Package 'ConFluxPro'

July 21, 2025

Type Package

Title Soil Gas Analysis and Flux Modeling

Version 1.3.1

Description Model soil gas fluxes with the Flux-Gradient Method. It includes functions for data handling, a forward and an inverse model for flux modeling and methods for calibration and uncertainty estimation. For more details see Gartiser et al. (2025a) <doi:10.21105/joss.08094> and Gartiser et al. (2025b) <doi:10.1111/ejss.70126>.

Imports stats (>= 4.1.0), lubridate (>= 1.9.2), tibble (>= 3.0.1), dplyr (>= 1.1.1), magrittr (>= 2.0.3), splines (>= 4.1.0), tidyr (>= 1.3.0), rlang (>= 1.1.0), furrr (>= 0.2.3), progressr (>= 0.10.0), ggplot2 (>= 3.4.2), scales (>= 1.2.1), lifecycle

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Suggests rmarkdown, testthat (>= 3.0.0), knitr, DiagrammeR, purrr

VignetteBuilder knitr, rmarkdown

URL https://confluxpro.valentingartiser.de/, https://github.com/valentingar/ConFluxPro, https://valentingar.github.io/ConFluxPro/

BugReports https://github.com/valentingar/ConFluxPro/issues

NeedsCompilation no

Author Valentin Gartiser [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-5320-374X>), Martin Maier [ctb] (ORCID: <https://orcid.org/0000-0002-7959-0108>), Verena Lang [ctb]

Maintainer Valentin Gartiser <code@valentingartiser.de>

Repository CRAN

Date/Publication 2025-07-10 15:20:21 UTC

Contents

`alternate` *Run parameter variation*

Description

Alternate `cfp_pfres()` / `cfp_fgres()` models for sensitivity analysis and more.

Usage

```
alternate(
  x,
  f,
  run_map,
  return_raw = TRUE,
  error_funs = NULL,
  error_args = NULL
)

alternate_model(run_map, x, f)
```

Arguments

`x`	A cfp_pfres or cfp_fgres model result.
`f`	A function taking in a soilphys object and recalculates the relevant columns. See `complete_soilphys()`.
`run_map`	A data.frame created by `run_map()` with the necessary information how the data is to be changed with each distinct `run_id`.
`return_raw`	Should the models be returned as is, or after applying any `error_funs`. Default is `TRUE` - exporting the models.
`error_funs`	A list of functions to be applied after flux calculation if `return_raw == FALSE`. This can be used to output not the models but quality parameters instead. Output must contain the column RMSE.
`error_args`	A list of additional function arguments to be passed to any of the `error_funs`. Must match the length of `error_funs`

Details

`alternate_model()` is used internally to change and rerun one model, but can also be used to update a model with a given unique run_map, e.g. by filtering the best run_id from the original `run_map`.

Value

A `list` of type `cfp_altres()`, each entry an updated model.

Examples

```
PROFLUX <- ConFluxPro::base_dat |>
  filter(site == "site_a") |> # use only 'site_a' for example
  pro_flux()

# Create a cfp_run_map where TPS is changed between 90 % and 110 %
# of the original value for 2 runs.
my_run_map <-
cfp_run_map(
  PROFLUX,
  list("TPS" = c(0.9, 1.1)),
  "factor",
  n_runs = 2)

# run the new models by providing a function `f`
# that updates the soilphys data.frame.
alternate(
  x = PROFLUX,
  f = \(x) complete_soilphys(x, "a+AFPS^b", quiet = TRUE),
  run_map = my_run_map)
```

---

base_dat *Example cfp_dat object*

Description

An example `cfp_dat()` object that combines all other example data.

Usage

```
base_dat
```

Format

A `cfp_dat()` object as a list with

profiles The profiles of the data.

gasdata The gasdata object.

soilphys The soilphys object.

layers_map The layers_map object.

bootstrap_error *Estimate model uncertainty*

Description

Estimate model uncertainty

Usage

```
bootstrap_error(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

## S3 method for class 'cfp_altres'
bootstrap_error(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

## S3 method for class 'cfp_dat'
bootstrap_error(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

## S3 method for class 'cfp_fgmod'
bootstrap_error(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)
```

```
## S3 method for class 'cfp_pfmod'
bootstrap_error(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

make_bootstrap_model(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

## S3 method for class 'cfp_pfmod'
make_bootstrap_model(
  x,
  n_samples = 50,
  sd_x_ppm = NULL,
  n_replicates = NULL,
  sample_from = "gasdata",
  rep_cols = NULL
)

calculate_bootstrap_error(x, y)

## S3 method for class 'cfp_pfmod'
calculate_bootstrap_error(x, y)
```

Arguments

x	A `cfp_pfres` model result from a call to `pro_flux()`.
n_samples	The number of samples to take in the bootstrapping.
sd_x_ppm	An optional estimate of the standard deviation of x_ppm. Can be either • a single value applied equally to all • a data.frame with a column of the same name that maps a value to every observation depth. See `depth_structure()` for an easy way to create it. • be provided as its own column already present in `x$gasdata`.
n_replicates	The number of replicates to be generated if sd_x_ppm is set.
sample_from	From which dataset to sample the bootstrapping dataset. Can either be `'gasdata'` or `'soilphys'` or `'both'`.

rep_cols	The id_cols that represent repetitions. If removed, the repetitions in soilphys of each profile must match in their structure exactly.
y	The result of the bootstrap model.

Value

x with added columns DELTA_flux and DELTA_prod as an estimate of the error of of the corresponding columns in the same units.

General procedure

bootstrap_error() is mostly a wrapper around two functions that can also be run separately.

In make_bootstrap_model(), for sample_from = "gasdata" the gasdata concentration data is resampled for every depth and profile a total number of n_samples. This is done by randomly sampling the observations at each depth without changing the number of observations but while allowing replacing. If rep_cols are given, these columns are removed from the id_cols and the resulting profiles combined as one.

For sample_from = "soilphys", the soilphys data is combined using the rep_cols as repetitions. Among every remaining profile and depth, one observation across all repetitions is chosen for each of n_samples. sample_from = "both" applies both methods above. Each newly sampled profile is identifiable by the added bootstrap_id column which is also added to id_cols.

After this new model is run again, the bootstap error is calculated in calculate_bootstrap_error(). This is the standard deviation of the production and flux parameters across all bootstrapped model runs and is calculated for each profile and layer of the original model, or for each distinct profile in the new model without rep_cols. These are returned together with the mean values of prod, flux and F0 across all runs in the PROFLUX data.frame and can thereby be extracted by efflux() and production().

Artificial observations in gasdata

If there are not enough observations per depth (e.g.) because there is only one measurement per depth, it is possible to create artificial observations by providing n_replicates and sd_x_ppm. Here, every depth of every profile is first averaged to its mean (redundant if there is only one observation). Then, a random dataset of n_replicates observations is generated that is normally distributed around the mean with a standard deviation (in ppm) of sd_x_ppm. These observations are then resampled as described above. Note that this error should be representative of the sampling error in the field and not the measurement error of the measurement device, which is much lower.

Examples

```
PROFLUX <- pro_flux(ConFluxPro::base_dat)
PROFLUX_BSE <- bootstrap_error(PROFLUX)
efflux(PROFLUX_BSE)

PROFLUX_BSE <- bootstrap_error(PROFLUX, n_replicates = 5, sd_x_ppm = 25)
efflux(PROFLUX_BSE)

make_bootstrap_model(PROFLUX) # internal
```

cfp_altapply *Apply a function over a list of models*

Description

Apply a function to a list of cfp_pfres pr cfp_fgres objects stored in an cfp_altres object. This can be used to summarise alternate() results.

Usage

```
cfp_altapply(X, FUN, ...)
```

Arguments

X	Either a cfp_altres object or a list.
FUN	the function to be applied to each element of X: see 'Details'. In the case of functions like +, %*%, the function name must be backquoted or quoted.
...	optional arguments to FUN.

Value

data.frame with the results of FUN bound together with added column run_id as identifier of the original list elements.

Examples

```
PROFLUX <- ConFluxPro::base_dat |> pro_flux()
model_list <- list('1' = PROFLUX, '2' = PROFLUX)

cfp_altapply(model_list, efflux)
```

cfp_dat *Model input data*

Description

cfp_dat is the essential object class that binds all necessary input data to run a ConFluxPro model. It automatically combines the different datasets and checks them for validity. It may split soilphys layers to correspond with layers_map and gasdata depths.

