Package 'bgfanalyzer'

September 26, 2026

Type Package

Title Analyze Microbial Biogas Fermentation Data

Version 1.1.0

Description Provides a new S3 class object and relevant methods to analyze biogas fermentation data.
It includes three workflows. One is specialized to a commercially available lab-scale fermentation system (see e.g. Nwaigwe (2018) <doi:10.1115/ES2018-7553>).
The second provides more flexibility and allows to import data from plain text files.
The last workflow offers the most flexibility as it doesn't expect external input files but relays on interactive user input.
Although the focus is set on biogas fermentations, concepts and workflows may be also applicable to other fermentations even if not a gaseous product is measured.
Furthermore, it provides functions that bridge to established plot engines (e.g. 'ggplot2' or 'plotly') for data visualisation.
'bgfanalyzer' catches up an idea of Hafner et al. (2018) <doi:10.1016/j.softx.2018.06.005> of using R to standardize research within the biogas field.
For more details on standardization efforts within the biogas research field see Hollinger et al. (2016) <doi:10.2166/wst.2016.336> and Hollinger et al. (2021) <doi:10.2166/wst.2020.569>

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports stats, utils, rlang, zoo, dplyr, graphics, ggplot2, plotly

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 3.5)

Config/roxygen2/version 8.1.0

VignetteBuilder knitr

NeedsCompilation no

Author Maximilian Strick [aut, cre, cph]

Maintainer Maximilian Strick <maximilianb.strick@gmail.com>

Repository CRAN

Date/Publication 2026-09-26 16:50:25 UTC

Contents

`add_bmp_measurement` *Supportive data import functions*

Description

A set of internal functions that bridge the data import and object creation when using from_AMPTSV2_report or from_standard_record to create an `BGF`. These functions are not meant for direct user interaction.

Usage

```
add_bmp_measurement(x, path, mode = "auto", feedback = FALSE)

add_standard_record(
  x,
  path,
  header = TRUE,
  dec = ".",
  sep = "\t",
  units = "hours",
```

```
  time_col = "time",
  RName = "R1",
  product_col = "GCounter..ml.",
  feedback = TRUE
)

add_ExpPara(x, rawReport, feedback = FALSE)

add_ExpSetup(x, rawReport, feedback = FALSE)

sort_AMPTSV2_reactors(x, rawReport, feedback = FALSE)

sort_standardReport(x, rawReport, RName, product_col)
```

Arguments

x	a BGF
path	a path pointing to an external data file. Either a .csv-file generated by the AMPTS II (BioProcess Control; Lund; Sweden), or a text-file with at least biogas production (cumulative volume) and time data.
mode	the mode add_bmp_measurement treats the imported data at path. Currently only 'auto' and 'AMPTSV2' are the only implemented options.
feedback	logic; if TRUE the function will print a textual feedback to the console
header	logic; does the matrix like structure at path has a header
dec	a single character specifying the decimal separator
sep	a single character specifying the column separator
units	the desired unit of the time difference calculated by calc_FR_time
time_col	a character specifying the name of the column with time information in the external data file
RName	a character specifying the reactor name of the biogas fermentation to be added. Defaults to 'R1'
product_col	a character specifying the name of the column with the cumulative biogas volume data in the external data file
rawReport	either a list or a data.frame created by read_raw_AMPTSV2_report or import_standard_record, respectively.

Details

The function add_bmp_mesurement is called by from_AMPTSV2_report. It imports a .csv-file created by the AMPTS II (BioProcess Control; Lund; Sweden) from a location on the system specified in path and adds it to a BGF specified in x.

Internally, add_bmp_mesurement calls read_raw_AMPTSV2_report to import the external file to a list and then it calls add_ExpPara, add_ExpSetup and sort_AMPTSV2_reactors to transfer the data to the BGF.

The function add_standard_record is similar to add_bmp_measurement, put in this case path must point to an external text file with a matrix like data structure. This matrix must have at least two columns, one with information on when the measurement was recorded (time_col), the other with a cumulative biogas volume measurement (product_col). Additional columns can be present and will be imported as well.

Internally, add_standard_record calls import_standard_record to import the data and next it calls calc_FR_time to calculate a standardized fermentation time for the imported data set. Finally, it calls sort_standardReport to transfer the imported data to the BGF The function will not remove empty rows resulting from object creation within the BGF. Thus, it is advised to use update_BGF to ensure the integrity of the BGF.

The function add_ExpPara is called by add_bmp_measurement and acts as a wrapper for add_ExpParam to transfer experimental meta data to a BGF object. It expects a list created by read_raw_AMPTSV2_report as a second argument.

Similarly,the function add_ExpSetup is also called by add_bmp_measurement, but acts as a wrapper for add_metaData to transfer fermentation meta data to a BGF object. Consequently it also expects a list created by read_raw_AMPTSV2_report as a second argument.

In addition, sort_AMPTSV2_reactors is also called by add_bmp_measurement and acts as a wrapper for add_BG_measurement and alter_BG_measurement to transfer cumulative biogas volume and biogas flow data to a BGF. Like the former two functions, it expects a list generated via read_raw_AMPTSV2_report as second argument

Additional details...

Value

a BGF

Examples

```
# create an example BGF
myBGF <- BGF(ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
             name = "myBGF",
             ProcessTemp = 52,
             MeasurementType = "AMPTSV2")

myBGF # print the BGF

# add data from external files
myBGF <- add_bmp_measurement(
        x = myBGF,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

myBGF # print the BGF

# create another example BGF
myBGF2 <- BGF("A")

myBGF2 # print the BGF

# add data from an external file (standard record)
```

```
myBGF2 <- add_standard_record(
        x = myBGF2,
       path = base::system.file("extdata","Fermentation_B.tsv",package = "bgfanalyzer"),
        RName = "R1",
        time_col = "UTC",
        product_col = "GCounter..ml.")

myBGF2 # print the BGF

# create an additional example BGF
myBGF3 <- BGF(c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"))

myBGF3 # print the BGF

# import standard record
rawReport <- read_raw_AMPTSV2_report(
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

# add experimental meta data to BGF
myBGF3 <- add_ExpPara(myBGF3,rawReport,TRUE)

myBGF3 # print the BGF

# add fermentation meta data to BGF
myBGF3 <- add_ExpSetup(myBGF3,rawReport,TRUE)

myBGF3 # print the BGF

# add biogas volume and flow data
myBGF3 <- sort_AMPTSV2_reactors(myBGF3,rawReport)

myBGF3 # print the BGF

# create an empty example BGF
myBGF4 <- BGF("myBGF4")

myBGF4 # print the BGF

# import a standard record
rawReport2 <- import_standard_record(
       ipath = base::system.file("extdata","Fermentation_B.tsv",package = "bgfanalyzer"))

# calculate a fermentaion/ observation time
rawReport2<-calc_FR_time(rawReport2,1,units="hours")

# add imported data to BGF
myBGF4 <- sort_standardReport(myBGF4,rawReport2,"R1",3)

myBGF4 # print BGF
```

add_whatever *Add data to a* BGF

Description

A set of functions to add data to each layer of a BGF.

Usage

```
add_whatever(x, layer, what, lab = "newData", feedback = FALSE)

add_ExpParam(x, what, feedback = FALSE)

add_metaData(x, what, makeCol = TRUE, lab = "newData", feedback = FALSE)

add_BG_measurement(x, reactor, time, col, measurement, feedback = FALSE)

add_BG_parameter(
  x,
  parameter,
  reactor,
  time = NULL,
  value = NULL,
  name = NULL,
  makeCol = TRUE,
  cut_zero = TRUE,
  interpolate_missing = TRUE,
  default_start = 0
)
```

Arguments

x	a BGF
layer	a character referring a layer of a BGF; either 'ExpParam' or 'metaData'
what	a character referring either to a list entry of the ExpParam layer, or a column of the metaData layer of a BGF
lab	a character providing the name for the new data
feedback	logic, if TRUE the function will print a feedback to the R console
makeCol	logic, default is TRUE. Should a new column be created? If FALSE the data will be added to an existing column named name
reactor	a character referring to a reactor/ fermentation of a BGF; in case of add_BG_parameter must have the same length as 'parameter'. If 'parameter' is a data.frame 'reactor' must have length 1 and specifies the reactor the measurements belong to

time	for add_BG_measurement, a numeric specifying the time that has passed since fermentation start; for add_BG_parameter either a integer indicating the position of the 'time' column in argument 'parameter' or a numeric of the same length as argument 'parameter' specifying the time that has passed since fermentation start (see Details)
col	either a character or numeric referring to an existing column of the BioGasData layer of a BGF
measurement	a numeric; measurement value to be added to the BioGasData layer of a BGF
parameter	either a data.frame with at least two columns or a numeric carrying the data to be added
value	defaults is NULL; must be an integer referring to the measurement data column in 'parameter' if 'parameter' is a data.frame
name	defaults is NULL; can be a character specifying the label of the new column being created in the BioGasData layer of a 'BGF"'
cut_zero	logic, default is TRUE. If FALSE measurements with a negative time (measurements that were acquired before the fermentation started) will NOT be removed from BioGasData
interpolate_missing	logic, default is TRUE. Indicating if missing values between to measurements should be interpolated
default_start	a numeric, a default measurement value that is assumed in the begin of the fermentation for the parameter to be added

Details

There are five functions that can be useful to add data to BGF. The first is add_whatever, which allows to add data either to the ExpParam or the metaData layer of a BGF. The functions add_ExpParam and add_metaData are wrapper functions that internally call add_whatever with the 'layer'

For add_ExpParam argument 'what' is expected to be a named vector of length 1. The name of the vector will be kept as label of a list entry in the ExpParam layer and the vectors value will be the value of that list entry.

