "Replicates", "Treatment A", "Treatment B", etc.). for each experimental group a different column must be present in edesign, coding with 1 and 0 whether each array belongs to that group or not.

make.design.matrix returns a design matrix where rows represent arrays and column variables of time, dummies and their interactions for up to the degree given. Dummies show the relative effect of each experimental group related to the first one. Single dummies indicate the abcissa component of each group. \$Time\*dummy\$ variables indicate slope changes, \$Time^2\*dummy\$ indicates curvature changes. Higher grade values could model complex responses. In case experimental groups share a initial state (i.e. common time 0), no single dummies are modeled.

**Value**

dis                    design matrix of dummies for fitting time series  
 groups.vector        vector coding the experimental group to which each variable belongs to  
 edesign              edesign value passed as argument

**Author(s)**

Ana Conesa, aconesa@cipf.es; Maria Jose Nueda, mj.nueda@ua.es

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2006. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments. *Bioinformatics* 22, 1096-1102

**Examples**

```
data(edesign.abiotic, edesignCT)
make.design.matrix(edesign.abiotic) # quadratic model
make.design.matrix(edesignCT, degree = 3) # cubic model with common starting time point
```

---

maSigPro	<i>Wrapping function for identifying significant differential gene expression profiles in micorarray time course experiments</i>
----------	--

---

**Description**

maSigPro performs a whole maSigPro analysis for a times series gene expression experiment. The function sucesively calls the functions [make.design.matrix\(optional\)](#), [p.vector](#), [T.fit](#), [get.siggenes](#) and [see.genes](#).

**Usage**

```
maSigPro(data, edesign, matrix = "AUTO", groups.vector = NULL,
  degree = 2, time.col = 1, repl.col = 2, group.cols = c(3:ncol(edesign)),
  Q = 0.05, alfa = Q, nvar.correction = FALSE, step.method = "backward", rsq = 0.7,
  min.obs = 3, vars = "groups", significant.intercept = "dummy", cluster.data = 1,
  add.IDs = FALSE, IDs = NULL, matchID.col = 1, only.names = FALSE, k = 9, m = 1.45,
  cluster.method = "hclust", distance = "cor", agгло.method = "ward", iter.max = 500,
  summary.mode = "median", color.mode = "rainbow", trat.repl.spots = "none",
  index = IDs[, (matchID.col + 1)], match = IDs[, matchID.col], rs = 0.7,
  show.fit = TRUE, show.lines = TRUE, pdf = TRUE, cexlab = 0.8,
  legend = TRUE, main = NULL, ...)
```

**Arguments**

<code>data</code>	matrix with normalized gene expression data. Genes must be in rows and arrays in columns. Row names must contain geneIDs (argument of <code>p.vector</code> )
<code>edesign</code>	matrix of experimental design. Row names must contain arrayIDs (argument of <code>make.design.matrix</code> and <code>see.genes</code> )
<code>matrix</code>	design matrix for regression analysis. By default design is calculated with <code>make.design.matrix</code> (argument of <code>p.vector</code> and <code>T.fit</code> , by default computed by <code>make.design.matrix</code> )
<code>groups.vector</code>	vector indicating experimental group of each variable (argument of <code>get.siggenes</code> and <code>see.genes</code> , by default computed by <code>make.design.matrix</code> )
<code>degree</code>	the degree of the regression fit polynome. <code>degree = 1</code> returns lineal regression, <code>degree = 2</code> returns quadratic regression, etc... (argument of <code>make.design.matrix</code> )
<code>time.col</code>	column in <code>edesign</code> containing time values. Default is first column (argument of <code>make.design.matrix</code> and <code>see.genes</code> )
<code>repl.col</code>	column in <code>edesign</code> containing coding for replicates arrays. Default is second column (argument of <code>make.design.matrix</code> and <code>see.genes</code> )
<code>group.cols</code>	columns in <code>edesign</code> indicating the coding for each group of the experiment (see <code>make.design.matrix</code> ) (argument of <code>make.design.matrix</code> and <code>see.genes</code> )
<code>Q</code>	level of false discovery rate (FDR) control (argument of <code>p.vector</code> )
<code>alfa</code>	significance level used for variable selection in the stepwise regression (argument of <code>T.fit</code> )
<code>nvar.correction</code>	logical for indicating correcting of stepwise regression significance level (argument of <code>T.fit</code> )
<code>step.method</code>	argument to be passed to the step function. Can be either "backward", "forward", "two.ways.backward" or "two.ways.forward"
<code>rsq</code>	cut-off level at the R-squared value for the stepwise regression fit. Only genes with R-squared greater than <code>rsq</code> are selected
<code>min.obs</code>	genes with less than this number of true numerical values will be excluded from the analysis (argument of <code>p.vector</code> and <code>T.fit</code> )
<code>vars</code>	variables for which to extract significant genes (argument of <code>get.siggenes</code> )
<code>significant.intercept</code>	experimental groups for which significant intercept coefficients are considered (argument of <code>get.siggenes</code> )

<code>cluster.data</code>	Type of data used by the cluster algorithm (argument of <a href="#">see.genes</a> )
<code>add.IDs</code>	logical indicating whether to include additional gene id's in the significant genes result (argument of <a href="#">get.siggenes</a> )
<code>IDs</code>	matrix containing additional gene id information (required when <code>add.IDs</code> is TRUE) (argument of <a href="#">get.siggenes</a> )
<code>matchID.col</code>	number of matching column in matrix IDs for adding genes ids (argument of <a href="#">get.siggenes</a> )
<code>only.names</code>	logical. If TRUE, expression values are omitted in the significant genes result (argument of <a href="#">get.siggenes</a> )
<code>k</code>	number of clusters (argument of <a href="#">see.genes</a> )
<code>m</code>	m parameter when "mfuzz" clustering algorithm is used. See <a href="#">mfuzz</a> (argument of <a href="#">see.genes</a> )
<code>cluster.method</code>	clustering method for data partitioning (argument of <a href="#">see.genes</a> )
<code>distance</code>	distance measurement function used when <code>cluster.method</code> is "hclust" (argument of <a href="#">see.genes</a> )
<code>agglo.method</code>	aggregation method used when <code>cluster.method</code> is "hclust" (argument of <a href="#">see.genes</a> )
<code>iter.max</code>	number of iterations when <code>cluster.method</code> is "kmeans" (argument of <a href="#">see.genes</a> )
<code>summary.mode</code>	the method to condensate expression information when more than one gene is present in the data. Possible values are "representative" and "median" (argument of <a href="#">PlotGroups</a> )
<code>color.mode</code>	color scale for plotting profiles. Can be either "rainblow" or "gray" (argument of <a href="#">PlotProfiles</a> )
<code>trat.repl.spots</code>	treatment given to replicate spots. Possible values are "none" and "average" (argument of <a href="#">get.siggenes</a> )
<code>index</code>	argument of the <a href="#">average.rows</a> function to use when <code>trat.repl.spots</code> is "average" (argument of <a href="#">get.siggenes</a> )
<code>match</code>	argument of the <code>link{average.rows}</code> function to use when <code>trat.repl.spots</code> is "average" (argument of <a href="#">get.siggenes</a> )
<code>rs</code>	minimum pearson correlation coefficient for replicated spots profiles to be averaged (argument of <a href="#">get.siggenes</a> )

