Package 'plethy'

October 16, 2019

Version 1.22.0

Date 2013-4-26

Title R framework for exploration and analysis of respirometry data

Imports Streamer, IRanges, reshape2, plyr, RColorBrewer,ggplot2, Biobase

Depends R (>= 3.1.0), methods, DBI (>= 0.5-1), RSQLite (>= 1.1), BiocGenerics, S4Vectors

Suggests RUnit, BiocStyle

Description This package provides the infrastructure and tools to import, query and perform basic analysis of whole body plethysmography and metabolism data. Currently support is limited to data derived from Buxco respirometry instruments as exported by their FinePointe software.

License GPL-3

biocViews DataImport, biocViews, Infastructure, DataRepresentation,TimeCourse

Collate BuxcoDB.R RetList.R buxco_db_v2.R dep_parser.R utilities.R

Url https://github.com/dbottomly/plethy

git_url https://git.bioconductor.org/packages/plethy

git_branch RELEASE_3_9

git_last_commit 14c7a57

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

Author Daniel Bottomly [aut, cre],
Marty Ferris [ctb],
Beth Wilmot [aut],
Shannon McWeeney [aut]

Maintainer Daniel Bottomly <bottomly@ohsu.edu>

R topics documented:

`plethy-package` *R framework for exploration and analysis of respirometry data*

Description

This package provides the infrastructure and tools to import, query and perform basic analysis of whole body plethysmography and metabolism data. Currently support is limited to data derived from Buxco respirometry instruments as exported by their FinePointe software.

Details

Package:	plethy
Version:	0.99.4
Date:	2013-4-26
Imports:	Streamer, DBI, RSQLite, methods, IRanges, reshape2, batch
Depends:	R (>= 3.0.0)
Suggests:	RUnit, BiocGenerics
License:	GPL-3
biocViews:	DataImport, biocViews, Infastructure, DataRepresentation,TimeCourse
Collate:	BuxcoDB.R RetList.R buxco_db_v2.R dep_parser.R utilities.R
Packaged:	2013-08-01 21:02:18 UTC; bottomly
Built:	R 3.0.1; ; 2013-08-01 21:02:29 UTC; unix

Index:

```
BuxcoDB-class           Class '"BuxcoDB"'
dbImport                Import data into a BuxcoDB database
parse.buxco             Functions for parsing Buxco respirometry data
plethy-package          R framework for exploration and analysis of
                        respirometry data
buxco.sample.data.path  Path to a sample Buxco output file
sample.db.path          Path to a sample BuxcoDB database
```

Further information is available in the following vignettes:

`plethy` plethy (source, pdf)

Author(s)

Daniel Bottomly, Marty Ferris

Maintainer: Daniel Bottomly <bottomly@ohsu.edu>

References

www.buxco.com

Examples

```
tmp_db <- tempfile()

bux.db <- parse.buxco(file.name=buxco.sample.data.path(), db.name = tmp_db)

head(retrieveData(bux.db))

bux.db <- makeBuxcoDB(tmp_db)

samples(bux.db)
variables(bux.db)
```

Additional annotation queries

Builtin queries to add additional annotation to a BuxcoDB database

---

Description

These functions take BuxcoDB objects as input and return an SQL query (in the SQLite dialect) which results in the creation of a table containing additional annotation that is not parsed from the Buxco CSV file. These annotations are meant to be categorical labels such as whether a datapoint was part of an acclimation or experimental run which may not necessarily be encoded directly in the Buxco CSV file. A user can also specify functions in this form to add custom annotations to the database. Typically the end user does not need to call these functions directly, instead they are supplied to the `code` argument of the `addAnnotation` method. This method then calls the function internally in the process of generating the annotation table.

Currently implemented queries are:

`day.infer.query`: Computes the number of days past the first observed timestamp for a given sample. This day assignment should be compared to the experimental timepoint potentially recorded in the 'Phase' element of the Buxco CSV file. This value will exist as the 'Rec_Exp_date' column of the `data.frame` returned by `retrieveData`.

