Package ‘mkin’

July 12, 2022

Type Package

Title Kinetic Evaluation of Chemical Degradation Data

Version 1.1.1

Date 2022-07-08

Description Calculation routines based on the FOCUS Kinetics Report (2006, 2014). Includes a function for conveniently defining differential equation models, model solution based on eigenvalues if possible or using numerical solvers. If a C compiler (on windows: ‘Rtools’) is installed, differential equation models are solved using automatically generated C functions. Heteroscedasticity can be taken into account using variance by variable or two-component error models as described by Ranke and Meinecke (2018) <doi:10.3390/environments6120124>. Interfaces to several nonlinear mixed-effects model packages are available, some of which are described by Ranke et al. (2021) <doi:10.3390/environments8080071>. Please note that no warranty is implied for correctness of results or fitness for a particular purpose.

Depends R (>= 2.15.1), parallel

Imports stats, graphics, methods, deSolve, R6, inline (>= 0.3.19), numDeriv, lmtest, pkgbuild, nlme (>= 3.1-151), purrr, saemix (>= 3.0)

Suggests knitr, rbenchmark, tikzDevice, testthat, rmarkdown, covr, vdiffr, benchmarkme, tibble, stats4

License GPL

LazyLoad yes

LazyData yes

Encoding UTF-8

Language en-GB

VignetteBuilder knitr

BugReports https://github.com/jranke/mkin/issues/

URL https://pkgdown.jrwb.de/mkin/
RoxygenNote 7.2.0

NeedsCompilation no

Author Johannes Ranke [aut, cre, cph]
    (<https://orcid.org/0000-0003-4371-6538>),
    Katrin Lindenberger [ctb] (contributed to mkinresplot()),
    René Lehmann [ctb] (ilr() and invilr()),
    Eurofins Regulatory AG [cph] (copyright for some of the contributions
    of JR 2012-2014)

Maintainer Johannes Ranke <johannes.ranke@jrwb.de>

Repository CRAN

Date/Publication 2022-07-12 14:10:02 UTC

R topics documented:

add_err .......................................................... 4
AIC.mmkin .......................................................... 6
aw ................................................................. 7
CAKE_export ...................................................... 8
confint.mkinfit .................................................. 9
create_deg_func .................................................. 12
D24_2014 ........................................................... 13
DFOP.solution .................................................... 14
dimethenamid_2018 ............................................. 15
endpoints ........................................................ 17
experimental_data_for_UBA_2019 ................................. 18
FOCUS_2006_datasets ........................................... 20
FOCUS_2006_DFOP_ref_A_to_B .................................. 21
FOCUS_2006_FOMC_ref_A_to_F .................................. 22
FOCUS_2006_HS_ref_A_to_F ..................................... 23
FOCUS_2006_SFO_ref_A_to_F .................................... 24
focus_soil_moisture ............................................. 25
FOMC.solution ................................................... 25
f_time_norm_focus .............................................. 26
get_deg_func ..................................................... 28
HS.solution ....................................................... 28
ilr ............................................................... 29
intervals.saem.mmkin ............................................ 31
IORE.solution ................................................... 31
loftest .......................................................... 32
logistic.solution ............................................... 34
logLik.mkinfit ................................................. 36
lrtest.mkinfit .................................................. 37
max_twa_parent ................................................ 38
mccall81_245T ................................................... 39
mean_degpars ................................................... 40
mixed ........................................................ 41
R topics documented:

mkinds .................................................. 43
mkindsg .................................................. 44
mkinerrmin ............................................. 46
mkinerrplot ............................................ 47
mkinfit .................................................. 48
mkinmod .................................................. 54
mkinparplot ............................................ 57
mkinplot .................................................. 58
mkinpredict ............................................. 59
mkinresplot ............................................ 61
mkin_long_to_wide ..................................... 63
mkin_wide_to_long ..................................... 64
mmkin ..................................................... 64
nafta ....................................................... 66
NAFTA_SOP_2015 ......................................... 68
NAFTA_SOP_Attachment ................................ 68
nlme.mmkin ............................................... 69
nlme_function ........................................... 72
nobs.mkinfit ............................................ 74
parms ..................................................... 74
plot.mixed.mmkin ....................................... 75
plot.mkinfit ............................................. 78
plot.mmkin ............................................... 81
plot.nafta ............................................... 82
residuals.mkinfit ...................................... 83
saem ....................................................... 84
schaef07_complex_case ................................. 87
SFO.solution ............................................ 88
SFORB.solution .......................................... 89
sigma_twocomp .......................................... 90
summary.mkinfit ....................................... 91
summary.nlme.mmkin ................................... 93
summary.saem.mmkin .................................. 95
synthetic_data_for_UBA_2014 .......................... 97
test_data_from_UBA_2014 ............................... 100
transform_odeparms .................................... 101
update.mkinfit ......................................... 104
[.mmkin .................................................. 104

Index ..................................................... 106
Add normally distributed errors to simulated kinetic degradation data

**Description**

Normally distributed errors are added to data predicted for a specific degradation model using `mkinpredict`. The variance of the error may depend on the predicted value and is specified as a standard deviation.

**Usage**

```r
add_err(
  prediction,
  sdfunc,
  secondary = c("M1", "M2"),
  n = 10,
  LOD = 0.1,
  reps = 2,
  digits = 1,
  seed = NA
)
```

**Arguments**

- **prediction**: A prediction from a kinetic model as produced by `mkinpredict`.
- **sdfunc**: A function taking the predicted value as its only argument and returning a standard deviation that should be used for generating the random error terms for this value.
- **secondary**: The names of state variables that should have an initial value of zero.
- **n**: The number of datasets to be generated.
- **LOD**: The limit of detection (LOD). Values that are below the LOD after adding the random error will be set to NA.
- **reps**: The number of replicates to be generated within the datasets.
- **digits**: The number of digits to which the values will be rounded.
- **seed**: The seed used for the generation of random numbers. If NA, the seed is not set.

**Value**

A list of datasets compatible with `mmkin`, i.e. the components of the list are datasets compatible with `mkinfit`.

**Author(s)**

Johannes Ranke
References


Examples

# The kinetic model
m_SFO_SFO <- mkinmod(parent = mkinsub("SFO", "M1"),
                     M1 = mkinsub("SFO"), use_of_ff = "max")

# Generate a prediction for a specific set of parameters
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)

# This is the prediction used for the "Type 2 datasets" on the Piacenza poster
# from 2015
# It's not run:
d_SFO_SFO <- mkinpredict(m_SFO_SFO,
                          c(k_parent = 0.1, f_parent_to_M1 = 0.5,
                           k_M1 = log(2)/1000),
                          c(parent = 100, M1 = 0),
                          sampling_times)

# Add an error term with a constant (independent of the value) standard deviation
# of 10, and generate three datasets
# It's not run:
d_SFO_SFO_err <- add_err(d_SFO_SFO, function(x) 10, n = 3, seed = 123456789)

# Name the datasets for nicer plotting
names(d_SFO_SFO_err) <- paste("Dataset", 1:3)

# Name the model in the list of models (with only one member in this case) for
# nicer plotting later on. Be quiet and use only one core not to offend CRAN
# checks
## Not run:
f_SFO_SFO <- mmkin(list("SFO-SFO" = m_SFO_SFO),
                   d_SFO_SFO_err, cores = 1,
                   quiet = TRUE)

plot(f_SFO_SFO)

# We would like to inspect the fit for dataset 3 more closely
# Using double brackets makes the returned object an mkinfit object
# instead of a list of mkinfit objects, so plot.mkinfit is used
plot(f_SFO_SFO[[3]], show_residuals = TRUE)

# If we use single brackets, we should give two indices (model and dataset),
# and plot.mmkin is used
plot(f_SFO_SFO[1, 3])

## End(Not run)
Calculate the AIC for a column of an mmkin object

Description
Provides a convenient way to compare different kinetic models fitted to the same dataset.

Usage
```r
## S3 method for class 'mmkin'
AIC(object, ..., k = 2)

## S3 method for class 'mmkin'
BIC(object, ...)
```

Arguments
- `object`: An object of class `mmkin`, containing only one column.
- `...`: For compatibility with the generic method
- `k`: As in the generic method

Value
As in the generic method (a numeric value for single fits, or a dataframe if there are several fits in the column).

Author(s)
Johannes Ranke

Examples
```r
## Not run: # skip, as it takes > 10 s on winbuilder
f <- mmkin(c("SFO", "FOMC", "DFOP"),
  list("FOCUS A" = FOCUS_2006_A,
       "FOCUS C" = FOCUS_2006_C), cores = 1, quiet = TRUE)
# We get a warning because the FOMC model does not converge for the
# FOCUS A dataset, as it is well described by SFO
AIC(f["SFO", "FOCUS A"])
AIC(f[["SFO", "FOCUS A"]]) # or when extracting an mkinfit object

# For FOCUS A, the models fit almost equally well, so the higher the number
# of parameters, the higher (worse) the AIC
AIC(f[, "FOCUS A"])
AIC(f[, "FOCUS A"], k = 0) # If we do not penalize additional parameters, we get nearly the same
BIC(f[, "FOCUS A"])
```

# Comparing the BIC gives a very similar picture
# For FOCUS C, the more complex models fit better
AIC(f[, "FOCUS C"])
BIC(f[, "FOCUS C"])

## End(Not run)

---

### aw

**Calculate Akaike weights for model averaging**

#### Description

Akaike weights are calculated based on the relative expected Kullback-Leibler information as specified by Burnham and Anderson (2004).

#### Usage

```r
aw(object, ...)
```

#### Arguments

- `object`: An `mmkin` column object, containing two or more `mkinfit` models that have been fitted to the same data, or an `mkinfit` object. In the latter case, further `mkinfit` objects fitted to the same data should be specified as dots arguments.
- `...`: Not used in the method for `mmkin` column objects, further `mkinfit` objects in the method for `mkinfit` objects.

#### References


#### Examples

```r
## Not run:
f_sfo <- mkinfit("SFO", FOCUS_2006_D, quiet = TRUE)
f_dfop <- mkinfit("DFOP", FOCUS_2006_D, quiet = TRUE)
aw_sfo_dfop <- aw(f_sfo, f_dfop)
sum(aw_sfo_dfop)
aw_sfo_dfop # SFO gets more weight as it has less parameters and a similar fit
f <- mmkin(c("SFO", "FOMC", "DFOP"), list("FOCUS D" = FOCUS_2006_D), cores = 1, quiet = TRUE)
```
Export a list of datasets format to a CAKE study file

Description
In addition to the datasets, the pathways in the degradation model can be specified as well.

Usage

```r
CAKE_export(
  ds,
  map = c(parent = "Parent"),
  links = NA,
  filename = "CAKE_export.csf",
  path = ".",
  overwrite = FALSE,
  study = "Degradinol aerobic soil degradation",
  description = "",
  time_unit = "days",
  res_unit = "% AR",
  comment = "",
  date = Sys.Date(),
  optimiser = "IRLS"
)
```

Arguments

- **ds**
  A named list of datasets in long format as compatible with `mkinfit`.

- **map**
  A character vector with CAKE compartment names (Parent, A1, ...), named with the names used in the list of datasets.

- **links**
  An optional character vector of target compartments, named with the names of the source compartments. In order to make this easier, the names are used as in the datasets supplied.

- **filename**
  Where to write the result. Should end in `.csf` in order to be compatible with CAKE.

- **path**
  An optional path to the output file.

- **overwrite**
  If TRUE, existing files are overwritten.

- **study**
  The name of the study.

- **description**
  An optional description.
time_unit: The time unit for the residue data.
res_unit: The unit used for the residues.
comment: An optional comment.
date: The date of file creation.
optimiser: Can be OLS or IRLS.

Value
The function is called for its side effect.

Author(s)
Johannes Ranke

confint.mkinfit  Confidence intervals for parameters of mkinfit objects

Description
The default method 'quadratic' is based on the quadratic approximation of the curvature of the likelihood function at the maximum likelihood parameter estimates. The alternative method 'profile' is based on the profile likelihood for each parameter. The 'profile' method uses two nested optimisations and can take a very long time, even if parallelized by specifying 'cores' on unixoid platforms. The speed of the method could likely be improved by using the method of Venzon and Moolgavkar (1988).

Usage
```r
## S3 method for class 'mkinfit'
confint(
  object,
  parm,
  level = 0.95,
  alpha = 1 - level,
  cutoff,
  method = c("quadratic", "profile"),
  transformed = TRUE,
  backtransform = TRUE,
  cores = parallel::detectCores(),
  rel_tol = 0.01,
  quiet = FALSE,
  ...)
```

Arguments

object  
A vector of names of the parameters which are to be given confidence intervals. If missing, all parameters are considered.

parm  
The confidence level required

alpha  
The allowed error probability, overrides 'level' if specified.

cutoff  
Possibility to specify an alternative cutoff for the difference in the log-likelihoods at the confidence boundary. Specifying an explicit cutoff value overrides arguments 'level' and 'alpha'.

method  
The 'quadratic' method approximates the likelihood function at the optimised parameters using the second term of the Taylor expansion, using a second derivative (hessian) contained in the object. The 'profile' method searches the parameter space for the cutoff of the confidence intervals by means of a likelihood ratio test.

transformed  
If the quadratic approximation is used, should it be applied to the likelihood based on the transformed parameters?

backtransform  
If we approximate the likelihood in terms of the transformed parameters, should we backtransform the parameters with their confidence intervals?

cores  
The number of cores to be used for multicore processing. On Windows machines, cores > 1 is currently not supported.

rel_tol  
If the method is 'profile', what should be the accuracy of the lower and upper bounds, relative to the estimate obtained from the quadratic method?

quiet  
Should we suppress the message "Profiling the likelihood"

Value

A matrix with columns giving lower and upper confidence limits for each parameter.

References


Examples

f <- mkinfit("SFO", FOCUS_2006_C, quiet = TRUE)
confint(f, method = "quadratic")

## Not run:
confint(f, method = "profile")
# Set the number of cores for the profiling method for further examples
if (identical(Sys.getenv("NOT_CRAN"), "true")) {
  n_cores <- parallel::detectCores() - 1
} else {
  n_cores <- 1
}
if (Sys.getenv("TRAVIS") != "") n_cores = 1
if (Sys.info()["sysname"] == "Windows") n_cores = 1

SFO_SFO <- mkinmod(parent = mkinsub("SFO", "m1"), m1 = mkinsub("SFO"),
                    use_of_ff = "min", quiet = TRUE)
SFO_SFO.ff <- mkinmod(parent = mkinsub("SFO", "m1"), m1 = mkinsub("SFO"),
                      use_of_ff = "max", quiet = TRUE)
f_d_1 <- mkinfit(SFO_SFO, subset(FOCUS_2006_D, value != 0), quiet = TRUE)

system.time(ci_profile <- confint(f_d_1, method = "profile", cores = 1, quiet = TRUE))
# Using more cores does not save much time here, as parent_0 takes up most of the time
# If we additionally exclude parent_0 (the confidence of which is often of
# minor interest), we get a nice performance improvement if we use at least 4 cores
system.time(ci_profile_no_parent_0 <- confint(f_d_1, method = "profile",
                                             c("k_parent_sink", "k_parent_m1", "k_m1_sink", "sigma"), cores = n_cores))

ci_profile

CI_quadratic_transformed <- confint(f_d_1, method = "quadratic")
CI_quadratic_transformed
CI_quadratic_untransformed <- confint(f_d_1, method = "quadratic", transformed = FALSE)
CI_quadratic_untransformed

# Against the expectation based on Bates and Watts (1988), the confidence
# intervals based on the internal parameter transformation are less
# congruent with the likelihood based intervals. Note the superiority of the
# interval based on the untransformed fit for k_m1_sink
rel_diffs_transformed <- abs((CI_quadratic_transformed_transformed - CI_profile)/CI_profile)
rel_diffs_untransformed <- abs((CI_quadratic_untransformed - CI_profile)/CI_profile)
rel_diffs_transformed < rel_diffs_untransformed

# Investigate a case with formation fractions
f_d_2 <- mkinfit(SFO_SFO.ff, subset(FOCUS_2006_D, value != 0), quiet = TRUE)

CI_profile_ff <- confint(f_d_2, method = "profile", cores = n_cores)
CI_profile_ff

CI_quadratic_transformed_ff <- confint(f_d_2, method = "quadratic")
CI_quadratic_transformed_ff

CI_quadratic_untransformed_ff <- confint(f_d_2, method = "quadratic", transformed = FALSE)
CI_quadratic_untransformed_ff

rel_diffs_transformed_ff <- abs((CI_quadratic_transformed_ff - CI_profile_ff)/CI_profile_ff)
rel_diffs_untransformed_ff <- abs((CI_quadratic_untransformed_ff - CI_profile_ff)/CI_profile_ff)

# While the confidence interval for the parent rate constant is closer to
# the profile based interval when using the internal parameter
# transformation, the interval for the metabolite rate constant is 'better
# without internal parameter transformation.
rel_diffs_transformed_ff < rel_diffs_untransformed_ff
rel_diffs_transformed_ff
rel_diffs_untransformed_ff
# The profiling for the following fit does not finish in a reasonable time,
# therefore we use the quadratic approximation
m_synth_DFOP_par <- mkinmod(parent = mkinsub("DFOP", c("M1", "M2")),
    M1 = mkinsub("SFO"),
    M2 = mkinsub("SFO"),
    use_of_ff = "max", quiet = TRUE)
DFOP_par_c <- synthetic_data_for_UBA_2014[[12]]$data
f_tc_2 <- mkinfit(m_synth_DFOP_par, DFOP_par_c, error_model = "tc",
    error_model_algorithm = "direct", quiet = TRUE)
confint(f_tc_2, method = "quadratic")
confint(f_tc_2, "parent_0", method = "quadratic")

## End(Not run)

---

**create_deg_func**  
Create degradation functions for known analytical solutions

**Description**

Create degradation functions for known analytical solutions

**Usage**

```r
create_deg_func(spec, use_of_ff = c("min", "max"))
```

**Arguments**

- `spec`: List of model specifications as contained in mkinmod objects
- `use_of_ff`: Minimum or maximum use of formation fractions

**Value**

Degradation function to be attached to mkinmod objects

**Examples**

```r
SFO_SFO <- mkinmod(
    parent = mkinsub("SFO", "m1"),
    m1 = mkinsub("SFO"))
FOCUS_D <- subset(FOCUS_2006_D, value != 0) # to avoid warnings
fit_1 <- mkinfit(SFO_SFO, FOCUS_D, solution_type = "analytical", quiet = TRUE)
## Not run:
fit_2 <- mkinfit(SFO_SFO, FOCUS_D, solution_type = "deSolve", quiet = TRUE)
if (require(rbenchmark))
    benchmark(
        analytical = mkinfit(SFO_SFO, FOCUS_D, solution_type = "analytical", quiet = TRUE),
        deSolve = mkinfit(SFO_SFO, FOCUS_D, solution_type = "deSolve", quiet = TRUE),
```
D24_2014

replications = 2)
DFOP_SFO <- mkinmod(
  parent = mkinsub("DFOP", "m1"),
  m1 = mkinsub("SFO"))
benchmark(
  analytical = mkinfit(DFOP_SFO, FOCUS_D, solution_type = "analytical", quiet = TRUE),
  deSolve = mkinfit(DFOP_SFO, FOCUS_D, solution_type = "deSolve", quiet = TRUE),
  replications = 2)

## End(Not run)

---

D24_2014  Aerobic soil degradation data on 2,4-D from the EU assessment in 2014

### Description

The five datasets were extracted from the active substance evaluation dossier published by EFSA. Kinetic evaluations shown for these datasets are intended to illustrate and advance kinetic modelling. The fact that these data and some results are shown here does not imply a license to use them in the context of pesticide registrations, as the use of the data may be constrained by data protection regulations.