For add_metaData argument 'what' is expected vector of the same length as number of rows in the metaData layer. It will copy that vector to the metaData layer as a new column. That column can be named using the 'lab' argument. Otherwise a standard name is created. The function checks if the chosen name already exists in metaData and will eventually generate a new name.

The function add_BG_measurement adds a single measurement value to an existing column of the BioGasData layer. To this end the user must specify to which reactor of the BGF the measurement belongs to (argument 'reactor') and at which time the measurement was taken (argument 'time'). The argument 'col' is the name or position of the column in the BioGasData layer to which the measurement should be added.

The function add_BG_parameter allows to add a new data column to the BioGasData layer of a BGF. It is very useful if not all measurements for a fermentation can be imported from the same standard record, e.g. if exhaust gas volumes are measured by one device, while the gas composition is registered by another.

Value

a BGF

Examples

```
# create an example BGF
myBGF <- from_AMPTSV2_report(
        ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
        BlankLabel = "Blank",
        name = "Test",
        InocToSubRatio=2,
        ProcessTemp = 52,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer")
        )

# add a start date to 'ExpParam'
myBGF <- add_whatever(myBGF,"ExpParam",what = c("Start date"="2026-05-04 12:00:00"))

# add organic total solutes measurements to 'metaData'
myBGF <- add_whatever(
        x = myBGF,
        layer = "metaData",
        what = c("oTS"=c(2.3,2.3,98.8,4.5,4.4,4.6,3.5,3.6,3.5,3.8,3.9,4,3.5,3.6,3.8))
        )

# add a end date to 'ExpParam'
myBGF<-add_ExpParam(x = myBGF,what =  c("End date"="2026-06-04 12:00:00"))

# add total solutes measurement to 'metaData'
myBGF<-add_metaData(
        x = myBGF,
        what =  c(3.3,3.3,99.8,6.5,6.6,6.6,6.5,6.6,6.5,6.8,6.9,6,6.5,6.6,6.8),
        lab="TS")

# add a new 'product' gas measurement for reactor 'R1' at time '49' (days) after fermentation start
myBGF<-add_BG_measurement(x = myBGF,reactor = "R1",time = 49,col = "product",measurement = 7777)

# create a second example BGF from a standard record with exhaust gas data
myBGF2<-from_standard_record(
        ReactorLayout = "A",
        ProcessTemp = 80,
        InocToSubRatio = .1,
       path = base::system.file("extdata","Fermentation_A.tsv",package = "bgfanalyzer"),
        time_col = 1,
        product_col = 3)

# import another standard record that provides additional information on the 'BioGasData' layer
Gas_comp<-import_standard_record(
        ipath = base::system.file(
            "extdata",
            "gasq_A.tsv",
            package = "bgfanalyzer"
```

```
         ),
        mkFRTime = "2025-01-15 17:00:00",
        FRTime_col = 1,
        units = "hours")

# add the gas composition data to the BGF and interpolate missing values
myBGF2<-add_BG_parameter(
        x = myBGF2,
        parameter = Gas_comp,
        reactor = "R1",
        time = 3,
        value = 2,
        name = "H2",
        cut_zero = TRUE,
        interpolate_missing = TRUE)
```

`alter_whatever` *Alter data within a* `BGF`

Description

A set of functions that allow to alter data in each layer of a `BGF`.

Usage

```
alter_whatever(x, layer, what, value, ID = NULL, feedback = FALSE)

alter_BG_measurement(
  x,
  reactor_id,
  time_id,
  col,
  measurement,
  ID = NULL,
  feedback = FALSE
)
```

Arguments

`x`	a `BGF`
`layer`	a `character` referring a layer of a `BGF`; either 'ExpParam' or 'metaData'
`what`	a `character` referring either to a list entry of the `ExpParam` layer, or a column of the `metaData` layer of a `BGF`
`value`, `measurement`	either a `character`, `numeric` or `logic`. The desired new value

ID	defaults to NULL, for alter_whatever can be a character or integer referring to row names in the metaData layer is. For alter_BG_measurement it can be an integer referring to a row in the BioGasData layer of a BGF
feedback	logic, if TRUE the function will print a feedback to the R console
reactor_id	a character identifying a the reactor/ fermentation whose entry in BioGasData needs to be changed
time_id	a numeric specifying the time that has passed since fermentation start for the chosen reactor
col	a character specifying the name of the column of the BioGasData layer that needs to be changed

Details

The two functions alter_whatever and alter_BG_measurement can be used to change information in each layer of a BGF. The function alter_whatever can be used to change data in the ExpParam or metaData layer.

The function alter_BG_measurement changes a single entry in the BioGasData layer of the BGF. This entry can be selected by providing the name of the reactor for which a measurement should be changed, together with the time after fermentation start when the measurement was acquired. Alternatively, the row number in the BioGasData layer of that measurement can be provided to 'ID'.

Value

a BGF

Examples

```
# create an example BGF
myBGF <- from_AMPTSV2_report(
        ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
        BlankLabel = "Blank",
        name = "Test",
        InocToSubRatio=2,
        ProcessTemp = 52,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer")
        )

# change name to 'newName'
myBGF <- alter_whatever(myBGF,"ExpParam","name","newName",feedback=TRUE)

# exclude reactor 5,6 and 7
myBGF <- alter_whatever(myBGF,"metaData","Excluded",TRUE,c("R5","R6","R7"),feedback=TRUE)

# exclude all reactors
myBGF <- alter_whatever(myBGF,"metaData","Excluded",TRUE)

# change the 'product' column of reactor 1 at time 0 to 7777
myBGF <- alter_BG_measurement(myBGF,"R1",0,"production",7777)
```

```
#  do the same for reactor 2 but index via 'ID'
myBGF <- alter_BG_measurement(myBGF,ID=2,col="production",measurement=7777)
```

BGF *Helper to set up a* BGF *object*

Description

Three helper functions exist that allows users to set up BGF objects either from scratch or from external data files.

Usage

```
BGF(
  ReactorLayout,
  BlankLabel = "Blank",
  name = "new_BGF",
  ProcessTemp = NA,
  InocToSubRatio = 2,
  MeasurementType = NA
)

from_AMPTSV2_report(
  ReactorLayout,
  BlankLabel = "Blank",
  name = "new_BGF",
  ProcessTemp,
  InocToSubRatio,
  path,
  feedback = FALSE
)

from_standard_record(
  ReactorLayout,
  ProcessTemp,
  InocToSubRatio,
  path,
  time_col,
  product_col,
  BlankLabel = "Blank",
  name = "new_BGF",
  units = "hours",
  RName = "R1",
  feedback = FALSE,
  ...
)
```

Arguments

ReactorLayout	A character vector providing the reactor layout, e.g. the grouping factor used for plotting and yield calculation of fermentation(s) in a BGF object
BlankLabel	A character string indicating which group in ReactorLayout is the inoculum used for net gas/ yield calculation
name	A character vector specifying the name of the new BGF object
ProcessTemp	The process temperature of the fermentation(s) to be stored in the BGF object
InocToSubRatio	The inoculum to substrate ratio (= inoculation strength) of the fermentation(s) to be stored in the BGF object
MeasurementType	An optional character string indicating the measurement type of the BGF object to be created
path	A path pointing an external data input file must be. Must be in a tidy format for from_standard_record or a report generated by an AMPTS II (see Details)
feedback	Logical if TRUE the function prints a summary to the R console
time_col	numeric. Indicates the position of the time stamp column in the imported report
product_col	numeric. Indicates the position of the product column (e.g. cumulative biogas volume) in the imported report
units	character string. Units in which the results are desired. Can be abbreviated.
RName	character. Label to be added to the data of the imported report. Should match a value of ReactorLayout
...	further arguments passed to read.table

Details

All three helper functions internally call new_BGF() with distinct parameters arguments pre-set. The easiest way to generate a BGF object is the BGF() function. It only needs the ReaktorLayout argument to be specified by the user and will subsequently create a object of class BGF.