<code>show.fit</code>	logical indicating whether regression fit curves must be plotted (argument of <a href="#">see.genes</a> )
<code>show.lines</code>	logical indicating whether a line must be drawn joining plotted data points for reach group (argument of <a href="#">see.genes</a> )
<code>pdf</code>	logical indicating whether a pdf results file must be generated (argument of <a href="#">see.genes</a> )
<code>cexlab</code>	graphical parameter magnification to be used for x labels in plotting functions
<code>legend</code>	logical indicating whether legend must be added when plotting profiles (argument of <a href="#">see.genes</a> )
<code>main</code>	title for pdf results file
<code>...</code>	other graphical function arguments

### Details

maSigPro finds and display genes with significant profile differences in time series gene expression experiments. The main, compulsory, input parameters for this function are a matrix of gene expression data (see [p.vector](#) for details) and a matrix describing experimental design (see [make.design.matrix](#) or [p.vector](#) for details). In case extended gene ID information is wanted to be included in the result of significant genes, a third IDs matrix containing this information will be required (see [get.siggenes](#) for details).

Basically in the function calls subsequent steps of the maSigPro approach which is:

- Make a general regression model with dummies to indicate different experimental groups.
- Select significant genes on the basis of this general model, applying *fdr* control.
- Find significant variables for each gene, using stepwise regression.
- Extract and display significant genes for any set of variables or experimental groups.

### Value

<code>summary</code>	a vector or matrix listing significant genes for the variables given by the function parameters
<code>sig.genes</code>	a list with detailed information on the significant genes found for the variables given by the function parameters. Each element of the list is also a list containing: <code>sig.profiles</code> : expression values of significant genes. The cluster assignment of each gene is given in the last column <code>coefficients</code> : regression coefficients for significant genes <code>t.score</code> : value of the t statistics of significant genes <code>sig.pvalues</code> : p-values of the regression coefficients for significant genes <code>g</code> : number of genes <code>...</code> : arguments passed by previous functions
<code>input.data</code>	input analysis data
<code>G</code>	number of input genes



edesign	matrix of experimental design
dis	regression design matrix
min.obs	imputed value for minimal number of true observations
p.vector	vector containing the computed p-values of the general regression model for each gene
variables	variables in the general regression model
g	number of significant genes
p.vector.alfa	p-value at FDR = Q control
step.method	imputed step method for stepwise regression
Q	imputed value for false discovery rate (FDR) control
step.alfa	inputed significance level in stepwise regression
influ.info	data frame of genes containing influential data

**Author(s)**

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es); Maria Jose Nueda, [mj.nueda@ua.es](mailto:mj.nueda@ua.es)

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

**See Also**

[make.design.matrix](#), [p.vector](#), [T.fit](#), [get.siggenes](#), [see.genes](#)

**Examples**

```
##### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{

  tc.dat <- NULL
  for (i in 1:n) {
    Ct1 <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ct1 group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
```

```

    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ctl, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## Create 270 flat profiles
flat <- tc.GENE(n = 270, r = 3)
## Create 10 genes with profile differences between Ctl and Tr1 groups
twodiff <- tc.GENE(n = 10, r = 3, b2 = 0.5, c2 = 1.3)
## Create 10 genes with profile differences between Ctl, Tr2, and Tr3 groups
threediff <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
## Create 10 genes with profile differences between Ctl and Tr2 and different variance
vardiff <- tc.GENE(n = 10, r = 3, a3 = 0.7, b3 = 1, c3 = 1.2, var32 = 0.03, var33 = 0.03)
## Create dataset
tc.DATA <- rbind(flat, twodiff, threediff, vardiff)
rownames(tc.DATA) <- paste("feature", c(1:300), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")
tc.DATA[sample(c(1:(300*36)), 300)] <- NA # introduce missing values

#### CREATE EXPERIMENTAL DESIGN
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0, 9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")

#### RUN maSigPro
tc.test <- maSigPro(tc.DATA, edesign, degree = 2, vars = "groups", main = "Test")

tc.test$g # gives number of total significant genes
tc.test$summary # shows significant genes by experimental groups
tc.test$sig.genes$Treat1$sig.pvalues # shows pvalues of the significant coefficients
# in the regression models of the significant genes
# for Control.vs.Treat1 comparison

```

---

maSigProUsersGuide [View maSigPro User's Guide](#)

---

## Description

Finds the location of the maSigPro User's Guide and opens it.

## Usage

```
maSigProUsersGuide(view=TRUE)
```

**Arguments**

`view` logical, to specify if the document is opened using the PDF document reader.

**Details**

The function `vignette("maSigPro")` will find the short maSigPro Vignette which describes how to obtain the maSigPro User's Guide. The User's Guide is not itself a true vignette because it is not automatically generated using Sweave during the package build process. This means that it cannot be found using `vignette`, hence the need for this special function.

If the operating system is other than Windows, then the PDF viewer used is that given by `Sys.getenv("R_PDFVIEWER")`. The PDF viewer can be changed using `Sys.putenv(R_PDFVIEWER=)`.

**Value**

If `vignette(view=TRUE)`, the PDF document reader is started and the User's Guide is opened. If `vignette(view=FALSE)`, returns the file location.