`break.type.query`: Labels each datapoint as belonging to one of several categories. Typically each datapoint would be part of the 'ACC' or 'EXP' groups corresponding to acclimation or experimental readings. The assignment of these categories is based on observations regarding the current breakpoint number relative to the number of breakpoints observed for a given sample and given day. Typically the acclimation readings are reported first followed by the experimental readings. Additionally, 'UNK' or 'ERR' indicate deviations from the expected sample-breakpoint relationships. Specifically, 'UNK' refers to the case where only one breakpoint was observed for that animal and day so the category is unknown. The presence of 'ERR' categories indicate potential issues with how the data was parsed and should be reported to the package author for investigation. NOTE: Currently, `day.infer.query` needs to be run prior to this query as it draws upon the computed date as opposed to the labeled date.

Usage

```
break.type.query(obj)
day.infer.query(obj)
```

Arguments

obj	A BuxcoDB object.

Value

A character string representing an SQL query.

Author(s)

Daniel Bottomly

References

http://www.sqlite.org/

See Also

BuxcoDB

Examples

```
samp.file <- sample.db.path()
new.file <- file.path(tempdir(), basename(samp.file))

stopifnot(file.copy(samp.file, new.file, overwrite=TRUE))

bux.db <- makeBuxcoDB(new.file)

head(retrieveData(bux.db))

#query used to compute experiment day relative to the initial timepoint.
day.infer.query(bux.db)

addAnnotation(bux.db, query=day.infer.query)

head(retrieveData(bux.db))
```

Buxco file Parsers *Functions for parsing Buxco respirometry data*

Description

A typical Buxco respirometry experiment involves collecting repeated measures on both acclimation and experimental data related to metabolism and respiration. The parse.buxco function creates a local database representation of a given file to facilitate fast retrieval and ultimately analysis. The user should only use parse.buxco with parse.buxco.basic mainly used for testing purposes.

Usage

```
parse.buxco(file.name = NULL, table.delim = "Table", burn.in.lines = c("Measurement", "Create measu
  chunk.size = 500, db.name = "bux_test.db", max.run.time.minutes = 60, overwrite = TRUE, verbose=TRU
parse.buxco.basic(file.name=NULL, table.delim="Table", burn.in.lines=c("Measurement", "Create mea
```

Arguments

file.name	A path to the Buxco CSV file. See vignette for further description of the required file format.
table.delim	A character vector of length one containing the pattern used to divide the Buxco file into tables.
burn.in.lines	A character vector containing the patterns used to divided each Buxco table into readings for different animals.
chunk.size	The number of lines that should be read in at a given time, more lines results in more memory consumption and quicker parsing speed.
db.name	The file name of the local database to create.
max.run.time.minutes	The maximum time in minutes that a acclimation or experimental run should take. A warning will be given if this is exceeded and the data will be treated as if there were seperate runs.
overwrite	A logical value specifying whether the local database specified in db.name should be overwritten if exists.
verbose	A logical value specifying if additional information should be printed as parsing progresses.
make.package	A logical value indicating whether a package should be created in db.name instead of a database file.
author	If make.package == T, a string value indicating who the package author should be.
author.email	If make.package == T, a string value indicating what the package author's email address should be.

Details

The parse.buxco function reads in the specified file in chunks. It uses the lines specified in burn.in.lines to determine whether a 'break' has been reached. Each break signifies that a series of readings for several animals has been completed and so only upon reaching a break is measurement data written to the database for the completed measurement sets. Because of this, there will always be some memory overhead in proportion to the number of readings in each series irrespective of chunk.size. To access the database in R, use the convienience method retrieveData.

Value

The parse.buxco function returns a BuxcoDB object. The parse.buxco.basic function returns a data.frame.

Note

parse.buxco.basic should not be used directly as it is extremely memory intensive as it parses the entire file at once and returns a data.frame result.