### Usage

D24_2014

### Format

An **mkindsg** object grouping five datasets

### Details

Data for the first dataset are from p. 685. Data for the other four datasets were used in the preprocessed versions given in the kinetics section (p. 761ff.), with the exception of residues smaller than 1 for DCP in the soil from Site I2, where the values given on p. 694 were used.

The R code used to create this data object is installed with this package in the `dataset_generation` directory. In the code, page numbers are given for specific pieces of information in the comments.

### Source

Examples

```r
print(D24_2014)
# Not run:
print(D24_2014$ds[[1]], data = TRUE)

m_D24 = mkinmod(D24 = mkinsub("SFO", to = "DCP"),
                 DCP = mkinsub("SFO", to = "DCA"),
                 DCA = mkinsub("SFO"))
print(m_D24)

m_D24_2 = mkinmod(D24 = mkinsub("DFOP", to = "DCP"),
                  DCP = mkinsub("SFO", to = "DCA"),
                  DCA = mkinsub("SFO"))
print(m_D24_2)

# End(Not run)
```

**DFOP.solution**  
*Double First-Order in Parallel kinetics*

**Description**

Function describing decline from a defined starting value using the sum of two exponential decline functions.

**Usage**

```r
DFOP.solution(t, parent_0, k1, k2, g)
```

**Arguments**

- `t`  
  Time.
- `parent_0`  
  Starting value for the response variable at time zero.
- `k1`  
  First kinetic constant.
- `k2`  
  Second kinetic constant.
- `g`  
  Fraction of the starting value declining according to the first kinetic constant.

**Value**

The value of the response variable at time `t`.

**References**

dimethenamid_2018

See Also

Other parent solutions: FOMC.solution(), HS.solution(), IORE.solution(), SFO.solution(), SFORB.solution(), logistic.solution()

Examples

plot(function(x) DFOP.solution(x, 100, 5, 0.5, 0.3), 0, 4, ylim = c(0,100))

dimethenamid_2018  Aerobic soil degradation data on dimethenamid and dimethenamid-P from the EU assessment in 2018

Description

The datasets were extracted from the active substance evaluation dossier published by EFSA. Kinetic evaluations shown for these datasets are intended to illustrate and advance kinetic modelling. The fact that these data and some results are shown here does not imply a license to use them in the context of pesticide registrations, as the use of the data may be constrained by data protection regulations.

Usage

dimethenamid_2018

Format

An mkindsg object grouping seven datasets with some meta information

Details

The R code used to create this data object is installed with this package in the ‘dataset_generation’ directory. In the code, page numbers are given for specific pieces of information in the comments.

Source

Examples

```r
print(dimethenamid_2018)
dmta_ds <- lapply(1:7, function(i) {
  ds_i <- dimethenamid_2018$ds[[i]]$data
  ds_i[ds_i$name == "DMTAP", "name"] <- "DMTA"
  ds_i$time <- ds_i$time * dimethenamid_2018$f_time_norm[i]
  ds_i
})
names(dmta_ds) <- sapply(dimethenamid_2018$ds, function(ds) ds$title)
dmta_ds["Elliot"] <- rbind(dmta_ds["Elliot 1"], dmta_ds["Elliot 2"])
dmta_ds["Elliot 1"] <- NULL
dmta_ds["Elliot 2"] <- NULL
## Not run:
# We don't use DFOP for the parent compound, as this gives numerical
# instabilities in the fits
sfo_sfo3p <- mkinmod(
  DMTA = mkinsub("SFO", c("M23", "M27", "M31")),
  M23 = mkinsub("SFO"),
  M27 = mkinsub("SFO"),
  M31 = mkinsub("SFO", "M27", sink = FALSE),
  quiet = TRUE
)
dmta_sfo_sfo3p_tc <- mmkin(list("SFO-SFO3+" = sfo_sfo3p),
  dmta_ds, error_model = "tc", quiet = TRUE)
print(dmta_sfo_sfo3p_tc)
# The default (test_log_parms = FALSE) gives an undue
# influence of ill-defined rate constants that have
# extremely small values:
plot(mixed(dmta_sfo_sfo3p_tc), test_log_parms = FALSE)
# If we disregards ill-defined rate constants, the results
# look more plausible, but the truth is likely to be in
# between these variants
plot(mixed(dmta_sfo_sfo3p_tc), test_log_parms = TRUE)
# We can also specify a default value for the failing
# log parameters, to mimic FOCUS guidance
plot(mixed(dmta_sfo_sfo3p_tc), test_log_parms = TRUE,
  default_log_parms = log(2)/1000)
# As these attempts are not satisfying, we use nonlinear mixed-effects models
# f_dmta_nlme_tc <- nlme(dmta_sfo_sfo3p_tc)
# nlme reaches maxIter = 50 without convergence
f_dmta_saem_tc <- saem(dmta_sfo_sfo3p_tc)
# I am commenting out the convergence plot as rendering them
# with pkgdown fails (at least without further tweaks to the
# graphics device used)
#saemix::plot(f_dmta_saem_tc$so, plot.type = "convergence")
saemix::plot(f_dmta_saem_tc$so, plot.type = "convergence")
summary(f_dmta_saem_tc)
# As the confidence interval for the random effects of DMTA_0
# includes zero, we could try an alternative model without
# such random effects
# f_dmta_saem_tc_2 <- saem(dmta_sfo_sfo3p_tc,
#  covariance.model = diag(c(0, rep(1, 7))))
# saemix::plot(f_dmta_saem_tc_2$so, plot.type = "convergence")```
# This does not perform better judged by AIC and BIC
# saemix::compare.saemix(f_dmta_saem_tc$so, f_dmta_saem_tc_2$so)
## End(Not run)

### endpoints

**Function to calculate endpoints for further use from kinetic models fitted with mkinfit**

**Description**

This function calculates DT50 and DT90 values as well as formation fractions from kinetic models fitted with mkinfit. If the SFORB model was specified for one of the parents or metabolites, the Eigenvalues are returned. These are equivalent to the rate constants of the DFOP model, but with the advantage that the SFORB model can also be used for metabolites.

**Usage**

```r
endpoints(fit)
```

**Arguments**

- `fit` An object of class `mkinfit`, `nlme.mmkin` or `saem.mmkin`, or another object that has list components `mkinmod` containing an `mkinmod` degradation model, and two numeric vectors, `bparms.optim` and `bparms.fixed`, that contain parameter values for that model.

**Details**

Additional DT50 values are calculated from the FOMC DT90 and k1 and k2 from HS and DFOP, as well as from Eigenvalues b1 and b2 of any SFORB models.

**Value**

A list with a matrix of dissipation times named `distimes`, and, if applicable, a vector of formation fractions named `ff` and, if the SFORB model was in use, a vector of eigenvalues of these SFORB models, equivalent to DFOP rate constants.

**Note**

The function is used internally by `summary.mkinfit`, `summary.nlme.mmkin` and `summary.saem.mmkin`.

**Author(s)**

Johannes Ranke
Examples

```r
fit <- mkinfit("FOMC", FOCUS_2006_C, quiet = TRUE)
endpoints(fit)
## Not run:
fit_2 <- mkinfit("DFOP", FOCUS_2006_C, quiet = TRUE)
endpoints(fit_2)
fit_3 <- mkinfit("SFORB", FOCUS_2006_C, quiet = TRUE)
endpoints(fit_3)
## End(Not run)
```

---

**Experimental datasets used for development and testing of error models**

---

**Description**

The 12 datasets were extracted from active substance evaluation dossiers published by EFSA. Kinetic evaluations shown for these datasets are intended to illustrate and advance error model specifications. The fact that these data and some results are shown here do not imply a license to use them in the context of pesticide registrations, as the use of the data may be constrained by data protection regulations.

Preprocessing of data was performed based on the recommendations of the FOCUS kinetics workgroup (FOCUS, 2014) as described below.

Datasets 1 and 2 are from the Renewal Assessment Report (RAR) for imazamox (France, 2015, p. 15). For setting values reported as zero, an LOQ of 0.1 was assumed. Metabolite residues reported for day zero were added to the parent compound residues.

Datasets 3 and 4 are from the Renewal Assessment Report (RAR) for isofetamid (Belgium, 2014, p. 8) and show the data for two different radiolabels. For dataset 4, the value given for the metabolite in the day zero sampling in replicate B was added to the parent compound, following the respective FOCUS recommendation.

Dataset 5 is from the Renewal Assessment Report (RAR) for ethofumesate (Austria, 2015, p. 16).

Datasets 6 to 10 are from the Renewal Assessment Report (RAR) for glyphosate (Germany, 2013, pages 8, 28, 50, 51). For the initial sampling, the residues given for the metabolite were added to the parent value, following the recommendation of the FOCUS kinetics workgroup.

Dataset 11 is from the Renewal Assessment Report (RAR) for 2,4-D (Hellas, 2013, p. 644). Values reported as zero were set to NA, with the exception of the day three sampling of metabolite A2, which was set to one half of the LOD reported to be 1% AR.

Dataset 12 is from the Renewal Assessment Report (RAR) for thifensulfuron-methyl (United Kingdom, 2014, p. 81).
Usage

`experimental_data_for_UBA_2019`

Format

A list containing twelve datasets as an R6 class defined by `mkind`, each containing, among others, the following components:

- `title` The name of the dataset, e.g. `Soil 1`
- `data` A data frame with the data in the form expected by `mkinfit`

Source

Germany (2013). Renewal Assessment Report Glyphosate Volume 3 Annex B.8: Environmental Fate and Behaviour
Hellas (2013). Renewal Assessment Report 2,4-D Volume 3 Annex B.8: Fate and behaviour in the environment
Ranke (2019) Documentation of results obtained for the error model expertise written for the German Umweltbundesamt.

Examples

```r
## Not run:
# Model definitions
sfo_sfo <- mkinmod(
  parent = mkinsub("SFO", to = "A1"),
  A1 = mkinsub("SFO"),
  use_of_ff = "max"
)
dfop_sfo <- mkinmod(
  parent = mkinsub("DFOP", to = "A1"),
  A1 = mkinsub("SFO"),
  use_of_ff = "max"
)
sfo_sfo_sfo <- mkinmod(
```
FOCUS_2006_datasets

Datasets A to F from the FOCUS Kinetics report from 2006

Description


Usage

FOCUS_2006_A
FOCUS_2006_B
FOCUS_2006_C
FOCUS_2006_D
FOCUS_2006_E
FOCUS_2006_F

Format

6 datasets with observations on the following variables.

name  a factor containing the name of the observed variable
time  a numeric vector containing time points
value  a numeric vector containing concentrations in percent of applied radioactivity

```r
parent = mkinsub("SFO", to = "A1"),
A1 = mkinsub("SFO", to = "A2"),
A2 = mkinsub("SFO"),
use_of_ff = "max"
)
dfop_sfo_sfo <- mkinmod(
    parent = mkinsub("DFOP", to = "A1"),
    A1 = mkinsub("SFO", to = "A2"),
    A2 = mkinsub("SFO"),
    use_of_ff = "max"
)
d_1_2 <- lapply(experimental_data_for_UBA_2019[1:2], function(x) x$data)
names(d_1_2) <- paste("Soil", 1:2)

f_1_2_tc <- mmkin(list("DFOP-SFO-SFO" = dfop_sfo_sfo), d_1_2, error_model = "tc")
plot(f_1_2_tc, resplot = "errmod")

## End(Not run)
```
FOCUS_2006_DFOP_ref_A_to_B

Source


Examples

- FOCUS_2006_C

Results of fitting the DFOP model to Datasets A to B of FOCUS (2006)

Description

A table with the fitted parameters and the resulting DT50 and DT90 values generated with different software packages. Taken directly from FOCUS (2006). The results from fitting the data with the Topfit software was removed, as the initial concentration of the parent compound was fixed to a value of 100 in this fit.

Usage

- FOCUS_2006_DFOP_ref_A_to_B

Format

A data frame containing the following variables.

- package: a factor giving the name of the software package
- M0: The fitted initial concentration of the parent compound
- f: The fitted f parameter
- k1: The fitted k1 parameter
- k2: The fitted k2 parameter
- DT50: The resulting half-life of the parent compound
- DT90: The resulting DT90 of the parent compound
- dataset: The FOCUS dataset that was used

Source


Examples

- data(FOCUS_2006_DFOP_ref_A_to_B)
Results of fitting the FOMC model to Datasets A to F of FOCUS (2006)

Description

A table with the fitted parameters and the resulting DT50 and DT90 values generated with different software packages. Taken directly from FOCUS (2006). The results from fitting the data with the Topfit software was removed, as the initial concentration of the parent compound was fixed to a value of 100 in this fit.

Usage

FOCUS_2006_FOMC_ref_A_to_F

Format

A data frame containing the following variables.

package  a factor giving the name of the software package
M0     The fitted initial concentration of the parent compound
alpha  The fitted alpha parameter
beta   The fitted beta parameter
DT50   The resulting half-life of the parent compound
DT90   The resulting DT90 of the parent compound
dataset The FOCUS dataset that was used

Source


Examples

data(FOCUS_2006_FOMC_ref_A_to_F)
Results of fitting the HS model to Datasets A to F of FOCUS (2006)

Description

A table with the fitted parameters and the resulting DT50 and DT90 values generated with different software packages. Taken directly from FOCUS (2006). The results from fitting the data with the Topfit software was removed, as the initial concentration of the parent compound was fixed to a value of 100 in this fit.

Usage

FOCUS_2006_HS_ref_A_to_F

Format

A data frame containing the following variables.

- package: a factor giving the name of the software package
- M0: The fitted initial concentration of the parent compound
- tb: The fitted tb parameter
- k1: The fitted k1 parameter
- k2: The fitted k2 parameter
- DT50: The resulting half-life of the parent compound
- DT90: The resulting DT90 of the parent compound
- dataset: The FOCUS dataset that was used

Source


Examples

data(FOCUS_2006_HS_ref_A_to_F)
Results of fitting the SFO model to Datasets A to F of FOCUS (2006)

Description

A table with the fitted parameters and the resulting DT50 and DT90 values generated with different software packages. Taken directly from FOCUS (2006). The results from fitting the data with the Topfit software was removed, as the initial concentration of the parent compound was fixed to a value of 100 in this fit.

Usage

FOCUS_2006_SFO_ref_A_to_F

Format

A data frame containing the following variables.

- package  a factor giving the name of the software package
- M0 The fitted initial concentration of the parent compound
- k The fitted first-order degradation rate constant
- DT50 The resulting half-life of the parent compound
- DT90 The resulting DT90 of the parent compound
- dataset The FOCUS dataset that was used

Source


Examples

```r
data(FOCUS_2006_SFO_ref_A_to_F)
```
**focus_soil_moisture**

**FOCUS default values for soil moisture contents at field capacity, MWHC and 1/3 bar**

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
</table>

**Examples**

```
focus_soil_moisture
```
Details

The form given here differs slightly from the original reference by Gustafson and Holden (1990). The parameter beta corresponds to 1/beta in the original equation.

Value

The value of the response variable at time t.

Note

The solution of the FOMC kinetic model reduces to the SFO.solution for large values of alpha and beta with \( k = \frac{\beta}{\alpha} \).

References


See Also

Other parent solutions: DFOP.solution(), HS.solution(), IORE.solution(), SFO.solution(), SFORB.solution(), logistic.solution()

Examples

plot(function(x) FOMC.solution(x, 100, 10, 2), 0, 2, ylim = c(0, 100))

Description

Time step normalisation factors for aerobic soil degradation as described in Appendix 8 to the FOCUS kinetics guidance (FOCUS 2014, p. 369).
Usage

f_time_norm_focus(object, ...)

## S3 method for class 'numeric'
f_time_norm_focus(
  object,
  moisture = NA,
  field_moisture = NA,
  temperature = object,
  Q10 = 2.58,
  walker = 0.7,
  f_na = NA,
  ...
)

## S3 method for class 'mkindsg'
f_time_norm_focus(
  object,
  study_moisture_ref_source = c("auto", "meta", "focus"),
  Q10 = 2.58,
  walker = 0.7,
  f_na = NA,
  ...
)

Arguments

- **object** An object containing information used for the calculations
- **...** Currently not used
- **moisture** Numeric vector of moisture contents in % w/w
- **field_moisture** Numeric vector of moisture contents at field capacity (pF2) in % w/w
- **temperature** Numeric vector of temperatures in °C
- **Q10** The Q10 value used for temperature normalisation
- **walker** The Walker exponent used for moisture normalisation
- **f_na** The factor to use for NA values. If set to NA, only factors for complete cases will be returned.
- **study_moisture_ref_source** Source for the reference value used to calculate the study moisture. If 'auto', preference is given to a reference moisture given in the meta information, otherwise the focus soil moisture for the soil class is used

References

See Also

focus_soil_moisture

Examples

f_time_norm_focus(25, 20, 25) # 1.37, compare FOCUS 2014 p. 184

D24_2014$meta
# No moisture normalisation in the first dataset, so we use f_na = 1 to get
# temperature only normalisation as in the EU evaluation
f_time_norm_focus(D24_2014, study_moisture_ref_source = "focus", f_na = 1)

get_deg_func

Retrieve a degradation function from the mmkin namespace

Description

Retrieve a degradation function from the mmkin namespace

Usage

get_deg_func()

Value

A function that was likely previously assigned from within nlme.mmkin

HS.solution

Hockey-Stick kinetics

Description

Function describing two exponential decline functions with a break point between them.

Usage

HS.solution(t, parent_0, k1, k2, tb)
Arguments

- **t**: Time.
- **parent_0**: Starting value for the response variable at time zero.
- **k1**: First kinetic constant.
- **k2**: Second kinetic constant.
- **tb**: Break point. Before this time, exponential decline according to \(k1\) is calculated, after this time, exponential decline proceeds according to \(k2\).

Value

The value of the response variable at time \(t\).

References


See Also

Other parent solutions: `DFOP.solution()`, `FOMC.solution()`, `IORE.solution()`, `SFO.solution()`, `SFORB.solution()`, `logistic.solution()`

Examples

```r
plot(function(x) HS.solution(x, 100, 2, 0.3, 0.5), 0, 2, ylim=c(0,100))
```

---

**ilr**

*Function to perform isometric log-ratio transformation*

Description

This implementation is a special case of the class of isometric log-ratio transformations.

Usage

```r
ilr(x)
invilr(x)
```
Arguments

- A numeric vector. Naturally, the forward transformation is only sensible for vectors with all elements being greater than zero.

Value

The result of the forward or backward transformation. The returned components always sum to 1 for the case of the inverse log-ratio transformation.

Author(s)

René Lehmann and Johannes Ranke

References


See Also

Another implementation can be found in R package robCompositions.

Examples