Usage

```
cfp_dat(gasdata, soilphys, layers_map)

as_cfp_dat(x)

## S3 method for class 'cfp_dat'
as_cfp_dat(x)
```

Arguments

gasdata	A cfp_gasdata object created by running `cfp_gasdata()`.
soilphys	A cfp_soilphys object created by running `cfp_soilphys()`.
layers_map	A cfp_layers_map object created by running `cfp_layers_map`.
x	An object of class cfp_dat

Value

A cfp_dat object with the following parameters:

gasdata The gasdata object with added column "gd_id" that is unique for each profile.

soilphys The soilphys object with added columns "sp_id" that is unique for each profile, "step_id" indicating the position of each step from the bottom up, "height" in m of each layer, "pmap" indicating which layer it belongs to from the bottom up. Potentially, some original steps were split to account for the depths within gasdata or layers_map.

layers_map The layers_map object with added column "group_id" indicating each unique group of the same layer parameterization set by layers_map.

profiles A `data.frame` where each row indicates one unique profile that is characterised by all `id_cols` present in the original input as well as the corresponding "gd_id", "sp_id", and "group_id". Each row has a unique identifier "prof_id".

id_cols A character vector of all columns that identify a profile uniquely.

See Also

Other data formats: `cfp_gasdata()`, `cfp_layered_profile()`, `cfp_layers_map()`, `cfp_profile()`, `cfp_soilphys()`

Examples

```
gasdata <- cfp_gasdata(
  ConFluxPro::gasdata,
  id_cols = c("site", "Date"))
soilphys <- cfp_soilphys(
  ConFluxPro::soilphys,
  id_cols = c("site", "Date"))
layers_map <-
 cfp_layers_map(
   ConFluxPro::layers_map,
   gas = "CO2",
```

```
   lowlim = 0,
   highlim = 1000,
   id_cols = "site")
base_dat <- cfp_dat(gasdata, soilphys, layers_map)

### filter similar to dplyr::fliter
filter(base_dat, site == "site_a")
filter(base_dat, prof_id %in%  1:5)

### coersion from derived objects
PROFLUX <- pro_flux(base_dat)
as_cfp_dat(PROFLUX)
```

`cfp_fgmod` *Model frame for fg_flux*

Description

An S3 class for `fg_flux()` models. The class inherits from cfp_dat and adds any model specific parameters.

Usage

```
cfp_fgmod(
  x,
  gases = unique_gases(x),
  modes = "LL",
  param = c("c_air", "DS"),
  funs = c("arith", "harm")
)
```

Arguments

`x`	A `cfp_dat` object with all the necessary input datasets.
`gases`	(character) A character vector defining the gases for which fluxes shall be calculated.
`modes`	(character) A character vector specifying mode(s) for dcdz calculation. Can be `"LL"`,`"LS"`,`"EF"`.

LL local linear approach: within each layer a linear model is evaluated of concentration over the depth.

LS linear spline approach: A linear spline is fitted over the complete profile with nodes at the layer intersections.

EF exponential fit approach: An exponential function of form y=a+b*x^c is fit of concentration against depth. Using the first derivative of that function the concentration gradient is evaluated for each layer.

DA exponential fit approach: An exponential function of form `y=a+b*(1-exp(-a*x))` is fit of concentration against depth. Using the first derivative of that function the concentration gradient is evaluated for each layer. From Davidson (2006).

`param` (character) A vector containing the the parameters of soilphys, for which means should be calculated, must contain "c_air" and "DS", more parameters may help interpretation.

`funs` (character) A vector defining the type of mean to be used for each parameter in `param`. One of "arith" or "harm".

Value

A `cfp_fgmod` object. This inherits from `cfp_dat()` and adds model specific parameters.

References

DAVIDSON, E. A., SAVAGE, K. E., TRUMBORE, S. E., & BORKEN, W. (2006). Vertical partitioning of CO2 production within a temperate forest soil. In Global Change Biology (Vol. 12, Issue 6, pp. 944–956). Wiley. https://doi.org/10.1111/j.1365-2486.2005.01142.x

See Also

Other model frames: `cfp_altres()`, `cfp_fgres()`, `cfp_pfmod()`, `cfp_pfres()`

Examples

```
cfp_fgmod(ConFluxPro::base_dat)

### coercion from other object types (internal)
fg_flux(ConFluxPro::base_dat) |>
  as_cfp_fgmod()
```

---

cfp_fgres *Model result of fg_flux*

Description

A function to create an object of class cfp_fgres. This is the central result class generated by running `fg_flux()`. Intended for internal use only.

Usage

```
cfp_fgres(x, y)
```

Arguments

`x` A valid `cfp_fgmod` object

`y` The corresponding FLUX `data.frame`.

Value

A `cfp_fgres` object. This inherits from `cfp_fgmod()`.

See Also

Other model frames: `cfp_altres()`, `cfp_fgmod()`, `cfp_pfmod()`, `cfp_pfres()`

Examples

```
FLUX <- fg_flux(ConFluxPro::base_dat)
cfp_fgres(
  cfp_fgmod(ConFluxPro::base_dat),
  FLUX$FLUX
)
```

---

cfp_gasdata *Soil gas concentration data*

Description

Create a cfp_gasdata object. This is a data.frame containing gas concentration data for one or multiple soil profiles. Each soil profile is uniquely identified by columns in the data.frame specified by the `id_cols` attribute.

Usage

```
cfp_gasdata(x, ...)

## S3 method for class 'data.frame'
cfp_gasdata(x, id_cols, ...)

## S3 method for class 'cfp_dat'
cfp_gasdata(x, ...)
```

Arguments

`x`	A `data.frame` with the following columns: **gas** The gas of that observation. **depth (cm)** The depth of the observation. **x_ppm (ppm)** The concentration in ppm. **any of** `id_cols` All id_cols that identify one profile uniquely.
`...`	not used
`id_cols`	Column names in data.frame that uniquely identify each profile.

Value

A cfp_gasdata object.

See Also

Other data formats: cfp_dat(), cfp_layered_profile(), cfp_layers_map(), cfp_profile(), cfp_soilphys()

Examples

```
cfp_gasdata(
  ConFluxPro::gasdata,
  id_cols = c("site", "Date"))
### Also used to extract the gasdata object from cfp_dat
cfp_gasdata(ConFluxPro::base_dat)
```

---

cfp_layered_profile *Object for layered soil profiles*

Description

A subclass of cfp_profile() where each profile consists of layers that are defined by their upper and lower boundary without gaps or duplicates.

Usage

```
cfp_layered_profile(x, id_cols = NULL)
```

Arguments

x	A data.frame with columns upper and lower.
id_cols	Column names in data.frame that uniquely identify each profile.

Details

upper and lower define the upper and lower bounds of each layer in cm. Higher values lay on top of lower values.

Value

A cfp_layered_profile object. This is a [cfp_profile()] that is further subdivided into layers by the columns upper and lower.

See Also

Other data formats: cfp_dat(), cfp_gasdata(), cfp_layers_map(), cfp_profile(), cfp_soilphys()

Examples

```
df <- data.frame(
  site = rep(c("site_a", "site_b"), each = 2),
  upper = c(10, 0, 7, 0),
  lower = c(0, -100, 0, -100),
  variable = 1:4)

cfp_layered_profile(df, id_cols = "site")
```

`cfp_layers_map` *Model layers*

Description

A function to create a cfp_layers_map object that defines the layers of both `fg_flux()` and `pro_flux()` models.

Usage

```
cfp_layers_map(x, ...)

## S3 method for class 'cfp_dat'
cfp_layers_map(x, ...)

## S3 method for class 'data.frame'
cfp_layers_map(
  x,
  id_cols,
  gas = NULL,
  lowlim = NULL,
  highlim = NULL,
  layer_couple = 0,
  ...
)
```

Arguments

`x`	(data.frame) That defines the layers for which the production or flux is modeled. Note that some parameters can also be provided directly to the function call instead (see Details).