Author(s)

Daniel Bottomly

References

http://www.buxco.com/

See Also

BuxcoDB, retrieveData

Examples

```
bux.db <- parse.buxco(file.name=buxco.sample.data.path(), db.name =tempfile())

head(retrieveData(bux.db))
```

BuxcoDB-class *Class* "BuxcoDB"

Description

This is the main class of the plethy package. Each object of this class simply holds the name of the database as well as the name(s) of any additional tables added through addAnnotation.

Objects from the Class

Objects should be created by calls of the form `makeBuxcoDb(db.name=NULL,annotation.table="Additional_labels`

Slots

`db.name`**:** Object of class `"character"` Stores the path to the database.

`annotation.table`**:** Object of class `"character"` Stores the name of the additional annotation table in the database to be created if `addAnnotation` is called.

Methods

addAnnotation `signature(obj = "BuxcoDB")`: Carry out an additional query to populate columns in a new or existing table. Each query should be specified by a function taking a BuxcoDB object as an argument and returning valid SQL. Additionally if the `index` argument is set to TRUE, indexes will be placed on the columns in the table. See the vignette for an example of how this function should be used in practice.

annoTable `signature(obj = "BuxcoDB")`: Retrieve the name of the table where the additional annotation is to be stored. This table will not exist until the `addAnnotation` method is called first.

annoCols `signature(obj = "BuxcoDB")`: Returns a vector of the column names in the additional annotation table.

annoLevels `signature(obj = "BuxcoDB")`: Returns a list of the same length as the number of annotation columns in the additional annotation table with each element containing the unique values for each column.

dbName `signature(obj = "BuxcoDB")`: Retrieve the path to the plethy database

retrieveData `signature(obj = "BuxcoDB")`: With no arguments, this method will retrieve all available data in the database as a `data.frame`. Specifying single or a vector of values to one or more of `samples`, `variables`, `tables`, `phase` or one of the addtional annotation names specified using `addAnnotation` will retrieve a subset of Buxco data. Note that the valid variable (column) names for the annotation table can be found through the `annoCols` method and the valid variables for each variable name can be found through `annoLevels`.

retrieveMatrix `signature{obj = "BuxcoDB"}`: This method first uses 'retrieveData' to retrieve the specified subset of data and then summarizes it into a matrix or array by first applying the specified 'summary.func'. The default form of the array is Samples x Timepoint x Variable though this can be changed by supplying a different formula.

samples `signature(obj = "BuxcoDB")`: Returns a vector of the unique samples in the database

tables `signature(obj = "BuxcoDB")`: Returns a vector of the unique tables in the database

variables `signature(obj = "BuxcoDB")`: Returns a vector of the unique Buxco measurement variables in the database

tsplot `signature(obj = "BuxcoDB")`: Produces a line plot of the subset of the subset of the data as specified in 'retrieveData' after summarizing each sample for each timepoint by the function specified in 'summary.func'. Colors and a legend can be added for a single experimental variable specified in 'exp.factor'. The x,y and legend labels can be modified using the 'xlab', 'ylab' and 'legend.name' arguments respecitively.

mvtsplot `signature(obj = "BuxcoDB")`: Produces a multivariate timeseries plot adapted from the function in the 'mvtsplot' CRAN package. By default it will produce a plot containing a heatmap like image for each sample centered and scaled by row along with a boxplot depicting the overall distribution of the variable specified in 'plot.value'. At the bottom of the plot, a line plot shows the median trend over time. The data can also optionally be grouped as either an 'inner.group' or an 'outer.group'. The 'inner.group.name' defaults at the sample name though it can be used to specify a group in its own right. The 'outer.group.name' subdivides the plot visualy and the medians are computed over each group seperately. Colors of the 'outer.group' can be changed by supplying a color vector to 'outer.col' named by the levels of 'outer.group.name'. The colors used in the heatmap image can be influenced by supplying a different palette from RColorBrewer to 'colorbrewer.pal'. Note that the data is first summarized by sample and day using the function supplied to 'summary.func'.

