Package 'SQMtools'

March 21, 2023

Title Analyze Results Generated by the 'SqueezeMeta' Pipeline

Version 1.6.2

Description 'SqueezeMeta' is a versatile pipeline for the automated analysis of metagenomics/metatranscriptomics data (<https://github.com/jtamames/SqueezeMeta>). This package provides functions loading 'SqueezeMeta' results into R, filtering them based on different criteria, and visualizing the results using basic plots. The 'SqueezeMeta' project (and any subsets of it generated by the different filtering functions) is parsed into a single object, whose different components (e.g. tables with the taxonomic or functional composition across samples, contig/gene abundance profiles) can be easily analyzed using other R packages such as 'vegan' or 'DESeq2'. The methods in this package are further described in Puente-Sánchez et al., (2020) <doi:10.1186/s12859-020-03703-2>.

Author Fernando Puente-Sánchez [aut, cre], Natalia García-García [aut]

Maintainer Fernando Puente-Sánchez <fernando.puente.sanchez@slu.se>

Depends R (>= 3.5.0)

biocViews

LazyDataCompression xz

Imports reshape2, ggplot2, pathview, data.table

Suggests vegan, DESeq2

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

BugReports https://github.com/jtamames/SqueezeMeta/issues

URL https://github.com/jtamames/SqueezeMeta

R topics documented:

combineSQM *Combine several SQM objects*

Description

Combine an arbitrary number of SQM objects into a single SQM object. The input objects must be subsets of the same original SQM object (i.e. from the same SqueezeMeta run). For combining results from different runs please check `combineSQMlite`.

Usage

```
combineSQM(
  ...,
  tax_source = "orfs",
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = TRUE,
  rescale_copy_number = TRUE
)
```

Arguments

`...` an arbitrary number of SQM objects. Alternatively, a single list containing an arbitrary number of SQM objects.

`tax_source` character. Features used for calculating aggregated abundances at the different taxonomic ranks. Either `"orfs"` or `"contigs"` (default `"orfs"`). If the objects being combined contain a subset of taxa or bins, this parameter can be set to `TRUE`.

`trusted_functions_only`
logical. If `TRUE`, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If `FALSE`, best hit annotations will be used (default `FALSE`).

`ignore_unclassified_functions`
logical. If `FALSE`, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If `TRUE`, they will be ignored (default `FALSE`).

`rescale_tpm` logical. If `TRUE`, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object (default `TRUE`).

`rescale_copy_number`
logical. If `TRUE`, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object with the highest RecA/RadA coverages. By default it is set to `TRUE`, which means that the returned copy numbers will represent the average copy number per function *in the genomes of the selected bins or contigs*. If any SQM objects that are being combined contain a functional subset rather than a contig/bins subset, this parameter should be set to `FALSE`.

Value

A SQM object

See Also

`subsetFun`, `subsetTax`, `combineSQMlite`

Examples

```
data(Hadza)
# Select Carbohydrate metabolism ORFs in Bacteroidetes,
#  and Amino acid metabolism ORFs in Proteobacteria
bact = subsetTax(Hadza, "phylum", "Bacteroidetes")
bact.carb = subsetFun(bact, "Carbohydrate metabolism")
proteo = subsetTax(Hadza, "phylum", "Proteobacteria")
```

```
proteo.amins = subsetFun(proteo, "Amino acid metabolism")
bact.carb_proteo.amins = combineSQM(bact.carb, proteo.amins, rescale_copy_number=FALSE)
```

combineSQMlite *Combine several SQM or SQMlite objects*

Description

Combine an arbitrary number of SQM or SQMlite objects into a single SQMlite object. This function accepts objects originating from different projects (i.e. different SqueezeMeta runs).

Usage

```
combineSQMlite(...)
```

Arguments

`...` an arbitrary number of SQM or SQMlite objects. Alternatively, a single list containing an arbitrary number of SQMlite objects.

Value

A SQMlite object

See Also

combineSQM

Examples

```
## Not run:
data(Hadza)
# Load data coming from a different run
other = loadSQMlite("/path/to/other/project/tables") # e.g. if the project was run using sqm_reads
# (We could also use loadSQM to load the data as long as the data comes from a SqueezeMeta run)
combined = combineSQMlite(Hadza, other)
# Now we can plot together the samples from Hadza and the second project
plotTaxonomy(combined, 'family')

## End(Not run)
```

exportKrona *Export the taxonomy of a SQM object into a Krona Chart*

Description

Generate a krona chart containing the full taxonomy from a SQM object.

Usage

```
exportKrona(SQM, output_name = NA)
```

Arguments

`SQM`	A SQM or SQMlite object.
`output_name`	character. Name of the output file containing the Krona charts in html format (default `"<project_name>.krona.html"`).

Details

Original code was kindly provided by Giuseppe D'Auria (dauria_giu@gva.es).

Value

No return value, but a krona chart is produced in the current working directory.

See Also

`plotTaxonomy` for plotting the most abundant taxa of a SQM object.

Examples

```
data(Hadza)
exportKrona(Hadza)
```

exportPathway *Export the functions of a SQM object into KEGG pathway maps*

Description

This function is a wrapper for the pathview package (Luo *et al.*, 2017. *Nucleic acids research*, 45:W501-W508). It will generate annotated KEGG pathway maps showing which reactions are present in the different samples. It will also generate legends with the color scales for each sample in separate png files.

Usage

```
exportPathway(
  SQM,
  pathway_id,
  count = "tpm",
  samples = NULL,
  split_samples = FALSE,
  sample_colors = NULL,
  log_scale = FALSE,
  fold_change_groups = NULL,
  fold_change_colors = NULL,
  max_scale_value = NULL,
  color_bins = 10,
  output_suffix = "pathview"
)
```

Arguments

`SQM` A SQM or SQMlite object.

`pathway_id` character. The five-number KEGG pathway identifier. A list of all pathway identifiers can be found in https://www.genome.jp/kegg/pathway.html.

`count` character. Either "abund" for raw abundances, "percent" for percentages, "bases" for raw base counts, "tpm" for TPM normalized values or "copy_number" for copy numbers (default "tpm"). Note that a given count type might not available in this object (e.g. TPM or copy number in SQMlite objects originating from a SQM reads project).

`samples` character. An optional vector with the names of the samples to export. If absent, all samples will be exported (default `NULL`).

`split_samples` logical. Generate a different output file for each sample (default `FALSE`).

`sample_colors` character. An optional vector with the plotting colors for each sample (default `NULL`).

`log_scale` logical. Use a base 10 logarithmic transformation for the color scale. Will have no effect if `fold_change_groups` is provided (default `FALSE`).