- `id_cols` the relevant id_cols (see below)
- `gas`, the gas that is modelled.
- `upper`, `lower` the upper and lower boundaries of each layer
- `lowlim`, `highlim` as the lower and upper limits of the production rate to be modeled in $\mu\, mol\ m^{-3}$
- the parameter `layer_couple`, that indicates how strongly the layer should be linked to the one below it (0 for no coupling)

...	not used
id_cols	Column names in data.frame that uniquely identify each profile.
gas	(character vector) of gas names to be added to x which is then repeated for each gas.
lowlim	(numeric vector) the same length as gas with the lower limit of possible production allowed in pro_flux() models.
highlim	(numeric vector) the same length as gas with the upper limit of possible production allowed in pro_flux() models.
layer_couple	**[Experimental]** (numeric_vector) A vector the same length as gas that indicates how strongly the layer should be linked to the one below it (0 for no coupling, the default).

Value

A cfp_layered_profile() data.frame with the columns described above as well as layer and pmap columns that identify each layer with an integer (ascending from bottom to top).

Add lowlim and highlim for multiple gases

Sometimes it is practical to model different gases with different limits. For example, it is a reasonable assumption that CO2 is not consumed in relevant amounts in most soils, whereas CH4 may be both produced or consumed. Therefore we may want to limit production rates of CO2 to only positive values, whereas allowing for negative CH4 production rates (i.e. consumption) as well.

To make this setup easy, you can provide a gas vector to the function together with highlim and lowlim vectors of the same length. The provided layers_map data.frame will then be replicated for each gas with the respective values of the production limits provided.

See Also

Other data formats: cfp_dat(), cfp_gasdata(), cfp_layered_profile(), cfp_profile(), cfp_soilphys()

Examples

```
cfp_layers_map(
   ConFluxPro::layers_map,
   gas = "CO2",
   lowlim = 0,
   highlim = 1000,
   id_cols = "site")

### add multiple gases at once
cfp_layers_map(
   ConFluxPro::layers_map,
   id_cols = "site",
   gas = c("CO2", "CH4"),
   lowlim = c(0, -1000),
   highlim = c(1000, 1000))

### Extract from an existing cfp_dat
cfp_layers_map(ConFluxPro::base_dat)
```

cfp_parameter *Get parameter descriptions and units*

Description

Function to access parameter descriptions and units used in ConFluxPro

Usage

```
cfp_parameter(x = NULL)
```

Arguments

x	Any object or data.frame to match the parameters to, or a character vector of parameter names.

Value

A data.frame() with the name, description and unit of the parameter

Examples

```
#list parameters within an object
cfp_parameter(soilphys)
cfp_parameter(gasdata)

#list all paramters
cfp_parameter()
```

cfp_pfmod *Model frame for pro_flux*

Description

An S3 class for pro_flux() models. The class inherits from cfp_dat and adds any model specific parameters.

Usage

```
cfp_pfmod(
  x,
  zero_flux = TRUE,
  zero_limits = c(-Inf, Inf),
  DSD0_optim = FALSE,
  evenness_factor = 0,
  known_flux_factor = 0
)
```

Arguments

x	A cfp_dat object with all the necessary input datasets.
zero_flux	(logical) Applies the zero-flux boundary condition? If FALSE, F0 is optimized alongside the production rates.
zero_limits	(numeric vector) a vector of length 2 defining the lower and upper limit of the lowest flux if zero_flux = FALSE.
DSD0_optim	**[Deprecated]**
evenness_factor	**[Experimental]**(numeric) A user defined factor used to penalise strong differences between the optimised production rates. This must be identified by trial-and-error and can help prevent that production rates are simply set to zero basically the lower a production is relative to the the maximum of the absolute of all productions, the higher it is penalised. The evenness_factor then defines the weight of this penalty in the optimisation algorithm prod_optim.
known_flux_factor	**[Deprecated]**

Value

A cfp_pfmod object that inherits from cfp_dat()

See Also

Other model frames: cfp_altres(), cfp_fgmod(), cfp_fgres(), cfp_pfres()

Examples

```
cfp_pfmod(ConFluxPro::base_dat)

### coercion from other object types (internal)
pro_flux(ConFluxPro::base_dat) |>
  as_cfp_pfmod()
```

---

cfp_pfres *Model result of pro_flux*

Description

A function to create an object of class cfp_pfres. This is the central result class generated by running pro_flux(). Intended for internal use only.

Usage

```
cfp_pfres(x, y)
```

Arguments

x	A valid cfp_pfmod object
y	The corresponding PROFULX data.frame.

Value

A cfp_pfres object. This inherits from cfp_pfmod().

See Also

Other model frames: cfp_altres(), cfp_fgmod(), cfp_fgres(), cfp_pfmod()

Examples

```
PROFLUX <- pro_flux(ConFluxPro::base_dat)
cfp_pfres(
  cfp_pfmod(ConFluxPro::base_dat),
  PROFLUX$PROFLUX
)
```

---

cfp_profile *Object for soil profiles*

Description

A central S3 class that defines a data.frame where columns given in id_cols define distinct soil profiles.

Usage

```
cfp_profile(x, id_cols = NULL)
```

Arguments

x	A data.frame
id_cols	Column names in data.frame that uniquely identify each profile.

Value

A cfp_profile object. This is a data.frame with the id_cols attribute.

See Also

Other data formats: cfp_dat(), cfp_gasdata(), cfp_layered_profile(), cfp_layers_map(), cfp_soilphys()

Examples

```
df <- data.frame(
  site = rep(c("site_a", "site_b"), each = 2),
  variable = 1:4)

cfp_profile(df, id_cols = "site")

### multiple id_cols
df <- data.frame(
  site = rep(c("site_a", "site_b"), each = 4),
  replicate = rep(c(1,2), times = 4),
  variable = 1:8)

cfp_profile(df, id_cols = c("site", "replicate"))
```

`cfp_run_map` *Create a run plan for parameter variation*

Description

An S3 class cfp_run_map to be used in alternate(). Either create a new run map from a cfp_pfres or cfp_fgres model or extract an existing run_map from an cfp_altres object.

Usage

```
cfp_run_map(
  x,
  params = list(),
  type = NULL,
  method = NULL,
  n_runs = NULL,
  layers_different = FALSE,
  layers_from = "layers_map",
  layers_altmap = NULL,
  topheight_adjust = FALSE
)
```

Arguments

`x`	Either a cfp_pfres or cfp_fgres model result.
`params`	A named list of numeric vectors. Names indicate column names in soilphys, vectors either distinct values (method permutation) or limits (method random).
`type`	A vector of length param indicating what the values in params represent. One of **abs** Absolute values that are applied as-is. **factor** Factors to be multiplied with the original values.

addition Factors to be added to the original values.

`method` Either 'random', where a random value is chosen within the bounds set in `params` or 'permutation', where every permutation of the values in `params` is added.

`n_runs` Integer value of the number of alterations to be done for method = 'random'.

`layers_different` Should layers from layers_map be changed individually? If `TRUE` this allows for different changes at different depths.

`layers_from` (character) If layers_different is TRUE, from which source should the layers be created? One of:

layers_map (default) Use the layers that are defined in layers_map.

soilphys Use the layers as defined in soilphys

layers_altmap Use the layers as defined in the provided layers_altmap object.

`layers_altmap` An optional layers_map created using layers_map() that defines the layers to be used if layers_different = TRUE.

`topheight_adjust` (logical) If the proposed change in topheight is larger than the highest layer in soilphys, should the limits be automatically adjusted per id_cols individually? Default is FALSE, which leads to an error in that case.

Value

An object of type `cfp_run_map` that can be used within alternate.

Examples

```
PROFLUX <- ConFluxPro::base_dat |> pro_flux()
# Create a cfp_run_map where TPS is changed between 90 % and 110 %
# of the original value for 50 runs.
cfp_run_map(
  PROFLUX,
  list("TPS" = c(0.9, 1.1)),
  "factor",
  n_runs = 50)
```

`cfp_soilphys` *Soil physical parameters data*

Description

Create a cfp_soilphys object. This is a data.frame containing layered data of soil physical properties, at the minimum of the air density `c_air` and diffusion coefficient `DS` for one or multiple soil profiles. Each soil profile is uniquely identified by columns in the data.frame specified by the `id_cols` attribute. Each profile is further subdivided into layers by columns `upper` and `lower` (see cfp_layered_profile).

Usage

```
cfp_soilphys(x, ...)

## S3 method for class 'cfp_dat'
cfp_soilphys(x, ...)

## S3 method for class 'data.frame'
cfp_soilphys(x, id_cols, ...)
```

Arguments

x	A data.frame with (at least) the following columns: **upper (cm)** The upper bound of each step. **lower (cm)** The lower bound of each step. **gas** The gas of that step. **DS** (m^2s^-1) The specific diffusion coefficient of that gas in that step. **c_air** ($molm^-3$) The number density of air in that step. **any of** `id_cols` All id_cols that identify one profile uniquely.
...	Internal, must be empty.
id_cols	Column names in data.frame that uniquely identify each profile.