summaryMeasures `signature(obj = "BuxcoDB"),summary.type=c("time.to.max.response","max.response"`
`data.frame(Value=mean(x$Value)),samples=NULL,variables=NULL,tables=NULL,Break_type_label="E`
Returns a data.frame containing a summaries of the variables for each sample with respect to the main time element specified in 'day.summary.column' after first summarizing each variable and timepoint for each sample by 'sample.summary.func'. The data can be subsetted ahead of time using the samples, variables, tables and Break_type_label arguments. Note this assumes that `break.type.query` was previously run using `addAnnotation`.

Author(s)

Daniel Bottomly

See Also

`parse.buxco`

Examples

```
samp.file <- sample.db.path()
new.file <- file.path(tempdir(), basename(samp.file))

stopifnot(file.copy(samp.file, new.file, overwrite=TRUE))

bux.db <- makeBuxcoDB(new.file)

show(bux.db)

head(retrieveData(bux.db))

annoCols(bux.db)

annoLevels(bux.db)

dbName(bux.db)

samples(bux.db)

tables(bux.db)

variables(bux.db)

addAnnotation(bux.db, query=day.infer.query, index=FALSE)

annoCols(bux.db)
annoLevels(bux.db)

addAnnotation(bux.db, query=break.type.query, index=TRUE)

annoCols(bux.db)
annoLevels(bux.db)

head(retrieveData(bux.db))

retrieveMatrix(bux.db)[1:5,1,1:5]
```

---

dbImport *Import data into a BuxcoDB database*

Description

The main purpose of this function is to add data originally retrieved from the `retrieveData` method into a new or existing `BuxcoDB` database. This will most frequently be useful in the context of a merging procedure, however it also can facilitate data sharing and/or communication between seperate DBMS systems.

Usage

```
dbImport(bux.db = NULL, bux.dta, db.name = "merge_test_1.db", debug = FALSE)
```

Arguments

bux.db	Either NULL or a BuxcoDB object
bux.dta	A data.frame consistent with the database structure of the BuxcoDB database, most easily created from a call to retrieveData.
db.name	Path to the new SQLite database to create.
debug	Logical value indicating whether the function should be more verbose.

Details

If only bux.dta is supplied and not bux.db, then a new database will be created and populated at db.name from its contents. If both db.name and bux.dta are supplied then the data.frame will be loaded into the existing database.

Value

A BuxcoDB object pointing to the newly created database.

Author(s)

Daniel Bottomly

See Also

BuxcoDB, retrieveData

Examples

```
bux.db <- makeBuxcoDB(sample.db.path())

samp.1 <- retrieveData(bux.db, samples="8034x13140_5")

test.db <- "test_db.db"

if (file.exists(test.db))
{
file.remove(test.db)
}

#create a new database from the output
db.1 <- dbImport(bux.db=NULL, bux.dta=samp.1, db.name=test.db)

samples(db.1)

test.db.2 <- "test_db_2.db"

if (file.exists(test.db.2))
{
file.remove(test.db.2)
}

samp.2 <- retrieveData(bux.db, samples="8034x13140_11")

db.2 <- dbImport(bux.db=db.1, bux.dta=samp.2, db.name=test.db.2)
```

```
samples(db.2)

file.remove(test.db.2)
file.remove(test.db)
```

`Utility functions` *Utility functions to assist with QA/QC and analysis of plethysmography data*

Description

After creation of a database, often additional data needs to be added or modified. These functions assist with the common tasks that occur when working with Buxco whole body plethysmography data such as adding labels based on the sample IDs in the case of `add.labels.by.sample` or modifying labels that have previously been added in the case of `adjust.labels`. The `get.err.breaks` function produces a summary of the samples and timepoints that have the specified value for the 'Break_type_label' column (such as 'ERR' or 'UNK') and whether they are close to the expected value for either and experimental or acclimation run. This can occur if there was only an experimental run for some samples or if other anomalies occured. The user can then inspect these new labels wihtin the data.frame, modify them manually if necessary and use the data.frame as input to the `adjust.labels` function which replaces the original labels and moves the original labels to another column for future reference.

