Package ‘SQMtools’

July 10, 2019

Title Analyze results generated by the SqueezeMeta pipeline

Version 0.3.3

Description SqueezeMeta is a versatile pipeline for the automated analysis of metagenomics/metatranscriptomics data (http://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

Author Fernando Puente-Sánchez, Natalia García-García

Maintainer Fernando Puente-Sánchez <`fpuente@cnb.csic.es`>

Depends R (>= 3.4.0)

Imports reshape2, ggplot2

Suggests vegan, DESeq2

License GPLv3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1.9000

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.

Usage

```
combineSQM(..., tax_source = "orfs", trusted_functions_only = F,
  ignore_unclassified_functions = F, rescale_tpm = T,
  rescale_copy_number = T)
```

Arguments

`...`	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`

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=F)
```

`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.logical.
`output_name`	character. Name of the output file.

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 5 bins (and the associated contigs and genes) obtained from 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=T)
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

Usage

```
loadSQM(project_path, tax_mode = "allfilter")
```

Arguments

`project_path`	character, project directory generated by SqueezeMeta.
`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) and `prokfilter` for applying the threshold to Bacteria and Archaea, but not to Eukaryotes.

Value

SQM object containing the parsed project.

Prerequisites

1. 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`
2. Generate tabular outputs with the `sqm2tables.py` script included in the `path/to/SqueezeMeta/utils` directory:
   `/path/to/SqueezeMeta/utils/sqm2tables.py project_dir project_dir/results/tables`

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
	$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
	$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
	$tpm		*numeric matrix*	bins	samples	tpm
	$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
		$tpm	*numeric matrix*	KEGG ids	samples	tpm
		$copy_number	*numeric matrix*	KEGG ids	samples	avg. copies
	$COG	**$abund**	*numeric matrix*	COG ids	samples	abundances
		$tpm	*numeric matrix*	COG ids	samples	tpm
		$copy_number	*numeric matrix*	COG ids	samples	avg. copies
	$PFAM	**$abund**	*numeric matrix*	PFAM ids	samples	abundances
		$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
	$COG_names		*character vector*	COG ids	(n/a)	COG names

Examples

```
## Not run:
# (outside R)
/path/to/SqueezeMeta/scripts/SqueezeMeta.pl -p Hadza -f raw -m coassembly -s test.samples # Run SqueezeMeta on th
/path/to/SqueezeMeta/utils/sqm2tables.py Hadza Hadza/results/tables # Generate the tabular outputs! They must be
```

```
# now go into R
R
library(SQMtools)
Hadza = loadSQM("Hadza") # Where Hadza is the path to the SqueezeMeta output directory

## End(Not run)

data(Hadza)
# Which are the ten most abundant KEGG IDs in our data?
topKEGG = sort(rowSums(Hadza$functions$KEGG$tpm), decreasing=T)[1:11]
topKEGG = topKEGG[names(topKEGG)!="Unclassified"]
# Which functions do those KEGG IDs represent?
Hadza$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!
```

---

mostAbundant — *Get the N most abundant rows from a numeric table*

Description

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

Usage

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

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).

Value

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

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")
```

---

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)
```

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`).

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")
```

`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 = c(), ignore_unclassified = T, gradient_col = c("ghostwhite",
  "dodgerblue4"), base_size = 11)
```

Arguments

`SQM`	A SQM object.
`fun_level`	character. Either `"KEGG"`, `"COG"` or `"PFAM"` (default `"KEGG"`).
`count`	character. Either `"tpm"` for TPM normalized values, `"abund"` for raw abundances or `"copy_number"` for copy numbers (default `"tpm"`).
`N`	integer Plot the `N` most abundant functions (default `25`).
`fun`	character. Custom functions to plot. If provided, it will override `N` (default `NULL`).
`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`).

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)
```

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`).

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 = T, ignore_unclassified = F, rescale = F,
  color = NULL, base_size = 11)
```

Arguments

`SQM`	A SQM 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`).
`ignore_unclassified`	logical. Don't include unclassified contigs 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`).

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 = SQMtools:::aggregate.fun(Hadza, "COG", trusted_functions_only=T,
                                        ignore_unclassified_functions=F)$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)
```

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)
```

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 = F,
  ignore_unclassified_functions = F, rescale_tpm = T,
  rescale_copy_number = T)
```

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 two most complete bins?
topBinNames = rownames(Hadza$bins$table)[order(Hadza$bins$table[,"Completeness"], decreasing=T)][1:2]
topBins = subsetBins(Hadza, topBinNames)
```

`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 = F,
  ignore_unclassified_functions = F, rescale_tpm = F,
  rescale_copy_number = F)
```

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, ignore_case = T, fixed = F,
  trusted_functions_only = F, ignore_unclassified_functions = F,
  rescale_tpm = F, rescale_copy_number = F)
```

Arguments

`SQM`	SQM object to be subsetted.
`fun`	character, pattern to search for in the different functional classifications.
`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`, `combineSQM`. The most abundant items of a particular table contained in a SQM object can be eselected with `mostAbundant`.

Examples

```
data(Hadza)
Hadza.iron = subsetFun(Hadza, "iron")
Hadza.carb = subsetFun(Hadza, "Carbohydrate metabolism")
```

`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 = F,
  ignore_unclassified_functions = F, rescale_tpm = F,
  rescale_copy_number = F)
```

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.

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=T))[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`

`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 = F,
  ignore_unclassified_functions = F, rescale_tpm = T,
  rescale_copy_number = T)
```

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

Examples

```
data(Hadza)
Hadza.Escherichia = subsetTax(Hadza, "genus", "Escherichia")
Hadza.Bacteroidetes = subsetTax(Hadza, "phylum", "Bacteroidetes")
```

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(SQM)
```

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).

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.
KEGG.coverage = SQMtools:::aggregate.fun(Hadza, "KEGG", trusted_functions_only=T,
                                         ignore_unclassified_functions=F)$cov
USiCGs.cov = apply(KEGG.coverage[USiCGs,], 2, median)
# Sample-wise division by the median USiCG coverage.
KEGG.copynumber = t(t(KEGG.coverage) / USiCGs.cov)
```

Index