`fold_change_groups`
list. An optional list containing two vectors of samples. If provided, the function will generate a single plot displaying the log2 fold-change between the average abundances of both groups of samples (log(second group / first group)) (default `NULL`).

`fold_change_colors`
character. An optional vector with the plotting colors of both groups in the fold-change plot. Will be ignored if `fold_change_group` is not provided.

`max_scale_value`
numeric. Maximum value to include in the color scale. By default it is the maximum value in the selected samples (if plotting abundances in samples) or the maximum absolute log2 fold-change (if plotting fold changes) (default `NULL`).

color_bins numeric. Number of bins used to generate the gradient in the color scale (default 10).

output_suffix character. Suffix to be added to the output files (default "pathview").

Value

No return value, but Pathview figures are produced in the current working directory.

See Also

plotFunctions for plotting the most functions taxa of a SQM object.

Examples

```
data(Hadza)
exportPathway(Hadza, "00910", count = 'copy_number',
              output_suffix = "nitrogen_metabolism", sample_colors = c("red", "blue"))
exportPathway(Hadza, "00250", count = 'tpm',
              output_suffix = "ala_asp_glu_metabolism_FoldChange",
              fold_change_groups = list(c("H1"), c("H12")), max_scale_value=2)
```

exportTable *Export results in tabular format*

Description

This function is a wrapper for R's write.table function.

Usage

```
exportTable(table, output_name)
```

Arguments

table vector, matrix or data.frame. The table to be written.

output_name character. Name of the output file.

Value

No return value, but a table is produced in the current working directory.

Examples

```
data(Hadza)
Hadza.iron = subsetFun(Hadza, "iron")
# Write the taxonomic distribution at the genus level of all the genes related to iron.
exportTable(Hadza.iron$taxa$genus$percent, "Hadza.ironGenes.genus.tsv")
# Now write the distribution of the different iron-related COGs
#  (Clusters of Orthologous Groups) across samples.
exportTable(Hadza.iron$functions$COG$tpm, "Hadza.ironGenes.COG.tsv")
# Now write all the information contained in the ORF table.
exportTable(Hadza.iron$orfs$table, "Hadza.ironGenes.orftable.tsv")
```

Hadza *Hadza hunter-gatherer gut metagenomes*

Description

Subset of two bins (and the associated contigs and genes) generated by running SqueezeMeta on two gut metagenomic samples obtained from two hunter-gatherers of the Hadza ethnic group.

Usage

```
data(Hadza)
```

Format

A SQM object; see `loadSQM`.

Source

SRR1927149, SRR1929485.

References

Rampelli *et al.*, 2015. Metagenome Sequencing of the Hadza Hunter-Gatherer Gut Microbiota. *Curr. biol.* **25**:1682-93 (PubMed).

Examples

```
data(Hadza)
plotTaxonomy(Hadza, "genus", rescale=TRUE)
plotFunctions(Hadza, "COG")
```

loadSQM *Load a SqueezeMeta project into R*

Description

This function takes the path to a project directory generated by SqueezeMeta (whose name is specified in the -p parameter of the SqueezeMeta.pl script) and parses the results into a SQM object. Alternatively, it can load the project data from a zip file produced by sqm2zip.py.

Usage

```
loadSQM(
  project_path,
  tax_mode = "prokfilter",
  trusted_functions_only = FALSE,
  engine = "data.table"
)
```

Arguments

project_path — character, project directory generated by SqueezeMeta, or zip file generated by sqm2zip.py.

tax_mode — character, which taxonomic classification should be loaded? SqueezeMeta applies the identity thresholds described in Luo *et al.*, 2014. Use allfilter for applying the minimum identity threshold to all taxa, prokfilter for applying the threshold to Bacteria and Archaea, but not to Eukaryotes, and nofilter for applying no thresholds at all (default prokfilter).

trusted_functions_only — logical. If TRUE, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If FALSE, best hit annotations will be used (default FALSE). Will only have an effect if the project_dir/results/tables is not already present.

engine — character. Engine used to load the ORFs and contigs tables. Either data.frame or data.table (significantly faster if your project is large). Default data.table.

Value

SQM object containing the parsed project.

Prerequisites

Run SqueezeMeta! An example call for running it would be:

```
/path/to/SqueezeMeta/scripts/SqueezeMeta.pl
-m coassembly -f fastq_dir -s samples_file -p project_dir
```

The SQM object structure

The SQM object is a nested list which contains the following information:

lvl1	**lvl2**	**lvl3**	**type**	**rows/names**	**columns**	**data**
$orfs	**$table**		*dataframe*	orfs	misc. data	misc. data
	$abund		*numeric matrix*	orfs	samples	abundances
	$bases		*numeric matrix*	orfs	samples	abundances
	$cov		*numeric matrix*	orfs	samples	coverages
	$cpm		*numeric matrix*	orfs	samples	covs. / 10^6
	$tpm		*numeric matrix*	orfs	samples	tpm
	$seqs		*character vector*	orfs	(n/a)	sequences
	$tax		*character matrix*	orfs	tax. ranks	taxonomy
$contigs	**$table**		*dataframe*	contigs	misc. data	misc. data
	$abund		*numeric matrix*	contigs	samples	abundances
	$bases		*numeric matrix*	contigs	samples	abundances
	$cov		*numeric matrix*	contigs	samples	coverages
	$cpm		*numeric matrix*	contigs	samples	covs. / 10^6
	$tpm		*numeric matrix*	contigs	samples	tpm
	$seqs		*character vector*	contigs	(n/a)	sequences
	$tax		*character matrix*	contigs	tax. ranks	taxonomies
	$bins		*character matrix*	contigs	bin. methods	bins
$bins	**$table**		*dataframe*	bins	misc. data	misc. data
	$length		*numeric vector*	bins	(n/a)	length
	$abund		*numeric matrix*	bins	samples	abundances
	$percent		*numeric matrix*	bins	samples	abundances
	$bases		*numeric matrix*	bins	samples	abundances
	$cov		*numeric matrix*	bins	samples	coverages
	$cpm		*numeric matrix*	bins	samples	covs. / 10^6
	$tax		*character matrix*	bins	tax. ranks	taxonomy
$taxa	**$superkingdom**	**$abund**	*numeric matrix*	superkingdoms	samples	abundances
		$percent	*numeric matrix*	superkingdoms	samples	percentages
	$phylum	**$abund**	*numeric matrix*	phyla	samples	abundances
		$percent	*numeric matrix*	phyla	samples	percentages
	$class	**$abund**	*numeric matrix*	classes	samples	abundances
		$percent	*numeric matrix*	classes	samples	percentages
	$order	**$abund**	*numeric matrix*	orders	samples	abundances
		$percent	*numeric matrix*	orders	samples	percentages
	$family	**$abund**	*numeric matrix*	families	samples	abundances
		$percent	*numeric matrix*	families	samples	percentages
	$genus	**$abund**	*numeric matrix*	genera	samples	abundances
		$percent	*numeric matrix*	genera	samples	percentages
	$species	**$abund**	*numeric matrix*	species	samples	abundances
		$percent	*numeric matrix*	species	samples	percentages
$functions	**$KEGG**	**$abund**	*numeric matrix*	KEGG ids	samples	abundances
		$bases	*numeric matrix*	KEGG ids	samples	abundances
		$cov	*numeric matrix*	KEGG ids	samples	coverages
		$cpm	*numeric matrix*	KEGG ids	samples	covs. / 10^6
		$tpm	*numeric matrix*	KEGG ids	samples	tpm
		$copy_number	*numeric matrix*	KEGG ids	samples	avg. copies
	$COG	**$abund**	*numeric matrix*	COG ids	samples	abundances
		$bases	*numeric matrix*	COG ids	samples	abundances

