Category

November 11, 2009

R topics documented:

applyByCategory . . . 2
cateGOry . . . 3
categoryToEntrezBuilder . . . 4
cb_contingency . . . 5
cb_parse_band_Hs . . . 6
cb_parse_band_Mm . . . 7
cb_test . . . 7
ChrBandTree-class . . . 9
ChrMapHyperGParams-class . . . 10
ChrMapHyperGResult-class . . . 11
ChrMapLinearMParams-class . . . 12
ChrMapLinearMResult-class . . . 14
DatPkg-class . . . 15
effectSize . . . 16
exampleLevels . . . 16
findAMstats . . . 17
geneGoHyperGeoTest . . . 18
geneKeggHyperGeoTest . . . 19
getPathNames . . . 20
GOHyperGParams-class . . . 21
gseattperm . . . 22
HyperGParams-class . . . 23
HyperGResult-accessors . . . 24
HyperGResultBase-class . . . 27
HyperGResult-class . . . 28
hyperGTest . . . 29
KEGGHyperGParams-class . . . 30
LinearMParams-class . . . 31
LinearMResultBase-class . . . 32
LinearMResult-class . . . 33
linearMTest . . . 34
local_test_factory . . . 35
makeChrBandGraph . . . 36
makeEBcontr . . . 37
makeValidParams . . . 38
MAPAmat . . . 38
NewChrBandTree . . . 39

probes2MAP 40
probes2Path 41
tree_visitor 42
ttperm 43
universeBuilder 44

Index **45**

`applyByCategory` *Apply a function to a vector of statistics, by category*

Description

For each category, apply the function `FUN` to the set of values of `stats` belonging to that category.

Usage

```
applyByCategory(stats, Amat, FUN = mean, ...)
```

Arguments

`stats`	Numeric vector with test statistics of interest.
`Amat`	A logical matrix: the adjacency matrix of the bipartite genes - category graph. Its rows correspond to the categories, columns to the genes, and `TRUE` indicates membership. The columns are assumed to be aligned with the elements of `stats`.
`FUN`	A function to apply to the subsets `stats` by categories.
`...`	Extra parameters passed to `FUN`.

Details

For GO categories, the function `cateGOry` might be useful for the construction of `Amat`.

Value

The return value is a list or vector of length equal to the number of categories. Each element corresponds to the values obtained by applying `FUN` to the subset of values in `stats` according to the category defined for that row.

Author(s)

R. Gentleman, contributions from W. Huber

See Also

`apply`

Examples

```
set.seed(0xabcd)
st = rnorm(20)
names(st) = paste("gene", 1:20)

a = matrix(sample(c(FALSE, TRUE), 60, replace=TRUE), nrow=3,
       dimnames = list(paste("category", LETTERS[1:3]), names(st)))

applyByCategory(st, a, median)
```

`cateGOry` *Construct a category membership matrix from a list of gene identifiers and their annotated GO categories.*

Description

The function constructs a category membership matrix, such as used by `applyByCategory`, from a list of gene identifiers and their annotated GO categories. For each of the GO categories stated in `categ`, all less specific terms (ancestors) are also included, thus one need only obtain the most specific set of GO term mappings, which can be obtained from Bioconductor annotation packages or via **biomaRt**. The ancestor relationships are obtained from the **GO** package.

Usage

```
cateGOry(x, categ, sparse=FALSE)
```

Arguments

`x`	Character vector with (arbitrary) gene identifiers. They will be used for the column names of the resulting matrix.
`categ`	A character vector of the same length as `x` with GO annotations for the genes in `x`. If a gene has multiple GO annotations, it is expected to occur multiple times in `x`, once for each different annotation.
`sparse`	Logical. Currently, this is ignored. This argument might be used in future versions of the function to result in returning a sparse matrix representation.

Details

Requires the `GO` package.

For some subsequent analyses, it is useful to remove categories that have only a small number of members. Use the normal matrix subsetting syntax for this, see example.

If a GO category in `categ` is not found in the GO annotation package, a warning will be generated, and no ancestors for that GO category are added (but that category itself will be part of the returned adjacency matrix).

Value

The adjacency matrix of the bipartite category membership graph, rows are categories and columns genes.

Author(s)

W. Huber

See Also

applyByCategory

Examples

```
g = cateGOry(c("CG2671", "CG2671", "CG2950"),
             c("GO:0000074", "GO:0001738", "GO:0003676"))
g

rownames(g)
colnames(g)
rowSums(g)   ## number of genes in each category

## Filter out categories with less than minMem members.
## This is toy data, in real applications, a number higher
## than 2 will be more appropriate.
minMemb = 2
g[rowSums(g)>=minMemb,,drop=FALSE ]
```

---

categoryToEntrezBuilder

Return a list mapping category ids to Entrez Gene ids

Description

Return a list mapping category ids to the Entrez Gene ids annotated at the category id. Only those category ids that have at least one annotation in the set of Entrez Gene ids specified by the `geneIds` slot of `p` are included.

Usage

```
categoryToEntrezBuilder(p)
```

Arguments

`p`	A subclass of `HyperGParams-class`

Details

End users **should not** call this directly. This method gets called from `hyperGTest`. To add support for a new category, a new method for this generic must be defined. Its signature should match a subclass of `HyperGParams-class` appropriate for the new category.

Value

A list mapping category ids to Entrez Gene identifiers.

Author(s)

S. Falcon

See Also

hyperGTest HyperGParams-class

---

cb_contingency *Create and Test Contingency Tables of Chromosome Band Annotations*

Description

For each chromosome band identifier in chrVect, cb_contingency builds and performs a test on a 2 x k contingency table for the genes from selids found in the child bands of the given chrVect element.

cb_sigBands extracts the chromosome band identifiers that were in a contingency table that tested significant given the specified p-value cutoff.

cb_children returns the child bands of a given band in the chromosome band graph. The argument must have length equal to one.

Usage

```
cb_contingency(selids, chrVect, chrGraph, testFun = chisq.test,
               min.expected = 5L, min.k = 1L)

cb_sigBands(b, p.value = 0.01)

cb_children(n, chrGraph)
```

Arguments

selids	A vector of the selected gene identifiers (usual Entrez IDs).
chrVect	A character vector of chromosome band identifiers
chrGraph	A graph object as returned by makeChrBandGraph. The nodes should be chromosome band IDs and the edges should represent the tree structure of the bands. Furthermore, the graph is expected to have a "geneIds" node attribute providing a vector of gene IDs annotated at each band.
testFun	The function to use for testing the 2 x k contingency tables. The default is chisq.test. It will be called with a single argument, a 2 x k matrix representing the contingency table.
min.expected	A numeric value specifying the minimum expected count for columns to be included in the contingency table. The expected count is (rowSum * colSum) / n. Chromosome bands with a select cell count less than min.expected are dropped from the table before testing occurs. If NULL, then no bands will be dropped.
min.k	An integer giving the minimum number of chromosome bands that must be present in a contingency table in order to proceed with testing.
b	A list as returned by cb_contingency

`p.value`	A p-value cutoff to use in selecting significant contingency tables.
`n`	A length one character vector specifying a chromosome band annotation. Bands not found in `chrGraph` will return `character(0)` when passed to `cb_children`.

Details

`cb_sigBands` assumes that the p-value associated with a result of `testFun` can by accessed as `testFun(t)$p.value`. We should improve this to be a method call which can then be specialized based on the class of the object returned by `testFun`.

Value

`cb_contingency` returns a list with an element for each test performed. This will most often be shorter than `length(chrVect)` due to skipped tests based on `min.found` and `min.k`. Each element of the returned list is itself a list with components:

`table`	A 2 x k contingency table
`result`	The output of `testFun` applied to the `table`.

`cb_sigBands` returns a character vector of chromosome band identifiers that are in one of the contingency tables that had a p-value less than the cutoff specified by `p.value`.

Author(s)

Seth Falcon

`cb_parse_band_Hs` *Parse Homo Sapiens Chromosome Band Annotations*

Description

This function parses chromosome band annotations as found in the <chip>MAP map of Bioconductor annotation data packages. The return value is a vector of parent bands up to the relevant chromosome.

Usage

```
cb_parse_band_Hs(x)
```

Arguments

`x`	A chromosome band annotation given as a string.

Details

The former function cb_parse_band_hsa is now deprecated.

Value

A character vector giving the path to the relevant chromosome.

Author(s)

Seth Falcon

Examples

```
cb_parse_band_Hs("12q32.12")
```

---

cb_parse_band_Mm *Parse Mus Musculus Chromosome Band Annotations*

Description

This function parses chromosome band annotations as found in the <chip>MAP map of Bioconductor annotation data packages. The return value is a vector of parent bands up to the relevant chromosome.

Usage

```
cb_parse_band_Mm(x)
```

Arguments

x	A chromosome band annotation given as a string.

Value

A character vector giving the path to the relevant chromosome.

Author(s)

Seth Falcon & Nolwenn Le Meur

Examples

```
cb_parse_band_Mm("10 B3")
```

---

cb_test *Chromosome Band Tree-Based Hypothesis Testing*

Description

cb_test is a flexible tool for discovering interesting chromosome bands relative to a selected gene list. The function supports local and global tests which can be carried out in a top down or bottom up fashion on the tree of chromosome bands.