**Examples**

```
maSigProUsersGuide()  
maSigProUsersGuide(view=FALSE)
```

---

NBdata	<i>RNA-Seq dataset example</i>
--------	--------------------------------

---

**Description**

NBdata contains a subset of a bigger normalized negative binomial simulated dataset.

**Usage**

```
data(NBdata)
```

**Format**

A data frame with 100 observations on 36 numeric variables.

**Details**

This dataset is part of a larger simulated and normalized dataset with 2 experimental groups, 6 time-points and 3 replicates. Simulation has been done by using a negative binomial distribution. The first 20 genes are simulated with changes among time.

**Examples**

```
data(NBdata)
```

---

`NBdesign`*Experimental design for RNA-Seq example*

---

**Description**

NBdesign contains a subset of a bigger normalized negative binomial simulated dataset.

**Usage**

```
data(NBdesign)
```

**Format**

A matrix with 36 rows and 4 columns

rows [1:36] "G1.T1.1" "G1.T1.2" "G1.T1.3" "G1.T2.1" ...

columns [1:6] [1] "Time" "Replicates" "Group.1" "Group.2"

**Details**

Samples are given in rows and experiment descriptors are given in columns. Row names contain sample names.

"Time" indicates the values that variable Time takes in each experimental condition. There are 6 time points.

"Replicates" is an index indicating the same experimental condition.

"Group.1" and "Group.2" columns indicate assignment to experimental groups, coding with 1 and 0 whether each sample belongs to that group or not.

**Examples**

```
data(NBdesign)
```

---

`p.vector`*Make regression fit for time series gene expression experiments*

---

**Description**

p.vector performs a regression fit for each gene taking all variables present in the model given by a regression matrix and returns a list of FDR corrected significant genes.

**Usage**

```
p.vector(data, design = NULL, Q = 0.05, MT.adjust = "BH", min.obs = 3, counts=FALSE, family=NULL, thet
```

**Arguments**

data	matrix containing normalized gene expression data. Genes must be in rows and arrays in columns
design	design matrix for the regression fit such as that generated by the <a href="#">make.design.matrix</a> function
Q	significance level
MT.adjust	argument to pass to p.adjust function indicating the method for multiple testing adjustment of p.value
min.obs	genes with less than this number of true numerical values will be excluded from the analysis. Default is 3 (minimum value for a quadratic fit)
counts	a logical indicating whether your data are counts
family	the distribution function to be used in the glm model. It must be specified as a function: gaussian(), poisson(), negative.binomial(theta)... If NULL family will be negative.binomial(theta) when counts=TRUE or gaussian() when counts=FALSE
theta	theta parameter for negative.binomial family
epsilon	argument to pass to glm.control, convergence tolerance in the iterative process to estimate de glm model

**Details**

rownames(design) and colnames(data) must be identical vectors and indicate array naming.  
 rownames(data) should contain unique gene IDs.  
 colnames(design) are the given names for the variables in the regression model.

**Value**

SELEC	matrix containing the expression values for significant genes
p.vector	vector containing the computed p-values
G	total number of input genes
g	number of genes taken in the regression fit
BH.alfa	p-value at FDR Q control when Benjamini & Holderberg (BH) correction is used
i	number of significant genes
dis	design matrix used in the regression fit
dat	matrix of expression value data used in the regression fit
...	additional values from input parameters

**Author(s)**

Ana Conesa, <aconesa@cipf.es>; Maria Jose Nueda, <mj.nueda@ua.es>

## References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2006. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments. *Bioinformatics* 22, 1096-1102

## See Also

[T.fit, lm](#)

## Examples

```
##### GENERATE TIME COURSE DATA
## generates n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{

  tc.dat <- NULL
  for (i in 1:n) {
    Ctl <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ctl group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ctl, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## Create 270 flat profiles
flat <- tc.GENE(n = 270, r = 3)
## Create 10 genes with profile differences between Ctl and Tr1 groups
twodiff <- tc.GENE(n = 10, r = 3, b2 = 0.5, c2 = 1.3)
## Create 10 genes with profile differences between Ctl, Tr2, and Tr3 groups
threediff <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
## Create 10 genes with profile differences between Ctl and Tr2 and different variance
vardiff <- tc.GENE(n = 10, r = 3, a3 = 0.7, b3 = 1, c2 = 1.3, var32 = 0.03, var33 = 0.03)
## Create dataset
tc.DATA <- rbind(flat, twodiff, threediff, vardiff)
rownames(tc.DATA) <- paste("feature", c(1:300), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")
tc.DATA[sample(c(1:(300*36)), 300)] <- NA # introduce missing values

##### CREATE EXPERIMENTAL DESIGN
```

```

Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")

tc.p <- p.vector(tc.DATA, design = make.design.matrix(edesign), Q = 0.05)
tc.p$i # number of significant genes
tc.p$SELEC # expression value of significant genes
tc.p$BH.alfa # p.value at FDR control
tc.p$p.adjusted# adjusted p.values

```

---

PlotGroups	<i>Function for plotting gene expression profile at different experimental groups</i>
------------	---

---

### Description

This function displays the gene expression profile for each experimental group in a time series gene expression experiment.

### Usage

```

PlotGroups(data, edesign = NULL, time = edesign[,1], groups = edesign[,c(3:ncol(edesign))],
           repvect = edesign[,2], show.fit = FALSE, dis = NULL, step.method = "backward",
           min.obs = 2, alfa = 0.05, nvar.correction = FALSE, summary.mode = "median", show.lines = TRUE,
           xlab = "time", cex.xaxis = 1, ylim = NULL, main = NULL, cexlab = 0.8, legend = TRUE, sub = NULL)

```

### Arguments

data	vector or matrix containing the gene expression data
edesign	matrix describing experimental design. Rows must be arrays and columns experiment descriptors
time	vector indicating time assignment for each array
groups	matrix indicating experimental group to which each array is assigned
repvect	index vector indicating experimental replicates
show.fit	logical indicating whether regression fit curves must be plotted
dis	regression design matrix
step.method	stepwise regression method to fit models for cluster mean profiles. It can be either "backward", "forward", "two.ways.backward" or "two.ways.forward"
min.obs	minimal number of observations for a gene to be included in the analysis
alfa	significance level used for variable selection in the stepwise regression

<code>nvar.correction</code>	argument for correcting stepwise regression significance level. See <a href="#">T.fit</a>
<code>summary.mode</code>	the method to condensate expression information when more than one gene is present in the data. Possible values are "representative" and "median"
<code>show.lines</code>	logical indicating whether a line must be drawn joining plotted data points for reach group
<code>groups.vector</code>	vector indicating experimental group to which each variable belongs
<code>xlab</code>	label for the x axis
<code>cex.xaxis</code>	graphical parameter magnification to be used for x axis in plotting functions
<code>ylim</code>	range of the y axis
<code>main</code>	plot main title
<code>cexlab</code>	graphical parameter magnification to be used for x axis label in plotting functions
<code>legend</code>	logical indicating whether legend must be added when plotting profiles
<code>sub</code>	plot subtitle

### Details

To compute experimental groups either a edesign object must be provided, or separate values must be given for the `time`, `repvect` and `groups` arguments.