		$cov	*numeric matrix*	COG ids	samples	coverages
		$cpm	*numeric matrix*	COG ids	samples	covs. / 10^6
		$tpm	*numeric matrix*	COG ids	samples	tpm
		$copy_number	*numeric matrix*	COG ids	samples	avg. copies
	$PFAM	**$abund**	*numeric matrix*	PFAM ids	samples	abundances
		$bases	*numeric matrix*	PFAM ids	samples	abundances
		$cov	*numeric matrix*	PFAM ids	samples	coverages
		$cpm	*numeric matrix*	PFAM ids	samples	covs. / 10^6
		$tpm	*numeric matrix*	PFAM ids	samples	tpm
		$copy_number	*numeric matrix*	PFAM ids	samples	avg. copies
$total_reads			*numeric vector*	samples	(n/a)	total reads
$misc	**$project_name**		*character vector*	(empty)	(n/a)	project nam
	$samples		*character vector*	(empty)	(n/a)	samples
	$tax_names_long	**$superkingdom**	*character vector*	short names	(n/a)	full names
		$phylum	*character vector*	short names	(n/a)	full names
		$class	*character vector*	short names	(n/a)	full names
		$order	*character vector*	short names	(n/a)	full names
		$family	*character vector*	short names	(n/a)	full names
		$genus	*character vector*	short names	(n/a)	full names
		$species	*character vector*	short names	(n/a)	full names
	$tax_names_short		*character vector*	full names	(n/a)	short names
	$KEGG_names		*character vector*	KEGG ids	(n/a)	KEGG nam
	$KEGG_paths		*character vector*	KEGG ids	(n/a)	KEGG hiara
	$COG_names		*character vector*	COG ids	(n/a)	COG names
	$COG_paths		*character vector*	COG ids	(n/a)	COG hierar
	$ext_annot_sources		*character vector*	COG ids	(n/a)	external data

If external databases for functional classification were provided to SqueezeMeta via the `-extdb` argument, the corresponding abundance (reads and bases), coverages, tpm and copy number profiles will be present in `SQM$functions` (e.g. results for the CAZy database would be present in `SQM$functions$CAZy`). Additionally, the extended names of the features present in the external database will be present in `SQM$misc` (e.g. `SQM$misc$CAZy_names`).

Examples

```
## Not run:
## (outside R)
## Run SqueezeMeta on the test data.
 /path/to/SqueezeMeta/scripts/SqueezeMeta.pl -p Hadza -f raw -m coassembly -s test.samples
## Now go into R.
library(SQMtools)
Hadza = loadSQM("Hadza") # Where Hadza is the path to the SqueezeMeta output directory.

## End(Not run)

data(Hadza) # We will illustrate the structure of the SQM object on the test data
# Which are the ten most abundant KEGG IDs in our data?
topKEGG = names(sort(rowSums(Hadza$functions$KEGG$tpm), decreasing=TRUE))[1:11]
topKEGG = topKEGG[topKEGG!="Unclassified"]
```

```
# Which functions do those KEGG IDs represent?
Hadza$misc$KEGG_names[topKEGG]
# What is the relative abundance of the Gammaproteobacteria class across samples?
Hadza$taxa$class$percent["Gammaproteobacteria",]
# Which information is stored in the orf, contig and bin tables?
colnames(Hadza$orfs$table)
colnames(Hadza$contigs$table)
colnames(Hadza$bins$table)
# What is the GC content distribution of my metagenome?
boxplot(Hadza$contigs$table[,"GC perc"]) # Not weighted by contig length or abundance!
```

loadSQMlite *Load tables generated by* `sqm2tables.py`, `sqmreads2tables.py` *or* `combine-sqm-tables.py` *into R.*

Description

This function takes the path to the output directory generated by `sqm2tables.py`, `sqmreads2tables.py` or `combine-sqm-tables.py` a SQMlite object. The SQMlite object will contain taxonomic and functional profiles, but no detailed information on ORFs, contigs or bins. However, it will also have a much smaller memory footprint. A SQMlite object can be used for plotting and exporting, but it can not be subsetted.

Usage

```
loadSQMlite(tables_path, tax_mode = "allfilter")
```

Arguments

`tables_path`	character, tables directory generated by `sqm2table.py`, `sqmreads2tables.py` or `combine-sqm-tables.py`.
`tax_mode`	character, which taxonomic classification should be loaded? SqueezeMeta applies the identity thresholds described in Luo *et al.*, 2014. Use `allfilter` for applying the minimum identity threshold to all taxa (default), `prokfilter` for applying the threshold to Bacteria and Archaea, but not to Eukaryotes, and `nofilter` for applying no thresholds at all.

Value

SQMlite object containing the parsed tables.