Usage

```
add.labels.by.sample(bux.db, sample.labels)
get.err.breaks(bux.db, max.exp.count=150, max.acc.count=900, vary.perc=.1, label.val="ERR")
adjust.labels(bux.db, err.breaks.dta)
proc.sanity(bux.db, max.exp.time=300, max.acc.time=1800, max.exp.count=150, max.acc.count=900)
```

Arguments

`bux.db`	An object of class `BuxcoDB`
`sample.labels`	A data.frame with a column named 'samples' and optionally a column named 'phase' with values corresponding to the sample names and Phase values (e.g. recorded experimental timepoint) in the database. The other columns will be add to the annotation table and any sample not included in the data.frame will have their labels set to NULL.
`err.breaks.dta`	A data.frame produced by `get.err.breaks` function.
`max.exp.time`	The maximum time a given experimental run should take in seconds
`max.acc.time`	The maximum time a given acclimation run should take in seconds
`max.exp.count`	The maximum number of records expected for the experimental run.
`max.acc.count`	The maximum value of records expected for the acclimation run.
`vary.perc`	The size of a percent decrease relative to the maximum experimental or acclimation run tolerated and still allow assignment to that category. Needs to be a value between 0 and 1.
`label.val`	A single character string observed in the Break_type_labels column of the annotation table (cannot be 'ACC' or 'EXP').

Value

`add.labels.by.sample` and `adjust.labels` modify tables in the SQLite database pointed to in the BuxcoDB object so nothing is returned. `get.err.breaks` returns a data.frame summarizing the samples and timepoints with a given `label.var`.

Author(s)

Daniel Bottomly

See Also

`parse.buxco`,`BuxcoDB`

Examples

```
##set up a test dataset using internal functions
##should label sample_1 as ACC and EXP and samples 2 and 3 as UNK
##sample_3 should be too divergent from the expected 150 rows, so
##the inferred labels should remain 'UNK'

samples=c(NA, "sample_1", NA, "sample_1", "sample_2", "sample_3")
count = c(NA,900, NA,150, 150, 110)
measure_break = c(FALSE, FALSE, TRUE, FALSE, FALSE,FALSE)
table_break = c(TRUE, rep(FALSE, length(samples)-1))
phase = rep("D1", length(samples))

err.dta <- data.frame(samples=samples, count=count, measure_break=measure_break, table_break=table_break, pha

sim.bux.lines <- plethy:::generate.sample.buxco(err.dta)

temp.file <- tempfile()
temp.db.file <- tempfile()
write(sim.bux.lines, file=temp.file)
test.bux.db <- parse.buxco(file.name=temp.file, db.name=temp.db.file, chunk.size=10000)
addAnnotation(test.bux.db, query=day.infer.query, index=FALSE)
addAnnotation(test.bux.db, query=break.type.query, index=TRUE)

##quick test of data

test <- proc.sanity(test.bux.db)

head(test$count)

test$time

##get a summary of this

unk.summary <- get.err.breaks(test.bux.db, label.val="UNK")
table(unk.summary$Sample_Name, unk.summary$inferred_labs)

##use the summary to change the Break_type_label column in the annotation table

head(retrieveData(test.bux.db))

adjust.labels(test.bux.db, unk.summary)
```

```
head(retrieveData(test.bux.db))

##additional annotations can be added to the database based on sample ID

sample.labels <- data.frame(samples=c("sample_1","sample_3"), response_type=c("high", "low"),stringsAsFactor

add.labels.by.sample(test.bux.db, sample.labels)

final.dta <- retrieveData(test.bux.db)

head(final.dta)

##should be 'high' for sample_1 and 'low' for sample_3 with NAs for sample_2

table(final.dta$Sample_Name, final.dta$response_type, useNA="ifany")
```

Index