When data is a matrix, the average expression value is displayed.

When there are array replicates in the data (as indicated by `repvect`), values are averaged by `repvect`.

PlotGroups plots one single expression profile for each experimental group even if there are more than one genes in the data set. The way data is condensated for this is given by `summary.mode`. When this argument takes the value "representative", the gene with the lowest distance to all genes in the cluster will be plotted. When the argument is "median", then median expression value is computed.

When `show.fit` is TRUE the stepwise regression fit for the data will be computed and the regression curves will be displayed.

If data is a matrix of genes and `summary.mode` is "median", the regression fit will be computed for the median expression value.

### Value

Plot of gene expression profiles by-group.

### Author(s)

Ana Conesa, <aconesa@cipf.es>; Maria Jose Nueda, <mj.nueda@ua.es>

### References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.



**See Also**[PlotProfiles](#)**Examples**

```
##### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{
  tc.dat <- NULL
  for (i in 1:n) {
    Ct1 <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ct1 group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ct1, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## create 10 genes with profile differences between Ct1, Tr2, and Tr3 groups
tc.DATA <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
rownames(tc.DATA) <- paste("gene", c(1:10), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")

##### CREATE EXPERIMENTAL DESIGN
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Ct1 <- c(rep(1, 9), rep(0, 27))
Tr1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Tr2 <- c(rep(0, 18), rep(1, 9), rep(0, 9))
Tr3 <- c(rep(0, 27), rep(1, 9))

PlotGroups (tc.DATA, time = Time, repvect = Replicates, groups = cbind(Ct1, Tr1, Tr2, Tr3))
```

**Description**

PlotProfiles displays the expression profiles of a group of genes.

**Usage**

```
PlotProfiles(data, cond, main = NULL, cex.xaxis = 0.5, ylim = NULL,
             repvect, sub = NULL, color.mode = "rainbow")
```

**Arguments**

data	a matrix containing the gene expression data
cond	vector for x axis labeling, typically array names
main	plot main title
cex.xaxis	graphical parameter magnification to be used for x axis in plotting functions
ylim	index vector indicating experimental replicates
repvect	index vector indicating experimental replicates
sub	plot subtitle
color.mode	color scale for plotting profiles. Can be either "rainblow" or "gray"

**Details**

The repvect argument is used to indicate with vertical lines groups of replicated arrays.

**Value**

Plot of experiment-wide gene expression profiles.

**Author(s)**

Ana Conesa, aconesa@cipf.es, Maria Jose Nueda, mj.nueda@ua.es

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

**See Also**

[PlotGroups](#)

**Examples**

```
#### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
                   var11 = 0.01, var12 = 0.01, var13 = 0.01,
```

```

var21 = 0.01, var22 = 0.01, var23 = 0.01,
var31 = 0.01, var32 = 0.01, var33 = 0.01,
var41 = 0.01, var42 = 0.01, var43 = 0.01,
a1 = 0, a2 = 0, a3 = 0, a4 = 0,
b1 = 0, b2 = 0, b3 = 0, b4 = 0,
c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{
  tc.dat <- NULL
  for (i in 1:n) {
    Ctl <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ctl group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ctl, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## create 10 genes with profile differences between Ctl, Tr2, and Tr3 groups
tc.DATA <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
rownames(tc.DATA) <- paste("gene", c(1:10), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")

PlotProfiles (tc.DATA, cond = colnames(tc.DATA), main = "Time Course",
              repvect = rep(c(1:12), each = 3))

```

---

position

*Column position of a variable in a data frame*

---

### Description

Finds the column position of a character variable in the column names of a data frame.

### Usage

```
position(matrix, vari)
```

### Arguments

matrix	matrix or data.frame with character column names
vari	character variable

### Value

numerical. Column position for the given variable.

**Author(s)**

Ana Conesa, aconesa@cipf.es

**Examples**

```
x <- matrix(c(1, 1, 2, 2, 3, 3), ncol = 3, nrow = 2)
colnames(x) <- c("one", "two", "three")
position(x, "one")
```

---

reg.coeffs

*Calculate true variables regression coefficients*

---

**Description**

reg.coeffs calculates back regression coefficients for true variables (experimental groups) from dummy variables regression coefficients.

**Usage**

```
reg.coeffs(coefficients, indepen = groups.vector[nchar(groups.vector)==min(nchar(groups.vector))][1]
           group)
```

**Arguments**

coefficients	vector of regression coefficients obtained from a regression model with dummy variables
indepen	independent variable of the regression formula
groups.vector	vector indicating the true variable of each variable in coefficients
group	true variable for which regression coefficients are to be computed

**Details**

regression coefficients in coefficients vector should be ordered by polynomial degree in a regression formula, ie: intercept,  $x$  term,  $x^2$  term,  $x^3$  term, and so on...

**Value**

reg.coeff vector of calculated regression coefficients

**Author(s)**

Ana Conesa, aconesa@cipf.es; Maria Jose Nueda, mj.nueda@ua.es

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

## Examples

```
groups.vector <-c("CT", "T1vsCT", "T2vsCT", "CT", "T1vsCT", "T2vsCT", "CT", "T1vsCT", "T2vsCT")
coefficients <- c(0.1, 1.2, -0.8, 1.7, 3.3, 0.4, 0.0, 2.1, -0.9)
## calculate true regression coefficients for variable "T1"
reg.coefs(coefficients, groups.vector = groups.vector, group = "T1")
```

---

see.genes

*Wrapper function for visualization of gene expression values of time course experiments*

---

## Description

This function provides visualisation tools for gene expression values in a time course experiment. The function first calls the heatmap function for a general overview of experiment results. Next a partitioning of the data is generated using a clustering method. The results of the clustering are visualized both as gene expression profiles extended along all arrays in the experiment, as provided by the plot.profiles function, and as summary expression profiles for comparison among experimental groups.