```r
# Order matters
ilr(c(0.1, 1, 10))
ilr(c(10, 1, 0.1))
# Equal entries give ilr transformations with zeros as elements
ilr(c(3, 3, 3))
# Almost equal entries give small numbers
ilr(c(0.3, 0.4, 0.3))
# Only the ratio between the numbers counts, not their sum
invilr(ilr(c(0.7, 0.29, 0.01)))
invilr(ilr(2.1 * c(0.7, 0.29, 0.01)))
# Inverse transformation of larger numbers gives unequal elements
invilr(-10)
invilr(c(-10, 0))
# The sum of the elements of the inverse ilr is 1
sum(invilr(c(-10, 0))))
# This is why we do not need all elements of the inverse transformation to go back:
a <- c(0.1, 0.3, 0.5)
b <- invilr(a)
length(b) # Four elements
ilr(c(b[1:3], 1 - sum(b[1:3]))) # Gives c(0.1, 0.3, 0.5)
```
Description

Confidence intervals for parameters in saem.mmkin objects

Usage

## S3 method for class 'saem.mmkin'
intervals(object, level = 0.95, backtransform = TRUE, ...)

Arguments

- **object**: The fitted saem.mmkin object
- **level**: The confidence level. Must be the default of 0.95 as this is what is available in the saemix object
- **backtransform**: In case the model was fitted with mkin transformations, should we backtransform the parameters where a one to one correlation between transformed and backtransformed parameters exists?
- **...**: For compatibility with the generic method

Value

An object with 'intervals.saem.mmkin' and 'intervals.lme' in the class attribute

IORE.solution

Indeterminate order rate equation kinetics

Description

Function describing exponential decline from a defined starting value, with a concentration dependent rate constant.

Usage

IORE.solution(t, parent_0, k__iore, N)

Arguments

- **t**: Time.
- **parent_0**: Starting value for the response variable at time zero.
- **k__iore**: Rate constant. Note that this depends on the concentration units used.
- **N**: Exponent describing the nonlinearity of the rate equation
Value

The value of the response variable at time \( t \).

Note

The solution of the IORE kinetic model reduces to the \texttt{SFO.solution} if \( N = 1 \). The parameters of the IORE model can be transformed to equivalent parameters of the FOMC mode - see the NAFTA guidance for details.

References


See Also

Other parent solutions: \texttt{DFOP.solution()}, \texttt{FOMC.solution()}, \texttt{HS.solution()}, \texttt{SFO.solution()}, \texttt{SFORB.solution()}, \texttt{logistic.solution()}

Examples

```r
plot(function(x) IORE.solution(x, 100, 0.2, 1.3), 0, 2, ylim = c(0, 100))
## Not run:
fit.fomc <- mkinfit("FOMC", FOCUS_2006_C, quiet = TRUE)
fit.iore <- mkinfit("IORE", FOCUS_2006_C, quiet = TRUE)
fit.iore.deS <- mkinfit("IORE", FOCUS_2006_C, solution_type = "deSolve", quiet = TRUE)

print(data.frame(fit.fomc$par, fit.iore$par, fit.iore.deS$par,
    row.names = paste("model par", 1:4)))
print(rbind(fomc = endpoints(fit.fomc)$distimes, iore = endpoints(fit.iore)$distimes,
    iore.deS = endpoints(fit.iore.deS$distimes))

## End(Not run)
```

Description

This is a generic function with a method currently only defined for mkinfit objects. It fits an anova model to the data contained in the object and compares the likelihoods using the likelihood ratio test \texttt{lrtest.default} from the lmtest package.
Usage

loftest(object, ...)

## S3 method for class 'mkinfit'
loftest(object, ...)

Arguments

object A model object with a defined loftest method
...
Not used

Details

The anova model is interpreted as the simplest form of an mkinfit model, assuming only a constant variance about the means, but not enforcing any structure of the means, so we have one model parameter for every mean of replicate samples.

See Also

lrtest

Examples

## Not run:
test_data <- subset(synthetic_data_for_UBA_2014[[12]]$data, name == "parent")
sfo_fit <- mkinfit("SFO", test_data, quiet = TRUE)
plot_res(sfo_fit) # We see a clear pattern in the residuals
loftest(sfo_fit) # We have a clear lack of fit

# We try a different model (the one that was used to generate the data)
dfop_fit <- mkinfit("DFOP", test_data, quiet = TRUE)
plot_res(dfop_fit) # We don't see systematic deviations, but heteroscedastic residuals
# therefore we should consider adapting the error model, although we have
loftest(dfop_fit) # no lack of fit

# This is the anova model used internally for the comparison
test_data_anova <- test_data
test_data_anova$time <- as.factor(test_data_anova$time)
anova_fit <- lm(value ~ time, data = test_data_anova)
summary(anova_fit)
logLik(anova_fit) # We get the same likelihood and degrees of freedom

# test_data_2 <- synthetic_data_for_UBA_2014[[12]]$data
m_synth_SFO_lin <- mkinmod(parent = list(type = "SFO", to = "M1"),
M1 = list(type = "SFO", to = "M2"),
M2 = list(type = "SFO"), use_of_ff = "max")
sfo_lin_fit <- mkinfit(m_synth_SFO_lin, test_data_2, quiet = TRUE)
plot_res(sfo_lin_fit) # not a good model, we try parallel formation
loftest(sfo_lin_fit)

# m_synth_SFO_par <- mkinmod(parent = list(type = "SFO", to = c("M1", "M2")),


logistic.solution

Logistic kinetics

Description

Function describing exponential decline from a defined starting value, with an increasing rate constant, supposedly caused by microbial growth

Usage

logistic.solution(t, parent_0, kmax, k0, r)

Arguments

t Time.
parent_0 Starting value for the response variable at time zero.
kmax Maximum rate constant.
k0 Minimum rate constant effective at time zero.
r Growth rate of the increase in the rate constant.

Value

The value of the response variable at time t.

Note

The solution of the logistic model reduces to the SF0.solution if k0 is equal to kmax.
**logistic.solution**

**References**


**See Also**

Other parent solutions: DFOP.solution(), FOMC.solution(), HS.solution(), IORE.solution(), SFO.solution(), SFORB.solution()

**Examples**

```r
# Reproduce the plot on page 57 of FOCUS (2014)
plot(function(x) logistic.solution(x, 100, 0.08, 0.0001, 0.2),
     from = 0, to = 100, ylim = c(0, 100),
     xlab = "Time", ylab = "Residue")
plot(function(x) logistic.solution(x, 100, 0.08, 0.0001, 0.4),
     from = 0, to = 100, add = TRUE, lty = 2, col = 2)
plot(function(x) logistic.solution(x, 100, 0.08, 0.0001, 0.8),
     from = 0, to = 100, add = TRUE, lty = 3, col = 3)
plot(function(x) logistic.solution(x, 100, 0.08, 0.0001, 0.2),
     from = 0, to = 100, add = TRUE, lty = 4, col = 4)
plot(function(x) logistic.solution(x, 100, 0.08, 0.0001, 0.8),
     from = 0, to = 100, add = TRUE, lty = 5, col = 5)
legend("topright", inset = 0.05,
       legend = paste0("k0 = ", c(0.0001, 0.0001, 0.0001, 0.001, 0.08),
                    " r = ", c(0.2, 0.4, 0.8, 0.2, 0.2)),
       lty = 1:5, col = 1:5)

# Fit with synthetic data
logistic <- mkinmod(parent = mkinsub("logistic"))
sampling_times <- c(0, 1, 3, 7, 14, 28, 60, 90, 120)
parms_logistic <- c(kmax = 0.08, k0 = 0.0001, r = 0.2)
d_logistic <- mkinpredict(logistic, parms_logistic, c(parent = 100), sampling_times)
d_2_1 <- add_err(d_logistic, sdfunc = function(x) sigma_twocomp(x, 0.5, 0.07),
                     n = 1, reps = 2, digits = 5, LOD = 0.1, seed = 123456)[[1]]
m <- mkinfit("logistic", d_2_1, quiet = TRUE)
plot_sep(m)
summary(m)$bpar
endpoints(m)$distimes
```
logLik.mkinfit

Calculated the log-likelihood of a fitted mkinfit object

Description

This function returns the product of the likelihood densities of each observed value, as calculated as part of the fitting procedure using \texttt{dnorm}, i.e. assuming normal distribution, and with the means predicted by the degradation model, and the standard deviations predicted by the error model.

Usage

\begin{verbatim}
## S3 method for class 'mkinfit'
logLik(object, ...)
\end{verbatim}

Arguments

\begin{itemize}
  \item \textbf{object} \hspace{1cm} An object of class \texttt{mkinfit}.
  \item \textbf{...} \hspace{1cm} For compatibility with the generic method
\end{itemize}

Details

The total number of estimated parameters returned with the value of the likelihood is calculated as the sum of fitted degradation model parameters and the fitted error model parameters.

Value

An object of class \texttt{logLik} with the number of estimated parameters (degradation model parameters plus variance model parameters) as attribute.

Author(s)

Johannes Ranke

See Also

Compare the AIC of columns of \texttt{mmkin} objects using \texttt{AIC.mmkin}.

Examples

\begin{verbatim}
## Not run:
sfo_sfo <- mkinmod(
  parent = mkinsub("SF0", to = "m1"),
  m1 = mkinsub("SF0")
)
d_t <- subset(FOCUS_2006_D, value != 0)
f_nw <- mkinfit(sfo_sfo, d_t, quiet = TRUE) # no weighting (weights are unity)
f_obs <- update(f_nw, error_model = "obs")
\end{verbatim}
lrtest.mkinfit

## Description

Compare two mkinfit models based on their likelihood. If two fitted mkinfit objects are given as arguments, it is checked if they have been fitted to the same data. It is the responsibility of the user to make sure that the models are nested, i.e. one of them has less degrees of freedom and can be expressed by fixing the parameters of the other.

## Usage

```r
## S3 method for class 'mkinfit'
lrtest(object, object_2 = NULL, ...)
## S3 method for class 'mmkin'
lrtest(object, ...)
```

## Arguments

- **object**: An `mkinfit` object, or an `mmkin` column object containing two fits to the same data.
- **object_2**: Optionally, another mkinfit object fitted to the same data.
- **...**: Argument to `mkinfit`, passed to `update.mkinfit` for creating the alternative fitted object.

## Details

Alternatively, an argument to mkinfit can be given which is then passed to `update.mkinfit` to obtain the alternative model.

The comparison is then made by the `lrtest.default` method from the lmtest package. The model with the higher number of fitted parameters (alternative hypothesis) is listed first, then the model with the lower number of fitted parameters (null hypothesis).

## Examples

```r
## Not run:
test_data <- subset(synthetic_data_for_UBA_2014[[12]]$data, name == "parent")
sfo_fit <- mkinfit("SFO", test_data, quiet = TRUE)
sfo_fit <- update(sfo_fit, error_model = "tc")
AIC(sfo_fit, test_data, quiet = TRUE)
```

```r
dfop_fit <- mkinfit("DFOP", test_data, quiet = TRUE)
lrtest(dfop_fit, sfo_fit)
```
lrtest(sfo_fit, dfop_fit)

# The following two examples are commented out as they fail during
generation of the static help pages by pkgdown
#lrtest(dfop_fit, error_model = "tc")
#lrtest(dfop_fit, fixedParms = c(k2 = 0))

# However, this equivalent syntax also works for static help pages
lrtest(dfop_fit, update(dfop_fit, error_model = "tc"))
lrtest(dfop_fit, update(dfop_fit, fixedParms = c(k2 = 0)))

## End(Not run)

---

max_twa_parent

Function to calculate maximum time weighted average concentrations
from kinetic models fitted with mkinfit

Description

This function calculates maximum moving window time weighted average concentrations (TWAs) for kinetic models fitted with mkinfit. Currently, only calculations for the parent are implemented for the SFO, FOMC, DFOP and HS models, using the analytical formulas given in the PEC soil section of the FOCUS guidance.

Usage

max_twa_parent(fit, windows)

max_twa_sfo(M0 = 1, k, t)

max_twa_fomc(M0 = 1, alpha, beta, t)

max_twa_dfop(M0 = 1, k1, k2, g, t)

max_twa_hs(M0 = 1, k1, k2, tb, t)

Arguments

- **fit**: An object of class mkinfit.
- **windows**: The width of the time windows for which the TWAs should be calculated.
- **M0**: The initial concentration for which the maximum time weighted average over the decline curve should be calculated. The default is to use a value of 1, which means that a relative maximum time weighted average factor (f_twa) is calculated.
- **k**: The rate constant in the case of SFO kinetics.
- **t**: The width of the time window.
- **alpha**: Parameter of the FOMC model.
mccall81_245T

beta Parameter of the FOMC model.
k1 The first rate constant of the DFOP or the HS kinetics.
k2 The second rate constant of the DFOP or the HS kinetics.
g Parameter of the DFOP model.
tb Parameter of the HS model.

Value

For max_twa_parent, a numeric vector, named using the windows argument. For the other functions, a numeric vector of length one (also known as ’a number’).

Author(s)

Johannes Ranke

References


Examples

```r
fit <- mkinfit("FOMC", FOCUS_2006_C, quiet = TRUE)
max_twa_parent(fit, c(7, 21))
```

mccall81_245T Datasets on aerobic soil metabolism of 2,4,5-T in six soils

Description

Time course of 2,4,5-trichlorophenoxyacetic acid, and the corresponding 2,4,5-trichlorophenol and 2,4,5-trichloroanisole as recovered in diethylether extracts.

Usage

mccall81_245T
Format

A dataframe containing the following variables.

- `name`: the name of the compound observed. Note that T245 is used as an acronym for 2,4,5-T. T245 is a legitimate object name in R, which is necessary for specifying models using `mkinmod`.
- `time`: a numeric vector containing sampling times in days after treatment.
- `value`: a numeric vector containing concentrations in percent of applied radioactivity.
- `soil`: a factor containing the name of the soil.

Source


Examples

```r
SFO_SFO_SFO <- mkinmod(T245 = list(type = "SFO", to = "phenol"),
phenol = list(type = "SFO", to = "anisole"),
anisole = list(type = "SFO"))
## Not run:
fit.1 <- mkinfit(SFO_SFO_SFO, subset(mccall81_245T, soil == "Commerce"), quiet = TRUE)
summary(fit.1)$bpar
endpoints(fit.1)
# formation fraction from phenol to anisol is practically 1. As we cannot
# fix formation fractions when using the ilr transformation, we can turn off
# the sink in the model generation
SFO_SFO_SFO_2 <- mkinmod(T245 = list(type = "SFO", to = "phenol"),
phenol = list(type = "SFO", to = "anisole", sink = FALSE),
anisole = list(type = "SFO"))
fit.2 <- mkinfit(SFO_SFO_SFO_2, subset(mccall81_245T, soil == "Commerce"),
quiet = TRUE)
summary(fit.2)$bpar
endpoints(fit.1)
plot_sep(fit.2)
## End(Not run)
```

mean_degparms

### Calculate mean degradation parameters for an mkinmod row object

**Description**

Calculate mean degradation parameters for an mkinmod row object.
Usage

mean_degsparms(
  object,
  random = FALSE,
  test_logParms = FALSE,
  conf.level = 0.6,
  default_logParms = NA
)

Arguments

object An mmkin row object containing several fits of the same model to different datasets
random Should a list with fixed and random effects be returned?
test_logParms If TRUE, log parameters are only considered in the mean calculations if their untransformed counterparts (most likely rate constants) pass the t-test for significant difference from zero.
conf.level Possibility to adjust the required confidence level for parameter that are tested if requested by 'test_logParms'.
default_logParms If set to a numeric value, this is used as a default value for the tested log parameters that failed the t-test.

Value

If random is FALSE (default), a named vector containing mean values of the fitted degradation model parameters. If random is TRUE, a list with fixed and random effects, in the format required by the start argument of nlme for the case of a single grouping variable ds.

mixed

Create a mixed effects model from an mmkin row object

Description

Create a mixed effects model from an mmkin row object

Usage

mixed(object, ...)

## S3 method for class 'mmkin'
mixed(object, method = c("none"), ...)

## S3 method for class 'mixed.mmkin'
print(x, digits = max(3,getOption("digits") - 3), ...)
Arguments

object  An \texttt{mmkin} row object
...  Currently not used
method  The method to be used
x  A \texttt{mixed.mmkin} object to print
digits  Number of digits to use for printing.

Value

An object of class \texttt{'mixed.mmkin'} which has the observed data in a single dataframe which is convenient for plotting.

Examples

```r
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)
n_biphasic <- 8
err_1 = list(const = 1, prop = 0.07)

DFOP_SFO <- mkinmod(
  parent = mkinsub("DFOP", "m1"),
  m1 = mkinsub("SFO"),
  quiet = TRUE)

set.seed(123456)
log_sd <- 0.3
syn_biphasic_parms <- as.matrix(data.frame(
  k1 = rlnorm(n_biphasic, log(0.05), log_sd),
  k2 = rlnorm(n_biphasic, log(0.01), log_sd),
  g = plogis(rnorm(n_biphasic, 0, log_sd)),
  f_parent_to_m1 = plogis(rnorm(n_biphasic, 0, log_sd)),
  k_m1 = rlnorm(n_biphasic, log(0.002), log_sd)))

ds_biphasic_mean <- lapply(1:n_biphasic,
  function(i) {
    mkinpredict(DFOP_SFO, syn_biphasic_parms[i, ],
      c(parent = 100, m1 = 0), sampling_times)
  })

set.seed(123456L)
ds_biphasic <- lapply(ds_biphasic_mean, function(ds) {
  add_err(ds,
    sdfunc = function(value) sqrt(err_1$const^2 + value^2 * err_1$prop^2),
    n = 1, secondary = "m1")[[1]]
})

## Not run:
f_mmkin <- mmkin(list("DFOP-SFO" = DFOP_SFO), ds_biphasic, error_model = "tc", quiet = TRUE)
f_mixed <- mixed(f_mmkin)
```
**Description**

At the moment this dataset class is hardly used in mkin. For example, mkinfit does not take mkinds datasets as argument, but works with dataframes such as the one contained in the data field of mkinds objects. Some datasets provided by this package come as mkinds objects nevertheless.

**Usage**

```r
## S3 method for class 'mkinds'
print(x, data = FALSE, ...)
```

**Arguments**

- `x` An `mkinds` object.
- `data` Should the data be printed?
- `...` Not used.

**Public fields**

- `title` A full title for the dataset
- `sampling_times` The sampling times
- `time_unit` The time unit
- `observed` Names of the observed variables
- `unit` The unit of the observations
- `replicates` The maximum number of replicates per sampling time
- `data` A data frame with at least the columns name, time and value in order to be compatible with mkinfit

**Methods**

**Public methods:**

- `mkinds$new()`
- `mkinds$clone()`

**Method `new()`**: Create a new mkinds object

**Usage:**

```r
mkinds$new(title = "", data, time_unit = NA, unit = NA)
```
Arguments:
title  The dataset title
data  The data
time_unit  The time unit
unit  The unit of the observations

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

Usage:
mkinds$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples

mds <- mkinds$new("FOCUS A", FOCUS_2006_A)
print(mds)

Description

A container for working with datasets that share at least one compound, so that combined evaluations are desirable.

Time normalisation factors are initialised with a value of 1 for each dataset if no data are supplied.

Usage

## S3 method for class 'mkinds'
print(x, data = FALSE, verbose = data, ...)

Arguments

x  An mkinds object.
data  Should the mkinds objects be printed with their data?
verbose  Should the mkinds objects be printed?
...  Not used.
Public fields

- **title**: A title for the dataset group
- **ds**: A list of mkinds objects
- **observed_n**: Occurrence counts of compounds in datasets
- **f_time_norm**: Time normalisation factors
- **meta**: A data frame with a row for each dataset, containing additional information in the form of categorical data (factors) or numerical data (e.g., temperature, moisture, or covariates like soil pH).

Methods

**Public methods:**

- `mkindsg$new()`
- `mkindsg$clone()`

**Method new():** Create a new mkindsg object

*Usage:*