If from_AMPTSV2_report is used, path must point to the report_yyyy-mm-dd_HHMM.csv-file created by an AMPTS II (BioProcess Control; Lund; Sweden). This special feature was included as the AMPTS II is widely distributed among biogas labs and several R tutorials exist online, that also refer on this system.

When from_standard_record is used to create the BGF object any text file storing biogas fermentation data could serve as a template. It is a wrapper to call read.table with certain arguments pre-set. These are: dec=".", sep="\t" and header=TRUE.

Such a text file must have a matrix like structure in which each row represents a unique observation. It must have at least two columns. One with the time stamp of the observation (format: %y-%m-%d %H:%M:%S). The other one must be an accumulating volume measurement of exhaust gas (biogas) at this moment. Further columns providing additional information (such as pH, temperature or RedOx potential) can be present and will be imported as well.

Value

A BGF object

Examples

```
# create a simple BGF without data
BGF(LETTERS[c(1:5)])

# create a BGF based on the report of an AMPTS II
from_AMPTSV2_report(
        ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
        BlankLabel = "Blank",
        name = "Test",
        InocToSubRatio=2,
        ProcessTemp = 52,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

# create a BGF from a minimal external data file
from_standard_record(ReactorLayout = "A",
        ProcessTemp = 80,
        InocToSubRatio = .1,
       path = base::system.file("extdata","Fermentation_A.tsv",package = "bgfanalyzer"),
        time_col = 1,
        product_col = 3)
```

BGF_defaultcolors *The default color set to be used by* bgf_plot

---

Description

The default color set used when plotting a BGF using the advanced plotting functions. It contains 3 * 8 = 24 colors.

Usage

```
BGF_defaultcolors
```

Format

An object of class character of length 24.

---

bgf_interpolation *Data interpolation algorithm*

Description

A function that allows to interpolate missing values in a numeric vector of a data.frame. It requires a second column in the data.frame representing time, as well as a third column representing grouping information.

Usage

```
bgf_interpolation(df, x, t, group, end = TRUE, sub_zero = NULL, offset = 1)
```

Arguments

df	a data.frame
x	an integer specifying the position of the vector with missing values that should be interpolated
t	an integer specifying the position of the vector with time data over which to interpolate
group	an integer specifying the position of a grouping vector within the input data
end	logic; default is TRUE. Should interpolation end if the last valid 'x' value does not match the final 't' value?
sub_zero	default is NULL. Can be a numeric which will be used to replace all (interpolated) values in 'x' that are below '0'.
offset	an integer; default is 1. Can be used to specify the offset (distance between valid 'x' values) that is used during terminal data extrapolation (argument 'end = FALSE'). With the default setting ('offset = 1') the last valid 'x' value and 1 value before that are selected.

Details

The function bgf_interpolation takes a data.frame as input and returns a modified version of it as output. In particular, it interpolates missing values in a numeric vector (argument 'x') based on a second vector (argument 't') without missing values.

It is possible to stop the interpolation (argument 'end'), if no valid value in 'x' occurs after a missing value, or to use linear extrapolation to fill terminal NA's. The slope of the extrapolation is based on the last valid 'x' value and the n^{th} valid 'x' value before (argument 'offset').

Furthermore, if linear extrapolation leads to e.g. negative gas concentrations or volumes, it is possible to replace these wrong values with any replacement (argument 'sub_zero')

Interpolation and extrapolation are done specifically for groups within the data (argument 'group')

Value

a data.frame

Examples

```
# create an example data.frame
gap_data=data.frame(
              time=as.numeric(rep(c(1:30),3)),
              value=c(0,rep(NA,3),7,14,26,44,72,rep(NA,8),314,350,
                      377,rep(NA,5),471,rep(NA,4),0,rep(NA,7),100,
                      127,163,rep(NA,4),217,rep(NA,5),267,rep(NA,4),
                      0,rep(NA,3),0,rep(NA,3),5,rep(NA,3),12,rep(NA,3),
                      32,rep(NA,3),35,rep(NA,3),33,rep(NA,3),28,rep(NA,3),
                      5,NA),
```

```
                cat=c(rep("exGas",30),rep("exGas_2",30),rep("cBioG",30)))

# interpolate and extrapolate missing values, replace values below '0' with '0'
closed_data<- bgf_interpolation(gap_data,2,1,3,end = FALSE,sub_zero = 0)

# inspect the differences
plot(value ~ time,data=gap_data,col=as.factor(cat))
legend(legend = levels(as.factor(gap_data$cat)),col=c(1,2,3),x = "topleft",pch=15)

plot(value ~ time,data=closed_data,col=as.factor(cat))
legend(legend = levels(as.factor(closed_data$cat)),col=c(1,2,3),x = "topleft",pch=15)
```

---

bgf_plot *Advanced* BGF *visualization*

Description

A set of functions that allow advanced data visualization employing ggplot2 and plotly. The two main functions are bgf_plot, which is a high end wrapper to produce frequently needed plots and plot_curve, which is offers more freedom during plot creation.

Usage

```
bgf_plot(x, type = "all", ...)

plot_curve(
  x,
  what,
  color,
  col = bgfanalyzer::BGF_defaultcolors,
  coltitle = "Reactor:",
  col_names = NULL,
  title = NULL,
  subtitle = NULL,
  xlab = ggplot2::waiver(),
  ylab = ggplot2::waiver(),
  keep_excluded = TRUE,
  interaction = FALSE
)

plot_product_curve(x, ...)

plot_production_curve(x, ...)

plot_rel_production_curve(x, ...)

plot_netProduct_by_Layout(
```

```
  x,
  col = bgfanalyzer::BGF_defaultcolors[2 + 3 * c(0:7)],
  coltitle = "Layout:",
  col_names = NULL,
  title = NULL,
  subtitle = NULL,
  xlab = paste0("observation time [", x$ExpParam$timeScale, "s]"),
  ylab = "net exhaust gas volume",
  keep_excluded = TRUE,
  interaction = FALSE
)

colplot_yield(
  x,
  hide = NULL,
  Excluded = FALSE,
  col = bgfanalyzer::BGF_defaultcolors[2 + 3 * c(0:7)],
  coltitle = "Layout:",
  col_names = NULL,
  title = NULL,
  subtitle = NULL,
  yield_label = FALSE,
  yield_label_pos = 100,
  yield_unit = "Nml/gVS",
  interaction = FALSE
)

boxplot_yield(
  x,
  timep = "all",
  hide = NULL,
  Excluded = FALSE,
  col = bgfanalyzer::BGF_defaultcolors[2 + 3 * c(0:7)],
  coltitle = "Layout:",
  col_names = NULL,
  title = NULL,
  subtitle = NULL,
  yield_label = FALSE,
  yield_label_pos = 100,
  yield_unit = "Nml/gVS",
  interaction = FALSE
)
```

Arguments

x	a BGF
type	a character indicating which plots should be produced. Possible values are 'product', 'netProduct', 'production', 'relProduction','yield_col' or 'yield_box,

	which yield in a product line plot, a net product line plot, a production line plot, a relative production line plot, a yield col plot or a yield box plot, respectively. It is possible to use any combination of valid types (e.g. 'type = c("product","production","yield_box")'). The default selection is 'all', which will cause the function to produce all plots.
`...`	further arguments passed to `plot_curve`, `plot_netProduct_by_Layout`, `colplot_yield` or `boxplot_yield`
`what`	the name of a (`numeric`) column in the `BioGasData`-layer of a BGF. Must be specified without quotes
`color`	the name another column in the `BioGasData`-layer holding a grouping variable
`col`	a vector with colors to be used to differentiate between levels of the grouping variable. In the default setting, 24 colors are provided
`coltitle`	the title of the plot legend. default ='Reactor:'
`col_names`	default is `NULL`. If specified, a `character` with the same length as levels in the 'color' argument is expected. It can be used change the default labels in the legend
`title`	default is `NULL`. If specified, a title for the plot can be created
`subtitle`	default is `NULL`. If specified, a subtitle for the plot can be created
`xlab`	can be used to change the default title of the x-axis. Default is waiver
`ylab`	can be used to change the default title of the y-axis. Default is waiver
`keep_excluded`	`logic`, default is `TRUE`. Should fermentations that are marked as 'Excluded' in the `metaData`-layer be shown in the plot?
`interaction`	`logic`, default is `FALSE`. If `TRUE` the plot will be created using `plotly` as plot engine instead of `ggplot2`
`hide`	default is `NULL`. Can be a `character` specifying reactor layouts to hide from the plot
`Excluded`	`logic`; default is `FALSE`. Should fermentations that are marked as 'Excluded' in the `metaData`-layer appear in the plot?
`yield_label`	`logic`; default is `FAlSE`. Should the mean yield be shown as a label on the plot?
`yield_label_pos`	a `numeric` specifying the position of the yield label in the plot
`yield_unit`	a `character` specifying the unit of the yield. The default is 'Nml/gVS', which is read 'norm milliliter per gram of volatile solutes'
`timep`	a `numeric` specifying the time point at which to calculate the yield for the individual reactor layouts. Alternatively, using the `character` expressions 'all' will draw the box over all values in the 'yield' and 'max' will draw the box using the final 'yield' value

Details

The main function for advanced data visualization purposes is `bgf_plot`. It is a wrapper that successively calls `plot_product_curve`, `plot_production_curve`, `plot_rel_production_curve`, `plot_netProduct_by_Layout`, `colplot_yield` and `boxplot_yield`. It stores the output of each function in a `list` which is returned

The second function, that allows advanced data visualization is plot_curve. It can be used to draw a line plot from data in the BioGasData-layer of a BGF. In this plot, the 'time' column representing the fermentation/ observation time will be on the x-axis. The y-axis can be any other column of the BioGasData-layer and is specified via the 'what' argument. Furthermore, it is required to provide a grouping variable in the 'color' argument. If that grouping variable has more than 24 levels, the 'col' argument must be changed as in the default setting only 24 colors are available for plotting.