Usage

```
cb_test(selids, chrtree, level, dir = c("up", "down"),
       type = c("local", "global"), next.pval = 0.05,
       cond.pval = 0.05, conditional = FALSE)
```

Arguments

- `selids` — A vector of gene IDs. The IDs should match those used to annotatate the `ChrBandTree` given by `chrtree`. In most cases, these will be Entrez Gene IDs.
- `chrtree` — A `ChrBandTree` object representing the chromosome bands and the mapping to gene identifiers. The genes in the `ChrBandTree` are limited to the universe of gene IDs specified at object creation time.
- `level` — An integer giving the level of the chromosome band tree at which testing should begin. The level is conceptualized as the set of nodes with a given path length to the root (organism) node of the chromosome band tree. So level 1 is the chromosome and level 2 is the chromosome arms. You can get a better sense by calling `exampleLevels(chrtree)`
- `dir` — A string giving the direction in which the chromosome band tree will be traversed when carrying out the tests. A bottom up traversal, from leaves to root, is specified by `"up"`. A top down, from root to leaves, traversal is specified by `"down"`.
- `type` — A string giving the type of test to perform. The current choices are `"local"` and `"global"`. A local test carries out a chisq.test on each 2 x K contingency table induced by each set of siblings at a given level in the tree. A global test uses the Hypergeometric distribution to compute a p-value for the 2 x 2 tables induced by each band treated independently.
- `next.pval` — The p-value cutoff used to determine whether the parents or children of a node should be tested. After testing a given level of the tree, the decision of whether or not to continue testing the children (or parents) of the already tested nodes is made by comparing the p-value result for a given node with this cutoff; relatives of nodes with values strictly greater than the cutoff are skipped.
- `cond.pval` — The p-value cutoff used to determine whether a node is significant during a conditional test. See `conditional`.
- `conditional` — A logical value. Can only be used when `dir="up"` and `type="global"`. In this case, a `TRUE` value causes a conditional Hypergeometric calculation to be performed. The genes annotated at significant children of a given band are removed before testing.

Value

A list with an element for each level of the tree that was tested. Note that the first element will correspond to the level given by `level` and that subsequent elements will be the next or previous depending on `dir`.

Each level element is itself a list consisting of a result list for each node or set of nodes tested. These inner-most lists will have, at least, the following components:

- `nodes` — A character vector of the nodes involved in the test.
- `p.value` — The p-value for the test
- `observed` — The contingency table
- `method` — A brief description of the test method

Author(s)

Seth Falcon

ChrBandTree-class *Class "ChrBandTree"*

Description

This class represents chromosome band annotation data for a given experiment. The class is responsible for storing the mapping of band to set of gene IDs located within that band as well as for representing the tree structured relationship among the bands.

Objects from the Class

Objects should be created using `NewChrBandTree` or `ChrBandTreeFromGraph`.

Slots

`toParentGraph`: Object of class `"graph"` representing the tree of chromosome bands. Edges in this directed graph go from child to parent.

`toChildGraph`: Object of class `"graph"`. This is the same as `toParentGraph`, but with the edge directons reversed. This is not an ideal implementation due to the duplication of data, but it provides quick access to parents or children of a given node.

`root`: Object of class `"character"` giving the name of the root node. The convention is to use "ORGANISM:<organism>".

`level2nodes`: Object of class `"list"` providing a mapping of levels in the tree to the set of nodes at that level. Levels `X` is defined as the set of nodes with a path length of `X` from the root node.

Methods

allGeneIds Return a vector of gene IDs representing the gene universe for this `ChrBandTree`

childrenOf Return a list with an element for each the character vector `n`. Each element is a character vector of node names of the children of the named element.

geneIds Return a vector of gene IDs for a single band.

lgeneIds Return a list of vectors of gene IDs when given more than one band. The "l" prefix is for list.

parentOf Return the parents of the specified bands. See `childrenOf` for a description of the structure of the return value.

treeLevels Return an integer vector identifying the levels of the tree.

level2nodes(g, level) Return the nodes in the tree that are at the level specified by `level`. The `level` argument can be either numeric or character, but should match a level returned by `treeLevels`.

Note

Not all known chromosome bands will be represented in a given instance. The set of bands that will be present is determined by the available annotation data and the specified gene universe. The annotation source maps genes to their most specific band. Such bands and all bands on the path to the root will be represented in the resulting tree.

Currently there is only support for human and mouse data.

Author(s)

S. Falcon

Examples

```
library("hgu95av2.db")
set.seed(0xfeee)
univ = NULL ## use all Entrez Gene IDs on the chip (not recommended)
ct = NewChrBandTree("hgu95av2.db", univ)

length(allGeneIds(ct))

exampleLevels(ct)

geneIds(ct, "10p11")
lgeneIds(ct, "10p11")
lgeneIds(ct, c("10p11", "Yq11.22"))

pp = parentOf(ct, c("10p11", "Yq11.22"))
childrenOf(ct, unlist(pp))

treeLevels(ct)

level2nodes(ct, 0)
level2nodes(ct, 0L)
level2nodes(ct, "0")

level2nodes(ct, 1)
```

ChrMapHyperGParams-class

Class "ChrMapHyperGParams"

Description

This class encapsulates parameters needed for Hypergeometric testing of over or under representation of chromosome bands among a selected gene list using `hyperGTest`.

Objects from the Class

Objects can be created by calls of the form `new("ChrMapHyperGParams", ...)`.

Slots

`chrGraph`: Object of class `"graph"`. The nodes are the chromosome bands and the edges describe the tree structure of the bands. Each node has a "geneIds" node attributes (see `nodeData`) which contains a vector of gene IDs annotated at the given band.

`conditional`: Object of class `"logical"`, indicating whether the test performed should be a conditional test.

`geneIds`: Object of class `"ANY"`: A vector of gene identifiers. Numeric and character vectors are probably the only things that make sense. These are the gene ids for the selected gene set.

universeGeneIds: Object of class "ANY": A vector of gene ids in the same format as geneIds defining a subset of the gene ids on the chip that will be used as the universe for the hypergeometric calculation. If this is NULL or has length zero, then all gene ids on the chip will be used.

annotation: A string giving the name of the annotation data package for the chip used to generate the data.

cateogrySubsetIds: Object of class "ANY": If the test method supports it, can be used to specify a subset of category ids to include in the test instead of all possible category ids.

categoryName: A string describing the category. Usually set automatically by subclasses. For example "GO".

pvalueCutoff: The p-value to use as a cutoff for significance for testing methods that require it. This value will also be passed on to the result instance and used for display and counting of significant results. The default is 0.01.

testDirection: A string indicating whether the test should be for overrepresentation ("over") or underrepresentation ("under").

datPkg: Object of class "DatPkg" used to assist with dispatch based on type of annotation data available.

Extends

Class "HyperGParams", directly.

Methods

No methods defined with class "ChrMapHyperGParams" in the signature.

Author(s)

Seth Falcon

Examples

```
showClass("ChrMapHyperGParams")
```

ChrMapHyperGResult-class

Class "ChrMapHyperGResult"

Description

This class represents the results of a Hypergeometric test for over-representation of genes in a selected gene list in the chromosome band annotation. The hyperGTest function returns an instance of ChrMapHyperGResult when given a parameter object of class ChrMapHyperGParams. For details on accessing the results, see HyperGResult-accessors.

Objects from the Class

Objects can be created by calls of the form new("ChrMapHyperGResult", ...).

Slots

pvalue.order: Object of class "integer" that gives the order of the p-values.

conditional: Object of class "logical" is a flag indicating whether or not this result is from a conditional analysis.

chrGraph: Object of class "graph". The nodes are the chromosome bands with edges representing the tree structure of the bands. Each node has a "geneIds" attribute that gives the gene IDs annotated at that band.

annotation: A string giving the name of the chip annotation data package used.

geneIds: Object of class "ANY": the input vector of gene identifiers intersected with the universe of gene identifiers used in the computation. The class of this slot is specified as "ANY" because gene IDs may be integer or character vectors depending on the annotation package.

testName: A string identifying the testing method used.

pvalueCutoff: Numeric value used a a p-value cutoff. Used by the show method to count number of significant terms.

testDirection: Object of class "character" indicating whether the test was for over-representation ("over") or under-representation ("under").

Extends

Class "HyperGResultBase", directly.

Methods

See HyperGResult-accessors.

Author(s)

Seth Falcon

Examples

```
showClass("ChrMapHyperGResult")
## For details on accessing the results:
##     help("HyperGResult-accessors")
```

ChrMapLinearMParams-class

Class "ChrMapLinearMParams"

Description

This class encapsulates parameters needed for testing systematic variations in some gene-level statistic by chromosome bands using linearMTest.

Objects from the Class

Objects can be created by calls of the form new("ChrMapLinearMParams", ...).

Slots

chrGraph: Object of class "graph". The nodes are the chromosome bands and the edges describe the tree structure of the bands. Each node has a "geneIds" node attributes (see nodeData) which contains a vector of gene IDs annotated at the given band.

conditional: Object of class "logical", indicating whether the test performed should be a conditional test.

geneStats: Named vector of class "numeric", giving the gene-level statistics to be used in the tests.

universeGeneIds: Object of class "ANY": A vector of gene ids defining a subset of the gene ids on the chip that will be used as the universe for the hypergeometric calculation. If this is NULL or has length zero, then all gene ids on the chip will be used.

annotation: A string giving the name of the annotation data package for the chip used to generate the data.

datPkg: Object of class "DatPkg" used to assist with dispatch based on type of annotation data available.

cateogrySubsetIds: Object of class "ANY": If the test method supports it, can be used to specify a subset of category ids to include in the test instead of all possible category ids.

categoryName: A string describing the category. Usually set automatically by subclasses. For example "GO".

pvalueCutoff: The p-value to use as a cutoff for significance for testing methods that require it. This value will also be passed on to the result instance and used for display and counting of significant results. The default is 0.01.

minSize: An integer giving a minimum size for a gene set for it to be tested. The default is 5.

testDirection: A string indicating whether the test should test for systematic increase ("up") or decrease ("down") in the geneStats values within a gene set relative to the remaining genes.

Extends

Class "LinearMParams", directly.

Methods

conditional signature(r = "ChrMapLinearMParams"): Accessor for the conditional slot.

conditional<- signature(r = "ChrMapLinearMParams", value = "logical"): Replacement method for the conditional slot.

Author(s)

Deepayan Sarkar

See Also

linearMTest

Examples

```
showClass("ChrMapLinearMParams")
```

ChrMapLinearMResult-class

Class "ChrMapLinearMResult"

Description

This class represents the results of a linear model-based test for systematic changes in a per-gene statistic by chromosome band annotation. The `linearMTest` function returns an instance of `ChrMapLinearMResult` when given a parameter object of class `ChrMapLinearMParams`. Most slots can be queried using accessors.

