Package 'pagoo'

February 12, 2021

Version 0.3.8

Title Analyze Bacterial Pangenomes in R with 'Pagoo'

Description Provides an encapsulated, object-oriented class system for analyzing bacterial pangenomes. For a definition of this concept, see Tettelin, et al. (2005) <doi:10.1073/pnas.0506758102>. It uses the R6 package as backend. It was designed in order to facilitate and speed-up the comparative analysis of multiple bacterial genomes, standardizing and optimizing routine tasks performed everyday. There are a handful of things done everyday when working with bacterial pangenomes: subset, summarize, extract, visualize and store data. So, 'pagoo' is intended to facilitate these tasks as much as possible. For a description of the implemented data structure and methods, see Ferres & Iraola (2020), <doi:10.1101/2020.07.29.226951>.

Maintainer Ignacio Ferres <iferres@pasteur.edu.uy>

URL https://iferres.github.io/pagoo/, https://github.com/iferres/pagoo

BugReports https://github.com/iferres/pagoo/issues

Depends R (>= 3.5.0), S4Vectors, Biostrings, ggplot2

biocViews

Imports R6, reshape2, vegan, GenomicRanges, BiocGenerics, shinyWidgets, shinydashboard, DT, plotly, magrittr, heatmaply, dendextend, ggfortify, shiny

Suggests micropan, patchwork, ape, phangorn, pegas, DECIPHER, rhierbaps, knitr, rmarkdown, testthat, covr

License GPL-3

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Ignacio Ferres [aut, cre] (<https://orcid.org/0000-0003-0910-6568>),
Gregorio Iraola [aut] (<https://orcid.org/0000-0002-6516-3404>),
Institut Pasteur de Montevideo [fnd]

Repository CRAN

Date/Publication 2021-02-12 09:50:08 UTC

R topics documented:

load_pangenomeRDS 2
pagoo 3
PgR6 10
PgR6M 14
PgR6MS 19
roary_2_pagoo 21

Index **22**

`load_pangenomeRDS` *Load A Pagoo Pangenome*

Description

This function loads a pagoo pangenome from a '.RDS' file generated by the 'save_pangenomeRDS' method. Objects loaded by this functions keep their states, i.e : dropped/recovered organisms are conserved, as well as the 'core_level' setted when the object was originally saved.

Usage

```
load_pangenomeRDS(file, pkg, ...)
```

Arguments

`file`	The path to the pangenome '.RDS' file.
`pkg`	The package to use to load the object. Shouldn't be necessary to provide, but may be useful in some cases.
`...`	Arguments to be passed to the pagoo object. `sep` and `core_level` overwrite the values stored in the file.

Value

A `PgR6MS` class object, or a `PgR6M` object (with or without sequences, respectively).

pagoo *Create a Pagoo Object*

Description

This is the main function to load a pagoo object. It's safer and more friendly than using pagoo's class constructors (PgR6, PgR6M, and PgR6MS). This function returns either a `PgR6M` class object, or a `PgR6MS` class object, depending on the parameters provided. If sequences are provided, it returns the latter. See below for more details.

Usage

```
pagoo(
  data,
  org_meta,
  cluster_meta,
  sequences,
  core_level = 95,
  sep = "__",
  verbose = TRUE
)
```

Arguments

`data`	A `data.frame` or `DataFrame` containing at least the following columns: `gene` (gene name), `org` (organism name to which the gene belongs to), and `cluster` (group of orthologous to which the gene belongs to). More columns can be added as metadata for each gene.
`org_meta`	(optional) A `data.frame` or `DataFrame` containing additional metadata for organisms. This `data.frame` must have a column named "org" with valid organisms names (that is, they should match with those provided in `data`, column `org`), and additional columns will be used as metadata. Each row should correspond to each organism.
`cluster_meta`	(optional) A `data.frame` or `DataFrame` containing additional metadata for clusters. This `data.frame` must have a column named "cluster" with valid organisms names (that is, they should match with those provided in `data`, column `cluster`), and additional columns will be used as metadata. Each row should correspond to each cluster.
`sequences`	(optional) Can accept: 1) a named `list` of named `character` vector. Name of list are names of organisms, names of character vector are gene names; or 2) a named `list` of `DNAStringSetList` objects (same requirements as (1), but with BStringSet names as gene names); or 3) a `DNAStringSetList` (same requirements as (2) but `DNAStringSetList` names are organisms names). If this parameter is used, then a `PgR6MS` class object is returned.

core_level	The initial core_level (that's the percentage of organisms a core cluster must be in to be considered as part of the core genome). Must be a number between 100 and 85, (default: 95). You can change it later by using the $core_level field once the object was created.
sep	A separator. By default is '__'(two underscores). It will be used to create a unique gid (gene identifier) for each gene. gids are created by pasting org to gene, separated by sep.
verbose	logical. Whether to display progress messages when loading class.

Details

This package uses [R6](https://r6.r-lib.org/articles/Introduction.html) classes to provide a unified, comprehensive, standardized, but at the same time flexible, way to analyze a pangenome. The idea is to have a single object which contains both the data and the basic methods to analyze them, as well as manipulate fields, explore, and to use in harmony with the already existing and extensive list of R packages available created for comparative genomics and genetics.