Value

A `cfp_soilphys` object.

See Also

Other data formats: `cfp_dat()`, `cfp_gasdata()`, `cfp_layered_profile()`, `cfp_layers_map()`, `cfp_profile()`

Examples

```
cfp_soilphys(
  ConFluxPro::soilphys,
  id_cols = c("site", "Date", "gas")
)
### Also used to extract an soilphys object from cfp_dat
cfp_soilphys(ConFluxPro::base_dat)
```

check_soilphys *Check for complete and correct soil physical parameters*

Description

This function analyses the soilphys dataframe before the flux calculation. It presents a warning, if there are variables missing and also looks for suspicious patterns that suggest an error in the interpolation made by discretize_depth. Mainly checks if certain columns are present and if they are missing, if they can be calculated from the data present. Looks for the following columns by default: "upper","lower","TPS","SWC","AFPS","t","p","DSD0","D0","DS"

Usage

```
check_soilphys(df, extra_vars = c(), id_cols)
```

Arguments

df	(dataframe) the soilphys dataframe
extra_vars	(character vector) column names of additional variables to be checked.
id_cols	(character vector) the columns that, together, identify a site uniquely (e.g. site, repetition)

Value

data frame of 'suspicious' parameter/depth combinations, where all values are NA.

See Also

Other soilphys: complete_soilphys(), discretize_depth(), soilphys_layered()

Examples

```
check_soilphys(ConFluxPro::soilphys, id_cols = c("site", "Date"))
```

combine_models *Combine models*

Description

Combinea list of multiple models or cfp_dat() objects into a single object.

Usage

```
combine_models(x)

## S3 method for class 'cfp_altres'
combine_models(x)

## S3 method for class 'list'
combine_models(x)

combine_models_by_reference(x_ref, x)
```

Arguments

x	A list of models, must inherit from cfp_dat()
x_ref	Reference element of x that controls the return class and attributes.

Value

An object of the same type as the first object in x.

Examples

```
mod1 <- filter(base_dat, site == "site_a")
mod2 <- filter(base_dat, site == "site_b")
combine_models(list(mod1, mod2))

# use a reference model for coercion
combine_models_by_reference(mod1, list(mod1, mod2))
```

---

complete_soilphys *(Re-)calculate soil physical parameters*

Description

This function completes the soilphys dataset by calculating different parameters if necessary, as long as all required parameters are available. Diffusion coefficients, as well as the air density are calculated if missing.

Usage

```
complete_soilphys(
  soilphys,
  DSD0_formula = NULL,
  gases = NULL,
  overwrite = TRUE,
  quiet = FALSE
)
```

Arguments

soilphys	(dataframe) the soilphys dataframe
DSD0_formula	(character) A character vector defining the way DSD0 should be calculated. Must refer to existing columns in soilphys. See examples below.
gases	(character) A character vector defining the gases for which to calculate D0 and DS.
overwrite	(logical) If true, already existing columns are overwritten.
quiet	(logical) Suppress messages.

Value

A `data.frame()` with all necessary columns for cfp_soilphys.

See Also

D0_massman

Other soilphys: `check_soilphys()`, `discretize_depth()`, `soilphys_layered()`

Examples

```
soilphys_barebones <- ConFluxPro::soilphys |>
 dplyr::select(
   c("site",
     "Date",
     "upper",
     "lower",
     "depth",
     "t",
     "p",
     "TPS",
     "SWC",
     "a",
     "b")
     )

complete_soilphys(
  soilphys_barebones,
  DSD0_formula = "a*AFPS^b",
  gases = "CO2")
```

`D0_massman` *Calculate D0*

Description

This function calculates the free-air diffusion coefficients of different gases for a given temperature and pressure.

Usage

```
D0_massman(gas, t, p)
```

Arguments

`gas`	(character) One of "CO2","CH4","N2O","O2","N2"
`t`	(numeric) temperature in °C
`p`	(numeric) pressure in hpa

Value

A numeric vector of D0 in m^2/s

References

Massman, W. J. A review of the molecular diffusivities of H2O, CO2, CH4, CO, O3, SO2, NH3, N2O, NO, and NO2 in air, O2 and N2 near STP. Atmospheric Environment 1998, 32(6), 1111–1127

Examples

```
D0_massman("CO2", 10, 1013)
```

`deepflux` *Extract flux rates from deep soil*

Description

Extract the incoming and outgoing flux from below the deepest layer of a `pro_flux()` model. This returns zero, if `zero_flux=TRUE`.

Usage

```
deepflux(x, ...)
```

Arguments

x	A valid cfp_pfres() object.
...	Further parameters passed on to efflux() in case of cfp_fgres.

Details

F0 represents the flux below the lowest layer defined in the cfp_pfres() model

Value

data.frame with F0 ($mol/m^2/s$)

Examples

```
PROFLUX <- ConFluxPro::base_dat |> pro_flux()

deepflux(PROFLUX)
```

---

depth_structure *Unique layers depths*

Description

Get the unique layers or depths, i.e. the backbone of an object given a set of identifying columns.

Usage

```
depth_structure(x, id_cols = NULL, ...)

## S3 method for class 'cfp_layered_profile'
depth_structure(x, id_cols = NULL, ...)

## S3 method for class 'cfp_profile'
depth_structure(x, id_cols = NULL, ...)

## S3 method for class 'cfp_dat'
depth_structure(x, id_cols = NULL, structure_from = NULL, ...)
```

Arguments

x	An object to get general structure of.
id_cols	The columns that identify each set of depth structures to extract (e.g. a site identifier).
...	internal One of "gasdata" "soilphys" or "layers_map".
structure_from	From which element should the structure be returned?

Value

A cfp_profile with columns depth, or upper and lower.

Examples

```
depth_structure(cfp_soilphys(ConFluxPro::base_dat))
depth_structure(cfp_gasdata(ConFluxPro::base_dat))
```

---

discretize_depth *Interpolate over depth to layered profile*

Description

Interpolate and discretize data into a layered structure. The output is a data.frame where each profile is separated into layers that intersect at depths defined in the function call. See `cfp_layered_profile()`.

There are different interpolation methods implemented, which might be more practical for different parameters or tasks.

- A `'linear'` interpolation for continuous parameters, (e.g. soil temperature).
- The `'boundary'` interpolation is only suitable for data that is already layered. It selects the value from the old layer that in which the new layer will lay in.
- A `'linspline'` interpolation fits a linear spline model to the data with knots defined in `knots`
- `'nearest'` finds the closest value to the new layer. You can define whether the closest value should be nearest to the top `1`, or bottom `0` of the layer using `int_depth`
- `'harmonic'` is similar to a linear interpolation but it uses the harmonic mean `harm()` using the distance in depth to each value as weights.

Multiple variables can be discretized at the same time by supplying multiple column names in `param`. It is also possible to use different `method` and controlling parameters `int_depth` and `knots` for each `param`. Just provide a list of settings the same length as `param`. If only one value is given, but multiple `param` the settings are reused for each parameter.

Usage

```
discretize_depth(
  df,
  param,
  method,
  depth_target,
  boundary_nearest = FALSE,
  boundary_average = "none",
  int_depth = 0.5,
  knots = NULL,
  ...
)
```

```
## S3 method for class 'cfp_profile'
discretize_depth(
  df,
  param,
  method,
  depth_target,
  boundary_nearest = FALSE,
  boundary_average = "none",
  int_depth = 0.5,
  knots = NULL,
  ...
)

## S3 method for class 'data.frame'
discretize_depth(
  df,
  param,
  method,
  depth_target,
  boundary_nearest = FALSE,
  boundary_average = "none",
  int_depth = 0.5,
  knots = NULL,
  id_cols = NULL,
  ...
)
```

Arguments

`df`	(dataframe) The dataframe containing the parameters to be interpolated, as well as the columns "depth", "upper" and "lower".
`param`	(character vector) The column names name of the parameters to be interpolated.
`method`	(character vector) a character (-vector) specifying the methods to be used for interpolation. Must be in the same order as param. One of • linear • boundary • linspline • nearest • harmonic
`depth_target`	(numeric vector or data frame) specifying the new layers. Must include n+1 depths for n target layers. If the target layers are different for id_cols, enter a data.frame instead. This data frame must have a "depth" column, as well as well as all `id_cols` needed that must be at least a subset of the `id_cols` of the original data.