Objects from the Class

Objects can be created by calls of the form `new("ChrMapLinearMResult", ...)`, but is more commonly created by callinf `linearMTest`

Slots

`pvalue.order`: Object of class `"integer"` ~~

`conditional`: Object of class `"logical"` ~~

`chrGraph`: Object of class `"graph"` ~~

`pvalues`: Object of class `"numeric"` ~~

`effectSize`: Object of class `"numeric"` ~~

`catToGeneId`: Object of class `"list"` ~~

`annotation`: Object of class `"character"` ~~

`geneIds`: Object of class `"ANY"` ~~

`testName`: Object of class `"character"` ~~

`pvalueCutoff`: Object of class `"numeric"` ~~

`minSize`: Object of class `"integer"` ~~

`testDirection`: Object of class `"character"` ~~

Extends

Class `"LinearMResult"`, directly.

Class `"LinearMResultBase"`, by class "LinearMResult", distance 2.

Methods

chrGraph `signature(r = "ChrMapLinearMResult")`: ...

condGeneIdUniverse `signature(r = "ChrMapLinearMResult")`: ...

isConditional `signature(r = "ChrMapLinearMResult")`: ...

Author(s)

Deepayan Sarkar

See Also

linearMTest, ChrMapLinearMParams, LinearMResult, LinearMResultBase,

Examples

```
showClass("ChrMapLinearMResult")
```

DatPkg-class *Class "DatPkg"*

Description

DatPkg is a VIRTUAL class for representing annotation data packages.

AffyDatPkg is a subclass of DatPkg used to represent standard annotation data packages that follow the format of Affymetrix expression array annotation.

YeastDatPkg is a subclass of DatPkg used to represent the annotation data packages for yeast. The yeast chip packages are based on sgd and are internally different from the AffyDataPkg conforming packages.

Org.XX.egDatPkg is a subclass of DatPkg used to represent the org.*.eg.db organism-level Entez Gene based annotation data packages.

Objects from the Class

A virtual Class: No objects may be created from it.

Given the name of an annotation data package, DatPkgFactory can be used to create an appropriate DatPkg subclass.

Slots

name A string giving the name of the annotation data package.

Methods

See showMethods(classes="DatPkg"). The set of methods, ID2EntreizID map between the standard IDs for an organism, or Chip and EntrezIDs, typically to give a way to get the GO terms. Different organisms, such as S. cerevisae and A. thaliana have their own internal IDs, so we need specialized methods for them.

Author(s)

Seth Falcon

Examples

```
DatPkgFactory("hgu95av2")
## Not run:
DatPkgFactory("org.Sc.sgd")
DatPkgFactory("org.Hs.eg.db")
DatPkgFactory("ag")
## End(Not run)
```

effectSize — *Extract estimated effect sizes*

Description

This function extracts estimated effect sizes from the results of a linear model-based gene-set / category enrichment test.

Usage

```
effectSize(r)
```

Arguments

r	The results of the test

Value

A numeric vector.

Author(s)

Deepayan Sarkar

See Also

linkS4class{LinearMResult}

exampleLevels — *Display a sample node from each level of a ChrBandTree object*

Description

The "levels" of a chromosome band tree represented by a ChrBandTree object are the sets of nodes with a given path length to the root node. This function displays the available levels along with an example node from each level.

Usage

```
exampleLevels(g)
```

Arguments

g	A ChrBandTree object

Value

A list with an element for each level. The names of the list are the levels. Each element is an example of a node from the given level.

Author(s)

S. Falcon

`findAMstats` *Compute per category summary statistics*

Description

For a given incidence matrix, `Amat`, compute some per category statistics.

Usage

```
findAMstats(Amat, tstats)
```

Arguments

`Amat`	An incidence matrix, with categories as the rows and probes as the columns.
`tstats`	A vector of per probe test statistics (should be the same length as `ncol(Amat)`.

Details

Simple summary statistics are computed, such as the row sums and the vector of per category sums of the test statistics, `tstats`.

Value

A list with components,

`eDE`	per category sums of the test statistics
`lens`	row sums of `Amat`

Author(s)

R. Gentleman

See Also

`applyByCategory`

Examples

```
ts = rnorm(100)
Am = matrix(sample(c(0,1), 1000, replace=TRUE), ncol=100)
findAMstats(Am, ts)
```

geneGoHyperGeoTest *Hypergeometric Tests for GO (DEFUNCT)*

Description

This function is defunct. Use `hyperGTest` instead.

Usage

```
geneGoHyperGeoTest(entrezGeneIds, lib, ontology, universe=NULL)
```

Arguments

`entrezGeneIds`	A vector of Entrez Gene Identifiers
`lib`	A string giving the name of the annotation data package to use. This must correspond to the microarray chip type that the data came from
`ontology`	One of "BP", "CC", or "MF" used to determine which GO ontology to use.
`universe`	A character vector of unique Entrez Gene identifiers. This is the population (the urn) of the Hypergeometric test. When `NULL` (default), the population is all Entrez Gene ids in the annotation package that have a GO term annotation in the specified GO category (see details).

Details

The Entrez Gene ids given in `entrezGeneIds` define the selected set of genes. The universe of Entrez Gene ids is determined by the chip annotation data package (`lib`) or specified by the `universe` argument which must be a subset of the Entrez Gene ids represented on the chip. Both the selected genes and the universe are reduced by removing Entrez Gene ids that do not have any annotations in the specified GO category.

For each GO term in the specified category that has at least one annotation in the selected gene set (`entrezGeneIds`), we determine how many of its Entrez Gene annotations are in the universe set and how many are in the selected set. With these counts we perform a Hypergeometric test using `phyper`. This is equivalent to using Fisher's exact test.

It is important that the correct chip annotation data package be identified as it determines the GO term to Entrez Gene id mapping as well as the universe of Entrez Gene ids in the case that the 'universe' argument is omitted.

For S. cerevisiae if the 'lib' argument is set to `"org.Sc.sgd"` then comparisons and statistics are computed using common names and are with respect to all genes annotated in the S. cerevisiae genome not with respect to any microarray chip. This will **not** be the right thing to do if you are working with a yeast microarray.

Value

A `HyperGResult-class` instance.

Author(s)

S. Falcon

See Also

HyperGResult-class HyperGParams-class geneKeggHyperGeoTest hyperGTest

---

geneKeggHyperGeoTest *Hypergeometric Tests for KEGG (DEFUNCT)*

Description

This function is defunct. Use `hyperGTest` instead.

Usage

```
geneKeggHyperGeoTest(entrezGeneIds, lib, universe=NULL)
```

Arguments

`entrezGeneIds`	A vector of Entrez Gene Identifiers
`lib`	A string giving the name of the annotation data package to use. This must correspond to the microarray chip type that the data came from
`universe`	A character vector of unique Entrez Gene identifiers. This is the population (the urn) of the Hypergeometric test. When `NULL` (default), the population is all Entrez Gene ids in the annotation package that have a KEGG pathway annotation (see details).

Details

The Entrez Gene ids given in `entrezGeneIds` define the selected set of genes. The universe of Entrez Gene ids is determined by the chip annotation data package (`lib`) or specified by the `universe` argument which must be a subset of the Entrez Gene ids represented on the chip. Both the selected genes and the universe are reduced by removing Entrez Gene ids that do not have any KEGG pathway annotations.

For each KEGG pathway that has at least one annotation in the selected gene set (`entrezGeneIds`), we determine how many of its Entrez Gene annotations are in the universe set and how many are in the selected set. With these counts we perform a Hypergeometric test using `phyper`. This is equivalent to using Fisher's exact test.

It is important that the correct chip annotation data package be identified as it determines the KEGG pathway to Entrez Gene id mapping as well as the universe of Entrez Gene ids in the case that the 'universe' argument is omitted.

For S. cerevisiae if the 'lib' argument is set to `"YEAST"` then comparisons and statistics are computed using common names and are with respect to all genes annotated in the S. cerevisiae genome not with respect to any microarray chip. This will **not** be the right thing to do if you are working with a yeast microarray.

Value

A `HyperGResult-class` instance.

Author(s)

S. Falcon

See Also

HyperGResult-class HyperGParams-class geneGoHyperGeoTest hyperGTest

---

getPathNames — *A function to print pathway names given their numeric id.*

Description

Given a KEGG pathway ID this function returns the character name of the pathway.

Usage

```
getPathNames(iPW)
```

Arguments

iPW	A vector of KEGG pathway IDs.

Details

This function simply does a look up in KEGGPATHID2NAME and returns a list of the pathway names.

Possible extensions would be to extend it to work with the cMAP library as well.

Value

A list of pathway names.

Author(s)

R. Gentleman

See Also

KEGGPATHID2NAME

Examples

```
nms = "00031"
getPathNames(nms)
```

GOHyperGParams-class

Class "GOHyperGParams"

Description

A parameter class for representing all parameters needed for running the `hyperGTest` method with one of the GO ontologies (BP, CC, MF) as the category.

Objects from the Class

Objects can be created by calls of the form `new("GOHyperGParams", ...)`.

Slots

`ontology`: A string specifying the GO ontology to use. Must be one of "BP", "CC", or "MF".

`conditional`: A logical indicating whether the calculation should condition on the GO structure.

`geneIds`: Object of class `"ANY"`: A vector of gene identifiers. Numeric and character vectors are probably the only things that make sense. These are the gene ids for the selected gene set.

`universeGeneIds`: Object of class `"ANY"`: A vector of gene ids in the same format as `geneIds` defining a subset of the gene ids on the chip that will be used as the universe for the hypergeometric calculation. If this is `NULL` or has length zero, then all gene ids on the chip will be used.

`annotation`: A string giving the name of the annotation data package for the chip used to generate the data.

`categorySubsetIds`: Object of class `"ANY"`: If the test method supports it, can be used to specify a subset of category ids to include in the test instead of all possible category ids.