For more information, tutorials, and resources, please visit https://iferres.github.io/pagoo/ .

Index

Active Bindings:

- $pan_matrix
- $organisms
- $clusters
- $genes
- $sequences
- $core_level
- $core_genes
- $core_clusters
- $core_sequences
- $shell_genes
- $shell_clusters
- $shell_sequences
- $cloud_genes
- $cloud_clusters
- $cloud_sequences

Active bindings

- $pan_matrix The panmatrix. Rows are organisms, and columns are groups of orthologous. Cells indicates the presence (>=1) or absence (0) of a given gene, in a given organism. Cells can have values greater than 1 if contain in-paralogs.
- $organisms A DataFrame with available organism names, and organism number identifier as rownames(). (Dropped organisms will not be displayed in this field, see $dropped below). Additional metadata will be shown if provided, as additional columns.

- $clusters A DataFrame with the groups of orthologous (clusters). Additional metadata will be shown as additional columns, if provided before. Each row corresponds to each cluster.
- $genes A SplitDataFrameList object with one entry per cluster. Each element contains a DataFrame with gene ids (<gid>) and additional metadata, if provided. gid are created by pasteing organism and gene names, so duplication in gene names are avoided.
- $sequences A DNAStringSetList with the set of sequences grouped by cluster. Each group is accessible as were a list. All Biostrings methods are available.
- $core_level The percentage of organisms a gene must be in to be considered as part of the coregenome. core_level = 95 by default. Can't be set above 100, and below 85 raises a warning.
- $core_genes Like genes, but only showing core genes.
- $core_clusters Like $clusters, but only showing core clusters.
- $core_sequences Like $sequences, but only showing core sequences.
- $cloud_genes Like genes, but only showing cloud genes. These are defined as those clusters which contain a single gene (singletons), plus those which have more than one but its organisms are probably clonal due to identical general gene content. Colloquially defined as strain-specific genes.
- $cloud_clusters Like $clusters, but only showing cloud clusters as defined above.
- $cloud_sequences Like $sequences, but only showing cloud sequences as defined above.
- $shell_genes Like genes, but only showing shell genes. These are defined as those clusters than don't belong neither to the core genome, nor to cloud genome. Colloquially defined as genes that are present in some but not all strains, and that aren't strain-specific.
- $shell_clusters Like $clusters, but only showing shell clusters, as defined above.
- $shell_sequences Like $sequences, but only showing shell sequences, as defined above.
- $summary_stats A DataFrame with information about the number of core, shell, and cloud clusters, as well as the total number of clusters.
- $random_seed The last .Random.seed. Used for reproducibility purposes only.
- $dropped A character vector with dropped organism names, and organism number identifier as names()

Methods

Below is a comprehensive description of all the methods provided by the object.

Add metadata Description:: Add metadata to the object. You can add metadata to each organism, to each group of orthologous, or to each gene. Elements with missing data should be filled by NA (dimensions of the provided data.frame must be coherent with object data).

Usage:: `$add_metadata(map = 'org',df)`

Arguments::

- map: character identifying the metadata to map. Can be one of "org", "group", or "gid".
- df: data.frame or DataFrame with the metadata to add. For each case, a column named as "map" must exists, which should contain identifiers for each element. In the

case of adding gene (gid) metadata,each gene should be referenced by the name of the organism and the name of the gene as provided in the "data" data.frame, separated by the "sep" argument.

Return:: self invisibly, but with additional metadata.

Drop an organism **Description::** Drop an organism from the dataset. This method allows to hide an organism from the real dataset, ignoring it in downstream analyses. All the fields and methods will behave as it doesn't exist. For instance, if you decide to drop organism 1, the $pan_matrix field (see below) would not show it when called.

Usage:: $drop(x)

Arguments::

- x: character or numeric. The name of the organism wanted to be dropped, or its numeric id as returned in $organism field (see below).

Return:: self invisibly, but with x dropped. It isn't necessary to assign the function call to a new object, nor to re-write it as R6 objects are mutable.

Recover a dropped organism **Description::** Recover a previously $drop()ped organism (see above). All fields and methods will start to behave considering this organism again.

Usage:: $recover(x)

Arguments::

- x: character or numeric. The name of the organism wanted to be recover, or its numeric id as returned in $dropped field (see below).

Return:: self invisibly, but with x recovered. It isn't necessary to assign the function call to a new object, nor to re-write it as R6 objects are mutable.

Write a pangenome as flat (text) files. **Description::** Write the pangenome data as flat tables (text). Is not the most recommended way to save a pangenome, since you can loose information as numeric precision, column classes (factor, numeric, integer), and the state of the object itself (i.e. dropped organisms, or core_level), loosing reproducibility. Use save_pangenomeRDS for a more precise way of saving a pagoo object. Still, it is useful if you want to work with the data outside R, just keep the above in mind.

Usage:: $write_pangenome(dir = "pangenome",force = FALSE)

Arguments::

- dir: The unexisting directory name where to put the data files. Default is "pangenome".
- force: logical. Whether to overwrite the directory if it already exists. Default: FALSE.