The SQMlite object structure

The SQMlite object is a nested list which contains the following information:

lvl1	**lvl2**	**lvl3**	**type**	**rows/names**	**columns**	**data**
$taxa	**$superkingdom**	**$abund**	*numeric matrix*	superkingdoms	samples	abundances
		$percent	*numeric matrix*	superkingdoms	samples	percentages

	$phylum	**$abund**	*numeric matrix*	phyla	samples	abundances
		$percent	*numeric matrix*	phyla	samples	percentages
	$class	**$abund**	*numeric matrix*	classes	samples	abundances
		$percent	*numeric matrix*	classes	samples	percentages
	$order	**$abund**	*numeric matrix*	orders	samples	abundances
		$percent	*numeric matrix*	orders	samples	percentages
	$family	**$abund**	*numeric matrix*	families	samples	abundances
		$percent	*numeric matrix*	families	samples	percentages
	$genus	**$abund**	*numeric matrix*	genera	samples	abundances
		$percent	*numeric matrix*	genera	samples	percentages
	$species	**$abund**	*numeric matrix*	species	samples	abundances
		$percent	*numeric matrix*	species	samples	percentages
$functions	**$KEGG**	**$abund**	*numeric matrix*	KEGG ids	samples	abundances (reac
		$bases	*numeric matrix*	KEGG ids	samples	abundances (base
		$tpm	*numeric matrix*	KEGG ids	samples	tpm
		$copy_number	*numeric matrix*	KEGG ids	samples	avg. copies
	$COG	**$abund**	*numeric matrix*	COG ids	samples	abundances (reac
		$bases	*numeric matrix*	COG ids	samples	abundances (base
		$tpm	*numeric matrix*	COG ids	samples	tpm
		$copy_number	*numeric matrix*	COG ids	samples	avg. copies
	$PFAM	**$abund**	*numeric matrix*	PFAM ids	samples	abundances (reac
		$bases	*numeric matrix*	PFAM ids	samples	abundances (base
		$tpm	*numeric matrix*	PFAM ids	samples	tpm
		$copy_number	*numeric matrix*	PFAM ids	samples	avg. copies
$total_reads			*numeric vector*	samples	(n/a)	total reads
$misc	**$project_name**		*character vector*	(empty)	(n/a)	project name
	$samples		*character vector*	(empty)	(n/a)	samples
	$tax_names_long	**$superkingdom**	*character vector*	short names	(n/a)	full names
		$phylum	*character vector*	short names	(n/a)	full names
		$class	*character vector*	short names	(n/a)	full names
		$order	*character vector*	short names	(n/a)	full names
		$family	*character vector*	short names	(n/a)	full names
		$genus	*character vector*	short names	(n/a)	full names
		$species	*character vector*	short names	(n/a)	full names
	$tax_names_short		*character vector*	full names	(n/a)	short names
	$KEGG_names		*character vector*	KEGG ids	(n/a)	KEGG names
	$KEGG_paths		*character vector*	KEGG ids	(n/a)	KEGG hiararchy
	$COG_names		*character vector*	COG ids	(n/a)	COG names
	$COG_paths		*character vector*	COG ids	(n/a)	COG hierarchy
	$ext_annot_sources		*character vector*	(empty)	(n/a)	external database

If external databases for functional classification were provided to SqueezeMeta or SqueezeMeta_reads via the `-extdb` argument, the corresponding abundance, tpm and copy number profiles will be present in `SQM$functions` (e.g. results for the CAZy database would be present in `SQM$functions$CAZy`). Additionally, the extended names of the features present in the external database will be present in `SQM$misc` (e.g. `SQM$misc$CAZy_names`). Note that results generated by SqueezeMeta_reads will contain only read abundances, but not bases, tpm or copy number estimations.

See Also

`plotBars` and `plotFunctions` will plot the most abundant taxa and functions in a SQMlite object. `exportKrona` will generate Krona charts reporting the taxonomy in a SQMlite object.

Examples

```
## Not run:
## (outside R)
## Run SqueezeMeta on the test data.
/path/to/SqueezeMeta/scripts/SqueezeMeta.pl -p Hadza -f raw -m coassembly -s test.samples
## Generate the tabular outputs!
/path/to/SqueezeMeta/utils/sqm2tables.py Hadza Hadza/results/tables
## Now go into R.
library(SQMtools)
Hadza = loadSQMlite("Hadza/results/tables")
# Where Hadza is the path to the SqueezeMeta output directory.
# Note that this is not the whole SQM project, just the directory containing the tables.
# It would also work with tables generated by sqmreads2tables.py, or combine-sqm-tables.py
plotTaxonomy(Hadza)
plotFunctions(Hadza)
exportKrona(Hadza, 'myKronaTest.html')

## End(Not run)
```

`MGKOs` *Single Copy Phylogenetic Marker Genes from Sunagawa's group (KOs)*

Description

Lists of Single Copy Phylogenetic Marker Genes. These are useful for transforming coverages or tpms into copy numbers. This is an alternative way of normalizing data in order to be able to compare functional profiles in samples with different sequencing depths.

Usage

```
data(MGKOs)
```

Format

Character vector with the KEGG identifiers for 10 Single Copy Phylogenetic Marker Genes.

References

Salazar, G *et al.* (2019). Gene Expression Changes and Community Turnover Differentially Shape the Global Ocean Metatranscriptome *Cell* **179**:1068-1083. (PubMed).

See Also

MGOGs for an equivalent list using OGs instead of KOs; USiCGs for an alternative set of single copy genes, and for examples on how to generate copy numbers.

MGOGs *Single Copy Phylogenetic Marker Genes from Sunagawa's group (OGs)*

Description

Lists of Single Copy Phylogenetic Marker Genes. These are useful for transforming coverages or tpms into copy numbers. This is an alternative way of normalizing data in order to be able to compare functional profiles in samples with different sequencing depths.

Usage

```
data(MGOGs)
```

Format

Character vector with the COG identifiers for 10 Single Copy Phylogenetic Marker Genes.

References

Salazar, G *et al.* (2019). Gene Expression Changes and Community Turnover Differentially Shape the Global Ocean Metatranscriptome *Cell* **179**:1068-1083. (PubMed).

See Also

MGKOs for an equivalent list using KOs instead of OGs; USiCGs for an alternative set of single copy genes, and for examples on how to generate copy numbers.

---

mostAbundant *Get the N most abundant rows (or columns) from a numeric table*

Description

Return a subset of an input matrix or data frame, containing only the N most abundant rows (or columns), sorted. Alternatively, a custom set of rows can be returned.

Usage

```
mostAbundant(
  data,
  N = 10,
  items = NULL,
  others = FALSE,
  rescale = FALSE,
  bycol = FALSE
)
```

Arguments

`data`	numeric matrix or data frame
`N`	integer Number of rows to return (default `10`).
`items`	Character vector. Custom row names to return. If provided, it will override `N` (default `NULL`).
`others`	logical. If `TRUE`, an extra row will be returned containing the aggregated abundances of the elements not selected with `N` or `items` (default `FALSE`).
`rescale`	logical. Scale result to percentages column-wise (default `FALSE`).
`bycol`	logical. Operate on columns instead of rows (default `FALSE`).

Value

A matrix or data frame (same as input) with the selected rows (or columns).