`categoryName`: A string describing the category. Usually set automatically by subclasses. For example "GO".

Extends

Class `"HyperGParams"`, directly.

Methods

`hyperGTest(p)` Perform hypergeometric tests to assess overrepresentation of category ids in the gene set. See the documentation for the generic function for details. This method must be called with a proper subclass of `HyperGParams`.

`ontology(p), ontology(p) <- value` Accessors for the GO ontology. When setting, `value` should be one of "BP", "CC", or "MF".

`conditional(p), conditional(p) <- value` Accessors for the conditional flag. When setting, `value` must be `TRUE` or `FALSE`.

`isConditional(p)` An alias for `conditional`.

Author(s)

S. Falcon

See Also

HyperGResult-class GOHyperGParams-class geneKeggHyperGeoTest hyperGTest

---

gseattperm *Permutation p-values for GSEA*

Description

This function performs GSEA computations and returns p-values for each gene set based on repeated permutation of the phenotype labels.

Usage

```
gseattperm(eset, fac, mat, nperm)
```

Arguments

eset	An ExpressionSet object
fac	A factor identifying the phenotypes in eset. Usually, this will be one of the columns in the phenotype data associated with eset.
mat	A 0/1 incidence matrix with each row representing a gene set and each column representing a gene. A 1 indicates membership of a gene in a gene set.
nperm	Number of permutations to test to build the reference distribution.

Details

The t-statistic is used (via rowttests) to test for a difference in means between the phenotypes determined by fac within each gene set (given as a row of mat).

A reference distribution for these statistics is established by permuting fac and repeating the test B times.

Value

A matrix with the same number of rows as mat and two columns, "Lower" and "Upper". The "Lower" ("Upper") column gives the probability of seeing a t-statistic smaller (larger) than the observed.

Author(s)

Seth Falcon

Examples

```
## This example uses a random sample of probesets and a randomly
## generated category matrix.  The results, therefore, are not
## meaningful, but the code demonstrates how to use gseattperm without
## requiring any expensive computations.

## Obtain an ExpressionSet with two types of samples (mol.biol)
haveALL <- require("ALL")
if (haveALL) {
```

```
data(ALL)
set.seed(0xabcd)
rndIdx <- sample(1:nrow(ALL), 500)
Bcell <- grep("^B", as.character(ALL$BT))
typeNames <- c("NEG", "BCR/ABL")
bcrAblOrNegIdx <- which(as.character(ALL$mol.biol) %in% typeNames)
s <- ALL[rndIdx, intersect(Bcell, bcrAblOrNegIdx)]
s$mol.biol <- factor(s$mol.biol)

## Generate a random category matrix
nCats <- 100
set.seed(0xdcba)
rndCatMat <- matrix(sample(c(0L, 1L), replace=TRUE),
                    nrow=nCats, ncol=nrow(s),
                    dimnames=list(
                      paste("c", 1:nCats, sep=""),
                      featureNames(s)))

## Demonstrate use of gseattperm
N <- 10
pvals <- gseattperm(s, s$mol.biol, rndCatMat, N)
pvals[1:5, ]
}
```

HyperGParams-class *Class "HyperGParams"*

Description

An abstract (VIRTUAL) parameter class for representing all parameters needed by a method specializing the `hyperGTest` generic. You should only use subclasses of this class directly.

Objects from the Class

Objects of this class cannot be instantiated directly.

Slots

`geneIds:` Object of class `"ANY"`: A vector of gene identifiers. Numeric and character vectors are probably the only things that make sense. These are the gene ids for the selected gene set.

`universeGeneIds:` Object of class `"ANY"`: A vector of gene ids in the same format as `geneIds` defining a subset of the gene ids on the chip that will be used as the universe for the hypergeometric calculation. If this is `NULL` or has length zero, then all gene ids on the chip will be used.

`annotation:` A string giving the name of the annotation data package for the chip used to generate the data.

`cateogrySubsetIds:` Object of class `"ANY"`: If the test method supports it, can be used to specify a subset of category ids to include in the test instead of all possible category ids.

`categoryName:` A string describing the category. Usually set automatically by subclasses. For example "GO".

pvalueCutoff: The p-value to use as a cutoff for significance for testing methods that require it. This value will also be passed on to the result instance and used for display and counting of significant results. The default is 0.01.

testDirection: A string indicating whether the test should be for overrepresentation (`"over"`) or underrepresentation (`"under"`).

Methods

hyperGTest `signature(p = "HyperGParams")`: Perform hypergeometric tests to assess overrepresentation of category ids in the gene set. See the documentation for the generic function for details. This method must be called with a proper subclass of `HyperGParams`.

`geneIds(object), geneIds(object) <- value` Accessors for the gene identifiers that will be used as the selected gene list.

codeannotation(object) Accessor for annotation. If you want to change the annotation for an existing instance, use the replacement form.

`ontology(object)` Accessor for GO ontology.

`pvalueCutoff(r), pvalueCutoff(r) <- value` Accessor for the p-value cutoff. When setting, `value` should be a numeric value between zero and one.

`testDirection` Accessor for the test direction. When setting, `value` must be either "over" or "under".

`universeGeneIds(r)` accessor for vector of gene identifiers.

`isConditional(r)` Returns `TRUE` if the instance has its conditional flag set

Author(s)

S. Falcon

See Also

`HyperGResult-class` `GOHyperGParams-class` `KEGGHyperGParams-class` `geneKeggHyperGeoTes`
`hyperGTest`

`HyperGResult-accessors`

Accessors for HyperGResult Objects

Description

This manual page documents generic functions for extracting data from the result object returned from a call to `hyperGTest`. The result object will be a subclass of `HyperGResultBase`. Methods apply to all result object classes unless otherwise noted.

Usage

```
pvalues(r)
oddsRatios(r)
expectedCounts(r)

geneCounts(r)
universeCounts(r)
universeMappedCount(r)
geneMappedCount(r)

geneIds(object, ...)
geneIdUniverse(r, cond = TRUE)
condGeneIdUniverse(r)
geneIdsByCategory(r, catids = NULL)
sigCategories(r, p)

## R CMD check doesn't like these
## annotation(r)
## description(r)

testName(r)
pvalueCutoff(r)
testDirection(r)

chrGraph(r)
```

Arguments

`r, object`	An instance of a subclass of `HyperGResultBase`.
`catids`	A character vector of category identifiers.
`p`	Numeric p-value used as a cutoff for selecting a subset of the result.
`cond`	A logical value indicating whether to return conditional results for a conditional test. The default is `TRUE`. For non-conditional results, this argument is ignored.
`...`	Additional arguments that may be used by specializing methods.

Accessor Methods (Generic Functions)

geneCounts returns an `"integer"` vector: for each category term tested, the number of genes from the gene set that are annotated at the term.

pvalues returns a `"numeric"` vector: the ordered p-values for each category term tested.

universeCounts returns an `"integer"` vector: for each category term tested, the number of genes from the gene universe that are annotated at the term.

universeMappedCount returns an `"integer"` vector of length one giving the size of the gene universe set.

expectedCounts returns a `"numeric"` vector giving the expected number of genes in the selected gene list to be found at each tested category term. These values may surprise you if you forget that your gene list and gene universe might have had to undergo further filtering to ensure that each gene has been labeled by at least one GO term.

oddsRatios returns a "numeric" vector giving the odds ratio for each category term tested.

annotation returns the name of the annotation data package used.

geneIds returns the input vector of gene identifiers intersected with the universe of gene identifiers used in the computation.

geneIdUniverse returns a list named by the tested categories. Each element of the list is a vector of gene identifiers (from the gene universe) annotated at the corresponding category term.

geneIdsByCategory returns a list similar to geneIdUniverse, but each vector of gene IDs is intersected with the list of selected gene IDs from geneIds. The result is the selected gene IDs annotated at each category.

sigCategories returns a character vector of category identifiers with a significant p-value. If argument p is missing, then the cutoff obtained from pvalueCutoff(r) will be used.

geneMappedCount returns the size of the selected gene set used in the computation. This is simply length(geneIds(obj)).

pvalueCutoff accessor for the pvalueCutoff slot.

testDirection accessor for the testDirection slot. Contains a string indicating whether the test was for "over" or "under" representation of the categories.

description returns a character string description of the test result.

testName returns a string describing the testing method used.

isConditional returns TRUE if the result was obtained using a conditional algorithm.

summary returns a data.frame summarizing the test result. Optional arguments pvalue and categorySize allow specification of maximum p-value and minimum categorySize, respectively. The data frame contains the GOID, Pvalue, OddsRatio, ExpCount, Count, and Size. ExpCount is the expected count and the Count is how many instances of that term were actually oberved in your gene list while the Size is the number that could have been found in your gene list if every instance had turned up. Values like the ExpCount and the Size are going to be affected by what is included in the gene universe as well as by whether or not it was a conditional test.

htmlReport writes an HTML version of the table produced by the summary method. The first argument should be a HyperGResult instance (or subclass). The path of a file to write the report to can be specified using the file argument. The default is file="" which will cause the report to be printed to the screen. If you wish to create a single report comprising multiple results you can set append=TRUE. The default is FALSE (overwrite pre-existing report file). You can specify a string to use as an identifier for each table by providing a value for the label argument. The number of digits displayed in numerical columns can be controlled using digits (defaults to 3). The summary method is called on the HyperGResult instance to generate a data frame that is transformed to HTML. You can pass additional arguments to the summary method which is used to generate the data frame that is transformed to HTML by specifying a named list using summary.args.