Return:: A directory with at least 3 files. "data.tsv" contain the basic pangenome data as it is provided to the data argument in the initialization method ($new(...)). "clusters.tsv" contain any metadata associated to the clusters. "organisms.tsv" contain any metadata associated to the organisms. The latter 2 files will contain a single column if no metadata was provided.

Save a pangenome as a RDS (binary) file. **Description::** Save a pagoo pangenome object. This function provides a method for saving a pagoo object and its state into a "RDS" file. To load the pangenome, use the load_pangenomeRDS function in this package. It *should* be compatible between pagoo versions, so you could update pagoo and still recover the same pangenome. Even sep and core_level are restored unless the user provides those arguments in load_pangenomeRDS. dropped organisms also kept hidden, as you where working with the original object.

Usage:: `$save_pangenomeRDS(file = "pangenome.rds")`

Arguments::

- `file`: The name of the file to save. Default: "pangenome.rds".

Return:: Writes a list with all the information needed to restore the object by using the load_pangenomeRDS function, into an RDS (binary) file.

Clone a pagoo object. **Description::** The objects of this class are clonable with this method.

Usage:: `$clone(deep = FALSE)`

Arguments::

- `deep`: `character` identifying the metadata to map. Can be one of `"org"`, `"group"`, or `"gid"`.

Return:: Whether to make a deep clone.

Compute distances **Description::** Compute distance between all pairs of genomes. The default dist method is `"bray"` (Bray-Curtis distance). Another used distance method is `"jaccard"`, but you should set `binary = FALSE` (see below) to obtain a meaningful result. See `vegdist` for details, this is just a wrapper function.

Usage:: `$dist( method = "bray",binary = FALSE,diag = FALSE,upper = FALSE,na.rm = FALSE,... )`

Arguments::

- `method`: The distance method to use. See vegdist for available methods, and details for each one.
- `binary`: Transform abundance matrix into a presence/absence matrix before computing distance.
- `diag`: Compute diagonals.
- `upper`: Return only the upper diagonal.
- `na.rm`: Pairwise deletion of missing observations when computing dissimilarities.
- `...`: Other parameters. See vegdist for details.

Return:: A `dist` object containing all pairwise dissimilarities between genomes.

Compute a Principal Component Analysis **Description::** Performs a principal components analysis on the panmatrix.

Usage:: `$pan_pca( center = TRUE,scale. = FALSE,...)`

Arguments::

- `center`: a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
- `scale.`: a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is TRUE.
- `...`: Other arguments. See prcomp

Return:: Returns a list with class "prcomp". See prcomp for more information.

Fit a Power Law Function for the Pangenome **Description::** Fits a power law curve for the pangenome rarefaction simulation.

Usage:: `$pg_power_law_fit(raref,...)`

Arguments::

- raref: (Optional) A rarefaction matrix, as returned by rarefact().
- ...: Further arguments to be passed to rarefact(). If raref is missing, it will be computed with default arguments, or with the ones provided here.

Return:: A list of two elements: $formula with a fitted function, and $params with fitted parameters. An attribute "alpha" is also returned (If alpha>1, then the pangenome is closed, otherwise is open.

Fit an Exponential Decay Function for the Coregenome **Description::** Fits an exponential decay curve for the coregenome rarefaction simulation.

Usage:: `$cg_exp_decay_fit(raref,pcounts = 10,...)`

Arguments::

- raref: (Optional) A rarefaction matrix, as returned by rarefact().
- pcounts: An integer of pseudo-counts. This is used to better fit the function at small numbers, as the linearization method requires to subtract a constant C, which is the coregenome size, from y. As y becomes closer to the coregenome size, this operation tends to 0, and its logarithm goes crazy. By default pcounts=10.
- ...: Further arguments to be passed to rarefact(). If raref is missing, it will be computed with default arguments, or with the ones provided here.

Return:: A list of two elements: $formula with a fitted function, and $params with fitted intercept and decay parameters.

Compute Genomic Fluidity **Description::** Computes the genomic fluidity, which is a measure of population diversity. See fluidity for more details.

Usage:: `$fluidity(nsim = 10)`

Arguments::

- nsim:An integer specifying the number of random samples to use in the computations.

Return:: A list with two elements, the mean fluidity and its sample standard deviation over the n.sim computed values.

Plot Accessory Frequency Plot **Description::** Plot a barplot with the frequency of genes within the total number of genomes.

Usage:: `$gg_barplot()`

Return:: A barplot, and a gg object (ggplot2 package) invisibly.

Plot a Distance Heatmap **Description::** Plot a heatmap showing the computed distance between all pairs of organisms.

Usage:: `$gg_dist(method = "bray",...)`

Arguments::

- method: Distance method. One of "Jaccard" (default), or "Manhattan",see above.
- ...: More arguments to be passed to distManhattan

Return:: A heatmap (ggplot2::geom_tile()), and a gg object (ggplot2 package) invisibly.

Plot a Pangenome Binary Map **Description::** Plot a pangenome binary map representing the presence/absence of each gene within each organism.

Usage:: `$gg_binmap()`

Return:: A binary map (`ggplot2::geom_raster()`), and a gg object (`ggplot2` package) invisibly.

Plot a PCA **Description::** Plot a scatter plot of a Principal Components Analysis.