There exist several wrapper functions, that internally call plot_curve exist. These are:

- plot_product_curve: Produces a 'product' line plot. Argument 'what' set to 'product' and argument 'color' set to 'reactor'. Can be called via bgf_plot when 'type = "product"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'product_curve'.

- plot_production_curve: Produces a 'production' line plot. Argument 'what' set to 'production' and argument 'color' set to 'reactor'. Can be called via bgf_plot when 'type = "production"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'production_curve'.

- plot_rel_production_curve: Produces an inverted 'rel_production' line plot. Argument 'what' set to 'c(100-rel_production)' and argument 'color' set to 'reactor'. Can be called via bgf_plot when 'type = "relProduction"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'rel_production_curve'

The function plot_netProduct_by_Layout plots mean net gas curves per layout. It will extract 'net_product' values from the BioGasData-layer and calculates a mean for each 'Layout' specified in the metaData-layer at each 'time' before drawing the line plot.

Can be called via bgf_plot when 'type = "netProduct"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'net_product_curve'.

The function colplot_yield produces a col plot with reactor layouts on the x-axis and mean 'yield' values per layout on the y-axis. It can be used if a yield summary was transferred to the metaData-layer using summarize_yield. It will generate a column-plot with reactor layouts on the x-axis and mean yield values per layout on the y-axis. With the default settings, fermentations that are marked as 'Excluded' in the metaData-layer will not be included in the plot. The standard deviation around the mean value will be calculated and drawn as an errorbar if possible.

Can be called via bgf_plot when 'type = "yield_col"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'yield_col'.

The function boxplot_yield produces a box plot with reactor layouts on the x-axis and and 'yield' values per layout on the y-axis. The argument 'timep' can be used to specify how many observations will be used to draw the boxes. In the default setting, 'all', all available observations for a layout will be used. Alternatively, to select the final observations 'timep = "max"' can be set. Furthermore any numeric that matches a value in the 'time' column of the BioGasData-layer can be passed to 'timep' to select the specific 'yield' value at that time.

Can be called via bgf_plot when 'type = "yield_box"' or 'type = "all"'. In the later case, the plot is stored in a list with the label 'yield_box'.

Value

either a single ggplot2 object or a list of ggplot2 objects

Examples

```
# create an example BGF
myBGF <- from_AMPTSV2_report(
        ReactorLayout = c("2*Meso","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
                          "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Meso",
        name = "myBGF",
        ProcessTemp = 40,
        InocToSubRatio = 2,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))


# generate all plots for the BGF
Plots <- bgf_plot(myBGF)

# print 'product' curve
if(interactive()) Plots[["product_curve"]]

# print 'net product' curve
if(interactive()) Plots[["net_product_curve"]]

# print 'production' curve
if(interactive()) Plots[["production_curve"]]

# print 'relative production' curve
if(interactive()) Plots[["rel_production_curve"]]

# print 'yield' colplot
if(interactive()) Plots[["yield_col"]]

# print 'yield' boxplot
if(interactive()) Plots[["yield_box"]]


# in the following examples the same plots are produced by plot_curve and the respective wrappers
# plot 'product' curve
plot_curve(
        x = myBGF,
        what = product,
        color = reactor,
        ylab = "raw exaust gas volume",
        xlab = paste0("observation time [",myBGF$ExpParam$timeScale,"]"))
if(interactive()) plot_product_curve(myBGF)
if(interactive()) bgf_plot(myBGF,type="product")

# plot 'production' curve
plot_curve(
        x = myBGF,
        what = production,
        color = reactor,
        ylab = "raw exaust gas flow",
        xlab = paste0("observation time [",myBGF$ExpParam$timeScale,"s]"))
```

```
if(interactive()) plot_production_curve(myBGF)
if(interactive()) bgf_plot(myBGF,type="production")

# plot 'rel_production' curve(s)
plot_curve(
        x = myBGF,
        what = rel_production,
        color = reactor,
        ylab = "remaining gas production [%]",
        xlab = paste0("observation time [",myBGF$ExpParam$timeScale,"s]"))
if(interactive()) plot_rel_production_curve(myBGF)
# Note: this curve is inverted so it starts at 100 %

if(interactive()) bgf_plot(myBGF,type="relProduction")

# plot mean net product by reactor layout
plot_netProduct_by_Layout(myBGF)

# create a col plot of the yield
colplot_yield(myBGF)
if(interactive()) bgf_plot(myBGF,type="yield_col")

# create a box plot of the 'yield'
boxplot_yield(myBGF)
if(interactive()) bgf_plot(myBGF,type="yield_box")

# create a box plot of the final 'yield'
boxplot_yield(myBGF,timep="max")
if(interactive()) bgf_plot(myBGF,type="yield_box",timep="max")
```

`calc_FR_time` *Calculate Fermentation Time*

Description

Calculates the fermentation time for a minimal biogas fermentation data set

Usage

```
calc_FR_time(standardReport, time_col, ...)
```

Arguments

`standardReport`	a data.frame with at least one column representing a series of time stamps (%y-%m-%d %H:%M:%S)
`time_col`	numeric. indicates which column of standard report is used to calculate the fermentation time
`...`	further arguments passed to `difftime`

Value

A data.frame with an additional `$time` column

Examples

```
# import example biogas fermentation data
stRep<-import_standard_record(
      ipath = base::system.file("extdata","Fermentation_B.tsv",package = "bgfanalyzer"),
      header=TRUE,
      dec=".",
      sep="\t")

# calculating the fermentation time adds a new column ($time) to the input data.frame
stRep_FRT<-calc_FR_time(stRep,1,units="hours")
```

`calc_inoc_matrix_from_metaData`

Calculate an inoculation matrix

Description

The function calculates an inoculation matrix based on the choosen inoculum-to-substrate ratio (`InocToSubRatio`) of a BGF. To this end, the reactor volume and the concentrations of organics in inoculum and substrate should be known. Most commonly, this concentration is provided as total solutes, organic total solutes or chemical oxygen demand of inoculum and substrate.

Usage

```
calc_inoc_matrix_from_metaData(
  x,
  col,
  reactor = 400,
  ISRatio = NULL,
  VSInoc = NULL,
  subset = NULL,
  digits = 2
)
```

Arguments

`x`	a BGF
`col`	either a `character` or an `integer` specifying which column of the `metaData` layer provides information about the organics concentration of the fermentations.
`reactor`	a `numeric` providing the total filling volume or mass of the liquid phase of the biogas reactors; default = 400

ISRatio	default = NULL; if specified a numeric is expected, providing the inoculum to substrate ratio of a biogas fermentation. If left to the default, a value is extracted from the ExpParam layer of the BGF
VSInoc	NOT WORKING default = NULL; can be specified if the BGF does not contain any 'Blanks' (get_blanks returns 'R'). If specified, either a integer or character is expected that refers to the row in the metaData layer from which to take the organics concentration of the inoculum. If left the default, the same organics concentration is assumed for inoculum and substrate
subset	default = NULL; if specified, a character vector is expected, declaring a subset of fermentation by their respective row name within the metaData layer of the BGF
digits	argument passed to round; default = 2

Details

The function calc_inoc_matrix_from_metaData is meant to build a bridge between data analysis in R and experimental work in the lab. Furthermore, this function should support the design of real world biogas batch-fermentations, so it is usually used on a BGF before the data of the BioGasData layer has been generated.

The BGF this function is used on should have the concentration of organics of inoculum and substrate in its metaData layer. In a biogas fermentation, the term 'inoculum' refers to the source of biogas producing organisms, while the term 'substrate' refers to the source material these organisms produce the biogas from.