Author(s)

Seth Falcon

See Also

hyperGTest HyperGResult-class HyperGParams-class GOHyperGParams-class KEGGHyperGParams-class

Examples

```
## Note that more in-depth examples can be found in the GOstats
## vignette (Hypergeometric tests using GOstats).
library("hgu95av2.db")
library("annotate")

probids <- ls(hgu95av2GENENAME)[1:300]
## Select for probeids that have PFAM ids
hasPFAM <- sapply(mget(probids, hgu95av2PFAM), function(ids)
                  if(!is.na(ids) && length(ids) > 1) TRUE else FALSE)
probids <- probids[hasPFAM]
## get unique Entrez Gene IDs
probids <- unique(getLL(probids, "hgu95av2"))
## Now do the same for the universe
univ <- ls(hgu95av2GENENAME)
univHasPFAM <- sapply(mget(univ, hgu95av2PFAM), function(ids)
                      if(!is.na(ids) && length(ids) > 1) TRUE else
                         FALSE)
univ <- univ[univHasPFAM]
univ <- unique(getLL(univ, "hgu95av2"))

p <- new("PFAMHyperGParams", geneIds=probids, universeGeneIds=univ,
         annotation="hgu95av2")
## this takes a while...
if(interactive()){
hypt <- hyperGTest(p)
summary(hypt)
htmlReport(hypt, file="temp.html", summary.args=list("htmlLinks"=TRUE))
}
```

HyperGResultBase-class

Class "HyperGResultBase"

Description

This VIRTUAL class represents common elements of the return values of generic functions like `hyperGTest`. All subclasses are intended to implement the accessor functions documented at HyperGResult-accessors.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`annotation`: Object of class `"character"` giving the name of the annotation data package used.

`geneIds`: Object of class `"ANY"` (usually a character vector, but sometimes an integer vector). The input vector of gene identifiers intersected with the universe of gene identifiers used in the computation.

`testName`: Object of class `"character"` identifying the testing method used.

pvalueCutoff: Numeric value used by the testing method as a p-value cutoff. Not all testing methods use this. Also used by the `show` method to count number of significant terms.

testDirection: A string indicating whether the test performed was for overrepresentation (`"over"`) or underrepresentation(`"under"`).

Methods

See HyperGResult-accessors.

Author(s)

Seth Falcon

See Also

`HyperGResult-class` `GeneGoHyperGeoTestResult-class` HyperGResult-accessors

`HyperGResult-class` *Class "HyperGResult"*

Description

This class represents the results of a test for over-representation of categories among genes in a selected gene set based upon the Hypergeometric distribution. The `hyperGTest` generic function returns an instance of the `HyperGResult` class. For details on accessing the results, see HyperGResult-accessors.

Objects from the Class

Objects can be created by calls of the form `new("HyperGResult", ...)`.

Slots

pvalues: `"numeric"` vector: the ordered p-values for each category term tested.

catToGeneId: Object of class `"list"`. The names of the list are category IDs. Each element is a vector of gene IDs annotated at the given category ID and in the specified gene universe.

annotation: A string giving the name of the chip annotation data package used.

geneIds: Object of class `"ANY"`: the input vector of gene identifiers intersected with the universe of gene identifiers used in the computation. The class of this slot is specified as `"ANY"` because gene IDs may be integer or character vectors depending on the annotation package.

testName: A string identifying the testing method used.

pvalueCutoff: Numeric value used a a p-value cutoff. Used by the `show` method to count number of significant terms.

testDirection: A string indicating whether the test should be for overrepresentation (`"over"`) or underrepresentation (`"under"`).

oddsRatios a `"numeric"` vector giving the odds ratio for each category term tested.

expectedCounts a `"numeric"` vector giving the expected number of genes in the selected gene list to be found at each tested category term.

Extends

Class "HyperGResultBase", directly.

Methods

See HyperGResult-accessors.

Author(s)

Seth Falcon

See Also

HyperGResultBase-class GeneGoHyperGeoTestResult-class HyperGResult-accessors

---

hyperGTest *Hypergeometric Test for association of categories and genes*

Description

Given a subclass of HyperGParams, compute Hypergeomtric p-values for over or under-representation of each term in the specified category among the specified gene set.

Usage

```
hyperGTest(p)
```

Arguments

p	An instance of a subclass of HyperGParams. This parameter object determines the category of interest (e.g., GO or KEGG) as well as the gene set.

Details

The gene identifiers in the geneIds slot of p define the selected set of genes. The universe of gene ids is determined by the chip annotation found in the annotation slot of p. Both the selected genes and the universe are reduced by removing identifiers that do not have any annotations in the specified category.

For each term in the specified category that has at least one annotation in the selected gene set, we determine how many of its annotations are in the universe set and how many are in the selected set. With these counts we perform a Hypergeometric test using phyper. This is equivalent to using Fisher's exact test.

It is important that the correct chip annotation data package be identified as it determines the universe of gene identifiers and is often used to determine the mapping between the category term and the gene identifiers.

For S. cerevisiae if the annotation slot of p is set to '"org.Sc.sgd"' then comparisons and statistics are computed using common names and are with respect to all genes annotated in the S. cerevisiae genome not with respect to any microarray chip. This will *not* be the right thing to do if you are working with a yeast microarray.

Value

A `HyperGResult` instance.

Implementation Notes

In most cases, the provided method with signature matching any subclass of `HyperGParams` is all that will be needed. This method follows a template pattern. To add support for a new FOO category type, a developer would need to create a `FooHyperGParams` subclass and then define two methods specialized to the new subclass that get called from inside `hyperGTest`: `universeBuilder` and `categoryToEntrezBuilder`.

Author(s)

S. Falcon

See Also

`HyperGResult-class` `HyperGParams-class` `GOHyperGParams-class` `KEGGHyperGParams-class`

KEGGHyperGParams-class

Class "KEGGHyperGParams" and "PFAMHyperGParams"

Description

Parameter classes for representing all parameters needed for running the `hyperGTest` method with KEGG or PFAM as the category.

Objects from the Class

Objects can be created by calls of the form `new("KEGGHyperGParams", ...)` or `new("PFAMHyperGParams", ...)`.

Slots

`geneIds:` Object of class `"ANY"`: A vector of gene identifiers. Numeric and character vectors are probably the only things that make sense. These are the gene ids for the selected gene set.

`universeGeneIds:` Object of class `"ANY"`: A vector of gene ids in the same format as `geneIds` defining a subset of the gene ids on the chip that will be used as the universe for the hypergeometric calculation. If this is `NULL` or has length zero, then all gene ids on the chip will be used.

`annotation:` A string giving the name of the annotation data package for the chip used to generate the data.

`cateogrySubsetIds:` Object of class `"ANY"`: If the test method supports it, can be used to specify a subset of category ids to include in the test instead of all possible category ids.

`categoryName:` A string describing the category. This will be automatically set to "KEGG" or "PFAM" via the class's prototype.

pvalueCutoff: The p-value to use as a cutoff for significance for testing methods that require it. This value will also be passed on to the result instance and used for display and counting of significant results. The default is 0.01.

testDirection: A string indicating whether the test should be for overrepresentation (`"over"`) or underrepresentation (`"under"`).

Extends

Class `"HyperGParams"`, directly.

Methods

hyperGTest `signature(p = "HyperGParams")`: Perform hypergeometric tests to assess overrepresentation of category ids in the gene set. See the documentation for the generic function for details. This method must be called with a proper subclass of `HyperGParams`.

Author(s)

S. Falcon

See Also

`HyperGResult-class` `GOHyperGParams-class` `geneKeggHyperGeoTest` `hyperGTest`

LinearMParams-class

Class "LinearMParams"

Description

An abstract (VIRTUAL) parameter class for representing all parameters needed by a method specializing the `linearMTest` generic. You should only use subclasses of this class directly.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

geneStats: Object of class `"numeric"` ~~

universeGeneIds: Object of class `"ANY"` ~~

annotation: Object of class `"character"` ~~

datPkg: Object of class `"DatPkg"` ~~

cateogrySubsetIds: Object of class `"ANY"` ~~

categoryName: Object of class `"character"` ~~

pvalueCutoff: Object of class `"numeric"` ~~

minSize: Object of class `"integer"` ~~

testDirection: Object of class `"character"` ~~

Methods

annotation<- `signature(object = "LinearMParams", value = "character")`: ...

annotation `signature(object = "LinearMParams")`: ...

categoryName `signature(r = "LinearMParams")`: ...

conditional `signature(r = "LinearMParams")`: ...

geneIds<- `signature(object = "LinearMParams")`: ...

geneIds `signature(object = "LinearMParams")`: ...

pvalueCutoff<- `signature(r = "LinearMParams")`: ...

pvalueCutoff `signature(r = "LinearMParams")`: ...

show `signature(object = "LinearMParams")`: ...

testDirection<- `signature(r = "LinearMParams")`: ...

testDirection `signature(r = "LinearMParams")`: ...

universeGeneIds `signature(r = "LinearMParams")`: ...

Author(s)

Deepayan Sarkar

See Also

`ChrMapLinearMParams` for descriptions of the slots.

Examples

```
showClass("LinearMParams")
```

LinearMResultBase-class

Class "LinearMResultBase"

Description

This VIRTUAL class represents common elements of the return values of generic functions like `linearMTest`.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`annotation`: Object of class `"character"` ~~

`geneIds`: Object of class `"ANY"` ~~

`testName`: Object of class `"character"` ~~

`pvalueCutoff`: Object of class `"numeric"` ~~

`minSize`: Object of class `"integer"` ~~

`testDirection`: Object of class `"character"` ~~

Methods

annotation `signature(object = "LinearMResultBase")`: ...

condGeneIdUniverse `signature(r = "LinearMResultBase")`: ...

conditional `signature(r = "LinearMResultBase")`: ...

description `signature(object = "LinearMResultBase")`: ...

geneIdsByCategory `signature(object = "LinearMResultBase")`: ...

geneIds `signature(object = "LinearMResultBase")`: ...

geneMappedCount `signature(r = "LinearMResultBase")`: ...

isConditional `signature(r = "LinearMResultBase")`: ...

pvalueCutoff `signature(r = "LinearMResultBase")`: ...

show `signature(object = "LinearMResultBase")`: ...

sigCategories `signature(r = "LinearMResultBase")`: ...

summary `signature(object = "LinearMResultBase")`: ...

testDirection `signature(r = "LinearMResultBase")`: ...

testName `signature(r = "LinearMResultBase")`: ...

universeCounts `signature(r = "LinearMResultBase")`: ...

universeMappedCount `signature(r = "LinearMResultBase")`: ...