Usage:: `$gg_pca(colour = NULL,...))`

Arguments::

- `colour`:The name of the column in `$organisms` field from which points will take color (if provided). `NULL` (default) renders black points.
- `...`: More arguments to be passed to `ggplot2::autoplot()`.

Return:: A scatter plot (`ggplot2::autoplot()`), and a gg object (`ggplot2` package) invisibly.

Plot a Pie with Pangenome Categories **Description::** Plot a pie chart showing the number of clusters of each pangenome category: core, shell, or cloud.

Usage:: `$gg_pie()`

Return:: A pie chart (`ggplot2::geom_bar()` + `coord_polar()`), and a gg object (`ggplot2` package) invisibly.

Plot Pangenome Curves **Description::** Plot pangenome and/or coregenome curves with the fitted functions returned by `pg_power_law_fit()` and `cg_exp_decay_fit()`. You can add points by adding + `geom_points()`, of ggplot2 package.

Usage:: `$gg_curves(what = c("pangenome","coregenome",...)`

Arguments::

- `what`: `"pangenome"` and/or `"coregenome"`.
- `...`: ignored

Return:: A scatter plot, and a gg object (`ggplot2` package) invisibly.

Run a Shiny App **Description::** Launch an interactive shiny app. It contains a sidebar with controls and switches to interact with the pagoo object. You can drop/recover organisms from the dataset, modify the core_level, visualize statistics, plots, and browse cluster and gene information. In the main body, it contains 2 tabs to switch between summary statistics plots and core genome information on one side, and accessory genome plots and information on the other.
The lower part of each tab contains two tables, side by side. On the "Summary" tab, the left one contain information about core clusters, with one cluster per row. When one of them is selected (click), the one on the right is updated to show information about its genes (if provided), one gene per row. On the "Accessory" tab, a similar configuration is shown, but on this case only accessory clusters/genes are displayed. There is a slider on the sidebar where one can select the accessory frequency range to display.
Give it a try!
Take into account that big pangenomes can slow down the performance of the app. More than 50-70 organisms often leads to a delay in the update of the plots/tables.

Usage:: `$runShinyApp()`

Return:: Opens a shiny app on the browser.

Retrieve Core Genes for Phylogeny **Description::** A field for obtaining core gene sequences is available (see below), but for creating a phylogeny with this sets is useful to: 1) have the possibility of extracting just one sequence of each organism on each cluster, in case paralogues are present, and 2) filling gaps with empty sequences in case the core_level was set below 100%, allowing more genes (some not in 100% of organisms) to be incorporated to the phylogeny. That is the purpose of this special function.

Usage:: `$core_seqs_4_phylo(max_per_org = 1, fill = TRUE)`

Arguments::

- `max_per_org`: Maximum number of sequences of each organism to be taken from each cluster.
- `fill`: `logical`. If fill `DNAStringSet` with empty `DNAString` in cases where `core_level` is set below 100%, and some clusters with missing organisms are also considered.

Return:: A `DNAStringSetList` with core genes. Order of organisms on each cluster is conserved, so it is easier to concatenate them into a super-gene suitable for phylogenetic inference.

`PgR6` *PgR6 basic class*

Description

A basic `PgR6` class constructor. It contains basic fields and subset functions to handle a pangenome. Final users should use `pagoo` instead of this, since is more easy to understand.

Active bindings

`pan_matrix` The panmatrix. Rows are organisms, and columns are groups of orthologous. Cells indicates the presence (>=1) or absence (0) of a given gene, in a given organism. Cells can have values greater than 1 if contain in-paralogs.

`organisms` A `DataFrame` with available organism names, and organism number identifier as `rownames()`. (Dropped organisms will not be displayed in this field, see `$dropped` below). Additional metadata will be shown if provided, as additional columns.

`genes` A `SplitDataFrameList` object with one entry per cluster. Each element contains a `DataFrame` with gene ids (`<gid>`) and additional metadata, if provided. `gid` are created by `pasteing` organism and gene names, so duplication in gene names are avoided.

`clusters` A `DataFrame` with the groups of orthologous (clusters). Additional metadata will be shown as additional columns, if provided before. Each row corresponds to each cluster.

`core_level` The percentage of organisms a gene must be in to be considered as part of the coregenome. `core_level = 95` by default. Can't be set above 100, and below 85 raises a warning.

`core_genes` Like `genes`, but only showing core genes.

`core_clusters` Like `$clusters`, but only showing core clusters.

`cloud_genes` Like `genes`, but only showing cloud genes. These are defined as those clusters which contain a single gene (singletons), plus those which have more than one but its organisms are probably clonal due to identical general gene content. Colloquially defined as strain-specific genes.

`cloud_clusters` Like `$clusters`, but only showing cloud clusters as defined above.

`shell_genes` Like `genes`, but only showing shell genes. These are defined as those clusters than don't belong neither to the core genome, nor to cloud genome. Colloquially defined as genes that are present in some but not all strains, and that aren't strain-specific.

`shell_clusters` Like `$clusters`, but only showing shell clusters, as defined above.

`summary_stats` A `DataFrame` with information about the number of core, shell, and cloud clusters, as well as the total number of clusters.