Examples

```
data(Hadza)
Hadza.carb = subsetFun(Hadza, "Carbohydrate metabolism")
# Which are the 20 most abundant KEGG functions in the ORFs related to carbohydrate metabolism?
topCarb = mostAbundant(Hadza.carb$functions$KEGG$tpm, N=20)
# Now print them with nice names.
rownames(topCarb) = paste(rownames(topCarb),
                         Hadza.carb$misc$KEGG_names[rownames(topCarb)], sep="; ")
topCarb
# We can pass this to any R function.
heatmap(topCarb)
# But for convenience we provide wrappers for plotting ggplot2 heatmaps and barplots.
plotHeatmap(topCarb, label_y="TPM")
plotBars(topCarb, label_y="TPM")
```

mostVariable *Get the N most variable rows (or columns) from a numeric table*

Description

Return a subset of an input matrix or data frame, containing only the N most variable rows (or columns), sorted. Variability is calculated as the Coefficient of Variation (sd/mean).

Usage

```
mostVariable(data, N = 10, bycol = FALSE)
```

Arguments

data	numeric matrix or data frame
N	integer Number of rows to return (default 10).
bycol	logical. Operate on columns instead of rows (default FALSE).

Value

A matrix or data frame (same as input) with the selected rows or columns.

Examples

```
data(Hadza)
Hadza.carb = subsetFun(Hadza, "Carbohydrate metabolism")
# Which are the 20 most variable KEGG functions in the ORFs related to carbohydrate metabolism?
topCarb = mostVariable(Hadza.carb$functions$KEGG$tpm, N=20)
# Now print them with nice names
rownames(topCarb) = paste(rownames(topCarb),
                          Hadza.carb$misc$KEGG_names[rownames(topCarb)], sep="; ")
topCarb
# We can pass this to any R function
heatmap(topCarb)
# But for convenience we provide wrappers for plotting ggplot2 heatmaps and barplots
plotHeatmap(topCarb, label_y="TPM")
plotBars(topCarb, label_y="TPM")
```

plotBars *Plot a barplot using ggplot2*

Description

Plot a ggplot2 barplot from a matrix or data frame. The data should be in tabular format (e.g. features in rows and samples in columns).

Usage

```
plotBars(
  data,
  label_x = "Samples",
  label_y = "Abundances",
  label_fill = "Features",
  color = NULL,
  base_size = 11,
  max_scale_value = NULL,
  metadata_groups = NULL
)
```

Arguments

`data` Numeric matrix or data frame.

`label_x` character Label for the x axis (default `"Samples"`).

`label_y` character Label for the y axis (default `"Abundances"`).

`label_fill` character Label for color categories (default `"Features"`).

`color` Vector with custom colors for the different features. If empty, the default ggplot2 palette will be used (default `NULL`).

`base_size` numeric. Base font size (default `11`).

`max_scale_value`

numeric. Maximum value to include in the y axis. By default it is handled automatically by ggplot2 (default `NULL`).

`metadata_groups`

list. Split the plot into groups defined by the user: list(’G1’ = c(’sample1’, sample2’), ’G2’ = c(’sample3’, ’sample4’)) default `NULL`).

Value

a ggplot2 plot object.

See Also

`plotTaxonomy` for plotting the most abundant taxa of a SQM object; `plotHeatmap` for plotting a heatmap with arbitrary data; `mostAbundant` for selecting the most abundant rows in a dataframe or matrix.

Examples

```
data(Hadza)
sk = Hadza$taxa$superkingdom$abund
plotBars(sk, label_y = "Raw reads", label_fill = "Superkingdom")
```

plotBins *Barplot of the most abundant bins in a SQM object*

Description

This function selects the most abundant bins across all samples in a SQM object and represents their abundances in a barplot. Alternatively, a custom set of bins can be represented.

Usage

```
plotBins(
  SQM,
  count = "percent",
  N = 15,
  bins = NULL,
  others = TRUE,
  samples = NULL,
  ignore_unmapped = FALSE,
  ignore_nobin = FALSE,
  rescale = FALSE,
  color = NULL,
  base_size = 11,
  max_scale_value = NULL,
  metadata_groups = NULL
)
```

Arguments

`SQM`	A SQM or a SQMlite object.
`count`	character. Either `"abund"` for raw abundances, `"percent"` for percentages, `"cov"` for coverages, or `"cpm"` for coverages per million reads (default `"percent"`).
`N`	integer Plot the `N` most abundant bins (default `15`).
`bins`	character. Custom bins to plot. If provided, it will override `N` (default `NULL`).
`others`	logical. Collapse the abundances of least abundant bins, and include the result in the plot (default `TRUE`).
`samples`	character. Character vector with the names of the samples to include in the plot. Can also be used to plot the samples in a custom order. If not provided, all samples will be plotted (default `NULL`).
`ignore_unmapped`	logical. Don't include unmapped reads in the plot (default `FALSE`).
`ignore_nobin`	logical. Don't include reads which are not in a bin in the plot (default `FALSE`).
`rescale`	logical. Re-scale results to percentages (default `FALSE`).
`color`	Vector with custom colors for the different features. If empty, we will use our own hand-picked pallete if N<=15, and the default ggplot2 palette otherwise (default `NULL`).

`base_size` numeric. Base font size (default 11).

`max_scale_value`
numeric. Maximum value to include in the y axis. By default it is handled automatically by ggplot2 (default NULL).

`metadata_groups`
list. Split the plot into groups defined by the user: list('G1' = c('sample1', sample2'), 'G2' = c('sample3', 'sample4')) default NULL).

Value

a ggplot2 plot object.

See Also

plotBins for plotting the most abundant bins of a SQM object; plotBars and plotHeatmap for plotting barplots or heatmaps with arbitrary data.

Examples

```
data(Hadza)
# Bins distribution.
plotBins(Hadza)
```

plotFunctions *Heatmap of the most abundant functions in a SQM object*

Description

This function selects the most abundant functions across all samples in a SQM object and represents their abundances in a heatmap. Alternatively, a custom set of functions can be represented.

Usage

```
plotFunctions(
  SQM,
  fun_level = "KEGG",
  count = "tpm",
  N = 25,
  fun = NULL,
  samples = NULL,
  ignore_unmapped = TRUE,
  ignore_unclassified = TRUE,
  gradient_col = c("ghostwhite", "dodgerblue4"),
  base_size = 11,
  metadata_groups = NULL
)
```

Arguments

SQM A SQM or SQMlite object.

fun_level character. Either "KEGG", "COG", "PFAM" or any other custom database used for annotation (default "KEGG").

count character. Either "abund" for raw abundances, "percent" for percentages, "bases" for raw base counts, "cpm" for coverages per million reads, "tpm" for TPM normalized values or "copy_number" for copy numbers (default "tpm"). Note that a given count type might not available in this object (e.g. TPM or copy number in SQMlite objects originating from a SQM reads project).