Author(s)

Deepayan Sarkar

See Also

`LinearMResult`, `ChrMapLinearMParams`

Examples

```
showClass("LinearMResultBase")
```

LinearMResult-class

Class "LinearMResult"

Description

This class represents the results of a test for systematic change in some gene-level statistic by gene sets. The `linearMTest` generic function returns an instance of the `LinearMResult` class.

Objects from the Class

Objects can be created by calls of the form `new("LinearMResult", ...)`, but is more commonly created using a call to `linearMTest`.

Slots

pvalues: Object of class "numeric" ~~

effectSize: Object of class "numeric" ~~

catToGeneId: Object of class "list" ~~

annotation: Object of class "character" ~~

geneIds: Object of class "ANY" ~~

testName: Object of class "character" ~~

pvalueCutoff: Object of class "numeric" ~~

minSize: Object of class "integer" ~~

testDirection: Object of class "character" ~~

Extends

Class "LinearMResultBase", directly.

Methods

effectSize signature(r = "LinearMResult"): ...

geneIdUniverse signature(r = "LinearMResult"): ...

pvalues signature(r = "LinearMResult"): ...

Author(s)

Deepayan Sarkar

See Also

linearMTest

Examples

```
showClass("LinearMResult")
```

---

linearMTest — *A linear model-based test to detect enrichment of unusual genes in categories*

Description

Given a subclass of LinearMParams, compute p-values for detecting systematic up or downregulation of the specified gene set in the specified category.

Usage

```
linearMTest(p)
```

Arguments

p	An instance of a subclass of LinearMParams. This parameter object determines the category of interest (currently, only chromosome bands) as well as the gene set.

Details

The per-gene statistics in the geneStats slot of p give a measure of up or down regulation of the individual genes in the universe.

Value

A LinearMResult instance.

Author(s)

D. Sarkar

See Also

LinearMResult-class LinearMParams-class

---

local_test_factory *Local and Global Test Function Factories*

Description

These functions return functions appropriate for use as the tfun argument to topdown_tree_visitor or bottomup_tree_visitor. In particular, it is these functions that are associated with the "local" and "global" options for the type argument to cb_test.

Usage

```
local_test_factory(selids, tableTest = chisq.test)
hg_test_factory(selids, PCUT = 0.05, COND = FALSE, OVER = TRUE)
```

Arguments

selids	A vector of gene IDs. The IDs should match those used to annotatate the ChrBandTree given by chrtree. In most cases, these will be Entrez Gene IDs.
tableTest	A contingency table testing function. The behavior of this function must be reasonably close to that of chisq.test.
PCUT	A p-value cutoff that will be used to determine if a given test is significant or not when using hg_test_factory with COND=TRUE.
COND	A logical value indicating whether a conditional test should be performed.
OVER	If TRUE, test for over representation, if FALSE, test for under representation.

Details

The returned functions have signature `f(start, g, prev_ans)` where `start` is a vector of start nodes, `g` is a chromosome band tree graph, and `prev_ans` can contain the previous result returned by a call to this function.

Value

A function that can be used as the `tfun` argument to the tree visitor functions.

Author(s)

Seth Falcon

See Also

`cb_test`

`makeChrBandGraph` *Create a graph representing chromosome band annotation data*

Description

This function returns a `graph` object representing the nested structure of chromosome bands (also known as cytogenetic bands). The nodes of the graph are band identifiers. Each node has a `geneIds` node attribute that lists the gene IDs that are annotated at the band (the gene IDs will be Entrez IDs in most cases).

Usage

```
makeChrBandGraph(chip, univ = NULL)
```

Arguments

`chip`	A string giving the annotation source. For example, `"hgu133plus2"`
`univ`	A vector of gene IDs (these should be Entrez IDs for most annotation sources). The annotations attached to the graph will be limited to those specified by `univ`. If `univ` is `NULL` (default), then the gene IDs are those found in the annotation data source.

Details

This function parses the data stored in the `<chip>MAP` map from the appropriate annotation data package. Although cytogenetic bands are observed in all organisms, currently, only human and mouse band nomenclatures are supported.

Value

A `graph-class` instance. The graph will be a tree and the root node is labeled for the organism.

Author(s)

Seth Falcon

Examples

```
chrGraph <- makeChrBandGraph("hgu95av2.db")
chrGraph
```

---

makeEBcontr — *A function to make the contrast vectors needed for EBarrays*

Description

Using EBarrays to detect differential expression requires the construction of a set of contrasts. This little helper function computes these contrasts for a two level factor.

Usage

```
makeEBcontr(f1, hival)
```

Arguments

`f1`	The factor that will define the contrasts.
`hival`	The `level` of the factor to treat as the high level.

Details

Not much more to add, see EBarrays for more details. This is used in the Category package to let users compute the posterior probability of differential expression, and hence to compute expected numbers of differentially expressed genes, per category.

Value

An object of class “ebarraysPatterns”.

Author(s)

R. Gentleman

See Also

`ebPatterns`

Examples

```
if( require("EBarrays") ) {
 myfac = factor(rep(c("A", "B"), c(12, 24)))
 makeEBcontr(myfac, "B")
}
```

makeValidParams — *Non-standard Generic for Checking Validity of Parameter Objects*

Description

This function is not intended for end-users, but may be useful for developers extending the Hypergeometric testing capabilities provideded by the Category package.

`makeValidParams` is intended to validate a parameter object instance (e.g. HyperGParams or subclass). The idea is that unlike `validObject`, methods for this generic attempt to fix invalid instances when possible, and in this case issuing a warning, and only give an error if the object cannot be fixed.

Usage

```
makeValidParams(object)
```

Arguments

`object`	A parameter object. Consult `showMethods` to see signatures currently supported.

Value

The value must have the same class as the `object` argument.

Author(s)

Seth Falcon

MAPAmat — *Create an incidence matrix mapping chromosome bands to genes*

Description

These functions return a 0/1 incidence matrix with a row for each chromosme band and a column for each gene. Only those chromosome bands with at least one gene annotation will be included.

Usage

```
MAPAmat(chip, univ = NULL, minCount = 0)
chrBandInciMat(chip, univ = NULL, minCount = 0)
makeChrBandInciMat(chrGraph)
```

Arguments

chip	A string giving the annotation source. For example, "hgu133plus2"
univ	A vector of gene IDs (these should be Entrez IDs for most annotation sources). The the annotations will be limited to those in the set specified by univ. If univ is NULL (default), then the gene IDs are those found in the annotation data source.
chrGraph	A graph object as returned by makeChrBandGraph
minCount	Bands with less than minCount genes will be excluded from the returned matrix. If minCount is 0, no bands will be removed, this is the default.

Details

chrBandInciMat is a DEPRECATED alias for MAPAmat.

Value

A (0/1) incidence matrix with chromosome bands as rows and gene IDs as columns. A 1 in m[i, j] indicates that the chromosome band rownames(m)[i] contains the geneID colnames(m)[j].

Author(s)

Seth Falcon

See Also

makeChrBandGraph, cateGOry, probes2MAP

Examples

```
have_hgu95av2.db <- suppressWarnings(require("hgu95av2.db"))
if (have_hgu95av2.db)
   mam <- MAPAmat("hgu95av2.db")
```

NewChrBandTree *Create a new ChrBandTree object*

Description

NewChrBandTree and ChrBandTreeFromGraph provide constructors for the ChrBandTree class.

Usage

```
NewChrBandTree(chip, univ)
ChrBandTreeFromGraph(g)
```

Arguments

`chip`	The name of an annotation data package
`univ`	A vector of gene identifiers that defines the universe of genes. Usually, this will be a vector of Entez Gene IDs. If `univ` is `NULL`, then all genes probed on the specified chip will be in the universe. We strongly recommend using the set of genes that remains after applying a non-specific filter as the universe.
`g`	A `graph` instance as returned by `makeChrBandGraph`

Value

A new `ChrBandTree` instance.

Author(s)

S. Falcon

See Also

`ChrBandTree-class`

`probes2MAP` *Map probe IDs to MAP regions.*

Description

This function maps probe identifiers to MAP positions using the appropriate Bioconductor metadata package.

Usage

```
probes2MAP(pids, data = "hgu133plus2")
```

Arguments

`pids`	A vector of probe IDs for the chip in use.
`data`	The name of the chip, as a character string.

Details

Probes are mapped to regions, no checking for duplicate Entrez gene IDs is done.

Value

A vector, the same length as `pids`, with the MAP locations.

Author(s)

R. Gentleman

See Also

`probes2Path`

Examples

```
set.seed(123)
library("hgu95av2.db")
v1 = sample(names(as.list(hgu95av2MAP)), 100)
pp = probes2MAP(v1, "hgu95av2.db")
```

---

probes2Path — *A function to map probe identifiers to pathways.*

Description

Given a set of probe identifiers from a microarray this function looks up all KEGG pathways that the probe is documented to be involved in.

Usage

```
probes2Path(pids, data = "hgu133plus2")
```

Arguments

`pids`	A vector of probe identifiers.
`data`	The character name of the chip.

Details

This is a simple look up in the appropriate chip `PATH` data environment.

Value

A list of pathway vectors. One element for each value of `pid` that is mapped to at least one pathway.

Author(s)

R. Gentleman

See Also

`findAMstats`

Examples

```
library("hgu95av2.db")
x = c("1001_at", "1000_at")
probes2Path(x, "hgu95av2.db")
```

tree_visitor *Tree Visitor Function*

Description

This function visits each node in a tree-like object in an order determined by the `relationOf` function. The function given by `tfun` is called for each set of nodes and the function `nfun` determines which nodes to test next optionally making use of the result of the previous test.