To achieve high comparability in between experiments, fermentations should be started based on the same inoculum-to-substrate ratio. Furthermore, if the concentration of organics in inoculum and substrate are known, the inoculum-to-substrate ratio can be used to calculate the amount of biogas produced from the inoculum and substrate fraction of a biogas reactor.

This allows the calculation of substrate specific biogas potentials.

Value

a data.frame; the inoculation matrix to set up a biogas fermentation (series)

Examples

```
# create an example BGF
myBGF<-BGF(
        ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
        BlankLabel = "Blank",
        name = "myBGF",
        ProcessTemp = 52,
        InocToSubRatio = 2,
        MeasurementType = "manuel")

# add organic total solutes measurement for each biogas fermentation to 'metaData' layer
myBGF<-add_metaData(
        x = myBGF,
      what = c(3.63,3.63,98,1.65,1.65,1.65,1.76,1.76,1.76,1.57,1.57,1.57,1.68,1.68,1.68),
```

```
        lab = "oTS")

# calculate ionoculation matrix for all or a subset of fermentations
InocMatrix<-calc_inoc_matrix_from_metaData(myBGF,"oTS")
InocMatrix_subset<-calc_inoc_matrix_from_metaData(myBGF,"oTS",subset = c("R5","R12","R9"))

# create a second example BGF without a 'Blank'
myBGF2<-BGF(LETTERS[1:5],"Blank","myBGF2",80,.1,"manuel")

# add organic total solutes to 'metaData' layer
myBGF2<-add_metaData(myBGF2,what = c(3.63,2.9,3.1,4.19,1.53),lab = "oTS")
InocMatrix_2<-calc_inoc_matrix_from_metaData(myBGF2,4,2000)
InocMatrix_2_fixed<-calc_inoc_matrix_from_metaData(myBGF2,4,2000,VSInoc=5)
```

`calc_yield` *Data transformation functions*

Description

A set of functions that allow to calculate and summarize biogas yield and production on the one hand, or allow to remove certain data points of a fermentation based on the fermentation/ observation time.

Usage

```
calc_yield(x, pos = 6, feedback = FALSE)

summarize_yield(x, timep = "auto", feedback = FALSE)

relative_production(x, feedback = FALSE)

calculate_flow_from_volume(x)
```

Arguments

x	a BGF
pos	a integer or character referencing the column with reference masses in the metaData layer
feedback	logic; if TRUE, the function will print a feedback to the console
timep	a numeric. The fermentation/ observation time at which to calculate the yield. Can be the character string 'auto' instead (default). In this case the final time is automatically chosen

Details

The function `calc_yield` allows to calculates a biogas yield (volume per mass) based on the 'netGas' column of the `BioGasData` layer of a `BGF`. In addition, a reference mass for each fermentation must be present in the `metaData` layer of the respective `BGF`. If a fermentation is classified as 'Blank', this mass will not be used.

The function `summarize_yield` summarizes the 'production' and 'yield' column of a `BioGasData` layer at a chosen time. The summary will be transferred to the `metaData` layer of the same `BGF`. It contains the information of what was the mean yield, the may production and when did max production occur for each reactor layout. If several fermentations share the same reactor layout standard deviations for the those three parameters are calculated as well.

The function `relative_production` calculates the relative biogas production ((accumulated biogas volume at time)/(final accumulated biogas volume)). It uses the values of the 'production' column in the `BioGasData` layer of a `BGF` and stores its results in the 'rel_production' column.

The function `calculate_flow_from_volume` uses the 'product' column of a `BGFs` `BioGasData` layer and generates values for the 'production' column. In particular, it takes the 'product' value at each time and subtracts the previous 'product' value, specifically for each fermentation. This reflects the amount of product formed in between time_now and time_previous and has the unit 'volume/time'. However, if the 'product' column does not contain volumetric biogas data (e.g. gravimetric biogas data instead) the unit of the production column can be different (e.g. 'mass/time'). The first 'product' value is subtracted by itself.

Value

a `BGF`

Examples

```
# create an example BGF
myBGF <- BGF(
        ReactorLayout = c("2*Meso","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
                          "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Meso",
        name = "myBGF",
        ProcessTemp = 42,
        MeasurementType = "AMPTSV2")

# add data to BGF
myBGF <- add_bmp_measurement(
        x = myBGF,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

# correct data
myBGF <- cols_to_numeric(myBGF)
myBGF <- close_gaps(myBGF)

# calculate netGas
myBGF <- netGas(myBGF)

# print the BGF; the yield was not calculated yet
```

```
myBGF

# calculate yield
myBGF <- calc_yield(myBGF)

# summarize yield
myBGF <- summarize_yield(myBGF)

# print the BGF again; now the yield was calculated
myBGF

# calculate relative production
myBGF <- relative_production(myBGF,TRUE)

# create another example BGF
myBGF2 <- from_standard_record(
        ReactorLayout = "A",
        ProcessTemp = 80,
        InocToSubRatio = .1,
       path = base::system.file("extdata","Fermentation_A.tsv",package = "bgfanalyzer"),
        time_col = 1,
        product_col = 3)

# calculate production
myBGF2 <- calculate_flow_from_volume(myBGF2)
```

---

close_gaps *Close gaps in data*

Description

Two functions exist, that can be used to close gaps in the BioGasData layer of a BGF. Depending on the function, used either the last valid value of a fermentation is carried forward or values in between two measurements can be interpolated.

Usage

```
close_gaps(x, feedback = FALSE)

na_correction(x, which = "all_num", ...)
```

Arguments

x	a BGF
feedback	logic. If TRUE the function will print a feedback to the console
which	either a numeric or a characterreferring to numeric columns in the BioGasData-layer of a BGF. Can be 'all_num' (the default) to automatically select all numeric columns in the BioGasData-layer
...	further arguments passed to bgf_interpolation

Details

The function close_gaps can be used to quickly close gaps at the end of a series of fermentations. It will go through each 'time' in the BioGasData layer of a BGF and check if the corresponding 'product' and 'production' values are NA. If TRUE the respective previous value will be selected and replaces the NA. This action will be done specifically for each 'reactor' level. It is called by from_AMPTSV2_report when creating a BGF from an AMPTS II generated report.

The function na_correction can be used to close gaps in between or at the end of a fermentation. Target data columns can be specified as integer or character in the which argument. Alternatively, which can be 'all_num' (the default) to select all numeric columns in the BioGasData-layer of a BGF. Internally the function calls bgf_interpolation on each column specified via the which argument. Additional arguments passed to na_correction will be forwarded to bgf_interpolation as well.

Value

a BGF

Examples

```
# create an example BGF
myBGF <- BGF(
        ReactorLayout = c("2*Meso","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
        "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Meso",
        name = "myBGF",
        ProcessTemp = 42,
        MeasurementType = "AMPTSV2")

# add data generated by an AMPTS II
myBGF <- add_bmp_measurement(
        x = myBGF,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

# convert data columns
myBGF <- cols_to_numeric(myBGF)

# close gaps in data
myBGF <- close_gaps(myBGF)

# another example BGF
myBGF2 <- from_standard_record(
        ReactorLayout = "A",
        ProcessTemp = 80,
        InocToSubRatio = .1,
       path = base::system.file("extdata","Fermentation_A.tsv",package = "bgfanalyzer"),
        time_col = 1,
        product_col = 3)

# import gas quality measurements
gasq <- import_standard_record(
        ipath = base::system.file(
```

```
            "extdata",
            "gasq_A.tsv",
            package = "bgfanalyzer"),
        mkFRTime = "2025-01-15 17:00:00",
        FRTime_col = 1,
        units = "hours")

# add gas quality data to BGF
myBGF2 <- add_BG_parameter(
        x = myBGF2,
        parameter = gasq,
        reactor= "R1",
        time = 3,
        value = 2,
        name = "H2",
        cut_zero = TRUE,
        interpolate_missing = FALSE)

# ensure data structure integrity (recommended before using na_correction)
myBGF2 <- update_BGF(myBGF2)

# close gaps resulting from merging fermentation data and gas quality data
# in all numerics of the BGF's 'BioGasData'-layer
myBGF2 <- na_correction(x=myBGF2)
```

`correct_RLayout` *Data correction functions*

Description

A set of functions that provide shortcuts for frequently used operations conducted on a `BGF`. For instance, they allow to specify the reactor layout in a formula like manner or to change a the type of any column of a `BGF`'s `metaData` or `BioGasData` layer to `numeric`.