`boundary_nearest`
: (logical) = TRUE/FALSE if it is TRUE then for target depth steps (partially) outside of the parameter boundaries, the nearest neighbor is returned, else returns NA. Default is FALSE.

`boundary_average`
: ("character) Defines what happens if the new layer contains multiple old layers. one of

  none = the default
  the new layer is set to NA

  arith the new layer is calculated as the arithmetic mean of the old

  harm the new layer is calculated as the harmonic mean of the old

`int_depth`
: (numeric vector) = value between 0 and 1 for 1 = interpolation takes the top of each depth step, 0.5 = middle and 0= bottom. Default = 0.5

`knots`
: (numeric vector) = the depths at which knots for the 'linspline' method are to be placed. If this differs for the parameters, a list of numeric vectors with the same length as "param" can be provided. Cannot differ between id_cols.

`...`
: Internal, must be empty.

`id_cols`
: Column names in data.frame that uniquely identify each profile.

Value

A `cfp_layered_profile()` data.frame with the variables `upper` and `lower` defining the layers derived from depth_target. The column `depth` is the middle of each layer. And all variables from `param`

See Also

Other soilphys: `check_soilphys()`, `complete_soilphys()`, `soilphys_layered()`

Examples

```
data("soiltemp")
library(dplyr)

dt <- data.frame(
  site = rep(c("site_a", "site_b"), each = 12),
  depth = c(5, seq(0,-100,-10), 7, seq(0,-100,-10)))

discretize_depth(df = soiltemp,
                 param = "t",
                 method = "linear",
                 depth_target = dt,
                 id_cols = c(
                   "site","Date"))
```

DSD0 *Calculate DSD0*

Description

Different functions to estimate soil diffusivity from the air-filled pore space.

Usage

```
DSD0_millington_quirk(AFPS, TPS = NULL, tortuosity = NULL)

DSD0_moldrup(AFPS, AFPS_100, b_campbell)

DSD0_currie(AFPS, a_currie = 1.9, b_currie = 1.4)

DSD0_linear(AFPS, a_lin, b_lin)
```

Arguments

AFPS	The air-filled porosity.
TPS	Total pore space
tortuosity	the tortuosity of the soil
AFPS_100	Air filled porosity at -100cm soil water matric head.
b_campbell	Campbell (1974) PSD index
a_currie, b_currie	fit parameter of Currie-style models
a_lin, b_lin	linear model coefficients

Details

- DSD0_millington_quirk() is of the form $D_s/D_0 = \Xi \cdot \epsilon$ where Ξ is the tortuosity factor (tortuosity) calulcated as $\Xi = \frac{\epsilon^{(10/3)}}{\Phi^2}$; ϵ is the air-filled pore space (AFPS) and Φ is the porosity (TPS). From Millington & Quirk (1961).
- DSD0_moldrup() is of the form $D_s/D_0 = (2 \cdot \epsilon_{100}^3 + 0.04 \cdot \epsilon_{100}) \cdot (\frac{\epsilon}{\epsilon_{100}})^{(2+\frac{3}{b_{campbell}})}$ where ϵ_{100} is the air-filled pore space at a matric potential head of -100 cm and $b_{campbell}$ is the slope of the water retention curve. From Moldrup et al. (2000).
- DSD0_currie() is of the form $D_s/D_0 = a \cdot \epsilon^b$ where a and b are fit parameter of an exponential model. From Currie (1960) with default values (a=1.9; b=1.4)from Troeh (1982).
- DSD0_linear() is a linear model of form $D_s/D_0 = a \cdot \epsilon + b$.

Value

A numeric vector of DSD0.

References

Millington, R. J., & Quirk, J. P. (1961). Permeability of porous solids. In Transactions of the Faraday Society (Vol. 57, p. 1200). Royal Society of Chemistry (RSC). https://doi.org/10.1039/tf9615701200

Moldrup, P., Olesen, T., Schjønning, P., Yamaguchi, T., & Rolston, D. E. (2000). Predicting the Gas Diffusion Coefficient in Undisturbed Soil from Soil Water Characteristics. In Soil Science Society of America Journal (Vol. 64, Issue 1, pp. 94–100). Wiley. https://doi.org/10.2136/sssaj2000.64194x

Currie, J. A. (1960). Gaseous diffusion in porous media. Part 2. - Dry granular materials. In British Journal of Applied Physics (Vol. 11, Issue 8, pp. 318–324). IOP Publishing. https://doi.org/10.1088/0508-3443/11/8/303

Troeh, F. R., Jabro, J. D., & Kirkham, D. (1982). Gaseous diffusion equations for porous materials. In Geoderma (Vol. 27, Issue 3, pp. 239–253). Elsevier BV. https://doi.org/10.1016/0016-7061(82)90033-7

Examples

```
DSD0_millington_quirk(0.2, 0.6)
DSD0_moldrup(0.2, 0.6, 1)
DSD0_currie(0.2)
DSD0_linear(0.2, a_lin = 1.4, b_lin = 0)
```

`efflux` *Extract efflux rates*

Description

Calculate or extract the soil/atmosphere efflux from `cfp_pfres` or `cfp_fgres` model results.

Usage

```
efflux(x, ...)

## S3 method for class 'cfp_pfres'
efflux(x, ...)

## S3 method for class 'cfp_fgres'
efflux(x, ..., method = "lm", layers = NULL)

## S3 method for class 'cfp_altres'
efflux(x, ...)
```

Arguments

`x`	A cfp_pfres or cfp_fgres model result, or a cfp_altres.
`...`	Arguments passed to methods.

method	Method(s) used to interpolate the efflux at the top of the soil from partial fluxes within the soil. One of **top** Use the flux in the topmost model layer. **lm** A linear model where each partial flux is centered in the respective layer and the model is evaluated at the top of the soil. **lex** Linearly extrapolate using fluxes of two layers in the soil.
layers	Vector of two integers selecting the layers for the `lex` method. Layers are indexed from 1 (topmost) to the number of layers used in the flux calculation.

Value

A `data.frame` with one row for each combination of `id_cols` and the column `efflux` in $molm^{-2}s^{-1}$.

Examples

```
my_dat <- ConFluxPro::base_dat |>
  filter(Date < "2021-03-01") #subset to speed up example
PROFLUX <- pro_flux(my_dat)
FLUX <- fg_flux(my_dat)

efflux(PROFLUX)
efflux(FLUX)
```

---

error_concentration *Estimate model error*

Description

A set of functions that can be called on an cfp_pfres object (the result of a call to pro_flux) to assess the quality of the model.

Usage

```
error_concentration(x, param_cols = NULL, normer = "sd")

## S3 method for class 'cfp_pfres'
error_concentration(x, param_cols = NULL, normer = "sd")

## S3 method for class 'cfp_fgres'
error_concentration(x, param_cols = NULL, normer = "sd")

## S3 method for class 'cfp_altres'
error_concentration(x, param_cols = NULL, normer = "sd")

error_efflux(x, param_cols, EFFLUX, normer = "sd", ...)
```

```
## S3 method for class 'cfp_pfres'
error_efflux(x, param_cols, EFFLUX, normer = "sd", ...)

## S3 method for class 'cfp_fgres'
error_efflux(x, param_cols, EFFLUX, normer = "sd", ...)

## S3 method for class 'cfp_altres'
error_efflux(x, param_cols, EFFLUX, normer = "sd", ...)
```

Arguments

x	A cfp_pfres object, that is returned by a call to pro_flux()
param_cols	The columns that, together, define different parameters (e.g. different gases) for which NRMSEs should be calculated separately (e.g. "gas"). Defaults to the id_cols of layers_map. If no such distinction is wished, set to `character()`
normer	a character string defining the type of normalization to be applied. Can be one of **mean** the arithmetic mean of a **sd** the standard deviation of a (default). **range** the difference between the range of a **IQR** the difference between the interquantile range of a
EFFLUX	A data.frame with (at most) one value of efflux per profile of x. Must contain any id_cols of x needed.
...	Further arguments passed to efflux

Details

For error_concentration, the way the error parameter is calculated for cfp_fgres and cfp_pfres objects is entirely different and should not be used in comparison between the two. NRMSE of cfp_pfres objects are calculated as the mean of depth-wise NRMSEs of modelled versus input gas concentrations. 'NRMSE's of cfp_fgres objects simply calculate the mean of (dcdz_sd / dcdz_ppm) per group described in param_cols.