## Usage

```
see.genes(data, edesign = data$edesign, time.col = 1, repl.col = 2,
  group.cols = c(3:ncol(edesign)), names.groups = colnames(edesign)[3:ncol(edesign)],
  cluster.data = 1, groups.vector = data$groups.vector, k = 9, m = 1.45,
  cluster.method = "hclust", distance = "cor", agglom.method = "ward",
  show.fit = FALSE, dis = NULL, step.method = "backward", min.obs = 3,
  alfa = 0.05, nvar.correction = FALSE, show.lines = TRUE, iter.max = 500,
  summary.mode = "median", color.mode = "rainbow", cexlab = 1, legend = TRUE,
  newX11 = TRUE, ylim = NULL, main = NULL, ...)
```

## Arguments

data	either matrix or a list containing the gene expression data, typically a <a href="#">get.siggenes</a> object
edesign	matrix of experimental design
time.col	column in edesign containing time values. Default is first column
repl.col	column in edesign containing coding for replicates arrays. Default is second column
group.cols	columns indicating the coding for each group (treatment, tissue,...) in the experiment (see details)
names.groups	names for experimental groups
cluster.data	type of data used by the cluster algorithm (see details)
groups.vector	vector indicating the experimental group to which each variable belongs
k	number of clusters for data partitioning

<code>m</code>	<code>m</code> parameter when "mfuzz" clustering algorithm is used. See <a href="#">mfuzz</a>
<code>cluster.method</code>	clustering method for data partitioning. Currently "hclust", "kmeans" and "mfuzz" are supported
<code>distance</code>	distance measurement function for when <code>cluster.method</code> is <a href="#">hclust</a>
<code>aggllo.method</code>	aggregation method used when <code>cluster.method</code> is <a href="#">hclust</a>
<code>show.fit</code>	logical indicating whether regression fit curves must be plotted
<code>dis</code>	regression design matrix
<code>step.method</code>	stepwise regression method to fit models for cluster mean profiles. Can be either "backward", "forward", "two.ways.backward" or "two.ways.forward"
<code>min.obs</code>	minimal number of observations for a gene to be included in the analysis
<code>alfa</code>	significance level used for variable selection in the stepwise regression
<code>nvar.correction</code>	argument for correcting <code>T.fit</code> significance level. See <a href="#">T.fit</a>
<code>show.lines</code>	logical indicating whether a line must be drawn joining plotted data points for each group
<code>iter.max</code>	maximum number of iterations when <code>cluster.method</code> is <a href="#">kmeans</a>
<code>summary.mode</code>	the method <a href="#">PlotGroups</a> takes to condensate expression information when more than one gene is present in the data. Possible values are "representative" and "median"
<code>color.mode</code>	color scale for plotting profiles. Can be either "rainblow" or "gray"
<code>cexlab</code>	graphical parameter magnification to be used for x labels in plotting functions
<code>legend</code>	logical indicating whether legend must be added when plotting profiles
<code>main</code>	plot title
<code>ylim</code>	range of the y axis to be used by <a href="#">PlotProfiles</a> and <a href="#">PlotGroups</a>
<code>newX11</code>	when TRUE, plot each type of plot in a different graphical device
<code>...</code>	other graphical function argument

### Details

Data can be provided either as a single data matrix of expression values, or a [get.siggenes](#) object. In the later case the other argument of the fuction can be taken directly from data.

Data clustering can be done on the basis of either the original expression values, the regression coefficients, or the t.scores. In case data is a `get.siggenes` object, this is given by providing the element names of the list `c("sig.profiles", "coefficients", "t.score")` of their list position (1,2 or 3).

### Value

Experiment wide gene profiles and by group profiles plots are generated for each data cluster in the graphical device.

<code>cut</code>	vector indicating gene partitioning into clusters
<code>c.algo.used</code>	clustering algorithm used for data partitioning
<code>groups</code>	groups matrix used for plotting functions

**Author(s)**

Ana Conesa, aconesa@cipf.es; Maria Jose Nueda, mj.nueda@ua.es

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2006. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments. *Bioinformatics* 22, 1096-1102

**See Also**

[PlotProfiles](#), [PlotGroups](#)

**Examples**

```
##### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{
  tc.dat <- NULL
  for (i in 1:n) {
    Ctl <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ctl group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ctl, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## Create 270 flat profiles
flat <- tc.GENE(n = 270, r = 3)
## Create 10 genes with profile differences between Ctl and Tr1 groups
twodiff <- tc.GENE(n = 10, r = 3, b2 = 0.5, c2 = 1.3)
## Create 10 genes with profile differences between Ctl, Tr2, and Tr3 groups
threediff <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
## Create 10 genes with profile differences between Ctl and Tr2 and different variance
vardiff <- tc.GENE(n = 10, r = 3, a3 = 0.7, b3 = 1, c3 = 1.2, var32 = 0.03, var33 = 0.03)
## Create dataset
tc.DATA <- rbind(flat, twodiff, threediff, vardiff)
```

```

rownames(tc.DATA) <- paste("feature", c(1:300), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")
tc.DATA [sample(c(1:(300*36)), 300)] <- NA # introduce missing values

#### CREATE EXPERIMENTAL DESIGN
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")

see.genes(tc.DATA, edesign = edesign, k = 4, main = "Time Course")

# This will show the regression fit curve
dise <- make.design.matrix(edesign)
see.genes(tc.DATA, edesign = edesign, k = 4, main = "Time Course", show.fit = TRUE,
          dis = dise$dis, groups.vector = dise$groups.vector, distance = "euclidean")

```

---

stepback

*Fitting a linear model by backward-stepwise regression*


---

## Description

stepback fits a linear regression model applying a backward-stepwise strategy.

## Usage

```
stepback(y = y, d = d, alfa = 0.05, family = gaussian(), epsilon=0.00001 )
```

## Arguments

y	dependent variable
d	data frame containing by columns the set of variables that could be in the selected model
alfa	significance level to decide if a variable stays or not in the model
family	the distribution function to be used in the glm model
epsilon	argument to pass to <code>glm.control</code> , convergence tolerance in the iterative process to estimate de glm model

## Details

The strategy begins analysing a model with all the variables included in d. If all variables are statistically significant (all variables have a p-value less than alfa) this model will be the result. If not, the less statistically significant variable will be removed and the model is re-calculated. The process is repeated up to find a model with all the variables statistically significant.