```
mkindsg$new(title = "", ds, f_time_norm = rep(1, length(ds)), meta)
```

*Arguments:*

- `title`: The title
- `ds`: A list of mkinds objects
- `f_time_norm`: Time normalisation factors
- `meta`: The meta data

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

*Usage:*

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

*Arguments:*

- `deep`: Whether to make a deep clone.

Examples

```
mdsg <- mkindsg$new("Experimental X", experimental_data_for_UBA_2019[6:10])
print(mdsg)
print(mdsg, verbose = TRUE)
print(mdsg, verbose = TRUE, data = TRUE)
```
Calculate the minimum error to assume in order to pass the variance test

Description

This function finds the smallest relative error still resulting in passing the chi-squared test as defined in the FOCUS kinetics report from 2006.

Usage

mkinerrmin(fit, alpha = 0.05)

Arguments

- **fit**: an object of class `mkinfit`.
- **alpha**: The confidence level chosen for the chi-squared test.

Details

This function is used internally by `summary.mkinfit`.

Value

A dataframe with the following components:

- **err.min**: The relative error, expressed as a fraction.
- **n.optim**: The number of optimised parameters attributed to the data series.
- **df**: The number of remaining degrees of freedom for the chi2 error level calculations. Note that mean values are used for the chi2 statistic and therefore every time point with observed values in the series only counts one time.

The dataframe has one row for the total dataset and one further row for each observed state variable in the model.

References

Examples

```r
SFO_SFO = mkinmod(parent = mkinsub("SFO", to = "m1"),
                   m1 = mkinsub("SFO"),
                   use_of_ff = "max")

fit_FOCUS_D = mkinfit(SFO_SFO, FOCUS_2006_D, quiet = TRUE)
round(mkinerrmin(fit_FOCUS_D), 4)
## Not run:
fit_FOCUS_E = mkinfit(SFO_SFO, FOCUS_2006_E, quiet = TRUE)
round(mkinerrmin(fit_FOCUS_E), 4)
## End(Not run)
```

Description

This function plots the squared residuals for the specified subset of the observed variables from an mkinfit object. In addition, one or more dashed line(s) show the fitted error model. A combined plot of the fitted model and this error model plot can be obtained with `plot.mkinfit` using the argument `show_errplot = TRUE`.

Usage

```r
mkinerrplot(
  object,
  obs_vars = names(object$mkinmod$map),
  xlim = c(0, 1.1 * max(object$data$predicted)),
  xlab = "Predicted",
  ylab = "Squared residual",
  maxy = "auto",
  legend = TRUE,
  lpos = "topright",
  col_obs = "auto",
  pch_obs = "auto",
  frame = TRUE,
  ...
)
```

Arguments

- **object**: A fit represented in an `mkinfit` object.
- **obs_vars**: A character vector of names of the observed variables for which residuals should be plotted. Defaults to all observed variables in the model.
mkinfit

Fit a kinetic model to data with one or more state variables

Description

This function maximises the likelihood of the observed data using the Port algorithm \texttt{stats::nlminb()}, and the specified initial or fixed parameters and starting values. In each step of the optimisation, the kinetic model is solved using the function \texttt{mkinpredict()}, except if an analytical solution is implemented, in which case the model is solved using the degradation function in the \texttt{mkinmod} object. The parameters of the selected error model are fitted simultaneously with the degradation model parameters, as both of them are arguments of the likelihood function.

### Parameters

- \texttt{xlim} plot range in x direction.
- \texttt{xlab} Label for the x axis.
- \texttt{ylab} Label for the y axis.
- \texttt{maxy} Maximum value of the residuals. This is used for the scaling of the y axis and defaults to "auto".
- \texttt{legend} Should a legend be plotted?
- \texttt{lpos} Where should the legend be placed? Default is "topright". Will be passed on to \texttt{legend}.
- \texttt{col_obs} Colors for the observed variables.
- \texttt{pch_obs} Symbols to be used for the observed variables.
- \texttt{frame} Should a frame be drawn around the plots?
- ... further arguments passed to \texttt{plot}.

### Value

Nothing is returned by this function, as it is called for its side effect, namely to produce a plot.

### Author(s)

Johannes Ranke

### See Also

\texttt{mkinplot}, for a way to plot the data and the fitted lines of the mkinfit object.

### Examples

```r
## Not run:
model <- mkinmod(parent = mkinsub("SFO", "m1"), m1 = mkinsub("SFO"))
fit <- mkinfit(model, FOCUS_2006_D, error_model = "tc", quiet = TRUE)
mkinerrplot(fit)
## End(Not run)
```
mkinfit

Usage

mkinfit(
  mkinmod, 
  observed, 
  parms.ini = "auto", 
  state.ini = "auto", 
  err.ini = "auto", 
  fixed_parms = NULL, 
  fixed_initials = names(mkinmod$diffs)[-1], 
  from_max_mean = FALSE, 
  solution_type = c("auto", "analytical", "eigen", "deSolve"), 
  method.ode = "lsoda", 
  use_compiled = "auto", 
  control = list(eval.max = 300, iter.max = 200), 
  transform_rates = TRUE, 
  transform_fractions = TRUE, 
  quiet = FALSE, 
  atol = 1e-08, 
  rtol = 1e-10, 
  error_model = c("const", "obs", "tc"), 
  error_model_algorithm = c("auto", "d_3", "direct", "twostep", "threestep", 
                           "fourstep", "IRLS", "OLS"), 
  reweight.tol = 1e-08, 
  reweight.max.iter = 10, 
  trace_parms = FALSE, 
  test_residuals = FALSE, 
  ...
)

Arguments

mkinmod A list of class mkinmod, containing the kinetic model to be fitted to the data, 
or one of the shorthand names ("SFO", "FOMC", "DFOP", "HS", "SFORB", 
"IORE"). If a shorthand name is given, a parent only degradation model is 
generated for the variable with the highest value in observed.

observed A dataframe with the observed data. The first column called "name" must con- 
tain the name of the observed variable for each data point. The second column 
must contain the times of observation, named "time". The third column must 
be named "value" and contain the observed values. Zero values in the "value" 
column will be removed, with a warning, in order to avoid problems with fitting 
the two-component error model. This is not expected to be a problem, because 
in general, values of zero are not observed in degradation data, because there is 
a lower limit of detection.

parms.ini A named vector of initial values for the parameters, including parameters to be 
optimised and potentially also fixed parameters as indicated by fixed_parms. 
If set to "auto", initial values for rate constants are set to default values. Using 
parameter names that are not in the model gives an error.
It is possible to only specify a subset of the parameters that the model needs. You can use the parameter lists "bparms.ode" from a previously fitted model, which contains the differential equation parameters from this model. This works nicely if the models are nested. An example is given below.

state.ini A named vector of initial values for the state variables of the model. In case the observed variables are represented by more than one model variable, the names will differ from the names of the observed variables (see map component of mkinmod). The default is to set the initial value of the first model variable to the mean of the time zero values for the variable with the maximum observed value, and all others to 0. If this variable has no time zero observations, its initial value is set to 100.

er.err.ini A named vector of initial values for the error model parameters to be optimised. If set to "auto", initial values are set to default values. Otherwise, initial values for all error model parameters must be given.

fixed_parms The names of parameters that should not be optimised but rather kept at the values specified in parms.ini. Alternatively, a named numeric vector of parameters to be fixed, regardless of the values in parms.ini.

fixed_initials The names of model variables for which the initial state at time 0 should be excluded from the optimisation. Defaults to all state variables except for the first one.

from_max_mean If this is set to TRUE, and the model has only one observed variable, then data before the time of the maximum observed value (after averaging for each sampling time) are discarded, and this time is subtracted from all remaining time values, so the time of the maximum observed mean value is the new time zero.

solution_type If set to "eigen", the solution of the system of differential equations is based on the spectral decomposition of the coefficient matrix in cases that this is possible. If set to "deSolve", a numerical ode solver from package deSolve is used. If set to "analytical", an analytical solution of the model is used. This is only implemented for relatively simple degradation models. The default is "auto", which uses "analytical" if possible, otherwise "deSolve" if a compiler is present, and "eigen" if no compiler is present and the model can be expressed using eigenvalues and eigenvectors.

method.ode The solution method passed via mkinpredict() to deSolve::ode() in case the solution type is "deSolve". The default "lsoda" is performant, but sometimes fails to converge.

use_compiled If set to FALSE, no compiled version of the mkinmod model is used in the calls to mkinpredict() even if a compiled version is present.

control A list of control arguments passed to stats::nlminb().

transform_rates Boolean specifying if kinetic rate constants should be transformed in the model specification used in the fitting for better compliance with the assumption of normal distribution of the estimator. If TRUE, also alpha and beta parameters of the FOMC model are log-transformed, as well as k1 and k2 rate constants for the DFOP and HS models and the break point tb of the HS model. If FALSE, zero is used as a lower bound for the rates in the optimisation.
transform_fractions
Boolean specifying if formation fractions should be transformed in the model specification used in the fitting for better compliance with the assumption of normal distribution of the estimator. The default (TRUE) is to do transformations. If TRUE, the g parameter of the DFOP model is also transformed. Transformations are described in transform_odeparms.

quiet
Suppress printing out the current value of the negative log-likelihood after each improvement?

atol
Absolute error tolerance, passed to deSolve::ode(). Default is 1e-8, which is lower than the default in the deSolve::lsoda() function which is used per default.

rtol
Absolute error tolerance, passed to deSolve::ode(). Default is 1e-10, much lower than in deSolve::lsoda().

error_model
If the error model is "const", a constant standard deviation is assumed. If the error model is "obs", each observed variable is assumed to have its own variance. If the error model is "tc" (two-component error model), a two component error model similar to the one described by Rocke and Lorenzato (1995) is used for setting up the likelihood function. Note that this model deviates from the model by Rocke and Lorenzato, as their model implies that the errors follow a lognormal distribution for large values, not a normal distribution as assumed by this method.

error_model_algorithm
If "auto", the selected algorithm depends on the error model. If the error model is "const", unweighted nonlinear least squares fitting ("OLS") is selected. If the error model is "obs", or "tc", the "d_3" algorithm is selected.
The algorithm "d_3" will directly minimize the negative log-likelihood and independently also use the three step algorithm described below. The fit with the higher likelihood is returned.
The algorithm "direct" will directly minimize the negative log-likelihood.
The algorithm "twostep" will minimize the negative log-likelihood after an initial unweighted least squares optimisation step.
The algorithm "threestep" starts with unweighted least squares, then optimizes only the error model using the degradation model parameters found, and then minimizes the negative log-likelihood with free degradation and error model parameters.
The algorithm "fourstep" starts with unweighted least squares, then optimizes only the error model using the degradation model parameters found, then optimizes the degradation model again with fixed error model parameters, and finally minimizes the negative log-likelihood with free degradation and error model parameters.
The algorithm "IRLS" (Iteratively Reweighted Least Squares) starts with unweighted least squares, and then iterates optimization of the error model parameters and subsequent optimization of the degradation model using those error model parameters, until the error model parameters converge.

reweight.tol
Tolerance for the convergence criterion calculated from the error model parameters in IRLS fits.
mkinfit

- **Reweight.max.iter**
  - Maximum number of iterations in IRLS fits.

- **Trace_parms**
  - Should a trace of the parameter values be listed?

- **Test_residuals**
  - Should the residuals be tested for normal distribution?

- **...**
  - Further arguments that will be passed on to deSolve::ode().

**Details**

Per default, parameters in the kinetic models are internally transformed in order to better satisfy the assumption of a normal distribution of their estimators.

**Value**

A list with "mkinfit" in the class attribute.

**Note**

When using the "IORE" submodel for metabolites, fitting with "transform_rates = TRUE" (the default) often leads to failures of the numerical ODE solver. In this situation it may help to switch off the internal rate transformation.

**Author(s)**

Johannes Ranke

**References**


**See Also**

summary.mkinfit, plot.mkinfit, parms and lrtest.

Comparisons of models fitted to the same data can be made using AIC by virtue of the method logLik.mkinfit.

Fitting of several models to several datasets in a single call to mmkin.

**Examples**

```r
# Use shorthand notation for parent only degradation
fit <- mkinfit("FOMC", FOCUS_2006_C, quiet = TRUE)
summary(fit)

# One parent compound, one metabolite, both single first order.
# We remove zero values from FOCUS dataset D in order to avoid warnings
FOCUS_D <- subset(FOCUS_2006_D, value != 0)
```

# Use mkinsub for convenience in model formulation. Pathway to sink included per default.
SFO_SFO <- mkinmod(
  parent = mkinsub("SFO", "m1"),
  m1 = mkinsub("SFO"))

# Fit the model quietly to the FOCUS example dataset D using defaults
fit <- mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE)
plot_sep(fit)

# As lower parent values appear to have lower variance, we try an alternative error model
fit.tc <- mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE, error_model = "tc")
# This avoids the warning, and the likelihood ratio test confirms it is preferable
lrtest(fit.tc, fit)

# We can also allow for different variances of parent and metabolite as error model
fit.obs <- mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE, error_model = "obs")
# The two-component error model has significantly higher likelihood
lrtest(fit.obs, fit.tc)

# We can show a quick (only one replication) benchmark for this case, as we
# have several alternative solution methods for the model. We skip
# uncompiled deSolve, as it is so slow. More benchmarks are found in the
# benchmark vignette
## Not run:
if(require(rbenchmark)) {
  benchmark(replications = 1, order = "relative", columns = c("test", "relative", "elapsed"),
    deSolve_compiled = mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE, error_model = "tc",
      solution_type = "deSolve", use_compiled = TRUE),
    eigen = mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE, error_model = "tc",
      solution_type = "eigen"),
    analytical = mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE, error_model = "tc",
      solution_type = "analytical"))
}

## Not run:
# Use stepwise fitting, using optimised parameters from parent only fit, FOMC-SFO
## Not run:
FOMC_SFO <- mkinmod(
  parent = mkinsub("FOMC", "m1"),
  m1 = mkinsub("SFO"))
fit.FOMC_SFO <- mkinfit(FOMC_SFO, FOCUS_D, quiet = TRUE)
# Again, we get a warning and try a more sophisticated error model
fit.FOMC_SFO.tc <- mkinfit(FOMC_SFO, FOCUS_D, quiet = TRUE, error_model = "tc")
# This model has a higher likelihood, but not significantly so
lrtest(fit.tc, fit.FOMC_SFO.tc)

# Also, the missing standard error for log_beta and the t-tests for alpha
# and beta indicate overparameterisation
summary(fit.FOMC_SFO.tc, data = FALSE)

# We can easily use starting parameters from the parent only fit (only for illustration)
fit.FOMC = mkinfit("FOMC", FOCUS_2006_D, quiet = TRUE, error_model = "tc")
fit.FOMC_SFO <- mkinfit(FOMC_SFO, FOCUS_D, quiet = TRUE,
Function to set up a kinetic model with one or more state variables

Description
This function is usually called using a call to `mkinsub()` for each observed variable, specifying the corresponding submodel as well as outgoing pathways (see examples).

Print `mkinmod` objects in a way that the user finds his way to get to its components.

Usage

```r
mkinmod(..., use_of_ff = "max", name = NULL, speclist = NULL, quiet = FALSE, verbose = FALSE, dll_dir = NULL, unload = FALSE, overwrite = FALSE)
```

## S3 method for class 'mkinmod'

```r
print(x, ...)
```

```r
mkinsub(submodel, to = NULL, sink = TRUE, full_name = NA)
```

Arguments

... For each observed variable, a list as obtained by `mkinsub()` has to be specified as an argument (see examples). Currently, single first order kinetics "SFO", indeterminate order rate equation kinetics "IORE", or single first order with reversible binding "SFORB" are implemented for all variables, while "FOMC", "DFOP", "HS" and "logistic" can additionally be chosen for the first variable which is assumed to be the source compartment. Additionally, `mkinsub()` has an argument to, specifying names of variables to which a transfer is to be assumed in the model. If the argument `use_of_ff` is set to "min" (default) and the model for the compartment is "SFO" or "SFORB", an additional `mkinsub()` argument can be `sink = FALSE`, effectively fixing the flux to sink to zero. In `print.mkinmod`, this argument is currently not used.
**use_of_ff** Specification of the use of formation fractions in the model equations and, if applicable, the coefficient matrix. If "max", formation fractions are always used (default). If "min", a minimum use of formation fractions is made, i.e. each first-order pathway to a metabolite has its own rate constant.

**name** A name for the model. Should be a valid R object name.

**speclist** The specification of the observed variables and their submodel types and pathways can be given as a single list using this argument. Default is NULL.

**quiet** Should messages be suppressed?

**verbose** If TRUE, passed to `inline::cfunction()` if applicable to give detailed information about the C function being built.

**dll_dir** Directory where an DLL object, if generated internally by `inline::cfunction()`, should be saved. The DLL will only be stored in a permanent location for use in future sessions, if ’dll_dir’ and ’name’ are specified.

**unload** If a DLL from the target location in ’dll_dir’ is already loaded, should that be unloaded first?

**overwrite** If a file exists at the target DLL location in ’dll_dir’, should this be overwritten?

**x** An `mkinmod` object.

**submodel** Character vector of length one to specify the submodel type. See `mkinmod` for the list of allowed submodel names.

**to** Vector of the names of the state variable to which a transformation shall be included in the model.

**sink** Should a pathway to sink be included in the model in addition to the pathways to other state variables?

**full_name** An optional name to be used e.g. for plotting fits performed with the model. You can use non-ASCII characters here, but then your R code will not be portable, i.e. may produce unintended plot results on other operating systems or system configurations.

**Details**

For the definition of model types and their parameters, the equations given in the FOCUS and NAFTA guidance documents are used.

For kinetic models with more than one observed variable, a symbolic solution of the system of differential equations is included in the resulting mkinmod object in some cases, speeding up the solution.

If a C compiler is found by `pkgbuild::has_compiler()` and there is more than one observed variable in the specification, C code is generated for evaluating the differential equations, compiled using `inline::cfunction()` and added to the resulting mkinmod object.

**Value**

A list of class `mkinmod` for use with `mkinfit()`, containing, among others,

**diffs** A vector of string representations of differential equations, one for each modelling variable.
map  A list containing named character vectors for each observed variable, specifying the modelling variables by which it is represented.

use_of_ff  The content of use_of_ff is passed on in this list component.

deg_func  If generated, a function containing the solution of the degradation model.

coeffmat  The coefficient matrix, if the system of differential equations can be represented by one.

cf  If generated, a compiled function calculating the derivatives as returned by cfunction.

A list for use with mkinmod.

Notes

The IORE submodel is not well tested for metabolites. When using this model for metabolites, you may want to read the note in the help page to mkinfit.

Author(s)

Johannes Ranke

References


Examples

# Specify the SFO model (this is not needed any more, as we can now mkinfit("SFO", ...)
SFO <- mkinmod(parent = mkinsub("SFO"))

# One parent compound, one metabolite, both single first order
SFO_SFO <- mkinmod(
  parent = mkinsub("SFO", "m1"),
  m1 = mkinsub("SFO"))
print(SFO_SFO)

## Not run:
fit_sfo_sfo <- mkinfit(SFO_SFO, FOCUS_2006_D, quiet = TRUE, solution_type = "deSolve")

# Now supplying compound names used for plotting, and write to user defined location
# We need to choose a path outside the session tempdir because this gets removed
DLL_dir <- "~/local/share/mkin"
if (dir.exists(DLL_dir)) dir.create(DLL_dir)
SFO_SFO.2 <- mkinmod(
  parent = mkinsub("SFO", "m1", full_name = "Test compound"),
mkinparplot

Function to plot the confidence intervals obtained using mkinfit

Description

This function plots the confidence intervals for the parameters fitted using mkinfit.

Usage

mkinparplot(object)

Arguments

object             A fit represented in an mkinfit object.

Value

Nothing is returned by this function, as it is called for its side effect, namely to produce a plot.
mkinplot

Author(s)
Johannes Ranke

Examples