`random_seed` The last `.Random.seed`. Used for reproducibility purposes only.

`dropped` A `character` vector with dropped organism names, and organism number identifier as `names()`

Methods

Public methods:

- `PgR6$new()`
- `PgR6$add_metadata()`
- `PgR6$drop()`
- `PgR6$recover()`
- `PgR6$write_pangenome()`
- `PgR6$save_pangenomeRDS()`
- `PgR6$clone()`

Method `new()`**:** A basic `PgR6` class constructor. It contains basic fields and subset functions to handle a pangenome.

Usage:

```
PgR6$new(
  data,
  org_meta,
  cluster_meta,
  core_level = 95,
  sep = "__",
  verbose = TRUE,
  DF,
  group_meta
)
```

Arguments:

`data` A `data.frame` or `DataFrame` containing at least the following columns: `gene` (gene name), `org` (organism name to which the gene belongs to), and `cluster` (group of orthologous to which the gene belongs to). More columns can be added as metadata for each gene.

`org_meta` (optional) A `data.frame` or `DataFrame` containing additional metadata for organisms. This `data.frame` must have a column named "org" with valid organisms names (that is, they should match with those provided in `data`, column `org`), and additional columns will be used as metadata. Each row should correspond to each organism.

`cluster_meta` (optional) A `data.frame` or `DataFrame` containing additional metadata for clusters. This `data.frame` must have a column named "cluster" with valid organisms names (that is, they should match with those provided in `data`, column `cluster`), and additional columns will be used as metadata. Each row should correspond to each cluster.

`core_level` The initial core_level (that's the percentage of organisms a core cluster must be in to be considered as part of the core genome). Must be a number between 100 and 85, (default: 95). You can change it later by using the `$core_level` field once the object was created.

`sep` A separator. By default is '__'(two underscores). It will be used to create a unique `gid` (gene identifier) for each gene. `gids` are created by pasting `org` to `gene`, separated by `sep`.

`verbose` `logical`. Whether to display progress messages when loading class.

`DF` Deprecated. Use `data` instead.

`group_meta` Deprecated. Use `cluster_meta` instead.

Returns: An R6 object of class PgR6. It contains basic fields and methods for analyzing a pangenome.

Method `add_metadata()`**:** Add metadata to the object. You can add metadata to each organism, to each group of orthologous, or to each gene. Elements with missing data should be filled by `NA` (dimensions of the provided data.frame must be coherent with object data).

Usage:

```
PgR6$add_metadata(map = "org", data)
```

Arguments:

`map` `character` identifying the metadata to map. Can be one of `"org"`, `"group"`, or `"gid"`.

`data` `data.frame` or `DataFrame` with the metadata to add. For each case, a column named as `"map"` must exists, which should contain identifiers for each element. In the case of adding gene (`gid`) metadata,each gene should be referenced by the name of the organism and the name of the gene as provided in the `"data"` data.frame, separated by the `"sep"` argument.

Returns: `self` invisibly, but with additional metadata.

Method `drop()`**:** Drop an organism from the dataset. This method allows to hide an organism from the real dataset, ignoring it in downstream analyses. All the fields and methods will behave as it doesn't exist. For instance, if you decide to drop organism 1, the `$pan_matrix` field (see below) would not show it when called.

Usage:

```
PgR6$drop(x)
```

Arguments:

`x` `character` or `numeric`. The name of the organism wanted to be dropped, or its numeric id as returned in `$organism` field (see below).

Returns: `self` invisibly, but with `x` dropped. It isn't necessary to assign the function call to a new object, nor to re-write it as R6 objects are mutable.

Method `recover()`**:** Recover a previously `$drop()`ped organism (see above). All fields and methods will start to behave considering this organism again.

Usage:

```
PgR6$recover(x)
```

Arguments:

`x` `character` or `numeric`. The name of the organism wanted to be recover, or its numeric id as returned in `$dropped` field (see below).

Returns: `self` invisibly, but with `x` recovered. It isn't necessary to assign the function call to a new object, nor to re-write it as R6 objects are mutable.

Method `write_pangenome()`**:** Write the pangenome data as flat tables (text). Is not the most recommended way to save a pangenome, since you can loose information as numeric precision, column classes (factor, numeric, integer), and the state of the object itself (i.e. dropped organisms, or core_level), loosing reproducibility. Use `$save_pangenomeRDS` for a more precise way of saving a pagoo object. Still, it is useful if you want to work with the data outside R, just keep the above in mind.

Usage:

```
PgR6$write_pangenome(dir = "pangenome", force = FALSE)
```

Arguments:

`dir` The non-existing directory name where to put the data files. Default is "pangenome".

`force` `logical`. Whether to overwrite the directory if it already exists. Default: `FALSE`.

Returns: A directory with at least 3 files. "data.tsv" contain the basic pangenome data as it is provided to the `data` argument in the initialization method (`$new(...)`). "clusters.tsv" contain any metadata associated to the clusters. "organisms.tsv" contain any metadata associated to the organisms. The latter 2 files will contain a single column if no metadata was provided.