**Value**

stepback returns an object of the class `lm`, where the model uses `y` as dependent variable and all the selected variables from `d` as independent variables.

The function `summary` are used to obtain a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

**Author(s)**

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es); Maria Jose Nueda, [mj.nueda@ua.es](mailto:mj.nueda@ua.es)

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. `maSigPro`: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

**See Also**

[lm](#), [step](#), [stepfor](#), [two.ways.stepback](#), [two.ways.stepfor](#)

**Examples**

```
## create design matrix
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")
dise <- make.design.matrix(edesign)
dis <- as.data.frame(dise$dis)

## expression vector
y <- c(0.082, 0.021, 0.010, 0.113, 0.013, 0.077, 0.068, 0.042, -0.056, -0.232, -0.014, -0.040,
-0.055, 0.150, -0.027, 0.064, -0.108, -0.220, 0.275, -0.130, 0.130, 1.018, 1.005, 0.931,
-1.009, -1.101, -1.014, -0.045, -0.110, -0.128, -0.643, -0.785, -1.077, -1.187, -1.249, -1.463)

s.fit <- stepback(y = y, d = dis)
summary(s.fit)
```

---

`stepfor`*Fitting a linear model by forward-stepwise regression*

---

**Description**

`stepfor` fits a linear regression model applying forward-stepwise strategy.

**Usage**

```
stepfor(y = y, d = d, alfa = 0.05, family = gaussian(), epsilon=0.00001 )
```

**Arguments**

<code>y</code>	dependent variable
<code>d</code>	data frame containing by columns the set of variables that could be in the selected model
<code>alfa</code>	significance level to decide if a variable stays or not in the model
<code>family</code>	the distribution function to be used in the glm model
<code>epsilon</code>	argument to pass to <code>glm.control</code> , convergence tolerance in the iterative process to estimate de glm model

**Details**

The strategy begins analysing all the possible models with only one of the variables included in `d`. The most statistically significant variable (with the lowest p-value) is included in the model and then it is considered to introduce in the model another variable analysing all the possible models with two variables (the selected variable in the previous step plus a new variable). Again the most statistically significant variable (with lowest p-value) is included in the model. The process is repeated till there are no more statistically significant variables to include.

**Value**

`stepfor` returns an object of the class `lm`, where the model uses `y` as dependent variable and all the selected variables from `d` as independent variables.

The function `summary` are used to obtain a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

**Author(s)**

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es); Maria Jose Nueda, [mj.nueda@ua.es](mailto:mj.nueda@ua.es)

**References**

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. `maSigPro`: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

**See Also**

[lm](#), [step](#), [stepback](#), [two.ways.stepback](#), [two.ways.stepfor](#)

**Examples**

```
## create design matrix
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")
dise <- make.design.matrix(edesign)
dis <- as.data.frame(dise$dis)

## expression vector
y <- c(0.082, 0.021, 0.010, 0.113, 0.013, 0.077, 0.068, 0.042, -0.056, -0.232, -0.014, -0.040,
-0.055, 0.150, -0.027, 0.064, -0.108, -0.220, 0.275, -0.130, 0.130, 1.018, 1.005, 0.931,
-1.009, -1.101, -1.014, -0.045, -0.110, -0.128, -0.643, -0.785, -1.077, -1.187, -1.249, -1.463)

s.fit <- stepfor(y = y, d = dis)
summary(s.fit)
```

---

suma2Venn

*Creates a Venn Diagram from a matrix of characters*

---

**Description**

suma2Venn transforms a matrix of characters into a binary matrix and creates a vennDiagram of the common elements between columns

**Usage**

```
suma2Venn(x, ...)
```

**Arguments**

x                    data frame of character values  
 ...                 plotting arguments for the vennDiagram function

**Details**

suma2Venn creates a list of all elements of a matrix or data frame of characters and computes the presence/absence of each element in each column of the matrix. This results is a numeric matrix of 1 and 0 which can be taken by the [vennDiagram](#) to generate a Venn Plot

**Value**

suma2Venn returns a Venn Plot such as that created by the [vennDiagram](#) function

**Author(s)**

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es)

**See Also**

[vennDiagram](#)

**Examples**

```
a <- c("a","b","c", "d", "e", NA, NA)
b <- c("a","b","f", NA, NA, NA, NA)
c <- c("b","e","f", "h", "i", "j", "k")
x <- cbind(a, b,c)
suma2Venn(x)
```

---

T.fit

*Makes a stepwise regression fit for time series gene expression experiments*

---

**Description**

T.fit selects the best regression model for each gene using stepwise regression.

**Usage**

```
T.fit(data, design = data$dis, step.method = "backward",
      min.obs = data$min.obs, alfa = data$Q, nvar.correction = FALSE, family = gaussian(), epsilon=0.001)
```

**Arguments**

data	can either be a <a href="#">p.vector</a> object or a matrix containing expression data with the same requirements as for the <a href="#">p.vector</a> function
design	design matrix for the regression fit such as that generated by the <a href="#">make.design.matrix</a> function. If data is a <a href="#">p.vector</a> object, the same design matrix is used by default
step.method	argument to be passed to the step function. Can be either "backward", "forward", "two.ways.backward" or "two.ways.forward"
min.obs	genes with less than this number of true numerical values will be excluded from the analysis
alfa	significance level used for variable selection in the stepwise regression
nvar.correction	argument for correcting T.fit significance level. See details

family	the distribution function to be used in the glm model. It must be the same used in p.vector
epsilon	argument to pass to glm.control, convergence tolerance in the iterative process to estimate de glm model

### Details

In the maSigPro approach `p.vector` and `T.fit` are subsequent steps, meaning that significant genes are first selected on the basis of a general model and then the significant variables for each gene are found by step-wise regression.