```r
## Not run:
model <- mkinmod(
  T245 = mkinsub("SFO", to = c("phenol"), sink = FALSE),
  phenol = mkinsub("SFO", to = c("anisole")),
  anisole = mkinsub("SFO"), use_of_ff = "max")
fit <- mkinfit(model, subset(mccall81_245T, soil == "Commerce"), quiet = TRUE)
mkinparplot(fit)
## End(Not run)
```

---

### mkinplot

*Plot the observed data and the fitted model of an mkinfit object*

---

Description

Deprecated function. It now only calls the plot method `plot.mkinfit`.

Usage

```r
mkinplot(fit, ...)
```

Arguments

- `fit`: an object of class `mkinfit`.
- `...`: further arguments passed to `plot.mkinfit`.

Value

The function is called for its side effect.

Author(s)
Johannes Ranke
Description

This function produces a time series for all the observed variables in a kinetic model as specified by `mkinmod`, using a specific set of kinetic parameters and initial values for the state variables.

Usage

```r
mkinpredict(x, odeparms, odeini, outtimes, ...)
```

## S3 method for class 'mkinmod'

```r
mkinpredict(
x,
  odeparms = c(k_parent_sink = 0.1),
  odeini = c(parent = 100),
  outtimes = seq(0, 120, by = 0.1),
  solution_type = "deSolve",
  use_compiled = "auto",
  method.ode = "lsoda",
  atol = 1e-08,
  rtol = 1e-10,
  map_output = TRUE,
  na_stop = TRUE,
  ...
)
```

## S3 method for class 'mkinfit'

```r
mkinpredict(
x,
  odeparms = x$bparms.ode,
  odeini = x$bparms.state,
  outtimes = seq(0, 120, by = 0.1),
  solution_type = "deSolve",
  use_compiled = "auto",
  method.ode = "lsoda",
  atol = 1e-08,
  rtol = 1e-10,
  map_output = TRUE,
  ...
)
```

Arguments

- `x` A kinetic model as produced by `mkinmod`, or a kinetic fit as fitted by `mkinfit`. In the latter case, the fitted parameters are used for the prediction.
odeparms A numeric vector specifying the parameters used in the kinetic model, which is generally defined as a set of ordinary differential equations.

odeini A numeric vector containing the initial values of the state variables of the model. Note that the state variables can differ from the observed variables, for example in the case of the SFORB model.

outtimes A numeric vector specifying the time points for which model predictions should be generated.

... Further arguments passed to the ode solver in case such a solver is used.

solution_type The method that should be used for producing the predictions. This should generally be "analytical" if there is only one observed variable, and usually "deSolve" in the case of several observed variables. The third possibility "eigen" is faster but not applicable to some models e.g. using FOMC for the parent compound.

use_compiled If set to FALSE, no compiled version of the mkinmod model is used, even if is present.

method.ode The solution method passed via mkinpredict to ode in case the solution type is "deSolve". The default "lsoda" is performant, but sometimes fails to converge.

atol Absolute error tolerance, passed to ode. Default is 1e-8, lower than in lsoda.

rtol Absolute error tolerance, passed to ode. Default is 1e-10, much lower than in lsoda.

map_output Boolean to specify if the output should list values for the observed variables (default) or for all state variables (if set to FALSE). Setting this to FALSE has no effect for analytical solutions, as these always return mapped output.

na_stop Should it be an error if deSolve::ode returns NaN values

Value
A matrix with the numeric solution in wide format

Author(s)
Johannes Ranke

Examples

SFO <- mkinmod(degradinol = mkinsub("SFO"))
# Compare solution types
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20, solution_type = "analytical")
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20, solution_type = "deSolve")
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20, solution_type = "deSolve", use_compiled = FALSE)
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20, solution_type = "eigen")
# Compare integration methods to analytical solution
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20,
solution_type = "analytical")[21,]
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20,
method = "lsoda")[21,]
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20,
method = "ode45")[21,]
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100), 0:20,
method = "rk4")[21,]
# rk4 is not as precise here

# The number of output times used to make a lot of difference until the
# default for atol was adjusted
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100),
seq(0, 20, by = 0.1))[[201,]
mkinpredict(SFO, c(k_degradinol = 0.3), c(degradinol = 100),
seq(0, 20, by = 0.01))[[2001,]

# Comparison of the performance of solution types
SFO_SFO = mkinmod(parent = list(type = "SFO", to = "m1"),
m1 = list(type = "SFO"), use_of_ff = "max")
if(require(rbenchmark)) {
  benchmark(replications = 10, order = "relative", columns = c("test", "relative", "elapsed"),
eigen = mkinpredict(SFO_SFO,
c(k_parent = 0.15, f_parent_to_m1 = 0.5, k_m1 = 0.01),
c(parent = 100, m1 = 0), seq(0, 20, by = 0.1),
solution_type = "eigen")[[201,],
deSolve_compiled = mkinpredict(SFO_SFO,
c(k_parent = 0.15, f_parent_to_m1 = 0.5, k_m1 = 0.01),
c(parent = 100, m1 = 0), seq(0, 20, by = 0.1),
solution_type = "deSolve")[[201,],
deSolve = mkinpredict(SFO_SFO,
c(k_parent = 0.15, f_parent_to_m1 = 0.5, k_m1 = 0.01),
c(parent = 100, m1 = 0), seq(0, 20, by = 0.1),
solution_type = "deSolve", use_compiled = FALSE)[[201,],
analytical = mkinpredict(SFO_SFO,
c(k_parent = 0.15, f_parent_to_m1 = 0.5, k_m1 = 0.01),
c(parent = 100, m1 = 0), seq(0, 20, by = 0.1),
solution_type = "analytical", use_compiled = FALSE)[[201,]]
}

## Not run:
# Predict from a fitted model
f <- mkinfit(SFO_SFO, FOCUS_2006.C, quiet = TRUE)
f <- mkinfit(SFO_SFO, FOCUS_2006.C, quiet = TRUE, solution_type = "deSolve")
head(mkinpredict(f))

## End(Not run)
Description

This function plots the residuals for the specified subset of the observed variables from an mkinfit object. A combined plot of the fitted model and the residuals can be obtained using `plot.mkinfit` using the argument `show_residuals = TRUE`.

Usage

```r
mkinresplot(
  object,
  obs_vars = names(object$mkinmod$map),
  xlim = c(0, 1.1 * max(object$data$time)),
  standardized = FALSE,
  xlab = "Time",
  ylab = ifelse(standardized, "Standardized residual", "Residual"),
  maxabs = "auto",
  legend = TRUE,
  lpos = "topright",
  col_obs = "auto",
  pch_obs = "auto",
  frame = TRUE,
  ...
)
```

Arguments

- **object**: A fit represented in an `mkinfit` object.
- **obs_vars**: A character vector of names of the observed variables for which residuals should be plotted. Defaults to all observed variables in the model.
- **xlim**: Plot range in x direction.
- **standardized**: Should the residuals be standardized by dividing by the standard deviation given by the error model of the fit?
- **xlab**: Label for the x axis.
- **ylab**: Label for the y axis.
- **maxabs**: Maximum absolute value of the residuals. This is used for the scaling of the y axis and defaults to "auto".
- **legend**: Should a legend be plotted?
- **lpos**: Where should the legend be placed? Default is "topright". Will be passed on to `legend`.
- **col_obs**: Colors for the observed variables.
- **pch_obs**: Symbols to be used for the observed variables.
- **frame**: Should a frame be drawn around the plots?
- **...**: Further arguments passed to `plot`.

Value

Nothing is returned by this function, as it is called for its side effect, namely to produce a plot.
**mkin_long_to_wide**

**Author(s)**

Johannes Ranke and Katrin Lindenberger

**See Also**

- `mkinplot`, for a way to plot the data and the fitted lines of the mkinfit object, and `plot_res` for a function combining the plot of the fit and the residual plot.

**Examples**

```r
model <- mkinmod(parent = mkinsub("SFO", "m1"), m1 = mkinsub("SFO"))
fit <- mkinfit(model, FOCUS_2006_D, quiet = TRUE)
mkinresplot(fit, "m1")
```

---

**mkin_long_to_wide**  
*Convert a dataframe from long to wide format*

**Description**

This function takes a dataframe in the long form, i.e. with a row for each observed value, and converts it into a dataframe with one independent variable and several dependent variables as columns.

**Usage**

```r
mkin_long_to_wide(long_data, time = "time", outtime = "time")
```

**Arguments**

- `long_data` The dataframe must contain one variable called "time" with the time values specified by the time argument, one column called "name" with the grouping of the observed values, and finally one column of observed values called "value".
- `time` The name of the time variable in the long input data.
- `outtime` The name of the time variable in the wide output data.

**Value**

Dataframe in wide format.

**Author(s)**

Johannes Ranke

**Examples**

```r
mkin_long_to_wide(FOCUS_2006_D)
```
**mkin_wide_to_long**  
*Convert a dataframe with observations over time into long format*

**Description**

This function simply takes a dataframe with one independent variable and several dependent variables and converts it into the long form as required by `mkinfit`.

**Usage**

```r
mkin_wide_to_long(wide_data, time = "t")
```

**Arguments**

- `wide_data`: The dataframe must contain one variable with the time values specified by the `time` argument and usually more than one column of observed values.
- `time`: The name of the time variable.

**Value**

Dataframe in long format as needed for `mkinfit`.

**Author(s)**

Johannes Ranke

**Examples**

```r
wide <- data.frame(t = c(1,2,3), x = c(1,4,7), y = c(3,4,5))
mkin_wide_to_long(wide)
```

---

**mkin**  
*Fit one or more kinetic models with one or more state variables to one or more datasets*

**Description**

This function calls `mkinfit` on all combinations of models and datasets specified in its first two arguments.
Usage

mmkin(models = c("SFO", "FOMC", "DFOP"),
       datasets,
       cores = if (Sys.info()["sysname"] == "Windows") 1 else parallel::detectCores(),
       cluster = NULL,
       ...
)

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

Arguments

models Either a character vector of shorthand names like c("SFO", "FOMC", "DFOP", "HS", "SFORB"), or an optionally named list of mkinmod objects.
datasets An optionally named list of datasets suitable as observed data for mkinfit.
cores The number of cores to be used for multicore processing. This is only used when the cluster argument is NULL. On Windows machines, cores > 1 is not supported, you need to use the cluster argument to use multiple logical processors. Per default, all cores detected by parallel::detectCores() are used, except on Windows where the default is 1.
cluster A cluster as returned by makeCluster to be used for parallel execution.
... Not used.
x An mmkin object.

Value

A two-dimensional array of mkinfit objects and/or try-errors that can be indexed using the model names for the first index (row index) and the dataset names for the second index (column index).

Author(s)

Johannes Ranke

See Also

[.mmkin for subsetting, plot.mmkin for plotting.

Examples

## Not run:
m_synth_SFO_lin <- mkinmod(parent = mkinsub("SFO", "M1"),
                           M1 = mkinsub("SFO", "M2"),
                           M2 = mkinsub("SFO"), use_of_ff = "max")

m_synth_FOMC_lin <- mkinmod(parent = mkinsub("FOMC", "M1"),
Evaluate parent kinetics using the NAFTA guidance

Description

The function fits the SFO, IORE and DFOP models using `mmkin` and returns an object of class `nafta` that has methods for printing and plotting.

Print `nafta` objects. The results for the three models are printed in the order of increasing model complexity, i.e. SFO, then IORE, and finally DFOP.
Usage

nafta(ds, title = NA, quiet = FALSE, ...)

## S3 method for class 'nafta'
print(x, quiet = TRUE, digits = 3, ...)

Arguments

ds A dataframe that must contain one variable called "time" with the time values specified by the time argument, one column called "name" with the grouping of the observed values, and finally one column of observed values called "value".
title Optional title of the dataset
quiet Should the evaluation text be shown?
... Further arguments passed to mmkin (not for the printing method).
x An nafta object.
digits Number of digits to be used for printing parameters and dissipation times.

Value

An list of class nafta. The list element named "mmkin" is the mmkin object containing the fits of the three models. The list element named "title" contains the title of the dataset used. The list element "data" contains the dataset used in the fits.

Author(s)

Johannes Ranke

Source


Examples

nafta_evaluation <- nafta(NAFTA_SOP_Appendix_D, cores = 1)
print(nafta_evaluation)
plot(nafta_evaluation)
Example datasets from the NAFTA SOP published 2015

Description


Usage

NAFTA_SOP_Appendix_B
NAFTA_SOP_Appendix_D

Format

2 datasets with observations on the following variables.

- name: a factor containing the name of the observed variable
- time: a numeric vector containing time points
- value: a numeric vector containing concentrations

Source


Examples

nafta_evaluation <- nafta(NAFTA_SOP_Appendix_D, cores = 1)
print(nafta_evaluation)
plot(nafta_evaluation)

Example datasets from Attachment 1 to the NAFTA SOP published 2015

Description

Data taken from from Attachment 1 of the SOP.

Usage

NAFTA_SOP_Attachment
Format

A list (NAFTA_SOP_Attachment) containing 16 datasets suitable for the evaluation with \texttt{nafta}.

Source


Examples

\begin{verbatim}
nafta_att_p5a <- nafta(NAFTA_SOP_Attachment[['p5a']], cores = 1)
print(nafta_att_p5a)
plot(nafta_att_p5a)
\end{verbatim}

---

\texttt{nlme.nlmkin} \hspace{1cm} \textit{Create an nlme model for an mmkin row object}

Description

This function sets up a nonlinear mixed effects model for an \texttt{mmkin} row object. An \texttt{mmkin} row object is essentially a list of \texttt{mkinfit} objects that have been obtained by fitting the same model to a list of datasets.

Usage

\begin{verbatim}
## S3 method for class 'mmkin'
nlme(model,
   data = "auto",
   fixed = lapply(as.list(names(mean_degparms(model))), function(el) eval(parse(text = 
     paste(el, 1, sep = "-")))),
   random = pdDiag(fixed),
   groups,
   start = mean_degparms(model, random = TRUE, test_log_parms = TRUE),
   correlation = NULL,
   weights = NULL,
   subset,
   method = c("ML", "REML"),
   na.action = na.fail,
   naPattern,
   control = list(),
   verbose = FALSE)
\end{verbatim}
## S3 method for class 'nlme.mmkin'
print(x, digits = max(3, getOption("digits") - 3), ...)

## S3 method for class 'nlme.mmkin'
update(object, ...)

### Arguments

**model**  
An *mmkin* row object.

**data**  
Ignored, data are taken from the *mmkin* model

**fixed**  
Ignored, all degradation parameters fitted in the *mmkin* model are used as fixed parameters

**random**  
If not specified, no correlations between random effects are set up for the optimised degradation model parameters. This is achieved by using the *nlme::pdDiag* method.

**groups**  
See the documentation of *nlme*

**start**  
If not specified, mean values of the fitted degradation parameters taken from the *mmkin* object are used

**correlation**  
See the documentation of *nlme*

**weights**  
passed to *nlme*

**subset**  
passed to *nlme*

**method**  
passed to *nlme*

**na.action**  
passed to *nlme*

**naPattern**  
passed to *nlme*

**control**  
passed to *nlme*

**verbose**  
passed to *nlme*

**x**  
An *nlme.mmkin* object to print

**digits**  
Number of digits to use for printing

**...**  
Update specifications passed to *update.nlme*

**object**  
An *nlme.mmkin* object to update

### Details

Note that the convergence of the *nlme* algorithms depends on the quality of the data. In degradation kinetics, we often only have few datasets (e.g. data for few soils) and complicated degradation models, which may make it impossible to obtain convergence with *nlme*.

### Value

Upon success, a fitted 'nlme.mmkin' object, which is an *nlme* object with additional elements. It also inherits from 'mixed.mmkin'.
Note

As the object inherits from `nlme::nlme`, there is a wealth of methods that will automatically work on `nlme.mmkin` objects, such as `nlme::intervals()`, `nlme::anova.lme()` and `nlme::coef.lme()`.

See Also

`nlme_function()`, `plot.mixed.mmkin`, `summary.nlme.mmkin`

Examples

ds <- lapply(experimental_data_for_UBA_2019[6:10],
  function(x) subset(x$data[c("name", "time", "value")], name == "parent"))

## Not run:
  f <- mmkin(c("SFO", "DFOP"), ds, quiet = TRUE, cores = 1)
  library(nlme)
  f_nlme_sfo <- nlme(f["SFO", ])
  f_nlme_dfop <- nlme(f["DFOP", ])
  anova(f_nlme_sfo, f_nlme_dfop)
  print(f_nlme_dfop)
  plot(f_nlme_dfop)
  endpoints(f_nlme_dfop)

ds_2 <- lapply(experimental_data_for_UBA_2019[6:10],
  function(x) x$data[c("name", "time", "value")])

  m_sfo_sfo <- mkinmod(parent = mkinsub("SFO", "A1"),
    A1 = mkinsub("SFO"), use_of_ff = "min", quiet = TRUE)
  m_sfo_sfo_ff <- mkinmod(parent = mkinsub("SFO", "A1"),
    A1 = mkinsub("SFO"), use_of_ff = "max", quiet = TRUE)
  m_dfop_sfo <- mkinmod(parent = mkinsub("DFOP", "A1"),
    A1 = mkinsub("SFO"), quiet = TRUE)

  f_2 <- mmkin(list("SFO-SFO" = m_sfo_sfo,
    "SFO-SFO-ff" = m_sfo_sfo_ff,
    "DFOP-SFO" = m_dfop_sfo),
    ds_2, quiet = TRUE)

  f_nlme_sfo_sfo <- nlme(f_2["SFO-SFO", ])
  plot(f_nlme_sfo_sfo)

  # With formation fractions this does not converge with defaults
  # f_nlme_sfo_sfo_ff <- nlme(f_2["SFO-SFO-ff", ])
  # plot(f_nlme_sfo_sfo_ff)

  # For the following, we need to increase pnlsMaxIter and the tolerance
  # to get convergence
  f_nlme_dfop_sfo <- nlme(f_2["DFOP-SFO", ],
    control = list(pnlsMaxIter = 120, tolerance = 5e-4))

  plot(f_nlme_dfop_sfo)

  anova(f_nlme_dfop_sfo, f_nlme_sfo_sfo)
endpoints(f_nlme_sfo_sfo)
endpoints(f_nlme_dfop_sfo)

if (length(findFunction("varConstProp")) > 0) {
  # tc error model for nlme available
  # Attempts to fit metabolite kinetics with the tc error model are possible,
  # but need tweaking of control values and sometimes do not converge
  f_tc <- mmkin(c("SFO", "DFOP"), ds, quiet = TRUE, error_model = "tc")
  f_nlme_sfo_tc <- nlme(f_tc["SFO", ])
  f_nlme_dfop_tc <- nlme(f_tc["DFOP", ])
  AIC(f_nlme_sfo, f_nlme_sfo_tc, f_nlme_dfop, f_nlme_dfop_tc)
  print(f_nlme_dfop_tc)
}

f_2_obs <- update(f_2, error_model = "obs")
f_nlme_sfo_sfo_obs <- nlme(f_2_obs["SFO-SFO", ])
print(f_nlme_sfo_sfo_obs)
  f_nlme_dfop_sfo_obs <- nlme(f_2_obs["DFOP-SFO", ],
                             control = list(pnlsMaxIter = 120, tolerance = 5e-4))

f_2_tc <- update(f_2, error_model = "tc")
# f_nlme_sfo_sfo_tc <- nlme(f_2_tc["SFO-SFO", ])
# No convergence with 50 iterations
# f_nlme_dfop_sfo_tc <- nlme(f_2_tc["DFOP-SFO", ],
# control = list(pnlsMaxIter = 120, tolerance = 5e-4)) # Error in X[, fmap[[nm]]] <- gradnm
  anova(f_nlme_dfop_sfo, f_nlme_dfop_sfo_obs)

## End(Not run)

---

**nlme_function**

*Helper functions to create nlme models from mmkin row objects*

**Description**

These functions facilitate setting up a nonlinear mixed effects model for an mmkin row object. An mmkin row object is essentially a list of mkinfit objects that have been obtained by fitting the same model to a list of datasets. They are used internally by the `nlme.mmkin()` method.

**Usage**