Usage

```
correct_RLayout(lo)

cols_to_numeric(x, vec = c(2:4), layer = "BioGasData")
```

Arguments

`lo`	a `character` specifying a reactor layout (via the 'ReactorLayout' argument of certain `bgfanalyzer` functions)
`x`	a `BGF`
`vec`	either a `character` of column names or an `integer` indicating the positions of columns in the chosen `layer`
`layer`	a `character` specifying a layer of a `BGF`. Can be 'metaData' or 'BioGasData'

Details

The function correct_RLayout allows to write 'ReactorLayout=c("3*Blank","3*Cellulose","3*Treatment A")' instead of 'ReactorLayout=c("Blank","Blank","Blank","Cellulose","Cellulose","Cellulose","Treatment A","Treatment A","Treatment A")' or 'ReactorLayout=c(rep("Blank",3),rep("Cellulose",3),rep("Treatment A",3))'.

The function will recognize the '*' and expects an integer on the left side specifying how often the right side should be repeated.

It is useful when dealing with replicates among the fermentations of a BGF, as it can be used to add redundant information to the 'metaData' layer of a (see examples).

The function cols_to_numeric allows to change the type of any column in the metaData or BioGasData layer of a BGF to numeric. It is possible to apply this change to several columns of the same layer at once.

Value

a character (correct_RLayout)

a BGF (cols_to_numeric)

Examples

```
# define a reactor layout
RL <- c("3*Blank","3*Cellulose","3*Treatment A")

# correct reactor layout
RL_cor <- correct_RLayout(RL)

RL_cor # print corrected layout

# example code
myBGF <- BGF(RL)

# add dry total solutes concentration to 'metaData'
myBGF<-add_metaData(x = myBGF,what = correct_RLayout(c("3*2.9","3*98.7","3*8.4")),lab="TS")

# change the new 'TS' column in 'metaData' layer to numeric
myBGF <- cols_to_numeric(myBGF,"TS","metaData")
```

---

get_layer *Extract information from a BGF*

Description

A set of functions that can be used to extract various information from a BGF.

Usage

```
get_layer(x, layer, feedback = FALSE)

get_whatever(x, layer, what, feedback = FALSE)

get_MeasurementType(x, feedback = FALSE)

get_ReactorLayout(x, feedback = FALSE)

get_BlankLabel(x, feedback = FALSE)

get_Excluded(x, feedback = FALSE)

get_blanks(x, feedback = FALSE)

get_yield_summary(x, Excluded = FALSE, feedback = TRUE)
```

Arguments

x	a BGF
layer	a layer of a BGF; either 'ExpParam', 'metaData' or 'BioGasData'
feedback	logic, if TRUE the function will print a feedback to the R console
what	a character referring either to a list entry of the ExpParam layer, or a column of the metaData or BioGasData layer of a BGF
Excluded	logic if TRUE, excluded reactors are not included in the summary. Default = FALSE

Details

The functions get_MeasurementType, get_ReactorLayout, get_BlankLabel,get_Excluded and get_blanks are wrapper for get_whatever and extract frequently needed information from a BGF.

The function get_whatever internally calls get_layer to extract any layer from a BGF, and subsequently return only a single list entry or a data.frame column.

The function get_yield_summary returns a data.frame if a yield summary was generated for the BGF via summarize_yield.

Value

get_layer returns a either a data.farme or list

get_whatever returns a either a data.farme column or list entry

get_MeasurementType returns a character

get_ReactorLayout returns a factor

get_BlankLabel returns a factor

get_Excluded returns a character

get_blanks returns a character

get_yield_summary returns a data.frame

Examples

```
# create an example BGF
myBGF<-BGF(LETTERS[1:5],"A","myBGF",52,2,"manuel")

# extract 'ExpParam' layer
ExpParam <-get_layer(myBGF,"ExpParam",TRUE)

# extract 'metaData' layer
metaData <-get_layer(myBGF,"metaData",TRUE)

# extract 'BioGasData' layer
BioGasData <-get_layer(myBGF,"BioGasData",TRUE)

# extract 'name'-attribute
get_whatever(myBGF,"ExpParam","name")

# extract 'MeasurmentType'-attribute
get_MeasurementType(myBGF)

# extract the reactor layout
get_ReactorLayout(myBGF)

# extract the Blank label
get_BlankLabel(myBGF)

# Exclude reactor 1 and 5
myBGF$metaData$Excluded[c(1,5)] <- TRUE

# extract the excluded reactors
get_Excluded(myBGF)

# extract names of blank reactors
get_blanks(myBGF)

# create a second example BGF
myBGF2<-from_AMPTSV2_report(
        ReactorLayout = c("2*Blank","Cellulose","3*neg ctrl","3*FR1","3*FR2","3*FR3"),
        BlankLabel = "Blank",
        name = "myBGF2",
        InocToSubRatio = 2,
        ProcessTemp = 52,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))

# extract yield summary
get_yield_summary(myBGF2)
```

`import_standard_record` *Data import functions*

Description

The bgfanalyzer package has three data import functions. Two are called internally by the helper functions from_standard_report or from_AMPTV2_report when creating a new BGF object. The third allows to import a BGF object, that was previously exported from R.

Usage

```
import_standard_record(
  ipath,
  dec = ".",
  sep = "\t",
  header = TRUE,
  mkFRTime = NULL,
  FRTime_col = NULL,
  units = NULL,
  ...
)

read_raw_AMPTSV2_report(path, sub = "\\\"")

import_BGF_object(path)
```

Arguments

ipath, path	a path pointing to an external file
dec	the character used in the file for decimal points.
sep	the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.
header	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.
mkFRTime	NULL by default. Can be a character string representing a start date (%y-%m-%d %H:%M:%S) for the calculation of the fermentation time
FRTime_col	NULL by default. Can be a character string with the name, or an integer representing the position of the date column after the input file was read via read.table.
units	character string. Units in which the results are desired. Can be abbreviated.
...	Further arguments to be passed to read.table.
sub	a character, which will be eliminated from the read in text connection

Details

Two of the three data import functions, import_standard_record and read_raw_AMPTSV2_report are unlikely to be directly called by a package user. Instead, they are called by from_standard_record or from_AMPTSV2_report, respectively, when importing an external data file.

The first import function `import_standard_record` is more than a wrapper for read.table with the arguments dec=".", sep="\t" and header=TRUE pre-set. It furthermore allows to calculate a fermentation time directly when importing the data. To this end, `mkFRTime` must be a `character` string representing a date in the format %y-%m-%d %H:%M:%S, `FRTime_col` an `integer` specifying the position of the time stamp within the data, and `units` must be a `character` specifying the desired units of the calculated fermentation time.

In case of `import_standard_record` a `data.frame` is returned.

The function `read_raw_AMPTSV2_report` calls readLines and expects a relative `path` to a `report_yyyy-mm-dd_HHMM.csv`-file generated by the AMPTS II web interface (Login > Download report > Generate report > Download generated report as raw text file (CSV)). **The function expects the original file generated by the AMPTS II**, NOT a .CSV version previously opened and saved by other software as this will replace the original AMPTS II-generated formatting the downstream function is build on. The argument `sub` is used in a call to gsub, which is needed to eliminate an artifact character introduced by calling `readLines`. Upon artifact elimination the raw data is converted into a `list` of three:

ExpPara: A `list`. Information concerning the all reactors part of the AMPTS II experiment. Serves as template for `ExpParam` of a `BGF`

ExpSetup: A `data.frame`. Information on individual reactors being part of the AMPTS II experiment. All information collected here will be added to metaData of a BGF

ExpData: A `data.frame`. Biogas volumes and flow data of the AMPTS II experiment. This data will be moved to `BioGasData` of a `BGF`

This list is returned by `read_raw_AMPTSV2_report`.

The third data import function, `import_BGF_object` allows to import a `BGF` from either an .RDS-file or an .csv-file as produced by save_BGF. A detailed format description of the .csv-file this function can read can be found elsewhere. The function expects a `path` to a file as a single argument. It will check the ending of the file path. If it's '.csv', a `BGF` is rebuild based on the read in data. Else the function serves as a wrapper to `readRDS`. Consequently, a `BGF` is returned.

Value

Either a data.frame, a list or a BGF

Examples

```
# import biogas fermentation from a .tsv file
stRep <- import_standard_record(
      ipath = base::system.file("extdata","Fermentation_B.tsv",package ="bgfanalyzer"),
      header=TRUE,
      dec=".",
      sep="\t")

# calculate a fermentation time while importing the data
stRep_frt <- import_standard_record(
       ipath = base::system.file("extdata","Fermentation_B.tsv",package ="bgfanalyzer"),
        header=TRUE,
        dec=".",
        sep="\t",
        mkFRTime = "2026-01-29 23:00:00",
        FRTime_col = 2,
```

```
        units = "hours")


# create a list that can be used as a template to build a BGF
RawReport <- read_raw_AMPTSV2_report(
        path = base::system.file("extdata","AMPTSV2.csv",package ="bgfanalyzer"))



# import a BGF from a '.csv'-file
BGF_csv <- import_BGF_object(
    path = base::system.file("extdata","importable_BGF_object.csv",package = "bgfanalyzer"))

# import a BGF from a '.RDS'-file
BGF_rds <- import_BGF_object(
    path = base::system.file("extdata","importable_BGF_object.RDS",package = "bgfanalyzer"))
```

`LabscaleBiogas` *A data set resulting from a lab scale biogas fermentation*

Description

The data set contains a `BGF` that was build from an external file.