Usage

```
tree_visitor(g, start, tfun, nfun, relationOf)
topdown_tree_visitor(g, start, tfun, nfun)
bottomup_tree_visitor(g, start, tfun, nfun)
```

Arguments

`g`	A tree-like object that supports the method given by `relationOf`.
`start`	The set of nodes to start the computation (can be a list of siblings), but the nodes should all belong to the same level of the tree (same path length to root node).
`tfun`	The test function applied to each list of siblings at each level starting with `start`. The signature of `tfun` should be `(start, g, prev_ans)`.
`nfun`	A function with signature `(ans, g)` that processes the result of `tfun` and returns a character vector of node names corresponding to nodes that were involved in an "interesting" test. This is used to determine the next set of nodes to test (see details).
`relationOf`	The method used to traverse the tree. For example `childrenOf` or `parentOf`.

Details

The `tree_visitor` function is intended to allow developers to quickly prototype different statistical testing paradigms on trees. It may be possible to extend this to work for DAGs.

The visit begins by calling `tfun` with the nodes provided by `start`. The result of each call to `tfun` is stored in an environment. The concept is visitation by tree level and so each result is stored using a key representing the level (this isn't quite right since the nodes in `start` need not be first level, but they will be assigned key "1". After storing the result, `nfun` is used to obtain a vector of accepted node labels. The idea is that the user should have a way of determining which nodes in the next level of the tree are worth testing. The next `start` set is determined by `start <- relationOf(g, accepted)` where accepted is `unique(nfun(ans, g))`.

Value

A list. See the return value of `cb_test` to get an idea. Each element of the list represents a call to `tfun` at a given level of the tree.

Author(s)

Seth Falcon

`ttperm` *A simple function to compute a permutation t-test.*

Description

The data matrix, `x`, with two-level factor, `fac`, is used to compute t-tests. The values of `fac` are permuted `B` times and the complete set of t-tests is performed for each permutation.

Usage

```
ttperm(x, fac, B = 100, tsO = TRUE)
```

Arguments

`x`	A data matrix. The number of columns should be the same as the length of `fac`.
`fac`	A factor with two levels.
`B`	An integer specifying the number of permutations.
`tsO`	A logical indicating whether to compute only the t-test statistic for each permuation. If `FALSE` then p-values are also computed - but this can be very slow.

Details

Not much more to say. Probably there is a generic function somewhere, but I could not find it.

Value

A list, the first element is named `obs` and contains the true, observed, values of the t-statistic. The second element is named `ans` and contains a list of length `B` containing the different permuations.

Author(s)

R. Gentleman

See Also

`rowttests`

Examples

```
x=matrix(rnorm(100), nc=10)
y = factor(rep(c("A","B"), c(5,5)))
ttperm(x, y, 10)
```

`universeBuilder` *Return a vector of gene identifiers with category annotations*

Description

Return all gene ids that are annotated at one or more terms in the category. If the `universeGeneIds` slot of `p` has length greater than zero, then the intersection of the gene ids specified in that slot and the normal return value is given.

Usage

```
universeBuilder(p)
```

Arguments

`p`	A subclass of `HyperGParams-class`

Details

End users **should not** call this directly. This method gets called from `hyperGTest`. To add support for a new category, a new method for this generic must be defined. Its signature should match a subclass of `HyperGParams-class` appropriate for the new category.

Value

A vector of gene identifiers.

Author(s)

S. Falcon

See Also

`hyperGTest` `HyperGParams-class`

Index

∗Topic **classes**
ChrBandTree-class, 8
ChrMapHyperGParams-class, 10
ChrMapHyperGResult-class, 11
ChrMapLinearMParams-class, 12
ChrMapLinearMResult-class, 13
DatPkg-class, 14
GOHyperGParams-class, 20
HyperGParams-class, 22
HyperGResult-class, 27
HyperGResultBase-class, 26
KEGGHyperGParams-class, 29
LinearMParams-class, 30
LinearMResult-class, 33
LinearMResultBase-class, 32
∗Topic **htest**
geneGoHyperGeoTest, 17
geneKeggHyperGeoTest, 18
HyperGResult-accessors, 24
hyperGTest, 28
linearMTest, 34
∗Topic **internal**
local_test_factory, 35
tree_visitor, 42
∗Topic **manip**
applyByCategory, 1
cateGOry, 2
categoryToEntrezBuilder, 3
findAMstats, 16
getPathNames, 19
makeChrBandGraph, 36
makeEBcontr, 37
makeValidParams, 38
MAPAmat, 38
probes2MAP, 40
probes2Path, 41
ttperm, 43
universeBuilder, 44

AffyDatPkg-class *(DatPkg-class)*, 14
allGeneIds *(ChrBandTree-class)*, 8
allGeneIds,ChrBandTree-method *(ChrBandTree-class)*, 8
annotation *(HyperGResult-accessors)*, 24
annotation,GOHyperGParams-method *(GOHyperGParams-class)*, 20
annotation,HyperGParams-method *(HyperGParams-class)*, 22
annotation,HyperGResultBase-method *(HyperGResult-accessors)*, 24
annotation,LinearMParams-method *(LinearMParams-class)*, 30
annotation,LinearMResultBase-method *(LinearMResultBase-class)*, 32
annotation<-,HyperGParams,character-method *(HyperGParams-class)*, 22
annotation<-,LinearMParams,character-method *(LinearMParams-class)*, 30
apply, *2*
applyByCategory, 1, *2*, *3*, *17*

bottomup_tree_visitor *(tree_visitor)*, 42

cateGOry, *1*, 2, *39*
categoryName *(HyperGParams-class)*, 22
categoryName,GOHyperGParams-method *(GOHyperGParams-class)*, 20
categoryName,HyperGParams-method *(HyperGParams-class)*, 22
categoryName,LinearMParams-method *(LinearMParams-class)*, 30
categoryToEntrezBuilder, 3
categoryToEntrezBuilder,GOHyperGParams-method *(categoryToEntrezBuilder)*, 3
categoryToEntrezBuilder,KEGGHyperGParams-metho *(categoryToEntrezBuilder)*, 3
categoryToEntrezBuilder,PFAMHyperGParams-metho *(categoryToEntrezBuilder)*, 3

cb_children (*cb_contingency*), 4
cb_contingency, 4
cb_parse_band_Hs, 5
cb_parse_band_hsa
(*cb_parse_band_Hs*), 5
cb_parse_band_Mm, 6
cb_sigBands (*cb_contingency*), 4
cb_test, 7, *35*
childrenOf (*ChrBandTree-class*), 8
childrenOf,ChrBandTree,character-method
(*ChrBandTree-class*), 8
chrBandInciMat (*MAPAmat*), 38
ChrBandTree-class, *40*
ChrBandTree-class, 8
ChrBandTreeFromGraph
(*NewChrBandTree*), 39
chrGraph
(*HyperGResult-accessors*),
24
chrGraph,ChrMapHyperGResult-method
(*HyperGResult-accessors*),
24
chrGraph,ChrMapLinearMResult-method
(*ChrMapLinearMResult-class*),
13
ChrMapHyperGParams-class, 10
ChrMapHyperGResult-class, 11
ChrMapLinearMParams, *14*, *31*, *33*
ChrMapLinearMParams-class, 12
ChrMapLinearMResult-class, 13
condGeneIdUniverse
(*HyperGResult-accessors*),
24
condGeneIdUniverse,ChrMapHyperGResult-method
(*HyperGResult-accessors*),
24
condGeneIdUniverse,ChrMapLinearMResult-method
(*ChrMapLinearMResult-class*),
13
condGeneIdUniverse,HyperGResultBase-method
(*HyperGResult-accessors*),
24
condGeneIdUniverse,LinearMResultBase-method
(*LinearMResultBase-class*),
32
conditional (*HyperGParams-class*),
22
conditional,ChrMapHyperGParams-method
(*ChrMapHyperGParams-class*),
10
conditional,ChrMapHyperGResult-method
(*ChrMapHyperGResult-class*),
11
conditional,ChrMapLinearMParams-method
(*ChrMapLinearMParams-class*),
12
conditional,GOHyperGParams-method
(*GOHyperGParams-class*), 20
conditional,HyperGParams-method
(*HyperGParams-class*), 22
conditional,HyperGResultBase-method
(*HyperGResultBase-class*),
26
conditional,LinearMParams-method
(*LinearMParams-class*), 30
conditional,LinearMResultBase-method
(*LinearMResultBase-class*),
32
conditional<-
(*HyperGParams-class*), 22
conditional<-,ChrMapHyperGParams,logical-metho
(*ChrMapHyperGParams-class*),
10
conditional<-,ChrMapLinearMParams,logical-meth
(*ChrMapLinearMParams-class*),
12
conditional<-,GOHyperGParams,logical-method
(*GOHyperGParams-class*), 20