```r
nlme_function(object)
nlme_data(object)
```

**Arguments**

- `object`  
  An mmkin row object containing several fits of the same model to different datasets
Value

A function that can be used with nlme

A groupedData object

See Also

nlme.mmkin

Examples

```r
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)
m_SFO <- mkinmod(parent = mkinsub("SFO"))
d_SFO_1 <- mkinpredict(m_SFO,
c(k_parent = 0.1),
c(parent = 98), sampling_times)
d_SFO_1_long <- mkin_wide_to_long(d_SFO_1, time = "time")
d_SFO_2 <- mkinpredict(m_SFO,
c(k_parent = 0.05),
c(parent = 102), sampling_times)
d_SFO_2_long <- mkin_wide_to_long(d_SFO_2, time = "time")
d_SFO_3 <- mkinpredict(m_SFO,
c(k_parent = 0.02),
c(parent = 103), sampling_times)
d_SFO_3_long <- mkin_wide_to_long(d_SFO_3, time = "time")

f <- mmkin("SFO", ds, cores = 1, quiet = TRUE)
grouped_data <- nlme_data(f)
nlme_f <- nlme_function(f)
# These assignments are necessary for these objects to be
# visible to nlme and augPred when evaluation is done by
# pkgdown to generate the html docs.
assign("nlme_f", nlme_f, globalenv())
assign("grouped_data", grouped_data, globalenv())

library(nlme)
m_nlme <- nlme(value ~ nlme_f(name, time, parent_0, log_k_parent_sink),
data = grouped_data,
fixed = parent_0 + log_k_parent_sink ~ 1,
random = pdDiag(parent_0 + log_k_parent_sink ~ 1),
start = mean_dp)
summary(m_nlme)
plot(augPred(m_nlme, level = 0:1), layout = c(3, 1))
# augPred does not work on fits with more than one state
# variable
#
# The procedure is greatly simplified by the nlme.mmkin function
```
f_nlme <- nlme(f)
plot(f_nlme)

nobs.mkinfit

**Number of observations on which an mkinfit object was fitted**

### Description

Number of observations on which an mkinfit object was fitted

### Usage

```r
## S3 method for class 'mkinfit'
nobs(object, ...)
```

### Arguments

- `object`: An mkinfit object
- `...`: For compatibility with the generic method

### Value

The number of rows in the data included in the mkinfit object

---

**parms**

*Extract model parameters from mkinfit models*

### Description

This function always returns degradation model parameters as well as error model parameters, in order to avoid working with a fitted model without considering the error structure that was assumed for the fit.

### Usage

```r
parms(object, ...)
```

```r
## S3 method for class 'mkinfit'
parms(object, transformed = FALSE, ...)
```

```r
## S3 method for class 'mmkin'
parms(object, transformed = FALSE, ...)
```
Arguments

object A fitted model object. Methods are implemented for `mkinfit()` objects and for `mmkin()` objects.

... Not used

transformed Should the parameters be returned as used internally during the optimisation?

Value

For `mkinfit` objects, a numeric vector of fitted model parameters. For `mmkin` row objects, a matrix with the parameters with a row for each dataset. If the `mmkin` object has more than one row, a list of such matrices is returned.

Examples

```r
# mkinfit objects
fit <- mkinfit("SFO", FOCUS_2006_C, quiet = TRUE)
parms(fit)
parms(fit, transformed = TRUE)

# mmkin objects
ds <- lapply(experimental_data_for_UBA_2019[6:10],
    function(x) subset(x$data[c("name", "time", "value")]))
names(ds) <- paste("Dataset", 6:10)
# Not run:
fits <- mmkin(c("SFO", "FOMC", "DFOP"), ds, quiet = TRUE, cores = 1)
parms(fits["SFO", ])
parms(fits[, 2])
parms(fits)
parms(fits, transformed = TRUE)
# End(Not run)
```

Description

Plot predictions from a fitted nonlinear mixed model obtained via an `mmkin` row object

Usage

```r
## S3 method for class 'mixed.mmkin'
plot(x,
i = 1:ncol(x$mmkin),
obs_vars = names(x$mkinmod$map),
standardized = TRUE,
```
plot.mixed.mmkin

Arguments

xlab = "Time",
xlim = range(x$data$time),
resplot = c("predicted", "time"),
pred_over = NULL,
test_log_parms = FALSE,
conf.level = 0.6,
default_log_parms = NA,
ymax = "auto",
maxabs = "auto",
ncol.legend = ifelse(length(i) <= 3, length(i) + 1, ifelse(length(i) <= 8, 3, 4)),
nrow.legend = ceiling((length(i) + 1)/ncol.legend),
rel.height.legend = 0.02 + 0.07 * nrow.legend,
rel.height.bottom = 1.1,
pch_ds = 1:length(i),
col_ds = pch_ds + 1,
lty_ds = col_ds,
frame = TRUE,
...

Arguments

x
An object of class mixed.mmkin, saem.mmkin or nlme.mmkin
i
A numeric index to select datasets for which to plot the individual predictions, in case plots get too large
obs_vars
A character vector of names of the observed variables for which the data and the model should be plotted. Defaults to all observed variables in the model.
standardized
Should the residuals be standardized? Only takes effect if resplot = "time".
xlab
Label for the x axis.
xlim
Plot range in x direction.
resplot
Should the residuals plotted against time or against predicted values?
pred_over
Named list of alternative predictions as obtained from mkinpredict with a compatible mkinmod.
test_log_parms
Passed to mean_degparms in the case of a mixed.mmkin object
conf.level
Passed to mean_degparms in the case of a mixed.mmkin object
default_log_parms
Passed to mean_degparms in the case of a mixed.mmkin object
ymax
Vector of maximum y axis values
maxabs
Maximum absolute value of the residuals. This is used for the scaling of the y axis and defaults to "auto".
ncol.legend
Number of columns to use in the legend
nrow.legend
Number of rows to use in the legend
rel.height.legend
The relative height of the legend shown on top
The function is called for its side effect.

Examples

```r
ds <- lapply(experimental_data_for_UBA_2019[6:10],
  function(x) x$data[c("name", "time", "value")])
names(ds) <- paste0("ds ", 6:10)
dfop_sfo <- mkinmod(parent = mkinsub("DFOP", "A1"),
  A1 = mkinsub("SFO"), quiet = TRUE)
## Not run:
f <- mmkin(list("DFOP-SFO" = dfop_sfo), ds, quiet = TRUE)
plot(f[, 3:4], standardized = TRUE)
# For this fit we need to increase pnlsMaxIter, and we increase the
# tolerance in order to speed up the fit for this example evaluation
# It still takes 20 seconds to run
f_nlme <- nlme(f, control = list(pnlsMaxIter = 120, tolerance = 1e-3))
plot(f_nlme)

f_saem <- saem(f, transformations = "saemix")
plot(f_saem)

f_obs <- mmkin(list("DFOP-SFO" = dfop_sfo), ds, quiet = TRUE, error_model = "obs")
f.nlmix <- nlmix(f_obs)
plot(f.nlmix)

# We can overlay the two variants if we generate predictions
pred_nlme <- mkinpredict(dfop_sfo,
  f_nlme$bparsms.optim[-1],
  c(parent = f_nlme$bparsms.optim[[1]], A1 = 0),
  seq(0, 180, by = 0.2))
plot(f.saem, pred_over = list(nlme = pred_nlme))
```

## End(Not run)
plot.mkinfit

Plot the observed data and the fitted model of an mkinfit object

Description

Solves the differential equations with the optimised and fixed parameters from a previous successful call to mkinfit and plots the observed data together with the solution of the fitted model.

Usage

```r
## S3 method for class 'mkinfit'
plot(
  x,
  fit = x,
  obs_vars = names(fit$mkinmod$map),
  xlab = "Time",
  ylab = "Residue",
  xlim = range(fit$data$time),
  ylim = "default",
  col_obs = 1:length(obs_vars),
  pch_obs = col_obs,
  lty_obs = rep(1, length(obs_vars)),
  add = FALSE,
  legend = !add,
  show_residuals = FALSE,
  show_errplot = FALSE,
  maxabs = "auto",
  sep_obs = FALSE,
  rel.height.middle = 0.9,
  row_layout = FALSE,
  lpos = "topright",
  inset = c(0.05, 0.05),
  show_errmin = FALSE,
  errmin_digits = 3,
  frame = TRUE,
  ...
)
```

```r
plot_sep(
  fit,
  show_errmin = TRUE,
  show_residuals = ifelse(identical(fit$err_mod, "const"), TRUE, "standardized"),
  ...
)
```

```r
plot_res(
  fit,
```
sep_obs = FALSE,
show_errmin = sep_obs,
standardized = ifelse(identical(fit$err_mod, "const"), FALSE, TRUE),

plot_err(fit, sep_obs = FALSE, show_errmin = sep_obs, ...)

Arguments

x
  Alias for fit introduced for compatibility with the generic S3 method.

fit
  An object of class mkinfit.

obs_vars
  A character vector of names of the observed variables for which the data and the
  model should be plotted. Defaults to all observed variables in the model.

xlab
  Label for the x axis.

ylab
  Label for the y axis.

xlim
  Plot range in x direction.

ylim
  Plot range in y direction. If given as a list, plot ranges for the different plot rows
  can be given for row layout.

col_obs
  Colors used for plotting the observed data and the corresponding model predic-
  tion lines.

cpych_obs
  Symbols to be used for plotting the data.

lty_obs
  Line types to be used for the model predictions.

add
  Should the plot be added to an existing plot?

legend
  Should a legend be added to the plot?

show_residuals
  Should residuals be shown? If only one plot of the fits is shown, the residual
  plot is in the lower third of the plot. Otherwise, i.e. if "sep_obs" is given, the
  residual plots will be located to the right of the plots of the fitted curves. If this
  is set to 'standardized', a plot of the residuals divided by the standard deviation
  given by the fitted error model will be shown.

show_errplot
  Should squared residuals and the error model be shown? If only one plot of
  the fits is shown, this plot is in the lower third of the plot. Otherwise, i.e. if
  "sep_obs" is given, the residual plots will be located to the right of the plots of the
  fitted curves.

maxabs
  Maximum absolute value of the residuals. This is used for the scaling of the y
  axis and defaults to "auto".

sep_obs
  Should the observed variables be shown in separate subplots? If yes, residual
  plots requested by "show_residuals" will be shown next to, not below the plot of
  the fits.

rel.height.middle
  The relative height of the middle plot, if more than two rows of plots are shown.

row_layout
  Should we use a row layout where the residual plot or the error model plot is
  shown to the right?
lpos  Position(s) of the legend(s). Passed to legend as the first argument. If not length one, this should be of the same length as the obs_var argument.

inset  Passed to legend if applicable.

show_errmin  Should the FOCUS chi2 error value be shown in the upper margin of the plot?

errmin_digits  The number of significant digits for rounding the FOCUS chi2 error percentage.

frame  Should a frame be drawn around the plots?

...  Further arguments passed to plot.

standardized  When calling ’plot_res’, should the residuals be standardized in the residual plot?

Details

If the current plot device is a tikz device, then latex is being used for the formatting of the chi2 error level, if show_errmin = TRUE.

Value

The function is called for its side effect.

Author(s)

Johannes Ranke

Examples

# One parent compound, one metabolite, both single first order, path from # parent to sink included
## Not run:
SFO_SFO <- mkinmod(parent = mkinsub("SFO", "m1", full = "Parent"),
                   m1 = mkinsub("SFO", full = "Metabolite M1" ))
fit <- mkinfit(SFO_SFO, FOCUS_2006_D, quiet = TRUE)
fit <- mkinfit(SFO_SFO, FOCUS_2006_D, quiet = TRUE, error_model = "tc")
plot(fit)
plot_res(fit)
plot_res(fit, standardized = FALSE)
plot_err(fit)

# Show the observed variables separately, with residuals
plot(fit, sep_obs = TRUE, show_residuals = TRUE, lpos = c("topright", "bottomright"),
     show_errmin = TRUE)

# The same can be obtained with less typing, using the convenience function plot_sep
plot_sep(fit, lpos = c("topright", "bottomright"))

# Show the observed variables separately, with the error model
plot(fit, sep_obs = TRUE, show_errplot = TRUE, lpos = c("topright", "bottomright"),
     show_errmin = TRUE)

## End(Not run)
plot.mmkin

Plot model fits (observed and fitted) and the residuals for a row or column of an mmkin object

Description

When `x` is a row selected from an mmkin object ([.mmkin), the same model fitted for at least one dataset is shown. When it is a column, the fit of at least one model to the same dataset is shown.

Usage

```r
## S3 method for class 'mmkin'
plot(
x,  
main = "auto",  
legends = 1,  
resplot = c("time", "errmod"),  
ylab = "Residue",  
standardized = FALSE,  
show_errmin = TRUE,  
errmin_var = "All data",  
errmin_digits = 3,  
cex = 0.7,  
rel.height.middle = 0.9,  
ymax = "auto",  
...)
```

Arguments

- `x`: An object of class `mmkin`, with either one row or one column.
- `main`: The main title placed on the outer margin of the plot.
- `legends`: An index for the fits for which legends should be shown.
- `resplot`: Should the residuals plotted against time, using `mkinresplot`, or as squared residuals against predicted values, with the error model, using `mkinerrplot`.
- `ylab`: Label for the y axis.
- `standardized`: Should the residuals be standardized? This option is passed to `mkinresplot`, it only takes effect if `resplot = "time"`.
- `show_errmin`: Should the chi2 error level be shown on top of the plots to the left?
- `errmin_var`: The variable for which the FOCUS chi2 error value should be shown.
- `errmin_digits`: The number of significant digits for rounding the FOCUS chi2 error percentage.
- `cex`: Passed to the plot functions and `mtext`. 
plot.nafta

Plot the results of the three models used in the NAFTA scheme.

Description

The plots are ordered with increasing complexity of the model in this function (SFO, then IORE, then DFOP).

Details

If the current plot device is a \texttt{tikz} device, then \texttt{latex} is being used for the formatting of the \texttt{chi2} error level.

Value

The function is called for its side effect.

Author(s)

Johannes Ranke

Examples

```r
## Not run:
# Only use one core not to offend CRAN checks
fits <- mmkin(c("FOMC", "HS"),
              list("FOCUS B" = FOCUS_2006_B, "FOCUS C" = FOCUS_2006_C), # named list for titles
cores = 1, quiet = TRUE, error_model = "tc")
plot(fits[, "FOCUS C"])
plot(fits["FOMC", ])
plot(fits["FOMC", ], show_errmin = FALSE)

# We can also plot a single fit, if we like the way plot.mmkin works, but then the plot
# height should be smaller than the plot width (this is not possible for the html pages
# generated by pkgdown, as far as I know).
plot(fits["FOMC", "FOCUS C"])) # same as plot(fits[1, 2])

# Show the error models
plot(fits["FOMC", ], resplot = "errmod")

## End(Not run)
```
residuals.mkinfit

Usage

## S3 method for class 'nafta'
plot(x, legend = FALSE, main = "auto", ...)

Arguments

x An object of class nafta.
legend Should a legend be added?
main Possibility to override the main title of the plot.
... Further arguments passed to plot.mmkin.

Details

Calls plot.mmkin.

Value

The function is called for its side effect.

Author(s)

Johannes Ranke

residuals.mkinfit Extract residuals from an mkinfit model

Description

Extract residuals from an mkinfit model

Usage

## S3 method for class 'mkinfit'
residuals(object, standardized = FALSE, ...)

Arguments

object A mkinfit object
standardized Should the residuals be standardized by dividing by the standard deviation obtained from the fitted error model?
... Not used

Examples

f <- mkinfit("DFOP", FOCUS_2006_C, quiet = TRUE)
residuals(f)
residuals(f, standardized = TRUE)
Fit nonlinear mixed models with SAEM

**Description**

This function uses `saemix::saemix()` as a backend for fitting nonlinear mixed effects models created from `mmkin` row objects using the Stochastic Approximation Expectation Maximisation algorithm (SAEM).

**Usage**

```r
saem(object, ...)  
## S3 method for class 'mmkin'
saem(
  object,
  transformations = c("mkin", "saemix"),
  degparms_start = numeric(),
  test_log_parms = TRUE,
  conf.level = 0.6,
  solution_type = "auto",
  nbiter.saemix = c(300, 100),
  control = list(displayProgress = FALSE, print = FALSE, nbiter.saemix = nbiter.saemix,
                 save = FALSE, save.graphs = FALSE),
  fail_with_errors = TRUE,
  verbose = FALSE,
  ...  
)

## S3 method for class 'saem.mmkin'
print(x, digits = max(3, getOption("digits") - 3), ...)

saemix_model(
  object,
  solution_type = "auto",
  transformations = c("mkin", "saemix"),
  degparms_start = numeric(),
  test_log_parms = FALSE,
  conf.level = 0.6,
  verbose = FALSE,
  ...
)

saemix_data(object, verbose = FALSE, ...)
```
Arguments

- **object**
  - An `mmkin` row object containing several fits of the same `mkinmod` model to different datasets

- **transformations**
  - Per default, all parameter transformations are done in mkin. If this argument is set to 'saemix', parameter transformations are done in 'saemix' for the supported cases. Currently this is only supported in cases where the initial concentration of the parent is not fixed, SFO or DFOP is used for the parent and there is either no metabolite or one.

- **degsparms_start**
  - Parameter values given as a named numeric vector will be used to override the starting values obtained from the 'mmkin' object.

- **test_log_parms**
  - If TRUE, an attempt is made to use more robust starting values for population parameters fitted as log parameters in mkin (like rate constants) by only considering rate constants that pass the t-test when calculating mean degradation parameters using `mean_degsparms`.

- **conf.level**
  - Possibility to adjust the required confidence level for parameter that are tested if requested by 'test_log_parms'.

- **solution_type**
  - Possibility to specify the solution type in case the automatic choice is not desired

- **nbiter.saemix**
  - Convenience option to increase the number of iterations

- **control**
  - Passed to `saemix::saemix`.

- **fail_with_errors**
  - Should a failure to compute standard errors from the inverse of the Fisher Information Matrix be a failure?

- **verbose**
  - Should we print information about created objects of type `saemix::SaemixModel` and `saemix::SaemixData`?

- **quiet**
  - Should we suppress the messages saemix prints at the beginning and the end of the optimisation process?

- **x**
  - An `saem.mmkin` object to print

- **digits**
  - Number of digits to use for printing

Details

An `mmkin` row object is essentially a list of `mkinfit` objects that have been obtained by fitting the same model to a list of datasets using `mkinfit`.

Starting values for the fixed effects (population mean parameters, argument `psi0` of `saemix::saemixModel()`) are the mean values of the parameters found using `mmkin`.

Value

An S3 object of class 'saem.mmkin', containing the fitted `saemix::SaemixObject` as a list component named 'so'. The object also inherits from 'mixed.mmkin'.

An `saemix::SaemixModel` object.

An `saemix::SaemixData` object.
See Also

summary.saem.mmkin plot.mixed.mmkin

Examples