N integer Plot the N most abundant functions (default 25).

fun character. Custom functions to plot. If provided, it will override N (default NULL).

samples character. Character vector with the names of the samples to include in the plot. Can also be used to plot the samples in a custom order. If not provided, all samples will be plotted (default NULL).

ignore_unmapped
logical. Don’t include unmapped reads in the plot (default TRUE).

ignore_unclassified
logical. Don’t include unclassified ORFs in the plot (default TRUE).

gradient_col A vector of two colors representing the low and high ends of the color gradient (default c("ghostwhite","dodgerblue4")).

base_size numeric. Base font size (default 11).

metadata_groups
list. Split the plot into groups defined by the user: list(’G1’ = c(’sample1’, sample2’), ’G2’ = c(’sample3’, ’sample4’)) default NULL).

Value

a ggplot2 plot object.

See Also

plotTaxonomy for plotting the most abundant taxa of a SQM object; plotBars and plotHeatmap for plotting barplots or heatmaps with arbitrary data.

Examples

```
data(Hadza)
plotFunctions(Hadza)
```

plotHeatmap *Plot a heatmap using ggplot2*

Description

Plot a ggplot2 heatmap from a matrix or data frame. The data should be in tabular format (e.g. features in rows and samples in columns).

Usage

```
plotHeatmap(
  data,
  label_x = "Samples",
  label_y = "Features",
  label_fill = "Abundance",
  gradient_col = c("ghostwhite", "dodgerblue4"),
  base_size = 11,
  metadata_groups = NULL
)
```

Arguments

data	numeric matrix or data frame.
label_x	character Label for the x axis (default "Samples").
label_y	character Label for the y axis (default "Features").
label_fill	character Label for color scale (default "Abundance").
gradient_col	A vector of two colors representing the low and high ends of the color gradient (default c("ghostwhite","dodgerblue4")).
base_size	numeric. Base font size (default 11).
metadata_groups	list. Split the plot into groups defined by the user: list('G1' = c('sample1', sample2'), 'G2' = c('sample3', 'sample4')) default NULL).

Value

A ggplot2 plot object.

See Also

plotFunctions for plotting the top functional categories of a SQM object; plotBars for plotting a barplot with arbitrary data; mostAbundant for selecting the most abundant rows in a dataframe or matrix.

Examples

```
data(Hadza)
topPFAM = mostAbundant(Hadza$functions$PFAM$tpm)
topPFAM = topPFAM[rownames(topPFAM) != "Unclassified",] # Take out the Unclassified ORFs.
plotHeatmap(topPFAM, label_x = "Samples", label_y = "PFAMs", label_fill = "TPM")
```

plotTaxonomy *Barplot of the most abundant taxa in a SQM object*

Description

This function selects the most abundant taxa across all samples in a SQM object and represents their abundances in a barplot. Alternatively, a custom set of taxa can be represented.

Usage

```
plotTaxonomy(
  SQM,
  rank = "phylum",
  count = "percent",
  N = 15,
  tax = NULL,
  others = TRUE,
  samples = NULL,
  nocds = "treat_separately",
  ignore_unmapped = FALSE,
  ignore_unclassified = FALSE,
  no_partial_classifications = FALSE,
  rescale = FALSE,
  color = NULL,
  base_size = 11,
  max_scale_value = NULL,
  metadata_groups = NULL
)
```

Arguments

`SQM` A SQM or a SQMlite object.

`rank` Taxonomic rank to plot (default `phylum`).

`count` character. Either `"percent"` for percentages, or `"abund"` for raw abundances (default `"percent"`).

`N` integer Plot the `N` most abundant taxa (default `15`).

`tax` character. Custom taxa to plot. If provided, it will override `N` (default `NULL`).

`others` logical. Collapse the abundances of least abundant taxa, and include the result in the plot (default `TRUE`).

samples: character. Character vector with the names of the samples to include in the plot. Can also be used to plot the samples in a custom order. If not provided, all samples will be plotted (default NULL).

nocds: character. Either "treat_separately" to treat reads annotated as No CDS separately, "treat_as_unclassified" to treat them as Unclassified or "ignore" to ignore them in the plot (default "treat_separately").

ignore_unmapped: logical. Don't include unmapped reads in the plot (default FALSE).

ignore_unclassified: logical. Don't include unclassified reads in the plot (default FALSE).

no_partial_classifications: logical. Treat reads not fully classified at the requested level (e.g. "Unclassified bacteroidetes" at the class level or below) as fully unclassified. This takes effect before ignore_unclassified, so if both are TRUE the plot will only contain fully classified contigs (default FALSE).

rescale: logical. Re-scale results to percentages (default FALSE).

color: Vector with custom colors for the different features. If empty, we will use our own hand-picked pallete if N<=15, and the default ggplot2 palette otherwise (default NULL).

base_size: numeric. Base font size (default 11).

max_scale_value: numeric. Maximum value to include in the y axis. By default it is handled automatically by ggplot2 (default NULL).

metadata_groups: list. Split the plot into groups defined by the user: list('G1' = c('sample1', sample2'), 'G2' = c('sample3', 'sample4')) default NULL).

Value

a ggplot2 plot object.

See Also

plotFunctions for plotting the most abundant functions of a SQM object; plotBars and plotHeatmap for plotting barplots or heatmaps with arbitrary data.

Examples

```
data(Hadza)
Hadza.amin = subsetFun(Hadza, "Amino acid metabolism")
# Taxonomic distribution of amino acid metabolism ORFs at the family level.
plotTaxonomy(Hadza.amin, "family")
```

RecA *RecA/RadA recombinase*

Description

The recombination protein RecA/RadA is essential for the repair and maintenance of DNA, and has homologs in every bacteria and archaea. By dividing the coverage of functions by the coverage of RecA, abundances can be transformed into copy numbers, which can be used to compare functional profiles in samples with different sequencing depths. RecA-derived copy numbers are available in the SQM object (`SQM$functions$<annotation_type>$copy_number`).

Usage

```
data(RecA)
```

Format

Character vector with the COG identifier for RecA/RadA.

Source

EggNOG Database.

Examples

```
data(Hadza)
data(RecA)
### Let's calculate the average copy number of each function in our samples.
# We do it for COG annotations here, but we could also do it for KEGG or PFAMs.
COG.coverage = Hadza$functions$COG$cov
COG.copynumber = t(t(COG.coverage) / COG.coverage[RecA,]) # Sample-wise division by RecA coverage.
```

rowMaxs *Return a vector with the row-wise maxima of a matrix or dataframe.*

Description

Return a vector with the row-wise maxima of a matrix or dataframe.