DatPkg-class, 14
DatPkgFactory (*DatPkg-class*), 14
description
(*HyperGResult-accessors*),
24
description,HyperGResultBase-method
(*HyperGResult-accessors*),
24
description,LinearMResultBase-method
(*LinearMResultBase-class*),
32

ebPatterns, *37*
effectSize, 15
effectSize,LinearMResult-method
(*LinearMResult-class*), 33
exampleLevels, 16
expectedCounts
(*HyperGResult-accessors*),
24
expectedCounts,ChrMapHyperGResult-method
(*HyperGResult-accessors*),
24
expectedCounts,HyperGResult-method
(*HyperGResult-accessors*),
24

findAMstats, 16, *41*

geneCounts (*HyperGResult-accessors*), 24
geneCounts,HyperGResultBase-method (*HyperGResult-accessors*), 24
geneGoHyperGeoTest, 17, *19*
GeneGoHyperGeoTestResult-class, *27, 28*
geneIds (*HyperGResult-accessors*), 24
geneIds,ChrBandTree-method (*ChrBandTree-class*), 8
geneIds,HyperGParams-method (*HyperGParams-class*), 22
geneIds,HyperGResultBase-method (*HyperGResult-accessors*), 24
geneIds,LinearMParams-method (*LinearMParams-class*), 30
geneIds,LinearMResultBase-method (*LinearMResultBase-class*), 32
geneIds<- (*HyperGParams-class*), 22
geneIds<-,HyperGParams,ANY-method (*HyperGParams-class*), 22
geneIds<-,HyperGParams,logical-method (*HyperGParams-class*), 22
geneIds<-,LinearMParams,ANY-method (*LinearMParams-class*), 30
geneIdsByCategory (*HyperGResult-accessors*), 24
geneIdsByCategory,HyperGResultBase-method (*HyperGResult-accessors*), 24
geneIdsByCategory,LinearMResultBase-method (*LinearMResultBase-class*), 32
geneIdUniverse (*HyperGResult-accessors*), 24
geneIdUniverse,ChrMapHyperGResult-method (*HyperGResult-accessors*), 24
geneIdUniverse,HyperGResult-method (*HyperGResult-accessors*), 24
geneIdUniverse,LinearMResult-method (*LinearMResult-class*), 33
geneKeggHyperGeoTest, 18, *18*, *21*, *23*, *30*
geneMappedCount (*HyperGResult-accessors*), 24
geneMappedCount,HyperGResultBase-method (*HyperGResult-accessors*), 24
geneMappedCount,LinearMResultBase-method (*LinearMResultBase-class*), 32
getPathNames, 19
GO, *2*
GO2AllProbes (*DatPkg-class*), 14
GO2AllProbes,DatPkg-method (*DatPkg-class*), 14
GO2AllProbes,Org.XX.egDatPkg-method (*DatPkg-class*), 14
GO2AllProbes,YeastDatPkg-method (*DatPkg-class*), 14
GOHyperGParams-class, *21*, *23*, *26*, *29*, *30*
GOHyperGParams-class, 20
graph-class, *36*
gseattperm, 21

hg_test_factory (*local_test_factory*), 35
htmlReport (*HyperGResult-accessors*), 24
htmlReport,HyperGResultBase-method (*HyperGResult-accessors*), 24
htmlReport,KEGGHyperGResult-method (*HyperGResult-accessors*), 24
htmlReport,PFAMHyperGResult-method (*HyperGResult-accessors*), 24
HyperGParams, *10*
HyperGParams-class, *4*, *18*, *19*, *26*, *29*, *44*
HyperGParams-class, 22
HyperGResult-class, *18*, *19*, *21*, *23*, *26*, *27*, *29*, *30*
HyperGResult-accessors, *11*, 24, *26–28*
HyperGResult-class, 27
HyperGResultBase, *11*
HyperGResultBase-class, *28*
HyperGResultBase-class, 26

hyperGTest, *4*, *10*, *18*, *19*, *21*, *23*, *26*, 28, *30*, *44*
hyperGTest,ChrMapHyperGParams-method *(hyperGTest)*, 28
hyperGTest,HyperGParams-method *(hyperGTest)*, 28
hyperGTest,KEGGHyperGParams-method *(hyperGTest)*, 28
hyperGTest,PFAMHyperGParams-method *(hyperGTest)*, 28

ID2EntrezID *(DatPkg-class)*, 14
ID2EntrezID,AffyDatPkg-method *(DatPkg-class)*, 14
ID2EntrezID,ArabidopsisDatPkg-method *(DatPkg-class)*, 14
ID2EntrezID,Org.XX.egDatPkg-method *(DatPkg-class)*, 14
ID2EntrezID,YeastDatPkg-method *(DatPkg-class)*, 14
ID2GO *(DatPkg-class)*, 14
ID2GO,DatPkg-method *(DatPkg-class)*, 14
initialize,HyperGParams-method *(HyperGParams-class)*, 22
isConditional *(HyperGParams-class)*, 22
isConditional,ChrMapHyperGParams-method *(ChrMapHyperGParams-class)*, 10
isConditional,ChrMapHyperGResult-method *(HyperGResult-accessors)*, 24
isConditional,ChrMapLinearMResult-method *(ChrMapLinearMResult-class)*, 13
isConditional,GOHyperGParams-method *(GOHyperGParams-class)*, 20
isConditional,HyperGParams-method *(HyperGParams-class)*, 22
isConditional,HyperGResultBase-method *(HyperGResult-accessors)*, 24
isConditional,LinearMResultBase-method *(LinearMResultBase-class)*, 32

KEGGHyperGParams-class, *23*, *26*, *29*
KEGGHyperGParams-class, 29
KEGGHyperGResult-class *(HyperGResult-class)*, 27
KEGGPATHID2NAME, *20*

level2nodes *(ChrBandTree-class)*, 8
level2nodes,ChrBandTree,character-method *(ChrBandTree-class)*, 8
level2nodes,ChrBandTree,numeric-method *(ChrBandTree-class)*, 8
lgeneIds *(ChrBandTree-class)*, 8
lgeneIds,ChrBandTree-method *(ChrBandTree-class)*, 8
LinearMParams, *13*
LinearMParams-class, *34*
LinearMParams-class, 30
LinearMResult, *14*, *33*
LinearMResult-class, *34*
LinearMResult-class, 33
LinearMResultBase, *14*, *33*
LinearMResultBase-class, 32
linearMTest, *12–14*, *33*, 34, *34*
linearMTest,LinearMParams-method *(linearMTest)*, 34
local_test_factory, 35

makeChrBandGraph, 36, *39*
makeChrBandInciMat *(MAPAmat)*, 38
makeEBcontr, 37
makeValidParams, 38
makeValidParams,HyperGParams-method *(HyperGParams-class)*, 22
MAPAmat, 38

NewChrBandTree, 39

oddsRatios *(HyperGResult-accessors)*, 24
oddsRatios,ChrMapHyperGResult-method *(HyperGResult-accessors)*, 24
oddsRatios,HyperGResult-method *(HyperGResult-accessors)*, 24
ontology *(HyperGParams-class)*, 22
ontology,GOHyperGParams-method *(GOHyperGParams-class)*, 20
ontology,HyperGParams-method *(HyperGParams-class)*, 22
ontology<- *(HyperGParams-class)*, 22
ontology<-,GOHyperGParams,character-method *(GOHyperGParams-class)*, 20
Org.XX.egDatPkg-class *(DatPkg-class)*, 14

parentOf *(ChrBandTree-class)*, 8

parentOf,ChrBandTree,character-method (ChrBandTree-class), 8
PFAMHyperGParams-class (KEGGHyperGParams-class), 29
PFAMHyperGResult-class (HyperGResult-class), 27
probes2MAP, *39*, 40
probes2Path, *40*, 41
pvalueCutoff (HyperGResult-accessors), 24
pvalueCutoff,HyperGParams-method (HyperGParams-class), 22
pvalueCutoff,HyperGResultBase-method (HyperGResult-accessors), 24
pvalueCutoff,LinearMParams-method (LinearMParams-class), 30
pvalueCutoff,LinearMResultBase-method (LinearMResultBase-class), 32
pvalueCutoff<- (HyperGParams-class), 22
pvalueCutoff<-,HyperGParams-method (HyperGParams-class), 22
pvalueCutoff<-,LinearMParams-method (LinearMParams-class), 30
pvalues (HyperGResult-accessors), 24
pvalues,ChrMapHyperGResult-method (HyperGResult-accessors), 24
pvalues,HyperGResult-method (HyperGResult-accessors), 24
pvalues,LinearMResult-method (LinearMResult-class), 33

rowttests, *43*

show,ChrBandTree-method (ChrBandTree-class), 8
show,GOHyperGParams-method (GOHyperGParams-class), 20
show,HyperGParams-method (HyperGParams-class), 22
show,HyperGResultBase-method (HyperGResultBase-class), 26
show,LinearMParams-method (LinearMParams-class), 30
show,LinearMResultBase-method (LinearMResultBase-class), 32
sigCategories (HyperGResult-accessors), 24
sigCategories,HyperGResultBase-method (HyperGResult-accessors), 24
sigCategories,LinearMResultBase-method (LinearMResultBase-class), 32
summary,HyperGResultBase-method (HyperGResult-accessors), 24
summary,KEGGHyperGResult-method (HyperGResult-accessors), 24
summary,LinearMResultBase-method (LinearMResultBase-class), 32
summary,PFAMHyperGResult-method (HyperGResult-accessors), 24

testDirection (HyperGResult-accessors), 24
testDirection,HyperGParams-method (HyperGParams-class), 22
testDirection,HyperGResultBase-method (HyperGResult-accessors), 24
testDirection,LinearMParams-method (LinearMParams-class), 30
testDirection,LinearMResultBase-method (LinearMResultBase-class), 32
testDirection<- (HyperGParams-class), 22
testDirection<-,HyperGParams-method (HyperGParams-class), 22
testDirection<-,LinearMParams-method (LinearMParams-class), 30
testName (HyperGResult-accessors), 24
testName,HyperGResultBase-method (HyperGResult-accessors), 24
testName,LinearMResultBase-method (LinearMResultBase-class), 32

topdown_tree_visitor
(*tree_visitor*), 42
tree_visitor, 42
treeLevels (*ChrBandTree-class*), 8
treeLevels,ChrBandTree-method
(*ChrBandTree-class*), 8
ttperm, 43

universeBuilder, 44
universeBuilder,GOHyperGParams-method
(*universeBuilder*), 44
universeBuilder,KEGGHyperGParams-method
(*universeBuilder*), 44
universeBuilder,PFAMHyperGParams-method
(*universeBuilder*), 44
universeCounts
(*HyperGResult-accessors*),
24
universeCounts,HyperGResultBase-method
(*HyperGResult-accessors*),
24
universeCounts,LinearMResultBase-method
(*LinearMResultBase-class*),
32
universeGeneIds
(*HyperGParams-class*), 22
universeGeneIds,HyperGParams-method
(*HyperGParams-class*), 22
universeGeneIds,LinearMParams-method
(*LinearMParams-class*), 30
universeMappedCount
(*HyperGResult-accessors*),
24
universeMappedCount,HyperGResultBase-method
(*HyperGResult-accessors*),
24
universeMappedCount,LinearMResultBase-method
(*LinearMResultBase-class*),
32

YeastDatPkg-class (*DatPkg-class*),
14