```r
## Not run:
d <- lapply(experimental_data_for_UBA_2019[6:10],
  function(x) subset(x$data[c("name", "time", "value")]))
names(d) <- paste("Dataset", 6:10)
f_mmkin_parent_p0_fixed <- mmkin("FOMC", d, state.ini = c(parent = 100), fixed_initials = "parent", quiet = TRUE)
f_saem_p0_fixed <- saem(f_mmkin_parent_p0_fixed)

f_mmkin_parent <- mmkin(c("SFO", "FOMC", "DFOP"), d, quiet = TRUE)
f_saem_sfo <- saem(f_mmkin_parent["SFO", ])
f_saem_fomc <- saem(f_mmkin_parent["FOMC", ])
f_saem_dfop <- saem(f_mmkin_parent["DFOP", ])

# The returned saem.mmkin object contains an SaemixObject, therefore we can use
# functions from saemix
library(saemix)
compare.saemix(f_saem_sfo$so, f_saem_fomc$so, f_saem_dfop$so)
plot(f_saem_fomc$so, plot.type = "convergence")
plot(f_saem_fomc$so, plot.type = "individual.fit")
plot(f_saem_fomc$so, plot.type = "npde")
plot(f_saem_fomc$so, plot.type = "vpc")

f_mmkin_parent_tc <- update(f_mmkin_parent, error_model = "tc")

f_saem_fomc_tc <- saem(f_mmkin_parent_tc["FOMC", ])
compare.saemix(f_saem_fomc$so, f_saem_fomc_tc$so)

sfo_sfo <- mkinmod(parent = mkinsub("SFO", "A1"),
                  A1 = mkinsub("SFO"))
fomc_sfo <- mkinmod(parent = mkinsub("FOMC", "A1"),
                    A1 = mkinsub("SFO"))
dfop_sfo <- mkinmod(parent = mkinsub("DFOP", "A1"),
                    A1 = mkinsub("SFO"))

# The following fit uses analytical solutions for SFO-SFO and DFOP-SFO,
# and compiled ODEs for FOMC that are much slower
f_mmkin <- mmkin(list(
  "SFO-SFO" = sfo_sfo, "FOMC-SFO" = fomc_sfo, "DFOP-SFO" = dfop_sfo),
  d, quiet = TRUE)

# saem fits of SFO-SFO and DFOP-SFO to these data take about five seconds
# each on this system, as we use analytical solutions written for saemix.
# When using the analytical solutions written for mkin this took around
# four minutes
f_saem_sfo_sfo <- saem(f_mmkin["SFO-SFO", ])
f_saem_dfop_sfo <- saem(f_mmkin["DFOP-SFO", ])

# We can use print, plot and summary methods to check the results
print(f_saem_dfop_sfo)
plot(f_saem_dfop_sfo)
summary(f_saem_dfop_sfo, data = TRUE)
```
The following takes about 6 minutes

```r
#f_saem_dfop_sfo_deSolve <- saem(f_mmkin["DFOP-SFO", ], solution_type = "deSolve",
# control = list(nbiter.saemix = c(200, 80), nbdisplay = 10))

#saemix::compare.saemix(list(
# f_saem_dfop_sfo$so,
# f_saem_dfop_sfo_deSolve$so))

# If the model supports it, we can also use eigenvalue based solutions, which
# take a similar amount of time

#f_saem_sfo_sfo_eigen <- saem(f_mmkin["SFO-SFO", ], solution_type = "eigen",
# control = list(nbiter.saemix = c(200, 80), nbdisplay = 10))

## End(Not run)
```

---

**schaefer07_complex_case**

*Metabolism data set used for checking the software quality of KinGUI*

### Description

This dataset was used for a comparison of KinGUI and ModelMaker to check the software quality of KinGUI in the original publication (Schäfer et al., 2007). The results from the fitting are also included.

### Usage

```r
schaefer07_complex_case
```

### Format

The data set is a data frame with 8 observations on the following 6 variables.

- **time**: a numeric vector
- **parent**: a numeric vector
- **A1**: a numeric vector
- **B1**: a numeric vector
- **C1**: a numeric vector
- **A2**: a numeric vector

The results are a data frame with 14 results for different parameter values.

### References

Examples

```r
data <- mkin_wide_to_long(schaefer07_complex_case, time = "time")
model <- mkinmod(
  parent = list(type = "SFO", to = c("A1", "B1", "C1"), sink = FALSE),
  A1 = list(type = "SFO", to = "A2"),
  B1 = list(type = "SFO"),
  C1 = list(type = "SFO"),
  A2 = list(type = "SFO"), use_of_ff = "max")
## Not run:
fit <- mkinfit(model, data, quiet = TRUE)
plot(fit)
endpoints(fit)
## End(Not run)
# Compare with the results obtained in the original publication
print(schaefer07_complex_results)
```

SFO.solution  

**Single First-Order kinetics**

**Description**

Function describing exponential decline from a defined starting value.

**Usage**

```r
SFO.solution(t, parent_0, k)
```

**Arguments**

- `t` Time.
- `parent_0` Starting value for the response variable at time zero.
- `k` Kinetic rate constant.

**Value**

The value of the response variable at time `t`.

**References**


See Also

Other parent solutions: \texttt{DFOP.solution()}, \texttt{FOMC.solution()}, \texttt{HS.solution()}, \texttt{IORE.solution()}, \texttt{SFORB.solution()}, \texttt{logistic.solution()}

Examples

\footnotesize
\begin{verbatim}
## Not run: plot(function(x) SFO.solution(x, 100, 3), 0, 2)
\end{verbatim}

\section*{SFOB solution}

\textit{Single First-Order Reversible Binding kinetics}

\subsection*{Description}

Function describing the solution of the differential equations describing the kinetic model with first-order terms for a two-way transfer from a free to a bound fraction, and a first-order degradation term for the free fraction. The initial condition is a defined amount in the free fraction and no substance in the bound fraction.

\subsection*{Usage}

\texttt{SFORB.solution}(t, parent_0, k_12, k_21, k_1output)

\subsection*{Arguments}

\begin{itemize}
  \item \texttt{t} Time.
  \item \texttt{parent_0} Starting value for the response variable at time zero.
  \item \texttt{k_12} Kinetic constant describing transfer from free to bound.
  \item \texttt{k_21} Kinetic constant describing transfer from bound to free.
  \item \texttt{k_1output} Kinetic constant describing degradation of the free fraction.
\end{itemize}

\subsection*{Value}

The value of the response variable, which is the sum of free and bound fractions at time \( t \).

\subsection*{References}


See Also

Other parent solutions: DFOP.solution(), FOMC.solution(), HS.solution(), IORE.solution(), SFO.solution(), logistic.solution()

Examples

```r
## Not run: plot(function(x) SFORB.solution(x, 100, 0.5, 2, 3), 0, 2)
```

---

### sigma_twocomp

#### Two-component error model

**Description**

Function describing the standard deviation of the measurement error in dependence of the measured value \( y \):

\[
\sigma = \sqrt{\sigma_{\text{low}}^2 + y^2 \cdot \text{rsd}_{\text{high}}^2}
\]

\( \sigma \) is the error model used for example by Werner et al. (1978). The model proposed by Rocke and Lorenzato (1995) can be written in this form as well, but assumes approximate lognormal distribution of errors for high values of \( y \).

**Usage**

\[ \text{sigma_twocomp}(y, \text{sigma_low}, \text{rsd_high}) \]

**Arguments**

- **y**: The magnitude of the observed value
- **sigma_low**: The asymptotic minimum of the standard deviation for low observed values
- **rsd_high**: The coefficient describing the increase of the standard deviation with the magnitude of the observed value

**Details**

\[
\sigma = \sqrt{\sigma_{\text{low}}^2 + y^2 \cdot \text{rsd}_{\text{high}}^2}
\]

\( \text{sigma} = \text{sqrt}((\text{sigma_low}^2 + y^2 \cdot \text{rsd_high}^2)) \)

**Value**

The standard deviation of the response variable.
**References**


**Examples**

```r
times <- c(0, 1, 3, 7, 14, 28, 60, 90, 120)
d_pred <- data.frame(time = times, parent = 100 * exp(- 0.03 * times))
set.seed(123456)
d_syn <- add_err(d_pred, function(y) sigma_twocomp(y, 1, 0.07),
  reps = 2, n = 1)[[1]]
f_nls <- nls(value ~ SSasymp(time, 0, parent_0, lrc), data = d_syn,
  start = list(parent_0 = 100, lrc = -3))
library(nlme)
f_gnls <- gnls(value ~ SSasymp(time, 0, parent_0, lrc),
  data = d_syn, na.action = na.omit,
  start = list(parent_0 = 100, lrc = -3))
if (length(findFunction("varConstProp")) > 0) {
  f_gnls_tc <- update(f_gnls, weights = varConstProp())
  f_gnls_tc_sf <- update(f_gnls_tc, control = list(sigma = 1))
}
f_mkin <- mkinfit("SFO", d_syn, error_model = "const", quiet = TRUE)
f_mkin_tc <- mkinfit("SFO", d_syn, error_model = "tc", quiet = TRUE)
plot_res(f_mkin_tc, standardized = TRUE)
AIC(f_nls, f_gnls, f_gnls_tc, f_gnls_tc_sf, f_mkin, f_mkin_tc)
```

**summary.mkinfit**

*Summary method for class "mkinfit"*

**Description**

Lists model equations, initial parameter values, optimised parameters with some uncertainty statistics, the chi2 error levels calculated according to FOCUS guidance (2006) as defined therein, formation fractions, DT50 values and optionally the data, consisting of observed, predicted and residual values.

**Usage**

```r
## S3 method for class 'mkinfit'
summary(object, data = TRUE, distimes = TRUE, alpha = 0.05, ...)

## S3 method for class 'summary.mkinfit'
print(x, digits = max(3, getOption("digits") - 3), ...)
```
summary.mkinfit

Arguments

- **object**: an object of class `mkinfit`.
- **data**: logical, indicating whether the data should be included in the summary.
- **distimes**: logical, indicating whether DT50 and DT90 values should be included.
- **alpha**: error level for confidence interval estimation from t distribution
- **...**: optional arguments passed to methods like `print`.
- **x**: an object of class `summary.mkinfit`.
- **digits**: Number of digits to use for printing

Value

The summary function returns a list with components, among others

- **version, Rversion**: The mkin and R versions used
- **date.fit, date.summary**: The dates where the fit and the summary were produced
- **diffs**: The differential equations used in the model
- **use_of_ff**: Was maximum or minimum use made of formation fractions
- **bpar**: Optimised and backtransformed parameters
- **data**: The data (see Description above).
- **start**: The starting values and bounds, if applicable, for optimised parameters.
- **fixed**: The values of fixed parameters.
- **errmin**: The chi2 error levels for each observed variable.
- **bparms.ode**: All backtransformed ODE parameters, for use as starting parameters for related models.
- **errparms**: Error model parameters.
- **ff**: The estimated formation fractions derived from the fitted model.
- **distimes**: The DT50 and DT90 values for each observed variable.
- **SFORB**: If applicable, eigenvalues of SFORB components of the model.

The print method is called for its side effect, i.e. printing the summary.

Author(s)

Johannes Ranke

References

Examples

```
summary(mkinfit(mkinmod(parent = mkinsub("SFO")), FOCUS_2006_A, quiet = TRUE))
```

Summary method for class "nlme.mmkin"

Description

Lists model equations, initial parameter values, optimised parameters for fixed effects (population), random effects (deviations from the population mean) and residual error model, as well as the resulting endpoints such as formation fractions and DT50 values. Optionally (default is FALSE), the data are listed in full.

Usage

```r
## S3 method for class 'nlme.mmkin'
summary(
  object,
  data = FALSE,
  verbose = FALSE,
  distimes = TRUE,
  alpha = 0.05,
  ...
)
## S3 method for class 'summary.nlme.mmkin'
print(x, digits = max(3, getOption("digits") - 3), verbose = x$verbose, ...)
```

Arguments

- **object**
  - an object of class `nlme.mmkin`
- **data**
  - logical, indicating whether the full data should be included in the summary.
- **verbose**
  - Should the summary be verbose?
- **distimes**
  - logical, indicating whether DT50 and DT90 values should be included.
- **alpha**
  - error level for confidence interval estimation from the t distribution
- **...**
  - optional arguments passed to methods like `print`
- **x**
  - an object of class `summary.nlme.mmkin`
- **digits**
  - Number of digits to use for printing
Value

The summary function returns a list based on the *nlme* object obtained in the fit, with at least the following additional components

- `nlmeversion`, `mkinversion`, `Rversion`  
  The *nlme*, *mkin* and *R* versions used
- `date.fit`, `date.summary`  
  The dates where the fit and the summary were produced
- `diffs`  
  The differential equations used in the degradation model
- `use_of_ff`  
  Was maximum or minimum use made of formation fractions
- `data`  
  The data
- `confint_trans`  
  Transformed parameters as used in the optimisation, with confidence intervals
- `confint_back`  
  Backtransformed parameters, with confidence intervals if available
- `ff`  
  The estimated formation fractions derived from the fitted model.
- `distimes`  
  The DT50 and DT90 values for each observed variable.
- `SFORB`  
  If applicable, eigenvalues of SFORB components of the model.

The print method is called for its side effect, i.e. printing the summary.

Author(s)

Johannes Ranke for the *mkin* specific parts José Pinheiro and Douglas Bates for the components inherited from *nlme*

Examples

```r
# Generate five datasets following SFO kinetics
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)
dt50_sfo_in_pop <- 50
k_in_pop <- log(2) / dt50_sfo_in_pop
set.seed(1234)
k_in <- rlnorm(5, log(k_in_pop), 0.5)
SFO <- mkinmod(parent = mkinsub("SFO"))

pred_sfo <- function(k) {
  mkinpredict(SFO,
    c(k_parent = k),
    c(parent = 100),
    sampling_times)
}

ds_sfo_mean <- lapply(k_in, pred_sfo)
names(ds_sfo_mean) <- paste("ds", 1:5)

set.seed(12345)
ds_sfo_syn <- lapply(ds_sfo_mean, function(ds) {
  add_err(ds,
```
sdfunc = function(value) sqrt(1^2 + value^2 * 0.07^2),
       n = 1)[[1]]
}

## Not run:
# Evaluate using mmkin and nlme
library(nlme)
f_mmkin <- mmkin("SFO", ds_sfo_syn, quiet = TRUE, error_model = "tc", cores = 1)
f_nlme <- nlme(f_mmkin)
summary(f_nlme, data = TRUE)

## End(Not run)

summary.saem.mmkin  Summary method for class "saem.mmkin"

Description

Lists model equations, initial parameter values, optimised parameters for fixed effects (population),
random effects (deviations from the population mean) and residual error model, as well as the
resulting endpoints such as formation fractions and DT50 values. Optionally (default is FALSE),
the data are listed in full.

Usage

## S3 method for class 'saem.mmkin'
summary(object, data = FALSE, verbose = FALSE, distimes = TRUE, ...)

## S3 method for class 'summary.saem.mmkin'
print(x, digits = max(3, getOption("digits") - 3), verbose = x$verbose, ...)

Arguments

object     an object of class saem.mmkin
data       logical, indicating whether the full data should be included in the summary.
verbose    Should the summary be verbose?
distimes   logical, indicating whether DT50 and DT90 values should be included.
...        optional arguments passed to methods like print.
x          an object of class summary.saem.mmkin
digits    Number of digits to use for printing
Value

The summary function returns a list based on the `saemix::SaemixObject` obtained in the fit, with at least the following additional components:

- `saemixversion`, `mkinversion`, `Rversion`: The saemix, mkin and R versions used.
- `date.fit`, `date.summary`: The dates where the fit and the summary were produced.
- `difs`: The differential equations used in the degradation model.
- `use_of_ff`: Was maximum or minimum use made of formation fractions.
- `data`: The data.
- `confint_trans`: Transformed parameters as used in the optimisation, with confidence intervals.
- `confint_back`: Backtransformed parameters, with confidence intervals if available.
- `confint_errmod`: Error model parameters with confidence intervals.
- `ff`: The estimated formation fractions derived from the fitted model.
- `distimes`: The DT50 and DT90 values for each observed variable.
- `SFORB`: If applicable, eigenvalues of SFORB components of the model.

The print method is called for its side effect, i.e. printing the summary.

Author(s)

Johannes Ranke for the mkin specific parts, saemix authors for the parts inherited from saemix.

Examples

```r
# Generate five datasets following DFOP-SFO kinetics
set.seed(1234)
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)
dfop_sfo <- mkinmod(parent = mkinsub("DFOP", "m1"),
    m1 = mkinsub("SFO"), quiet = TRUE)
pred_dfop_sfo <- function(k1, k2, g, f_parent_to_m1, k_m1) {
  mkinpredict(dfop_sfo,
    c(k1 = k1, k2 = k2, g = g, f_parent_to_m1 = f_parent_to_m1, k_m1 = k_m1),
    c(parent = 100, m1 = 0),
    sampling_times)
}
ds_mean_dfop_sfo <- lapply(1:5, function(i) {
  mkinpredict(dfop_sfo,
    c(k1 = k1_in[i], k2 = k2_in[i], g = g_in[i],
    f_parent_to_m1 = f_parent_to_m1_in[i], k_m1 = k_m1_in[i],
    k_m1_in <- rlnorm(5, log(0.02), 0.3))
})
```
synthetic_data_for_UBA_2014