Usage

```
LabscaleBiogas
```

Format

An object of class `BGF` of length 3.

`LabscaleBiogasLayout` *The reactor layout of the lab scale biogas example*

Description

The data set contains a reactor layout of a lab scale biogas fermentation conducted with the AMPTS II system (BPC, Lund, Sweden).

Usage

```
LabscaleBiogasLayout
```

Format

An object of class `character` of length 15.

netGas *Calculate the net product gas*

Description

Two functions that allow to calculate the net amount of biogas produced by fermentations in a BGF. Both allow to subtract the amount of gas produced by blanks from fermentations, but differ in the way the netGas is calculated and also in the experimental data they need as input.

Usage

```
netGas(x, purity = 1, substract_blank = TRUE, pos = 7, feedback = FALSE)

netGasGC(
  x,
  purity,
  percent = TRUE,
  substract_blank = TRUE,
  pos = 7,
  na_replace = 0,
  feedback = FALSE
)
```

Arguments

x	a BGF
purity	either a numeric indicating the assumed purity (0 - 1) of the product gas for netGas, or a character or an integer referencing the column providing gas quality measurements in the BioGasData layer of a BGF.
substract_blank	logic; default is TRUE. Should the mean product volume produced by blanks be subtracted from fermentations? Fermentations classified as Blanks must exist in the BGF (BGF$metaData$Blank==TRUE).
pos	an integer indicating where the mass amounts of blanks in each fermentation are stored (see *Note*)
feedback	logic, default is FALSE; If TRUE, the function will print a feedback to the console
percent	logic; default is TRUE. Are the values in the column referenced by the 'purity' argument in percent or in an interval from 0 to 1?
na_replace	a numeric. The default assumed concentration of target gas at the beginning of the fermentation.

Details

The function netGas uses the 'product' column of the BioGasData layer of a BGF to calculate the 'net_product' and can be used if no gas quality measurements are available. When called, it uses

the same assumed purity of the product gas for blanks and fermentations. The expected use case is the analysis of AMPTS II generated data, where a methane concentration of 100 vol.% can be assumed due to CO2 absorption units of that system.

The function `netGasGC` uses the 'production' column instead of the 'product' column to calculate the 'net_product' column of the `BioGasData` layer of a `BGF`. It furthermore needs a column providing gas quality measurements added to the `BioGasData` layer.

Value

a `BGF`

Note

To be able of subtracting the amount of gas produced by blank fermentations from the other fermentations, it is essential to provide the information of how much blank was used to inoculate each fermentation. If this information is not known, it can be deduced from the inoculum to substrate ratio and the working load (mass or volume) of the fermentation.

Examples

```
# create an example BGF
myBGF <- BGF(
        ReactorLayout = c("2*Blank","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
                          "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Blank",
        name = "myBGF",
        ProcessTemp = 42,
        MeasurementType = "AMPTSV2")

# add data generated by an AMPTS II
myBGF <- add_bmp_measurement(
        x = myBGF,
        path = base::system.file("extdata","AMPTSV2.csv",package ="bgfanalyzer"))

# convert data columns
myBGF <- cols_to_numeric(myBGF)

# close gaps in data
myBGF <- close_gaps(myBGF)

# calculate netGas
myBGF <- netGas(myBGF)

# create a second example BGF
myBGF2 <- from_standard_record(
        ReactorLayout = "A",
        ProcessTemp = 80,
        InocToSubRatio = .1,
        path = base::system.file("extdata","Fermentation_A.tsv",package ="bgfanalyzer"),
        time_col = 1,
        product_col = 3)
```

```
# import gas quality data
gasq <- import_standard_record(
      ipath = base::system.file(
              "extdata",
              "gasq_A.tsv",
              package ="bgfanalyzer"),
      mkFRTime = "2025-01-15 17:00:00",
      FRTime_col = 1,
      units = "hours")

# add gas quality data to
myBGF2 <- add_BG_parameter(myBGF2,gasq,"R1",3,2,name = "H2",cut_zero = TRUE)

# ensure data integrity
myBGF2 <- update_BGF(myBGF2)

# correct NA's
myBGF2 <- na_correction(myBGF2)

# calculate production
myBGF2 <- calculate_flow_from_volume(myBGF2)

# calculate net gas based on production
myBGF2 <- netGasGC(myBGF2,substract_blank = FALSE,purity = "H2")
```

---

new_BGF *Constructor to set up a Bio Gas Fermentation object*

Description

Builds a BGF object from input parameters.

Usage

```
new_BGF(
  name,
  ProcessTemp,
  InocToSubRatio,
  ReactorLayout,
  BlankLabel,
  MeasurementType = NA
)
```

Arguments

name	A character vector specifying the name of the new BGF object
ProcessTemp	The process temperature of the fermentation(s) to be stored in the BGF object

`InocToSubRatio`	The inoculum to substrate ratio (= inoculation strength) of the fermentation(s) to be stored in the BGF object
`ReactorLayout`	A character vector providing the reactor layout, e.g. the grouping factor used for plotting and yield calculation of fermentation(s) in a BGF object
`BlankLabel`	A character string indicating which group in ReactorLayout is the inoculum used for net gas/ yield calculation
`MeasurementType`	An optional character string indicating the measurement type of the BGF object to be created

Details

This function serves as a base constructor to create a new BGF object. The constructor first checks if all required input variables have the correct data type, then a minimal BGF object is created thereof. All BGFs object share the structure of such a minimal BGF object, e.g. a list with at least three elements:

ExpParam: A list that stores information on the BGF object itself, as well as meta data all fermentations of the object have in common. For example, the input variables name, ProcessTemp, InocToSubRatio and MeasurementType are stored as elements in this list

metaData: A matrix with one row for each fermentation of the BGF object. It is created from the input variables ReactorLayout and BlankLabel. The first column of that matrix provides the ReactorLayout of the corresponding fermentation/ row. The second holds the information whether this layout is the specified BlankLabel or not. The third column enables to classify a fermentation/ row as excluded. Excluded fermentations are not used for yield statistics and can be hidden in plots

BioGasData: A matrix in which experimental data of each fermentation is stored. Upon object creation, all values are set NA. Initially, seven columns are created but further column can be added via dedicated methods or R base syntax.

Value

An R object of class BGF

`plot.BGF` *Plot a Biogas Fermentation*

Description

Default plot method for a BGF.

Usage

```
## S3 method for class 'BGF'
plot(
  x,
  xlab = "Time",
  ylab = "Volume [Nml]",
```

```
  main = paste0("Raw exhaust gas volume plot of ", x$ExpParam$name),
  ...
)
```

Arguments

x	An object of class BGF
xlab, ylab, main	character strings specifying the title of the x-axis, the y-axis or the plot itself, respectively.
...	arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

type what type of plot should be drawn. Possible types are

- "p" for **p**oints,
- "l" for **l**ines,
- "b" for **b**oth,
- "c" for the lines part alone of "b",
- "o" for both '**o**verplotted',
- "h" for '**h**istogram' like (or 'high-density') vertical lines,
- "s" for stair **s**teps,
- "S" for other **s**teps, see 'Details' below,
- "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility. Note that some methods, e.g. plot.factor, do not accept this.

main an overall title for the plot: see title.

sub a subtitle for the plot: see title.

xlab a title for the x axis: see title.

ylab a title for the y axis: see title.

asp the y/x aspect ratio, see plot.window.

Details

Creates a raw exhaust gas volume plot of the fermentations stored in a BGF object.

Value

A plot of exhaust gas volumes

print.BGF *Print a Biogas Fermentation*

Description

Prints a BGF object to the R console

Usage

```
## S3 method for class 'BGF'
print(x, ...)
```

Arguments

x	An object of class BGF
...	further arguments passed to or from other methods.

Details

Default print method for a BGF.

Value

print's details of a BGF object to the console

save_BGF *Save a* BGF *to an external file*

Description

Saves a BGF to either a '.csv'- or a '.RDS'-file. It exists one end user function, which internally calls three sub functions, if a '.csv'-file is produced. Otherwise, it acts as a wrapper for saveRDS.