Value

The calculated error estimate for a single model, a list of models (cfp_altres) and for each parameter combination in `param_cols`

Examples

```
PROFLUX <- pro_flux(base_dat)

error_concentration(PROFLUX)
error_efflux(
  PROFLUX,
  EFFLUX = data.frame(efflux = 1),
  param_cols = c("site"))
```

`evaluate_models` *Evaluate model runs for calibration*

Description

Evaluate the model runs produced by a call to `alternate()` with user-defined error functions.

Usage

```
evaluate_models(
  x,
  eval_funs = NULL,
  eval_weights = 1,
  param_cols,
  eval_cols,
  n_best = NULL,
  f_best = 0.01,
  scaling_fun = scale_min_median,
  ...
)

## S3 method for class 'cfp_altres'
evaluate_models(
  x,
  eval_funs = NULL,
  eval_weights = 1,
  param_cols = cfp_id_cols(cfp_layers_map(cfp_og_model(x))),
  eval_cols = NULL,
  n_best = NULL,
  f_best = 0.01,
  scaling_fun = scale_min_median,
  ...
)
```

Arguments

`x`	A cfp_altres object, as returned by `alternate()`.
`eval_funs`	A named list of evaluation functions. Each function must accept the arguments `x` and `param_cols` that are passed from this function.
`eval_weights`	A vector of weights the same length of `eval_funs` or one. Alternatively a `data.frame()` that specifies the weight for any wished `error_parameter` (names of `eval_funs`) and `param_cols` combinations. Provide the weights as a numeric in the `parameter_weight` column.
`param_cols`	The columns that, together, define different parameters (e.g. different gases) for which NRMSEs should be calculated separately (e.g. "gas"). Defaults to the id_cols of layers_map. If no such distinction is wished, set to `character()`

eval_cols	A character vector of columns for which the model error should be returned separately. Must be a subset of param_cols and defaults to the complete set.
n_best	An integer number of runs to select as the best runs.
f_best	A numeric between 0 to 1 as the fraction of runs to select as the best. Defaults to 0.01.
scaling_fun	A scaling function. Defaults to min-median scaling.
...	Any arguments that need to be passed to the error_funs. Note that all matching arguments will be applied to each function!

Value

A list with components best_runs the runs with the lowest model error (ME), model_error the model error for all runs, models_evaluated the raw values returned by error_funs and best_runs_runmap, a cfp_run_map() which can be used to rerun the best_runs model configurations. Note, that for best_runs_runmap the value of run_id is remapped to values 1:n_best.

Examples

```
PROFLUX <- pro_flux(base_dat |> filter(site == "site_a"))

run_map <-
 cfp_run_map(
   PROFLUX,
   params = list(TPS = c(0.9, 1.1)),
   type = "factor",
   n_runs = 5)

PF_alt <- alternate(
  PROFLUX,
  \(x) complete_soilphys(x, DSD0_formula = "a*AFPS^b", quiet = TRUE),
  run_map)

evaluate_models(
  PF_alt,
  eval_funs = list("NRMSE_conc" = error_concentration)
    )
```

---

extractors *Extract elements from an object*

Description

These functions extract components from different objects that can be created in ConFluxPro.

Usage

```
cfp_og_model(x)

## S3 method for class 'cfp_altres'
cfp_og_model(x)

cfp_id_cols(x)

cfp_gases(x)

cfp_modes(x)

cfp_param(x)

cfp_funs(x)

cfp_zero_flux(x)

cfp_zero_limits(x)

cfp_DSD0_optim(x)

cfp_evenness_factor(x)

cfp_known_flux_factor(x)

cfp_runmap_type(x)

cfp_params_df(x)

cfp_n_runs(x)

cfp_layers_different(x)

cfp_layers_from(x)

cfp_layers_altmap(x)
```

Arguments

x	An object from which to extract the information.

Value

The extracted component, e.g. a `data.frame()` or `character()`.

Examples

```
my_data <- ConFluxPro::base_dat |>
```

```
  filter(Date == "2021-01-01") # subset for example = faster runtime

### from cfp_dat objects (and derivatives)
cfp_id_cols(my_data)

cfp_gasdata(my_data) |> head()
cfp_soilphys(my_data) |> head()
cfp_layers_map(my_data) |> head()
my_data$profiles |> head()

### from cfp_pfmod or cfp_pfres objects
PROFLUX <- my_data |> pro_flux()
cfp_zero_flux(PROFLUX)
cfp_zero_limits(PROFLUX)
cfp_DSD0_optim(PROFLUX) #deprecated
cfp_evenness_factor(PROFLUX)
cfp_known_flux_factor(PROFLUX)
PROFLUX$PROFLUX |> head()

### from cfp_fgmod or cfp_fgres objects
FLUX <- my_data |> fg_flux()

cfp_gases(FLUX)
cfp_modes(FLUX)
cfp_param(FLUX)
cfp_funs(FLUX)
FLUX$FLUX |> head()

### from cfp_run_map
set.seed(42)
my_run_map <-
cfp_run_map(
  PROFLUX,
  list("TPS" = c(0.9, 1.1)),
  "factor",
  n_runs = 2)

cfp_params_df(my_run_map)
cfp_n_runs(my_run_map)
cfp_layers_from(my_run_map)
cfp_layers_different(my_run_map)
cfp_runmap_type(my_run_map)
cfp_layers_altmap(my_run_map)

### from cfp_altres
my_altres <-
alternate(
  x = PROFLUX,
  f = \(x) complete_soilphys(x, "a+AFPS^b", quiet = TRUE),
  run_map = my_run_map)

cfp_og_model(my_altres)
cfp_run_map(my_altres)
```

`fg_flux` *Flux-gradient method*

Description

`fg_flux()` implements different approaches to the flux-gradient method (FGM). It takes a valid input dataset from `cfp_dat()` and calculates for each layer defined in `cfp_layers_map()`.

Usage

```
fg_flux(x, ...)

## S3 method for class 'cfp_dat'
fg_flux(x, ...)

## S3 method for class 'cfp_fgres'
fg_flux(x, ...)

## S3 method for class 'cfp_fgmod'
fg_flux(x, ...)
```

Arguments

`x`	A `cfp_dat` object with all the necessary input datasets.
`...`	Arguments passed on to `cfp_fgmod`

`gases` (character) A character vector defining the gases for which fluxes shall be calculated.

`modes` (character) A character vector specifying mode(s) for dcdz calculation. Can be `"LL"`,`"LS"`,`"EF"`.

- **LL** local linear approach: within each layer a linear model is evaluated of concentration over the depth.
- **LS** linear spline approach: A linear spline is fitted over the complete profile with nodes at the layer intersections.
- **EF** exponential fit approach: An exponential function of form y=a+b*x^c is fit of concentration against depth. Using the first derivative of that function the concentration gradient is evaluated for each layer.
- **DA** exponential fit approach: An exponential function of form `y=a+b*(1-exp(-a*x))` is fit of concentration against depth. Using the first derivative of that function the concentration gradient is evaluated for each layer. From Davidson (2006).

`param` (character) A vector containing the the parameters of soilphys, for which means should be calculated, must contain "c_air" and "DS", more parameters may help interpretation.

`funs` (character) A vector defining the type of mean to be used for each parameter in `param`. One of "arith" or "harm".

Details

The model result contains the original data, but adds the dataset FLUX, which contains the calculated flux rates. You can use functions efflux and production to calculate different elements or access the raw result with model_result$FLUX.

Value

A cfp_fgres model result.

References

DAVIDSON, E. A., SAVAGE, K. E., TRUMBORE, S. E., & BORKEN, W. (2006). Vertical partitioning of CO2 production within a temperate forest soil. In Global Change Biology (Vol. 12, Issue 6, pp. 944–956). Wiley. https://doi.org/10.1111/j.1365-2486.2005.01142.x

See Also

Other flux models: pro_flux()

Examples

```
fg_flux(ConFluxPro::base_dat)
```

---

filter *Filter profiles*

Description

Filter profiles by their id_cols or (where available) by their prof_id. This is built on dplyr::filter().