The step regression can be "backward" or "forward" indicating whether the step procedure starts from the model with all or none variables. With the "two.ways.backward" or "two.ways.forward" options the variables are both allowed to get in and out. At each step the p-value of each variable is computed and variables get in/out the model when this p-value is lower or higher than given threshold `alfa`. When `nva.correction` is TRUE the given significance level is corrected by the number of variables in the model

### Value

<code>sol</code>	matrix for summary results of the stepwise regression. For each selected gene the following values are given: <ul style="list-style-type: none"> <li>• p-value of the regression ANOVA</li> <li>• R-squared of the model</li> <li>• p-value of the regression coefficients of the selected variables</li> </ul>
<code>sig.profiles</code>	expression values for the genes contained in <code>sol</code>
<code>coefficients</code>	matrix containing regression coefficients for the adjusted models
<code>groups.coeffs</code>	matrix containing the coefficients of the implicit models of each experimental group
<code>variables</code>	variables in the complete regression model
<code>G</code>	total number of input genes
<code>g</code>	number of genes taken in the regression fit
<code>dat</code>	input analysis data matrix
<code>dis</code>	regression design matrix
<code>step.method</code>	imputed step method for stepwise regression
<code>edesign</code>	matrix of experimental design
<code>influ.info</code>	data frame of genes containing influential data

### Author(s)

Ana Conesa, <aconesa@cipf.es>; Maria Jose Nueda, <mj.nueda@ua.es>

### References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2006. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments. *Bioinformatics* 22, 1096-1102

**See Also**

[p.vector](#), [step](#)

**Examples**

```
##### GENERATE TIME COURSE DATA
## generate n random gene expression profiles of a data set with
## one control plus 3 treatments, 3 time points and r replicates per time point.

tc.GENE <- function(n, r,
  var11 = 0.01, var12 = 0.01, var13 = 0.01,
  var21 = 0.01, var22 = 0.01, var23 = 0.01,
  var31 = 0.01, var32 = 0.01, var33 = 0.01,
  var41 = 0.01, var42 = 0.01, var43 = 0.01,
  a1 = 0, a2 = 0, a3 = 0, a4 = 0,
  b1 = 0, b2 = 0, b3 = 0, b4 = 0,
  c1 = 0, c2 = 0, c3 = 0, c4 = 0)
{
  tc.dat <- NULL
  for (i in 1:n) {
    Ctl <- c(rnorm(r, a1, var11), rnorm(r, b1, var12), rnorm(r, c1, var13)) # Ctl group
    Tr1 <- c(rnorm(r, a2, var21), rnorm(r, b2, var22), rnorm(r, c2, var23)) # Tr1 group
    Tr2 <- c(rnorm(r, a3, var31), rnorm(r, b3, var32), rnorm(r, c3, var33)) # Tr2 group
    Tr3 <- c(rnorm(r, a4, var41), rnorm(r, b4, var42), rnorm(r, c4, var43)) # Tr3 group
    gene <- c(Ctl, Tr1, Tr2, Tr3)
    tc.dat <- rbind(tc.dat, gene)
  }
  tc.dat
}

## Create 270 flat profiles
flat <- tc.GENE(n = 270, r = 3)
## Create 10 genes with profile differences between Ctl and Tr1 groups
twodiff <- tc.GENE(n = 10, r = 3, b2 = 0.5, c2 = 1.3)
## Create 10 genes with profile differences between Ctl, Tr2, and Tr3 groups
threediff <- tc.GENE(n = 10, r = 3, b3 = 0.8, c3 = -1, a4 = -0.1, b4 = -0.8, c4 = -1.2)
## Create 10 genes with profile differences between Ctl and Tr2 and different variance
vardiff <- tc.GENE(n = 10, r = 3, a3 = 0.7, b3 = 1, c3 = 1.2, var32 = 0.03, var33 = 0.03)
## Create dataset
tc.DATA <- rbind(flat, twodiff, threediff, vardiff)
rownames(tc.DATA) <- paste("feature", c(1:300), sep = "")
colnames(tc.DATA) <- paste("Array", c(1:36), sep = "")
tc.DATA[sample(c(1:(300*36)), 300)] <- NA # introduce missing values

##### CREATE EXPERIMENTAL DESIGN
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0, 9))
```

```

Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")

## run T.fit from a p.vector object
tc.p <- p.vector(tc.DATA, design = make.design.matrix(edesign), Q = 0.01)
tc.tstep <- T.fit(data = tc.p , alfa = 0.05)

## run T.fit from a data matrix and a design matrix
dise <- make.design.matrix(edesign)
tc.tstep <- T.fit (data = tc.DATA[271:300,], design = dise$dis,
                 step.method = "two.ways.backward", min.obs = 10, alfa = 0.05)
tc.tstep$sol # gives the p.values of the significant
             # regression coefficients of the optimized models

```

---

two.ways.stepback	<i>Fitting a linear model by backward-stepwise regression</i>
-------------------	---

---

## Description

two.ways.stepback fits a linear regression model applying backward-stepwise strategy.

## Usage

```
two.ways.stepback(y = y, d = d, alfa = 0.05, family = gaussian(), epsilon=0.00001)
```

## Arguments

y	dependent variable
d	data frame containing by columns the set of variables that could be in the selected model
alfa	significance level to decide if a variable stays or not in the model
family	the distribution function to be used in the glm model
epsilon	argument to pass to glm.control, convergence tolerance in the iterative process to estimate de glm model

## Details

The strategy begins analysing a model with all the variables included in d. If all the variables are statistically significant (all the variables have a p-value less than alfa) this model will be the result. If not, the less statistically significant variable will be removed and the model is re-calculated. The process is repeated up to find a model with all the variables statistically significant (p-value < alpha). Each time that a variable is removed from the model, it is considered the possibility of one or more removed variables to come in again.

## Value

`two.ways.stepback` returns an object of the class `lm`, where the model uses `y` as dependent variable and all the selected variables from `d` as independent variables.

The function `summary` are used to obtain a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

## Author(s)

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es); Maria Jose Nueda, [mj.nueda@ua.es](mailto:mj.nueda@ua.es)

## References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. `maSigPro`: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

## See Also

[lm](#), [step](#), [stepfor](#), [stepback](#), [two.ways.stepfor](#)

## Examples

```
## create design matrix
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")
dise <- make.design.matrix(edesign)
dis <- as.data.frame(dise$dis)

## expression vector
y <- c(0.082, 0.021, 0.010, 0.113, 0.013, 0.077, 0.068, 0.042, -0.056, -0.232, -0.014, -0.040,
-0.055, 0.150, -0.027, 0.064, -0.108, -0.220, 0.275, -0.130, 0.130, 1.018, 1.005, 0.931,
-1.009, -1.101, -1.014, -0.045, -0.110, -0.128, -0.643, -0.785, -1.077, -1.187, -1.249, -1.463)

s.fit <- two.ways.stepback(y = y, d = dis)
summary(s.fit)
```



---

two.ways.stepfor	<i>Fitting a linear model by forward-stepwise regression</i>
------------------	--

---

## Description

`two.ways.stepfor` fits a linear regression model applying forward-stepwise strategy.