Method `save_pangenomeRDS()`**:** Save a pagoo pangenome object. This function provides a method for saving a pagoo object and its state into a "RDS" file. To load the pangenome, use the `load_pangenomeRDS` function in this package. It *should* be compatible between pagoo versions, so you could update pagoo and still recover the same pangenome. Even `sep` and `core_level` are restored unless the user provides those arguments in `load_pangenomeRDS`. `dropped` organisms also kept hidden, as you where working with the original object.

Usage:

```
PgR6$save_pangenomeRDS(file = "pangenome.rds")
```

Arguments:

`file` The name of the file to save. Default: "pangenome.rds".

Returns: Writes a list with all the information needed to restore the object by using the load_pangenomeRDS function, into an RDS (binary) file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PgR6$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PgR6M *PgR6 class with methods.*

Description

PgR6 with Methods. Final users should use pagoo instead of this, since is more easy to understand. Inherits: PgR6

Super class

pagoo::PgR6 -> PgR6M

Methods

Public methods:

- PgR6M$new()
- PgR6M$rarefact()
- PgR6M$dist()
- PgR6M$pan_pca()
- PgR6M$pg_power_law_fit()
- PgR6M$cg_exp_decay_fit()
- PgR6M$gg_barplot()
- PgR6M$gg_binmap()
- PgR6M$gg_dist()
- PgR6M$gg_pca()
- PgR6M$gg_pie()
- PgR6M$gg_curves()
- PgR6M$runShinyApp()
- PgR6M$clone()

Method `new()`**:** Create a PgR6M object.

Usage:

```
PgR6M$new(
  data,
  org_meta,
  cluster_meta,
  core_level = 95,
  sep = "__",
  verbose = TRUE,
  DF,
  group_meta
)
```

Arguments:

`data` A `data.frame` or `DataFrame` containing at least the following columns: `gene` (gene name), `org` (organism name to which the gene belongs to), and `cluster` (group of orthologous to which the gene belongs to). More columns can be added as metadata for each gene.

`org_meta` (optional) A `data.frame` or `DataFrame` containing additional metadata for organisms. This `data.frame` must have a column named "org" with valid organisms names (that is, they should match with those provided in `data`, column `org`), and additional columns will be used as metadata. Each row should correspond to each organism.

`cluster_meta` (optional) A `data.frame` or `DataFrame` containing additional metadata for clusters. This `data.frame` must have a column named "cluster" with valid organisms names (that is, they should match with those provided in `data`, column `cluster`), and additional columns will be used as metadata. Each row should correspond to each cluster.

`core_level` The initial core_level (that's the percentage of organisms a core cluster must be in to be considered as part of the core genome). Must be a number between 100 and 85, (default: 95). You can change it later by using the `$core_level` field once the object was created.

`sep` A separator. By default is '__'(two underscores). It will be used to create a unique `gid` (gene identifier) for each gene. `gid`s are created by pasting `org` to `gene`, separated by `sep`.

`verbose` `logical`. Whether to display progress messages when loading class.

`DF` Deprecated. Use `data` instead.

`group_meta` Deprecated. Use `cluster_meta` instead.

Returns: An R6 object of class PgR6M. It contains basic fields and methods for analyzing a pangenome. It also contains additional statistical methods for analyze it, and methods to make basic exploratory plots.

Method `rarefact()`**:** Rarefact pangenome or corgenome. Compute the number of genes which belong to the pangenome or to the coregenome, for a number of random permutations of increasingly bigger sample of genomes.

Usage:

```
PgR6M$rarefact(what = "pangenome", n.perm = 10)
```

Arguments:

`what` One of `"pangenome"` or `"coregenome"`.

`n.perm` The number of permutations to compute (default: 10).

Returns: A `matrix`, rows are the number of genomes added, columns are permutations, and the cell number is the number of genes in each category.

Method `dist()`**:** Compute distance between all pairs of genomes. The default dist method is `"bray"` (Bray-Curtis distance). Another used distance method is `"jaccard"`, but you should set `binary = FALSE` (see below) to obtain a meaningful result. See `vegdist` for details, this is just a wrapper function.

Usage:

```
PgR6M$dist(
  method = "bray",
  binary = FALSE,
  diag = FALSE,
```

```
  upper = FALSE,
  na.rm = FALSE,
  ...
)
```

Arguments:

`method` The distance method to use. See vegdist for available methods, and details for each one.

`binary` Transform abundance matrix into a presence/absence matrix before computing distance.

`diag` Compute diagonals.

`upper` Return only the upper diagonal.

`na.rm` Pairwise deletion of missing observations when computing dissimilarities.

`...` Other parameters. See vegdist for details.

Returns: A `dist` object containing all pairwise dissimilarities between genomes.

Method `pan_pca()`**:** Performs a principal components analysis on the panmatrix

Usage:

```
PgR6M$pan_pca(center = TRUE, scale. = FALSE, ...)
```

Arguments:

`center` a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.

`scale.` a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is TRUE.

`...` Other arguments. See prcomp

Returns: Returns a list with class "prcomp". See prcomp for more information.

Method `pg_power_law_fit()`**:** Fits a power law curve for the pangenome rarefaction simulation.

Usage:

```
PgR6M$pg_power_law_fit(raref, ...)
```

Arguments:

`raref` (Optional) A rarefaction matrix, as returned by `rarefact()`.