Usage

```
filter(.data, ..., .by = NULL, .preserve = FALSE)

## S3 method for class 'cfp_dat'
filter(.data, ..., .preserve = FALSE)
```

Arguments

.data	A cfp_dat() object or its derivatives.
...	<data-masking> Expressions that return a logical value, and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

.by	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
.preserve	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Value

A subset of the original data.

Examples

```
base_dat |>
  filter(site == "site_a")

base_dat |>
  filter(Date > "2022-03-01")
```

---

flux *Re-run model*

Description

A function to either run fg_flux() or pro_flux() models from valid cfp_fgmod or cfp_pfmod objects.

Usage

```
flux(x)
```

Arguments

x	A valid cfp_fgmod or cfp_pfmod object.

Value

Either a cfp_pfres or cfp_fgres model result.

Examples

```
FLUX <- ConFluxPro::base_dat |> fg_flux()
FLUX2 <- flux(FLUX)

all.equal(FLUX, FLUX2)
```

gasdata *Example soil CO2 concentrations*

Description

A synthetic dataset of soil CO2 concentrations at two sites over a one-year period.

Usage

```
gasdata
```

Format

A tibble with 312 rows and 6 variables:

site name of the site

Date Date in the format "YYYY-MM-DD"

depth depth from mineral soil in cm

repetition id for which repetition in each depth

x_ppm concentration, in ppm

gas name of the gas

harm *Harmonic mean*

Description

This function calculates harmonic mean of a vector and can be used analogous to the base functions mean() or median()

Usage

```
harm(x, w = 1, na.rm = FALSE)
```

Arguments

x	(numeric vector)
w	(numeric vector) optional vector of weights corresponding to x. Default is 1 for all.
na.rm	(logical) If TRUE, then NA values are omitted and the mean calculated with the remaining values. If FALSE (default) then returns NA if x contains NA values.

Value

(numeric) harmonic mean of x

Examples

```
harm(c(1:10))
harm(c(1:10),c(10:1))
```

`layers_map` *Example cfp_layers_map object*

Description

An example dataset for `layers_map` that devides each `site` into two layers.

Usage

```
layers_map
```

Format

A data.frame with 4 rows and 3 variables:

site name of the site

upper upper limit for layer in cm

lower lower limit for layer in cm

`n_groups` *Get number of groups/profiles*

Description

Get number of groups/profiles

Usage

```
n_groups(x)

n_profiles(x)
```

Arguments

`x`	A `cfp_dat` object.

Value

An integer giving the number of groups of the object.

An integer giving the number of profiles of the object.

Examples

```
n_groups(base_dat)

n_profiles(base_dat)
n_profiles(cfp_soilphys(base_dat))
```

`plot_profile` *Plot profiles*

Description

Plot vertical soil profiles of ConFluxPro objects using ggplot. This is mainly intended for diagnostic purposes and better understand the underlying data.

Supported objects:

cfp_pfres Displays TPS, SWC and AFPS, as well as production and measured and modelled gas concentrations.

cfp_fgres Displays TPS, SWC and AFPS, as well as the measured concentration profile, and concentration gradients for each layer.

cfp_soilphys Displays TPS, SWC and AFPS, as well as values of Ds and Temperature.

cfp_gasdata Displays the concentration profile.

cfp_layers_map Displays the layer names, pmap and layer_couple, as well as the allowed production range.

Usage

```
plot_profile(x)
```

Arguments

`x`	A cfp_pfres, cfp_fgres model result, or a cfp_soilphys, cfp_gasdata or cfp_layers_map object

Value

A ggplot2 plot with facets for each distinct profile. If more than 20 profiles are plotted a message is sent because this can take a long time.

Examples

```
data_subset <-  base_dat |>
  filter(Date == "2021-02-01")

plot_profile(cfp_soilphys(data_subset))
plot_profile(cfp_gasdata(data_subset))
plot_profile(cfp_layers_map(data_subset))
```

`production` *Extract production rates*

Description

Easily extract the production of `cfp_pfres()` and `cfp_fgres()` models per layer defined in `layers_map()` and calculate the relative contribution per layer.

Usage

```
production(x, ...)
```

Arguments

`x`	A valid `cfp_pfres()` or `cfp_fgres()` object.
`...`	Further parameters passed on to `efflux()` in case of `cfp_fgres`.

Details

For a `pro_flux()` model, the extraction is straightforward and simply the product of the optimised production rate (per volume) multiplied by the height of the layer.

For `fg_flux()`, the assumption is made that the production of the layer i is the difference of the flux in the layer above F_{i+1} and the layer below F_{i-1}. The flux below the lowest layer is assumed to be zero and the flux above the topmost layer is the efflux. This approach has some uncertainties and it should be evaluated if it applies to your model.

If there are error estimates available from a call to `bootstrap_error()`, the errors are propagated as follows:

$$\Delta prod_{rel} = |\Delta efflux \cdot \frac{prod_{abs}}{efflux^2}| + |\Delta prod_{abs} \cdot \frac{1}{efflux}|$$

Value

data.frame with `prod_abs` ($mol/m^2/s$), `efflux` ($mol/m^2/s$) and `prod_rel` where $prod_{rel} = efflux/prod_{abs}$.

Examples

```
PROFLUX <- pro_flux(base_dat)

production(PROFLUX)
```

`pro_flux` *Inverse model of production profiles*

Description

This implements an inverse modeling approach which optimizes vertically resolved production (or consumption) of the gases in question to fit a modeled concentration profile to observed data.

One boundary condition of this model is, that there is no incoming or outgoing flux at the bottom of the lowest layer of the profile. If this boundary condition is not met, the flux must be optimised as well. This can be set in `zero_flux`.

Usage

```
pro_flux(x, ...)

## S3 method for class 'cfp_dat'
pro_flux(x, ...)

## S3 method for class 'cfp_pfres'
pro_flux(x, ...)

## S3 method for class 'cfp_pfmod'
pro_flux(x, ...)
```

Arguments

`x`	A `cfp_dat` object with all the necessary input datasets.
`...`	Arguments passed on to `cfp_pfmod`
	`zero_flux` (logical) Applies the zero-flux boundary condition? If `FALSE`, `F0` is optimized alongside the production rates.
	`zero_limits` (numeric vector) a vector of length 2 defining the lower and upper limit of the lowest flux if `zero_flux` = `FALSE`.
	`DSD0_optim` **[Deprecated]**
	`evenness_factor` **[Experimental]**(numeric) A user defined factor used to penalise strong differences between the optimised production rates. This must be identified by trial-and-error and can help prevent that production rates are simply set to zero basically the lower a production is relative to the the maximum of the absolute of all productions, the higher it is penalised. The `evenness_factor` then defines the weight of this penalty in the optimisation algorithm `prod_optim`.

Value

A cfp_pfres() model result.

See Also

Other flux models: fg_flux()

Examples

```
soilphys <-
 cfp_soilphys(
   ConFluxPro::soilphys,
   id_cols = c("site", "Date")
 )

gasdata <-
 cfp_gasdata(
   ConFluxPro::gasdata,
   id_cols = c("site", "Date")
 )


lmap <-
 cfp_layers_map(
   ConFluxPro::layers_map,
   gas = "CO2",
   lowlim = 0,
   highlim = 1000,
   id_cols = "site"
 )

PROFLUX <-
 cfp_dat(gasdata,
         soilphys,
         lmap ) |>
 pro_flux()
```

---

rmse *(Normalized) root mean square error*

Description

(Normalized) root mean square error

Calculate the (normalized) root-mean-square-error of two vectors.

Usage

```
rmse(a, b)

nrmse(a, b, normer = "sd")
```

Arguments

a, b	numeric vectors of same length to be compared
normer	a character string defining the type of normalization to be applied. Can be one of **mean** the arithmetic mean of a **sd** the standard deviation of a (default). **range** the difference between the range of a **IQR** the difference between the interquantile range of a

Value

The (normalised) rmse of the provided vector.

Examples

```
set.seed(42)
a <- c(1, 2, 3, 4)
b <- a * rnorm(4, 1, 0.1)
rmse(a, b)
nrmse(a, b, normer = "sd")
nrmse(a, b, normer = "mean")
```

---

run_map *run_map*

Description

[Deprecated]

`run_map()` was deprecated in favor of cfp_run_map for consistency.