## Usage

```
two.ways.stepfor(y = y, d = d, alfa = 0.05, family = gaussian(), epsilon=0.00001 )
```

## Arguments

<code>y</code>	dependent variable
<code>d</code>	data frame containing by columns the set of variables that could be in the selected model
<code>alfa</code>	significance level to decide if a variable stays or not in the model
<code>family</code>	the distribution function to be used in the glm model
<code>epsilon</code>	argument to pass to <code>glm.control</code> , convergence tolerance in the iterative process to estimate de glm model

## Details

The strategy begins analysing all the possible models with only one of the variables included in `d`. The most statistically significant variable (with the lowest p-value) is included in the model and then it is considered to introduce in the model another variable analysing all the possible models with two variables (the selected variable in the previous step plus a new variable). Again the most statistically significant variable (with lowest p-value) is included in the model. The process is repeated till there are no more statistically significant variables to include. Each time that a variable enters the model, the p-values of the current model variables is recalculated and non significant variables will be removed.

## Value

`two.ways.stepfor` returns an object of the class `lm`, where the model uses `y` as dependent variable and all the selected variables from `d` as independent variables.

The function `summary` are used to obtain a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

## Author(s)

Ana Conesa, [aconesa@cipf.es](mailto:aconesa@cipf.es); Maria Jose Nueda, [mj.nueda@ua.es](mailto:mj.nueda@ua.es)

## References

Conesa, A., Nueda M.J., Alberto Ferrer, A., Talon, T. 2005. maSigPro: a Method to Identify Significant Differential Expression Profiles in Time-Course Microarray Experiments.

## See Also

[lm](#), [step](#), [stepback](#), [stepfor](#), [two.ways.stepback](#)

## Examples

```
## create design matrix
Time <- rep(c(rep(c(1:3), each = 3)), 4)
Replicates <- rep(c(1:12), each = 3)
Control <- c(rep(1, 9), rep(0, 27))
Treat1 <- c(rep(0, 9), rep(1, 9), rep(0, 18))
Treat2 <- c(rep(0, 18), rep(1, 9), rep(0,9))
Treat3 <- c(rep(0, 27), rep(1, 9))
edesign <- cbind(Time, Replicates, Control, Treat1, Treat2, Treat3)
rownames(edesign) <- paste("Array", c(1:36), sep = "")
dise <- make.design.matrix(edesign)
dis <- as.data.frame(dise$dis)

## expression vector
y <- c(0.082, 0.021, 0.010, 0.113, 0.013, 0.077, 0.068, 0.042, -0.056, -0.232, -0.014, -0.040,
-0.055, 0.150, -0.027, 0.064, -0.108, -0.220, 0.275, -0.130, 0.130, 1.018, 1.005, 0.931,
-1.009, -1.101, -1.014, -0.045, -0.110, -0.128, -0.643, -0.785, -1.077, -1.187, -1.249, -1.463)

s.fit <- two.ways.stepfor(y = y, d = dis)
summary(s.fit)
```

# Index

- \*Topic **UsersGuide**
  - maSigProUsersGuide, 18
- \*Topic **aplot**
  - PlotGroups, 23
  - PlotProfiles, 25
  - see.genes, 29
  - suma2Venn, 35
- \*Topic **arith**
  - average.rows, 2
  - i.rank, 11
  - position, 27
- \*Topic **datasets**
  - data.abiotic, 3
  - edesign.abiotic, 5
  - edesign.OD, 6
  - edesignCT, 6
  - edesignDR, 7
  - NBdata, 19
  - NBdesign, 20
- \*Topic **design**
  - make.design.matrix, 12
- \*Topic **manip**
  - get.siggenes, 8
  - maSigPro, 13
  - see.genes, 29
- \*Topic **misc**
  - reg.coeffs, 28
  - suma2Venn, 35
- \*Topic **models**
  - maSigPro, 13
  - T.fit, 36
- \*Topic **regression**
  - make.design.matrix, 12
  - p.vector, 20
  - stepback, 32
  - stepfor, 34
  - T.fit, 36
  - two.ways.stepback, 39
  - two.ways.stepfor, 41
- average.rows, 2, 9, 15
- coefficients, 33, 34, 40, 41
- data.abiotic, 3
- edesign.abiotic, 5
- edesign.OD, 6
- edesignCT, 6
- edesignDR, 7
- effects, 33, 34, 40, 41
- fitted.values, 33, 34, 40, 41
- get.siggenes, 8, 13–17, 29, 30
- hclust, 30
- i.rank, 11
- kmeans, 30
- lm, 22, 33–35, 40–42
- make.design.matrix, 12, 13, 14, 16, 17, 21, 36
- maSigPro, 13
- maSigProUsersGuide, 18
- mfuzz, 15, 30
- NBdata, 19
- NBdesign, 20
- order, 12
- p.vector, 13, 14, 16, 17, 20, 36–38
- PlotGroups, 15, 23, 26, 30, 31
- PlotProfiles, 15, 25, 25, 30, 31
- position, 27
- rank, 12
- reg.coeffs, 28

residuals, [33](#), [34](#), [40](#), [41](#)

see.genes, [13–17](#), [29](#)

step, [33](#), [35](#), [38](#), [40](#), [42](#)

stepback, [32](#), [35](#), [40](#), [42](#)

stepfor, [33](#), [34](#), [40](#), [42](#)

suma2Venn, [35](#)

summary, [33](#), [34](#), [40](#), [41](#)

T.fit, [13](#), [14](#), [17](#), [22](#), [24](#), [30](#), [36](#), [37](#)

two.ways.stepback, [33](#), [35](#), [39](#), [42](#)

two.ways.stepfor, [33](#), [35](#), [40](#), [41](#)

vennDiagram, [35](#), [36](#)