`...` Further arguments to be passed to `rarefact()`. If `raref` is missing, it will be computed with default arguments, or with the ones provided here.

Returns: A `list` of two elements: `$formula` with a fitted function, and `$params` with fitted parameters. An attribute `"alpha"` is also returned (If `alpha>1`, then the pangenome is closed, otherwise is open.

Method `cg_exp_decay_fit()`**:** Fits an exponential decay curve for the coregenome rarefaction simulation.

Usage:

```
PgR6M$cg_exp_decay_fit(raref, pcounts = 10, ...)
```

Arguments:

`raref` (Optional) A rarefaction matrix, as returned by `rarefact()`.

`pcounts` An integer of pseudo-counts. This is used to better fit the function at small numbers, as the linearization method requires to subtract a constant C, which is the coregenome size, from `y`. As `y` becomes closer to the coregenome size, this operation tends to 0, and its logarithm goes crazy. By default `pcounts=10`.

`...` Further arguments to be passed to `rarefact()`. If `raref` is missing, it will be computed with default arguments, or with the ones provided here.

Returns: A `list` of two elements: `$formula` with a fitted function, and `$params` with fitted intercept and decay parameters.

Method `gg_barplot()`**:** Plot a barplot with the frequency of genes within the total number of genomes.

Usage:

```
PgR6M$gg_barplot()
```

Returns: A barplot, and a gg object (`ggplot2` package) invisibly.

Method `gg_binmap()`**:** Plot a pangenome binary map representing the presence/absence of each gene within each organism.

Usage:

```
PgR6M$gg_binmap()
```

Returns: A binary map (`ggplot2::geom_raster()`), and a gg object (`ggplot2` package) invisibly.

Method `gg_dist()`**:** Plot a heatmap showing the computed distance between all pairs of organisms.

Usage:

```
PgR6M$gg_dist(method = "bray", ...)
```

Arguments:

`method` Distance method. One of "Jaccard" (default), or "Manhattan", see above.

`...` More arguments to be passed to `distManhattan`.

Returns: A heatmap (`ggplot2::geom_tile()`), and a gg object (`ggplot2` package) invisibly.

Method `gg_pca()`**:** Plot a scatter plot of a Principal Components Analysis.

Usage:

```
PgR6M$gg_pca(colour = NULL, ...)
```

Arguments:

`colour` The name of the column in `$organisms` field from which points will take colour (if provided). `NULL` (default) renders black points.

`...` More arguments to be passed to `ggplot2::autoplot()`.

Returns: A scatter plot (`ggplot2::autoplot()`), and a gg object (`ggplot2` package) invisibly.

Method `gg_pie()`**:** Plot a pie chart showing the number of clusters of each pangenome category: core, shell, or cloud.

Usage:

```
PgR6M$gg_pie()
```

Returns: A pie chart (`ggplot2::geom_bar()` + `coord_polar()`), and a gg object (`ggplot2` package) invisibly.

Method `gg_curves()`**:** Plot pangenome and/or coregenome curves with the fitted functions returned by `pg_power_law_fit()` and `cg_exp_decay_fit()`. You can add points by adding + `geom_points()`, of ggplot2 package

Usage:

```
PgR6M$gg_curves(what = c("pangenome", "coregenome"), ...)
```

Arguments:

`what` One of `"pangenome"` or `"coregenome"`.

`...` ????

Returns: A scatter plot, and a gg object (`ggplot2` package) invisibly.

Method `runShinyApp()`**:** Launch an interactive shiny app. It contains a sidebar with controls and switches to interact with the pagoo object. You can drop/recover organisms from the dataset, modify the core_level, visualize statistics, plots, and browse cluster and gene information. In the main body, it contains 2 tabs to switch between summary statistics plots and core genome information on one side, and accessory genome plots and information on the other.

The lower part of each tab contains two tables, side by side. On the "Summary" tab, the left one contain information about core clusters, with one cluster per row. When one of them is selected (click), the one on the right is updated to show information about its genes (if provided), one gene per row. On the "Accessory" tab, a similar configuration is shown, but on this case only accessory clusters/genes are displayed. There is a slider on the sidebar where one can select the accessory frequency range to display.

Give it a try!

Take into account that big pangenomes can slow down the performance of the app. More than 50-70 organisms often leads to a delay in the update of the plots/tables.

Usage:

```
PgR6M$runShinyApp()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PgR6M$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PgR6MS *PgR6 class with Methods and Sequences.*

Description

PgR6 with Methods and Sequences. Final users should use pagoo instead of this, since is more easy to understand. Inherits: PgR6M

Super classes

pagoo::PgR6 -> pagoo::PgR6M -> PgR6MS

Active bindings

`sequences` A DNAStringSetList with the set of sequences grouped by cluster. Each group is accessible as were a list. All Biostrings methods are available.

`core_sequences` Like $sequences, but only showing core sequences.

`cloud_sequences` Like $sequences, but only showing cloud sequences as defined above.

`shell_sequences` Like $sequences, but only showing shell sequences, as defined above.

Methods

Public methods:

- PgR6MS$new()
- PgR6MS$core_seqs_4_phylo()
- PgR6MS$clone()

Method `new()`**:** Create a PgR6MS object.

Usage:

```
PgR6MS$new(
  data,
  org_meta,
  cluster_meta,
  core_level = 95,
  sep = "__",
  DF,
  group_meta,
  sequences,
  verbose = TRUE
)
```

Arguments:

`data` A data.frame or DataFrame containing at least the following columns: gene (gene name), org (organism name to which the gene belongs to), and cluster (group of orthologous to which the gene belongs to). More columns can be added as metadata for each gene.

org_meta (optional) A data.frame or DataFrame containing additional metadata for organisms. This data.frame must have a column named "org" with valid organisms names (that is, they should match with those provided in data, column org), and additional columns will be used as metadata. Each row should correspond to each organism.

cluster_meta (optional) A data.frame or DataFrame containing additional metadata for clusters. This data.frame must have a column named "cluster" with valid organisms names (that is, they should match with those provided in data, column cluster), and additional columns will be used as metadata. Each row should correspond to each cluster.

core_level The initial core_level (that's the percentage of organisms a core cluster must be in to be considered as part of the core genome). Must be a number between 100 and 85, (default: 95). You can change it later by using the $core_level field once the object was created.

sep A separator. By default is '__'(two underscores). It will be used to create a unique gid (gene identifier) for each gene. gids are created by pasting org to gene, separated by sep.

DF Deprecated. Use data instead.

group_meta Deprecated. Use cluster_meta instead.

sequences Can accept: 1) a named list of named character vector. Name of list are names of organisms, names of character vector are gene names; or 2) a named list of DNAStringSetList objects (same requirements as (1), but with BStringSet names as gene names); or 3) a DNAStringSetList (same requirements as (2) but DNAStringSetList names are organisms names).

verbose logical. Whether to display progress messages when loading class.

Returns: An R6 object of class PgR6MS. It contains basic fields and methods for analyzing a pangenome. It also contains additional statistical methods for analyze it, methods to make basic exploratory plots, and methods for sequence manipulation.

Method core_seqs_4_phylo(): A field for obtaining core gene sequences is available (see below), but for creating a phylogeny with this sets is useful to: 1) have the possibility of extracting just one sequence of each organism on each cluster, in case paralogues are present, and 2) filling gaps with empty sequences in case the core_level was set below 100%, allowing more genes (some not in 100% of organisms) to be incorporated to the phylogeny. That is the purpose of this special function.

Usage:

```
PgR6MS$core_seqs_4_phylo(max_per_org = 1, fill = TRUE)
```

Arguments:

max_per_org Maximum number of sequences of each organism to be taken from each cluster.

fill logical. If fill DNAStringSet with empty DNAString in cases where core_level is set below 100%, and some clusters with missing organisms are also considered.

Returns: A DNAStringSetList with core genes. Order of organisms on each cluster is conserved, so it is easier to concatenate them into a super-gene suitable for phylogenetic inference.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PgR6MS$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

roary_2_pagoo *Read roary's output into a pagoo's R6 class object*

Description

This function handle conversion of roary's output files into a pagoo R6 class object. It takes the "gene_presence_absence.csv" file and (optionally but recommended) gff input file paths, and returns an object of class PgR6MS (or PgR6M if left empty the gffs argument).

Usage

```
roary_2_pagoo(gene_presence_absence_csv, gffs, sep = "__")
```

Arguments

gene_presence_absence_csv	character, path to the "gene_presence_absence.csv" file. (Do not confuse with the file with the same name but with .Rtab extension).
gffs	A character vector with paths to original gff files used as roary's input. Typically the return value of list.files() function. This parameter is optional but highly recommended if you want to manipulate sequences.
sep	character. Default: "__" (two underscores). See PgR6MS for a more detail argument description.

Value

A pagoo's R6 class object. Ethier PgR6M, if gffs argument is left empty, or PgR6MS if path to gff files is provided.

References

Andrew J. Page, Carla A. Cummins, Martin Hunt, Vanessa K. Wong, Sandra Reuter, Matthew T. G. Holden, Maria Fookes, Daniel Falush, Jacqueline A. Keane, Julian Parkhill, "Roary: Rapid large-scale prokaryote pan genome analysis", Bioinformatics, 2015;31(22):3691-3693

Examples

```
## Not run:
gffs <- list.files(path = "path/to/gffs/",
                   pattern = "[.]gff$",
                   full.names = TRUE)
gpa_csv <- "path/to/gene_presence_absence.csv"

library(pagoo)
pg <- roary_2_pagoo(gene_presence_absence_csv = gpa_csv,
                    gffs = gffs)

## End(Not run)
```

Index

DataFrame, *3–5*, *10–12*, *15*, *19, 20*
distManhattan, *8*, *17*
DNAStringSetList, *3*, *5*, *19, 20*

fluidity, *8*

load_pangenomeRDS, 2

pagoo, 3, *10*, *14*, *19*
pagoo::PgR6, *14*, *19*
pagoo::PgR6M, *19*
PgR6, *3*, 10, *14*
PgR6M, *3*, 14, *19*, *21*
PgR6MS, *3*, 19, *21*
prcomp, *7*, *16*

roary_2_pagoo, 21

SplitDataFrameList, *5*, *10*

vegdist, *7*, *15, 16*