Create a cfp_run_map for model alteration in alternate()

Usage

```
run_map(
  x,
  params = list(),
  type = NULL,
  method = NULL,
```

```
  n_runs = NULL,
  layers_different = FALSE,
  layers_from = "layers_map",
  layers_altmap = NULL,
  topheight_adjust = FALSE
)
```

Arguments

x	Either a cfp_pfres or cfp_fgres model result.
params	A named list of numeric vectors. Names indicate column names in soilphys, vectors either distinct values (method permutation) or limits (method random).
type	A vector of length param indicating what the values in params represent. One of **abs** Absolute values that are applied as-is. **factor** Factors to be multiplied with the original values. **addition** Factors to be added to the original values.
method	Either 'random', where a random value is chosen within the bounds set in params or 'permutation', where every permutation of the values in params is added.
n_runs	Integer value of the number of alterations to be done for method = 'random'.
layers_different	Should layers from layers_map be changed individually? If TRUE this allows for different changes at different depths.
layers_from	(character) If layers_different is TRUE, from which source should the layers be created? One of: **layers_map** (default) Use the layers that are defined in layers_map. **soilphys** Use the layers as defined in soilphys **layers_altmap** Use the layers as defined in the provided layers_altmap object.
layers_altmap	An optional layers_map created using layers_map() that defines the layers to be used if layers_different = TRUE.
topheight_adjust	(logical) If the proposed change in topheight is larger than the highest layer in soilphys, should the limits be automatically adjusted per id_cols individually? Default is FALSE, which leads to an error in that case.

Value

An object of type cfp_run_map that can be used within alternate.

Examples

```
PROFLUX <- ConFluxPro::base_dat |> pro_flux()
# Create a cfp_run_map where TPS is changed between 90 % and 110 %
# of the original value for 50 runs.
cfp_run_map(
  PROFLUX,
```

```
  list("TPS" = c(0.9, 1.1)),
  "factor",
  n_runs = 50)
```

`scale_min_median` *Scale a vector by its min and median*

Description

Scale a vector between its minimum and median.

Usage

```
scale_min_median(x)
```

Arguments

`x`	a numeric vector

Value

`x` scaled between min and median of `x`.

Examples

```
scale_min_median(1:10)
```

`season` *Get season of a date-time*

Description

A simple function to return a character (-vector) of the season from a Date (-vector). Months:

spring 3-5

summer 6-8

fall 9-11

winter 12-2

Usage

```
season(d)
```

Arguments

d	(Date) Any date object

Value

A character vector the same length as d

Examples

```
season(as.Date(c("1955-01-15","1985-06-15","2015-10-15")))
```

---

sobol_calc_indices *Calculate sobol indices*

Description

[Experimental]

From any result parameter and its corresponding `cfp_run_map` calculate first-order and total sobol indices using the Azzini (2021) method.

Usage

```
sobol_calc_indices(Y, effect_cols, id_cols = character(), run_map)
```

Arguments

Y	A data.frame with the desired effect parameter(s) of the model output, e.g. `efflux()`. The output should come from a list of model results produced by a call to `alternate()` with a valid `cfp_run_map` produced by `sobol_run_map()`.
effect_cols	character vector of the column names in `Y` for which sobol indices should be calculated, e.g. `'efflux'`.
id_cols	character vector of column names in `Y` specifying grouping variables. Indices are then calculated for each group individually.
run_map	The `cfp_run_map` used for the calculation of Y produced by a call to `sobol_run_map()`.

Details

This implements the approach outlined in Azzini et al (2021).

Value

A `data.frame` with the following columns

`...` Any `id_cols` specified

`param_id, param, pmap` Parameter identificators from the `cfp_run_map` used.

`effect_param` The parameter for which the effect was calculated.

`Vt, Vi, VY` Internal parameters for the indice calculation.

`Si` First order sobol indice.

`ST` Total order sobol indice.

References

Azzini, Ivano; Mara, Thierry A.; Rosati, Rossana: Comparison of two sets of Monte Carlo estimators of Sobol' indices, Environmental Modelling & Software, Volume 144, 2021, 105167, ISSN 1364-8152, https://doi.org/10.1016/j.envsoft.2021.105167

See Also

Other sobol: `sobol_run_map()`

Examples

```
PROFLUX <- pro_flux(base_dat)

sobol_map <- sobol_run_map(PROFLUX,
 params = list("TPS" = c(0.9, 1.1),
               "t" = c(0.9, 1.1)),
 type = c("factor", "factor"),
 n_runs = 10)

 PF_sobol <-
alternate(
  PROFLUX,
  \(x) complete_soilphys(x, DSD0_formula = "a*AFPS^b", quiet = TRUE),
  sobol_map)

sobol_calc_indices(efflux(PF_sobol), "efflux", c("site"), sobol_map)
```

---

sobol_run_map *Create a run plan for sobol indice calculation*

Description

[Experimental]

Modify an existing `cfp_run_map` for sobol indice estimation or create a new one from scratch.

Usage

```
sobol_run_map(x, ...)

## S3 method for class 'cfp_dat'
sobol_run_map(x, ...)

## S3 method for class 'cfp_run_map'
sobol_run_map(x, ...)
```

Arguments

x	Either an object of class cfp_run_map created by a call to cfp_run_map() with method = 'random', or a cfp_pfres or cfp_fgres model result.
...	Arguments passed on to run_map

params A named list of numeric vectors. Names indicate column names in soilphys, vectors either distinct values (method permutation) or limits (method random).

type A vector of length param indicating what the values in params represent. One of

abs Absolute values that are applied as-is.

factor Factors to be multiplied with the original values.

addition Factors to be added to the original values.

method Either 'random', where a random value is chosen within the bounds set in params or 'permutation', where every permutation of the values in params is added.

n_runs Integer value of the number of alterations to be done for method = 'random'.

layers_different Should layers from layers_map be changed individually? If TRUE this allows for different changes at different depths.

layers_from (character) If layers_different is TRUE, from which source should the layers be created? One of:

layers_map (default) Use the layers that are defined in layers_map.

soilphys Use the layers as defined in soilphys

layers_altmap Use the layers as defined in the provided layers_altmap object.

layers_altmap An optional layers_map created using layers_map() that defines the layers to be used if layers_different = TRUE.

topheight_adjust (logical) If the proposed change in topheight is larger than the highest layer in soilphys, should the limits be automatically adjusted per id_cols individually? Default is FALSE, which leads to an error in that case.

Value

A cfp_run_map to be used in alternate for sensitivity analysis.

See Also

Other sobol: sobol_calc_indices()

Examples

```
PROFLUX <- pro_flux(base_dat)

sobol_run_map(PROFLUX,
 params = list("TPS" = c(0.9, 1.1),
               "t" = c(0.9, 1.1)),
 type = c("factor", "factor"),
 n_runs = 10)
```

---

soildiff *Example soil diffusion models*

Description

A synthetic dataset of soil total pore space and diffusion models after the general formula a*AFPS^b.

Usage

```
soildiff
```

Format

A tibble with 8 rows and 6 variables:

site name of the site

upper upper limit for layer in cm

lower lower limit for layer in cm

TPS total pore space as fraction of volume

a diffusion-model fit parameter a

b diffusion-model fit parameter b

soilphys *Example cfp_soilphys object*

Description

An example dataset for `soilphys` based on the sets `soiltemp`, `soilwater` and `soildiff`

Usage

```
soilphys
```

Format

A tibble with 120 rows and 4 variables:

site name of the site

Date Date in the format "YYYY-MM-DD"

depth depth in cm

t temperature in °C

soiltemp *Example soil temperature*

Description

A synthetic dataset of soil temperature at discrete depths. The dates correspond to `gasdata`.

Usage

```
soiltemp
```

Format

A tibble with 120 rows and 4 variables:

site name of the site

Date Date in the format "YYYY-MM-DD"

depth depth in cm

t temperature in °C

soilwater *Example soil water content*

Description

A synthetic dataset of soil water content in a layered structure. The dates correspond to gasdata.

Usage

```
soilwater
```

Format

A tibble with 180 rows and 5 variables:

site name of the site
Date Date in the format "YYYY-MM-DD"
upper upper limit for layer in cm
lower lower limit for layer in cm
SWC soil water content as fraction of volume

unique_gases *Get the unique gases of an object*

Description

Get the gases from a CFP object.

Usage

```
unique_gases(x)
```

Arguments

x	the object to extract the gases from.

Value

A character vector of gases in that object.

Examples

```
unique_gases(base_dat)

data.frame(gas = c("CO2", "CH4")) |>
cfp_profile(id_cols = "gas") |>
unique_gases()
```

Index