Usage

```
save_BGF(
  x,
  opath = NULL,
  mkdir = FALSE,
  feedback = FALSE,
  append = FALSE,
  format = "csv",
  ...
)
```

```
save_ExpParam(
  x,
  opath = NULL,
  mkdir = FALSE,
  feedback = FALSE,
  append = FALSE,
  ...
)

save_metaData(
  x,
  opath = NULL,
  mkdir = FALSE,
  feedback = FALSE,
  append = FALSE,
  ...
)

save_BioGasData(
  x,
  opath = NULL,
  mkdir = FALSE,
  feedback = FALSE,
  append = FALSE,
  ...
)
```

Arguments

x	a BGF that should be exported
opath	a path pointing to a location on the system, where the BGF should be saved
mkdir	logic, if TRUE a new directory is created as specified in opath
feedback	logic, if TRUE the function will print a feedback to the R console
append	logic, if TRUE the function assumes that the path specified in opath already includes a file name with an appropriate ending as specified in format. Otherwise a file name will be generated
format	specifies the output generated by the function. Can be either 'csv' or 'RDS'
...	further arguments that can be passed to dir.create.

Details

The main function used to save a BGF as an external data file is save_BGF. The user can choose if the BGF should be saved as an '.RDS'-file or in a human readable '.csv'-file. A human readable '.csv'-file is generated by a consecutive call to the three sub-functions save_ExpParam, save_metaData and save_BioGasData. Each of these functions is designed to write a single layer of the input BGF to an external '.csv'-file.

Value

Either a '.csv' or '.RDS' file created of a BGF

Examples

```
# create a BGF
myBGF <- BGF(LETTERS[1:5],LETTERS[1],"myBGF",52,2,"manual")


# save it as '.csv' file
save_BGF(myBGF)

# save it as '.RDS' file
save_BGF(myBGF,format="RDS")

# remove files
file.remove("myBGF_BGF_object.csv")
file.remove("myBGF_BGF_object.RDS")

# save only the 'ExpParam' layer
save_ExpParam(myBGF)

# remove file
file.remove("myBGF_ExpParam.csv")

# save only the 'metaData' layer
save_metaData(myBGF)

# remove file
file.remove("myBGF_metaData.csv")

# save only the 'BioGasData' layer
save_BioGasData(myBGF)

# remove file
file.remove("myBGF_BioGasData.csv")
```

---

subset_BGF *Split or merge* BGF*'s*

Description

Functions that allow sub-setting and merging of 'BGF"'s.

Usage

```
subset_BGF(x, reactor = NULL, layout = NULL, name = NULL)

merge_BGF(x, y, mergeParam = FALSE, x_tag = NULL, y_tag = NULL, name = NULL)
```

Arguments

x, y	a BGF
reactor	defaults to NULL. Can be an integer or character specifying the reactors/ fermentations to keep based on the reactor name (= row name of the fermentation in metaData-layer)
layout	defaults to NULL.Can be an integer or character specifying the reactors/ fermentations to keep based on the reactor layout (= respective value in the 'Layout' column of the metaData-layer)
name	defaults to NULL. Can be a character providing a new name for the BGF
mergeParam	logic; defaults to FALSE. Should the ExpParam-layer of the BGF's be also merged?
x_tag, y_tag	defaults to NULL. Can be a character which is used as a tag when merging ExpParam-layers

Details

The function subset_BGF can be used to subset a BGF. The user can either use reactor/ fermentation names (= row names of metaData-layer, values in column 'reactor' of BioGasData-layer), or reactor layouts (= values in 'Layout' column of metaData-layer) as a basis for subsetting. Furthermore, it is possible to give a new name to the resulting subset.

The function merge_BGF can be used to merge two BGF's. On the way, it will generate new row names for the metaData-layer and adjust the values in the 'reactor' column of the BioGasData-layer appropriately.

In general, it is advised to only merge BGF's that have the same values in their ExpParam-layer, as with the default setting only the ExpParam-layer of the first BGF passed to merge_BGF is preserved. It is possible to change the default behavior with the 'mergeParam' argument. In that case, new names for entries in the merged ExpParam-layer will be generated if a entry is either present in only one ÈxpPram-layer of the BGF's or if the values for the same entry in both BGF's differs. The new names are prolonged by a specific tag (controlled via arguments 'x_tag' and 'y_tag'), so that it is clear where that entry originates from.

Furthermore, these tags will be added to the metaData- and BioGasData-layer.

Value

a BGF

Examples

```
# create an example BGF
myBGF <- from_AMPTSV2_report(
        ReactorLayout = c("2*Blank","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
                          "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Blank",
        name = "myBGF",
        InocToSubRatio=2,
        ProcessTemp = 52,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"))
myBGF
```

```
# subset using 'reactor' argument
myBGF_subset1 <- subset_BGF(myBGF,reactor=c(4:9),name="subset 1")
myBGF_subset1

# subset using 'layout' argument
myBGF_subset2 <- subset_BGF(myBGF,layout=c("S2 ctrl","S2 4d","S2 6d"),name="subset 2")
myBGF_subset2

# merge the BGF's
mergedBGF <- merge_BGF(myBGF_subset1,myBGF_subset2,TRUE)
mergedBGF
```

`trim_FR_time` *Re-orientate fermentations of a* `BGF` *by time*

Description

The function allows to remove data points of individual or all fermentations based on the fermentation/ observation time ('time' column in `BiGasData` layer of a `BGF`). The user can either remove data points later than a specified 'time' value, or subtract a specified value from each value in the 'time' column and afterwards optionally remove the resulting times < 0.

Usage

```
trim_FR_time(x, value, mode = "all", left_end = TRUE, cut_left = TRUE)
```

Arguments

`x`	a `BGF`
`value`	a `numeric`; either the time before or after which data points should be removed (or shifted)
`mode`	a `character`. Can be 'all' to apply the trimming to all fermentations of a `BGF`. Otherwise each element in `mode` must refer to a fermentation label (e.g. 'mode = c("R1","R3","R4","R15")' to trim fermentation R1, R3, R4 and R15)
`left_end`	`logic`; default `TRUE`. Should the fermentation/ observation time be shifted towards the left end? If `FALSE` data points later than the selected 'value' will be removed
`cut_left`	`logic`; default `TRUE`. Should data points with a negative fermentation/ observation times be removed?

Value

a `BGF`

Examples

```
# create example BGF
myBGF <- from_AMPTSV2_report(
        ReactorLayout = c("2*Meso","Cellulose","2*S1 ctrl","2*S1 7d","2*S1 4d",
                          "2*S2 ctrl","2*S2 4d","2*S2 6d"),
        BlankLabel = "Meso",
        name = "myBGF",
        ProcessTemp = 42,
        path = base::system.file("extdata","AMPTSV2.csv",package = "bgfanalyzer"),
        InocToSubRatio = 2,
        feedback = TRUE)

# BioGasData has 735 rows
nrow(myBGF$BioGasData)

# remove all data before day 30
myBGF <- trim_FR_time(myBGF,value= 30)

# BioGasData has now only 285 rows
nrow(myBGF$BioGasData)

# remove all data after day 15
myBGF <- trim_FR_time(myBGF,value= 15,left_end=FALSE,cut_left=FALSE)

# BioGasData has now only 225 rows
nrow(myBGF$BioGasData)
```

`validate_BGF` *Validation of data integrity*

Description

A set of two functions, that allow to validate and re-establish data integrity of a `BGF`.

Usage

```
validate_BGF(x)

update_BGF(x, delete_missing_BGD = FALSE)
```

Arguments

`x`	a `BGF`
`delete_missing_BGD`	`logic`; defaults to `FALSE`. If `TRUE` fermentations/ rows in the `metaData`-layer with no data in the `BioGasData`-layer will be removed from the `metaData`-layer.

Details

The function validate_BGF can be used to check whether an object is a BGF or not. It will raise an error if an object does not fulfill the criteria of a BGF.

The function update_BGF checks and updates several parameters, so that the internal logic of a BGF is consistent. First of all, it checks if all all fermentations in the metaData-layer occur in the BioGasData-layer of a BGF and vice versa. It than adds a row to the metaData-layer, if it detects any fermentation in the BioGasData-layer, that is not yet represented in the metaDatalayer.

In the opposite case, a fermentation has an entry in the metaData but not in the BioGasData-layer, the user can decide whether this fermentation should be removed from the metaData-layer or not.

The function will also remove rows that have NAin the 'reactor' column from the BioGasData-layer, sort observations in that layer by 'reactor' and 'time' columns and finally generates new valid row names representing the new row number of an observation in the reordered BioGasData-layer.

Value

a BGF

Examples

```
# create an example BGF
myBGF <- new_BGF("myBGF",52,2,LETTERS[1:15],"A","manuel")

# check if 'myBGF' fulfills the criteria of being a BGF
validate_BGF(myBGF)

# print myBGF
myBGF

# updating the BGF will remove observations from `BioGasData`-layer
myBGF <- update_BGF(myBGF)

# print myBGF again
myBGF
```

Index