Usage

```
rowMaxs(table)
```

Arguments

`table`	matrix or dataframe.

Value

a vector with the row-wise maxima.

rowMins *Return a vector with the row-wise minima of a matrix or dataframe.*

Description

Return a vector with the row-wise minima of a matrix or dataframe.

Usage

```
rowMins(table)
```

Arguments

`table`	matrix or dataframe.

Value

a vector with the row-wise minima.

seqvec2fasta *Print a named vector of sequences as a fasta-formatted string*

Description

Print a named vector of sequences as a fasta-formatted string

Usage

```
seqvec2fasta(seqvec)
```

Arguments

`seqvec`	vector. The vector to be written as a fasta string.

Value

a character string with the sequence/s written in the fasta format.

Examples

```
data(Hadza)
seqvec2fasta(Hadza$orfs$seqs[1:10])
```

subsetBins — *Create a SQM object containing only the requested bins, and the contigs and ORFs contained in them.*

Description

Create a SQM object containing only the requested bins, and the contigs and ORFs contained in them.

Usage

```
subsetBins(
  SQM,
  bins,
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = TRUE,
  rescale_copy_number = TRUE
)
```

Arguments

`SQM` SQM object to be subsetted.

`bins` character. Vector of bins to be selected.

`trusted_functions_only` logical. If `TRUE`, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If `FALSE`, best hit annotations will be used (default `FALSE`).

`ignore_unclassified_functions` logical. If `FALSE`, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If `TRUE`, they will be ignored (default `FALSE`).

`rescale_tpm` logical. If `TRUE`, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object. By default it is set to `TRUE`, which means that the returned TPMs will be scaled *by million of reads of the selected bins*.

`rescale_copy_number` logical. If `TRUE`, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object. By default it is set to `TRUE`, which means that the returned copy numbers for each function will represent the average copy number of that function *per genome of the selected bins*.

Value

SQM object containing only the requested bins.

See Also

`subsetContigs`, `subsetORFs`

Examples

```
data(Hadza)
# Which are the most complete bins?
topBinNames = rownames(Hadza$bins$table)[order(Hadza$bins$table[,"Completeness"],
                                              decreasing=TRUE)][1:2]
# Subset with the most complete bin.
topBin = subsetBins(Hadza, topBinNames[1])
```

---

subsetContigs *Select contigs*

Description

Create a SQM object containing only the requested contigs, the ORFs contained in them and the bins that contain them.

Usage

```
subsetContigs(
  SQM,
  contigs,
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = FALSE,
  rescale_copy_number = FALSE
)
```

Arguments

`SQM` SQM object to be subsetted.

`contigs` character. Vector of contigs to be selected.

`trusted_functions_only`
: logical. If `TRUE`, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If `FALSE`, best hit annotations will be used (default `FALSE`).

`ignore_unclassified_functions`
: logical. If `FALSE`, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If `TRUE`, they will be ignored (default `FALSE`).

rescale_tpm logical. If TRUE, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object (default FALSE).

rescale_copy_number

logical. If TRUE, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object. By default it is set to FALSE, which means that the returned copy numbers for each function will represent the average copy number of that function per genome in the parent object.

Value

SQM object containing only the selected contigs.

See Also

subsetORFs

Examples

```
data(Hadza)
# Which contigs have a GC content below 40?
lowGCcontigNames = rownames(Hadza$contigs$table[Hadza$contigs$table[,"GC perc"]<40,])
lowGCcontigs = subsetContigs(Hadza, lowGCcontigNames)
hist(lowGCcontigs$contigs$table[,"GC perc"])
```

subsetFun *Filter results by function*

Description

Create a SQM object containing only the ORFs with a given function, and the contigs and bins that contain them.

Usage

```
subsetFun(
  SQM,
  fun,
  columns = NULL,
  ignore_case = TRUE,
  fixed = FALSE,
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = FALSE,
  rescale_copy_number = FALSE
)
```

Arguments

`SQM`	SQM object to be subsetted.
`fun`	character. Pattern to search for in the different functional classifications.
`columns`	character. Restrict the search to the provided column names from `SQM$orfs$table`. If not provided the search will be performed in all the columns containing functional information (default `NULL`).
`ignore_case`	logical Make pattern matching case-insensitive (default `TRUE`).
`fixed`	logical. If `TRUE`, pattern is a string to be matched as is. If `FALSE` the pattern is treated as a regular expression (default `FALSE`).
`trusted_functions_only`	logical. If `TRUE`, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If `FALSE`, best hit annotations will be used (default `FALSE`).
`ignore_unclassified_functions`	logical. If `FALSE`, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If `TRUE`, they will be ignored (default `FALSE`).
`rescale_tpm`	logical. If `TRUE`, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object (default `FALSE`).
`rescale_copy_number`	logical. If `TRUE`, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object. By default it is set to `FALSE`, which means that the returned copy numbers for each function will represent the average copy number of that function per genome in the parent object.

Value

SQM object containing only the requested function.

See Also

`subsetTax`, `subsetORFs`, `subsetSamples`, `combineSQM`. The most abundant items of a particular table contained in a SQM object can be selected with `mostAbundant`.

Examples

```
data(Hadza)
Hadza.iron = subsetFun(Hadza, "iron")
Hadza.carb = subsetFun(Hadza, "Carbohydrate metabolism")
# Search for multiple patterns using regular expressions
Hadza.twoKOs = subsetFun(Hadza, "K00812|K00813", fixed=FALSE)
```

subsetORFs *Select ORFs*

Description

Create a SQM object containing only the requested ORFs, and the contigs and bins that contain them. Internally, all the other subset functions in this package end up calling subsetORFs to do the work for them.

Usage

```
subsetORFs(
  SQM,
  orfs,
  tax_source = "orfs",
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = FALSE,
  rescale_copy_number = FALSE,
  contigs_override = NULL
)
```

Arguments

SQM SQM object to be subsetted.

orfs character. Vector of ORFs to be selected.

tax_source character. Features used for calculating aggregated abundances at the different taxonomic ranks. Either "orfs" or "contigs" (default "orfs").

trusted_functions_only
logical. If TRUE, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If FALSE, best hit annotations will be used (default FALSE).

ignore_unclassified_functions
logical. If FALSE, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If TRUE, they will be ignored (default FALSE).

rescale_tpm logical. If TRUE, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object (default FALSE).

rescale_copy_number
logical. If TRUE, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object. By default it is set to FALSE, which means that the returned copy numbers for each function will represent the average copy number of that function per genome in the parent object.

`contigs_override`
: character. Optional vector of contigs to be included in the subsetted object.

Value

SQM object containing the requested ORFs.

A note on contig/bins subsetting

While this function selects the contigs and bins that contain the desired orfs, it DOES NOT recalculate contig/bin abundance and statistics based on the selected ORFs only. This means that the abundances presented in tables such as `SQM$contig$abund` or `SQM$bins$tpm` will still refer to the complete contigs and bins, regardless of whether only a fraction of their ORFs are actually present in the returned SQM object. This is also true for the statistics presented in `SQM$contigs$table` and `SQM$bins$table`.

Examples

```
data(Hadza)
# Select the 100 most abundant ORFs in our dataset.
mostAbundantORFnames = names(sort(rowSums(Hadza$orfs$tpm), decreasing=TRUE))[1:100]
mostAbundantORFs = subsetORFs(Hadza, mostAbundantORFnames)
```

---

subsetRand *Select random ORFs*

Description

Create a random subset of a SQM object.

Usage

```
subsetRand(SQM, N)
```

Arguments

`SQM`	SQM object to be subsetted.
`N`	numeric. number of random ORFs to select.

Value

SQM object containing a random subset of ORFs.

See Also

`subsetORFs`

subsetSamples *Filter results by sample*

Description

Create a SQM object containing only samples specified by the user, and the ORFs, contigs, bins, taxa and functions present in those samples.

Usage

```
subsetSamples(SQM, samples, remove_missing = TRUE)
```

Arguments

SQM	SQM object to be subsetted.
samples	character. Samples to be included in the subset.
remove_missing	bool. If TRUE, ORFs, contigs, bins, taxa and functions absent from the selected samples will be removed from the subsetted object (default TRUE).

Value

SQM object containing only the requested samples.

See Also

subsetTax, subsetFun, subsetORFs, combineSQM. The most abundant items of a particular table contained in a SQM object can be selected with mostAbundant.

subsetTax *Filter results by taxonomy*

Description

Create a SQM object containing only the contigs with a given consensus taxonomy, the ORFs contained in them and the bins that contain them.

Usage

```
subsetTax(
  SQM,
  rank,
  tax,
  trusted_functions_only = FALSE,
  ignore_unclassified_functions = FALSE,
  rescale_tpm = TRUE,
  rescale_copy_number = TRUE
)
```

Arguments

`SQM` SQM object to be subsetted.

`rank` character. The taxonomic rank from which to select the desired taxa (`superkingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species`)

`tax` character. The taxon to select.

`trusted_functions_only`
logical. If `TRUE`, only highly trusted functional annotations (best hit + best average) will be considered when generating aggregated function tables. If `FALSE`, best hit annotations will be used (default `FALSE`).

`ignore_unclassified_functions`
logical. If `FALSE`, ORFs with no functional classification will be aggregated together into an "Unclassified" category. If `TRUE`, they will be ignored (default `FALSE`).

`rescale_tpm` logical. If `TRUE`, TPMs for KEGGs, COGs, and PFAMs will be recalculated (so that the TPMs in the subset actually add up to 1 million). Otherwise, per-function TPMs will be calculated by aggregating the TPMs of the ORFs annotated with that function, and will thus keep the scaling present in the parent object. By default it is set to `TRUE`, which means that the returned TPMs will be scaled *by million of reads of the selected taxon*.

`rescale_copy_number`
logical. If `TRUE`, copy numbers with be recalculated using the RecA/RadA coverages in the subset. Otherwise, RecA/RadA coverages will be taken from the parent object. By default it is set to `TRUE`, which means that the returned copy numbers for each function will represent the average copy number of that function *per genome of the selected taxon*.

Value

SQM object containing only the requested taxon.

See Also

`subsetFun`, `subsetContigs`, `subsetSamples`, `combineSQM`. The most abundant items of a particular table contained in a SQM object can be selected with `mostAbundant`.

Examples

```
data(Hadza)
Hadza.Prevotella = subsetTax(Hadza, "genus", "Prevotella")
Hadza.Proteobacteria = subsetTax(Hadza, "phylum", "Proteobacteria")
```

summary.SQM *summary method for class SQM*

Description

Computes different statistics of the data contained in the SQM object.

Usage

```
## S3 method for class 'SQM'
summary(object, ...)
```

Arguments

`object`	SQM object to be summarized.
`...`	Additional parameters (ignored).

Value

A list of summary statistics.

summary.SQMlite *summary method for class SQMlite*

Description

Computes different statistics of the data contained in the SQMlite object.

Usage

```
## S3 method for class 'SQMlite'
summary(object, ...)
```

Arguments

`object`	SQMlite object to be summarized.
`...`	Additional parameters (ignored).

Value

A list of summary statistics.

USiCGs *Universal Single-Copy Genes*

Description

Lists of Universal Single Copy Genes for Bacteria and Archaea. These are useful for transforming coverages or tpms into copy numbers. This is an alternative way of normalizing data in order to be able to compare functional profiles in samples with different sequencing depths.

Usage

```
data(USiCGs)
```

Format

Character vector with the KEGG identifiers for 15 Universal Single Copy Genes.

Source

Carr *et al.*, 2013. Table S1.

References

Carr, Shen-Orr & Borenstein (2013). Reconstructing the Genomic Content of Microbiome Taxa through Shotgun Metagenomic Deconvolution *PLoS Comput. Biol.* **9**:e1003292. (PubMed).

See Also

`MGOGs` and `MGKOs` for an alternative set of single copy genes, and for examples on how to generate copy numbers.

Examples

```
data(Hadza)
data(USiCGs)
### Let's look at the Universal Single Copy Gene distribution in our samples.
KEGG.tpm = Hadza$functions$KEGG$tpm
all(USiCGs %in% rownames(KEGG.tpm)) # Are all the USiCGs present in our dataset?
# Plot a boxplot of USiCGs tpms and calculate median USiCGs tpm.
# This looks weird in the test dataset because it contains only a small subset of the metagenomes.
# In a set of complete metagenomes USiCGs should have fairly similar TPM averages
# and low dispersion across samples.
boxplot(t(KEGG.tpm[USiCGs,]), names=USiCGs, ylab="TPM", col="slateblue2")

### Now let's calculate the average copy numbers of each function.
# We do it for KEGG annotations here, but we could also do it for COGs or PFAMs.
USiCGs.cov = apply(Hadza$functions$KEGG$cov[USiCGs,], 2, median)
# Sample-wise division by the median USiCG coverage.
KEGG.copynumber = t(t(Hadza$functions$KEGG$cov) / USiCGs.cov)
```

Index