```r
c(parent = 100, m1 = 0),
sampling_times)
```
```r
}
```
```r
names(ds_mean_dfop_sfo) <- paste("ds", 1:5)
```
```r
ds_syn_dfop_sfo <- lapply(ds_mean_dfop_sfo, function(ds) {
  add_err(ds,
    sdfunc = function(value) sqrt(1^2 + value^2 * 0.07^2),
    n = 1)[[1]]
})
```
```r
## Not run:
# Evaluate using mmkin and saem
f_mmkin_dfop_sfo <- mmkin(list(dfop_sfo), ds_syn_dfop_sfo,
  quiet = TRUE, error_model = "tc", cores = 5)
f_saem_dfop_sfo <- saem(f_mmkin_dfop_sfo)
summary(f_saem_dfop_sfo, data = TRUE)
## End(Not run)
```

---

**synthetic_data_for_UBA_2014**

*Synthetic datasets for one parent compound with two metabolites*

**Description**

The 12 datasets were generated using four different models and three different variance components. The four models are either the SFO or the DFOP model with either two sequential or two parallel metabolites.

Variance component 'a' is based on a normal distribution with standard deviation of 3. Variance component 'b' is also based on a normal distribution, but with a standard deviation of 7. Variance component 'c' is based on the error model from Rocke and Lorenzato (1995), with the minimum standard deviation (for small y values) of 0.5, and a proportionality constant of 0.07 for the increase of the standard deviation with y. Note that this is a simplified version of the error model proposed by Rocke and Lorenzato (1995), as in their model the error of the measured values approximates lognormal distribution for high values, whereas we are using normally distributed error components all along.

Initial concentrations for metabolites and all values where adding the variance component resulted in a value below the assumed limit of detection of 0.1 were set to NA.

As an example, the first dataset has the title SFO_lin_a and is based on the SFO model with two sequential metabolites (linear pathway), with added variance component 'a'.

Compare also the code in the example section to see the degradation models.

**Usage**

`synthetic_data_for_UBA_2014`
Format

A list containing twelve datasets as an R6 class defined by `mkinds`, each containing, among others, the following components

- **title** The name of the dataset, e.g. `SFO_lin_a`
- **data** A data frame with the data in the form expected by `mkinfit`

Source

Ranke (2014) Prüfung und Validierung von Modellierungssoftware als Alternative zu ModelMaker 4.0, Umweltbundesamt Projektnummer 27452


Examples

```r
## Not run:
# The data have been generated using the following kinetic models
m_synth_SFO_lin <- mkinmod(parent = list(type = "SFO", to = "M1"),
    M1 = list(type = "SFO", to = "M2"),
    M2 = list(type = "SFO"), use_of_ff = "max")

m_synth_SFO_par <- mkinmod(parent = list(type = "SFO", to = c("M1", "M2"),
    sink = FALSE),
    M1 = list(type = "SFO"),
    M2 = list(type = "SFO"), use_of_ff = "max")

m_synth_DFOP_lin <- mkinmod(parent = list(type = "DFOP", to = "M1"),
    M1 = list(type = "SFO", to = "M2"),
    M2 = list(type = "SFO"), use_of_ff = "max")

m_synth_DFOP_par <- mkinmod(parent = list(type = "DFOP", to = c("M1", "M2"),
    sink = FALSE),
    M1 = list(type = "SFO"),
    M2 = list(type = "SFO"), use_of_ff = "max")

# The model predictions without intentional error were generated as follows
sampling_times = c(0, 1, 3, 7, 14, 28, 60, 90, 120)

d_synth_SFO_lin <- mkinpredict(m_synth_SFO_lin,
    c(k_parent = 0.7, f_parent_to_M1 = 0.8,
    k_M1 = 0.3, f_M1_to_M2 = 0.7,
    k_M2 = 0.02),
    c(parent = 100, M1 = 0, M2 = 0),
    sampling_times)

d_synth_DFOP_lin <- mkinpredict(m_synth_DFOP_lin,
    c(k1 = 0.2, k2 = 0.02, g = 0.5,
    f_parent_to_M1 = 0.5, k_M1 = 0.3,
    f_M1_to_M2 = 0.7, k_M2 = 0.02),
```

c(parent = 100, M1 = 0, M2 = 0),
sampling_times)

d_synth_SFO_par <- mkinpredict(m_synth_SFO_par,
c(k_parent = 0.2,
f_parent_to_M1 = 0.8, k_M1 = 0.01,
f_parent_to_M2 = 0.2, k_M2 = 0.02),
c(parent = 100, M1 = 0, M2 = 0),
sampling_times)

d_synth_DFOP_par <- mkinpredict(m_synth_DFOP_par,
c(k1 = 0.3, k2 = 0.02, g = 0.7,
f_parent_to_M1 = 0.6, k_M1 = 0.04,
f_parent_to_M2 = 0.4, k_M2 = 0.01),
c(parent = 100, M1 = 0, M2 = 0),
sampling_times)

# Construct names for datasets with errors
d_synth_names = paste0("d_synth_", c("SFO_lin", "SFO_par", "DFOP_lin", "DFOP_par"))

# Original function used or adding errors. The add_err function now published
# with this package is a slightly generalised version where the names of
# secondary compartments that should have an initial value of zero (M1 and M2
# in this case) are not hardcoded any more.
# add_err = function(d, sdfunc, LOD = 0.1, reps = 2, seed = 123456789)
# {
#  set.seed(seed)
#  d_long = mkin_wide_to_long(d, time = "time")
#  d_rep = data.frame(lapply(d_long, rep, each = 2))
#  d_rep$value = rnorm(length(d_rep$value), d_rep$value, sdfunc(d_rep$value))
#  d_rep[d_rep$time == 0 & d_rep$name %in% c("M1", "M2")] <- 0
#  d_NA <- transform(d_rep, value = ifelse(value < LOD, NA, value))
#  d_NA$value <- round(d_NA$value, 1)
#  return(d_NA)
# }

# The following is the simplified version of the two-component model of Rocke
# and Lorenzato (1995)
sdfunc_twocomp = function(value, sd_low, rsd_high) {
  sqrt(sd_low^2 + value^2 * rsd_high^2)
}

# Add the errors.
for (d_synth_name in d_synth_names)
{
  d_synth = get(d_synth_name)
  assign(paste0(d_synth_name, "_a"), add_err(d_synth, function(value) 3))
  assign(paste0(d_synth_name, "_b"), add_err(d_synth, function(value) 7))
  assign(paste0(d_synth_name, "_c"), add_err(d_synth,
     function(value) sdfunc_twocomp(value, 0.5, 0.07)))
}
test_data_from_UBA_2014

Three experimental datasets from two water sediment systems and one soil

Description

The datasets were used for the comparative validation of several kinetic evaluation software packages (Ranke, 2014).

Usage

test_data_from_UBA_2014

Format

A list containing three datasets as an R6 class defined by mkinds. Each dataset has, among others, the following components:

title The name of the dataset, e.g. UBA_2014_WS_river
data A data frame with the data in the form expected by mkinfit

Source

Ranke (2014) Prüfung und Validierung von Modellierungssoftware als Alternative zu ModelMaker 4.0, Umweltbundesamt Projektnummer 27452

Examples

## Not run:
# This is a level P-II evaluation of the dataset according to the FOCUS kinetics
# guidance. Due to the strong correlation of the parameter estimates, the
# covariance matrix is not returned. Note that level P-II evaluations are
# generally considered deprecated due to the frequent occurrence of such

transform_odeparms

Functions to transform and backtransform kinetic parameters for fitting

Description

The transformations are intended to map parameters that should only take on restricted values to the full scale of real numbers. For kinetic rate constants and other parameters that can only take on positive values, a simple log transformation is used. For compositional parameters, such as the formations fractions that should always sum up to 1 and cannot be negative, the ilr transformation is used.

Usage

transform_odeparms(
  parms,
  mkinmod,
  transform_rates = TRUE,
  transform_fractions = TRUE
)

backtransform_odeparms(
  transparms,
  mkinmod,
)
transform_odeparms

transform_rates = TRUE,
transform_fractions = TRUE
)

Arguments

parms Parameters of kinetic models as used in the differential equations.
mkinmod The kinetic model of class mkinmod, containing the names of the model variables that are needed for grouping the formation fractions before ilr transformation, the parameter names and the information if the pathway to sink is included in the model.

transform_rates Boolean specifying if kinetic rate constants should be transformed in the model specification used in the fitting for better compliance with the assumption of normal distribution of the estimator. If TRUE, also alpha and beta parameters of the FOMIC model are log-transformed, as well as k1 and k2 rate constants for the DFOP and HS models and the break point tb of the HS model.

transform_fractions Boolean specifying if formation fractions constants should be transformed in the model specification used in the fitting for better compliance with the assumption of normal distribution of the estimator. The default (TRUE) is to do transformations. The g parameter of the DFOP model is also seen as a fraction. If a single fraction is transformed (g parameter of DFOP or only a single target variable e.g. a single metabolite plus a pathway to sink), a logistic transformation is used stats::qlogis(). In other cases, i.e. if two or more formation fractions need to be transformed whose sum cannot exceed one, the ilr transformation is used.

transparms Transformed parameters of kinetic models as used in the fitting procedure.

Details

The transformation of sets of formation fractions is fragile, as it supposes the same ordering of the components in forward and backward transformation. This is no problem for the internal use in mkinfit.

Value

A vector of transformed or backtransformed parameters

Author(s)

Johannes Ranke

Examples

SFO_SFO <- mkinmod(
  parent = list(type = "SFO", to = "m1", sink = TRUE),
  m1 = list(type = "SFO"), use_of_ff = "min")
# Fit the model to the FOCUS example dataset D using defaults
FOCUS_D <- subset(FOCUS_2006_D, value != 0)  # remove zero values to avoid warning
fit <- mkinfit(SFO_SFO, FOCUS_D, quiet = TRUE)
fit.s <- summary(fit)
# Transformed and backtransformed parameters
print(fit.s$par, 3)
print(fit.s$bpar, 3)

## Not run:
# Compare to the version without transforming rate parameters (does not work
# with analytical solution, we get NA values for m1 in predictions)
fit.2 <- mkinfit(SFO_SFO, FOCUS_D, transform_rates = FALSE,
  solution_type = "deSolve", quiet = TRUE)
fit.2.s <- summary(fit.2)
print(fit.2.s$par, 3)
print(fit.2.s$bpar, 3)

## End(Not run)
initials <- fit$start$value
names(initials) <- rownames(fit$start)
transformed <- fit$start_transformed$value
names(transformed) <- rownames(fit$start_transformed)
transform_odeparms(initials, SFO_SFO)
backtransform_odeparms(transformed, SFO_SFO)

## Not run:
# The case of formation fractions (this is now the default)
SFO_SFO.ff <- mkinmod(
  parent = list(type = "SFO", to = "m1", sink = TRUE),
  m1 = list(type = "SFO"),
  use_of_ff = "max")
fit.ff <- mkinfit(SFO_SFO.ff, FOCUS_D, quiet = TRUE)
fit.ff.s <- summary(fit.ff)
print(fit.ff.s$par, 3)
print(fit.ff.s$bpar, 3)
initials <- c("f_parent_to_m1" = 0.5)
transformed <- transform_odeparms(initials, SFO_SFO.ff)
backtransform_odeparms(transformed, SFO_SFO.ff)

# And without sink
SFO_SFO.ff.2 <- mkinmod(
  parent = list(type = "SFO", to = "m1", sink = FALSE),
  m1 = list(type = "SFO"),
  use_of_ff = "max")

fit.ff.2 <- mkinfit(SFO_SFO.ff.2, FOCUS_D, quiet = TRUE)
fit.ff.2.s <- summary(fit.ff.2)
print(fit.ff.2.s$par, 3)
print(fit.ff.2.s$bpar, 3)
update.mkinfit  

Update an mkinfit model with different arguments

Description

This function will return an updated mkinfit object. The fitted degradation model parameters from the old fit are used as starting values for the updated fit. Values specified as 'parms.ini' and/or 'state.ini' will override these starting values.

Usage

## S3 method for class 'mkinfit'
update(object, ..., evaluate = TRUE)

Arguments

object  
An mkinfit object to be updated

...  
Arguments to mkinfit that should replace the arguments from the original call. Arguments set to NULL will remove arguments given in the original call

evaluate  
Should the call be evaluated or returned as a call

Examples

## Not run:
fit <- mkinfit("SFO", subset(FOCUS_2006_D, value != 0), quiet = TRUE)
parms(fit)
plot_err(fit)
fit_2 <- update(fit, error_model = "tc")
parms(fit_2)
plot_err(fit_2)

## End(Not run)

[.mmkin  

Subsetting method for mmkin objects

Description

Subsetting method for mmkin objects

Usage

## S3 method for class 'mmkin'
x[i, j, ..., drop = FALSE]
Arguments

x     An `mmkin` object

i     Row index selecting the fits for specific models

j     Column index selecting the fits to specific datasets

...   Not used, only there to satisfy the generic method definition

drop  If FALSE, the method always returns an `mmkin` object, otherwise either a list of `mkinfit` objects or a single `mkinfit` object.

Value

An object of class `mmkin`.

Author(s)

Johannes Ranke

Examples

```r
# Only use one core, to pass R CMD check --as-cran
fits <- mmkin(c("SFO", "FOMC"), list(B = FOCUS_2006_B, C = FOCUS_2006_C),
              cores = 1, quiet = TRUE)
fits["FOMC", ]
fits[, "B"]
fits["SFO", "B"]

head(  # This extracts an `mkinfit` object with lots of components
fits[["FOMC", "B"]]
)
```
Index

* datasets
  D24_2014, 13
  dimethenamid_2018, 15
  experimental_data_for_UBA_2019, 18
  FOCUS_2006_datasets, 20
  FOCUS_2006_DFOP_ref_A_to_B, 21
  FOCUS_2006_FOMC_ref_A_to_F, 22
  FOCUS_2006_HS_ref_A_to_F, 23
  FOCUS_2006_SFO_ref_A_to_F, 24
  focus_soil_moisture, 25
  mccall81_245T, 39
  NAFTA_SOP_2015, 68
  NAFTA_SOP_Attachment, 68
  schaefer07_complex_case, 87
  synthetic_data_for_UBA_2014, 97
  test_data_from_UBA_2014, 100

* hplot
  mkinerrplot, 47

* manip
  ilr, 29
  mkin_wide_to_long, 64
  mkinerrmin, 46

* optimize
  mmkin, 64

* parent solutions
  DFOP.solution, 14
  FOMC.solution, 25
  HS.solution, 28
  IORE.solution, 31
  logistic.solution, 34
  SFO.solution, 88
  SFORB.solution, 89
  [..mmkin, 65, 81, 104

  add_err, 4
  AIC, 52
  AIC.mmkin, 6, 36
  array, 65
  aw, 7

  backtransform_odeparms
    (transform_odeparms), 101
  BIC.mmkin (AIC.mmkin), 6
  CAKE_export, 8
  confint.mkinfit, 9
  create_deg_func, 12

  D24_2014, 13
  deSolve::lsoda(), 51
  deSolve::ode(), 50–52
  DFOP.solution, 14, 26, 29, 32, 35, 89, 90
  dimethenamid_2018, 15
  dnorm, 36

  endpoints, 17
  experimental_data_for_UBA_2019, 18

  f_time_norm_focus, 26
  FOCUS_2006_A (FOCUS_2006_datasets), 20
  FOCUS_2006_B (FOCUS_2006_datasets), 20
  FOCUS_2006_C (FOCUS_2006_datasets), 20
  FOCUS_2006_D (FOCUS_2006_datasets), 20
  FOCUS_2006_datasets, 20
  FOCUS_2006_DFOP_ref_A_to_B, 21
  FOCUS_2006_E (FOCUS_2006_datasets), 20
  FOCUS_2006_F (FOCUS_2006_datasets), 20
  FOCUS_2006_FOMC_ref_A_to_F, 22
  FOCUS_2006_HS_ref_A_to_F, 23
  FOCUS_2006_SFO_ref_A_to_F, 24
  focus_soil_moisture, 25, 28
  FOMC.solution, 15, 25, 29, 32, 35, 89, 90

  get_deg_func, 28
  groupedData, 73

  HS.solution, 15, 26, 28, 32, 35, 89, 90

  ilr, 29, 101, 102
  inline::cfunction(), 55
  intervals.saem.mmkin, 31
INDEX

invilr (ilr), 29
IORE.solution, 15, 26, 29, 31, 35, 89, 90
legend, 48, 62, 80
loftest, 32
logistic.solution, 15, 26, 29, 32, 34, 89, 90
logLik, 36
logLik.mkinfit, 36, 52
lrtest, 52
lrtest.default, 32, 37
lrtest.mkinfit, 37
lrtest.mkinfit(lrtest.mkinfit), 37
lsoda, 60
makeCluster, 65
max_twa_dfop (max_twa_parent), 38
max_twa_fomc (max_twa_parent), 38
max_twa_hs (max_twa_parent), 38
max_twa_parent, 38
max_twa_sfo (max_twa_parent), 38
mccall81_245T, 39
mean_degparms, 40, 76, 85
mixed, 41
mixed.mkin, 76
mkin_long_to_wide, 63
mkin_wide_to_long, 64
mkinds, 19, 43, 43, 98, 100
mkindsg, 13, 15, 44, 44
mkinerrmin, 46
mkinerrplot, 47, 81
mkinfit, 4, 7, 8, 10, 17, 19, 36–38, 46, 47, 48,
56–59, 62, 64, 65, 78, 79, 83, 85, 92,
98, 100, 102, 104
mkinfit(), 55, 75
mkinmod, 17, 40, 48–50, 54, 55, 56, 59, 60, 65,
76, 85, 102
mkinparplot, 57
mkinplot, 48, 58, 63
mkinpredict, 4, 59, 60, 76
mkinpredict(), 48, 50
mkinresplot, 61, 81, 82
mkinsub (mkinmod), 54
mkinsub(), 54
mkin, 4, 6, 7, 36, 37, 42, 52, 64, 65–67, 70,
81, 84, 85, 105
mkin(), 75
mtext, 81
nafta, 66, 67, 69, 83

NAFTA_SOP_2015, 68
NAFTA_SOP_Appendix_B (NAFTA_SOP_2015),
68
NAFTA_SOP_Appendix_D (NAFTA_SOP_2015),
68
NAFTA_SOP_Attachment, 68
nlme, 94
nlme.mmkin, 17, 69, 73, 76, 93
nlme.mmkin(), 72
nlme::anova.lme(), 71
nlme::coef.lme(), 71
nlme::intervals(), 71
nlme::nlme, 71
nlme::pdDiag, 70
nlme_data(nlme::function), 72
nlme::function, 72
nlme::function(), 71
nobs.mkinfit, 74
ode, 60
ode solver from package deSolve, 50
parallel::detectCores(), 65
pars, 52, 74
pkgbuild::has_compiler(), 55
plot, 48, 62, 77, 80
plot.mixed.mkin, 71, 75, 86
plot.mkinfit, 47, 52, 58, 62, 78, 82
plot.mkinfit, 65, 81, 83
plot.nafta, 82
plot_err(plot.mkinfit), 78
plot_res, 63
plot_res(plot.mkinfit), 78
plot_sep(plot.mkinfit), 78
print.mixed.mkin(mixed), 41
print.mkindsg(mkindsg), 43
print.mkinmod(mkinmod), 54
print.mkin(mkin), 64
print.nafta(nafta), 66
print.nlme.mmkin(nlme.mmkin), 69
print.saem.mmkin(saem), 84
print.summary.mkinfit
(summary.mkinfit), 91
print.summary.nlme.mmkin
(summary.nlme.mmkin), 93
print.summary.saem.mmkin
(summary.saem.mmkin), 95
residuals.mkinfit, 83

saem, 84
saem.mmkin, 17, 76, 95
saemix::saemix, 85
saemix::saemix(), 84
saemix::SaemixData, 85
saemix::SaemixModel, 85
saemix::saemixModel, 85
saemix::saemixModel(), 85
saemix::SaemixObject, 85, 96
saemix_data(saem), 84
saemix_model(saem), 84
schaefer07_complex_case, 87
schaefer07_complex_results
    (schaefer07_complex_case), 87
SFO.solution, 15, 26, 29, 32, 34, 35, 88, 90
SFORB.solution, 15, 26, 29, 32, 35, 89, 89
sigma_twocomp, 90
stats::nlminb(), 48, 50
stats::qlogis(), 102
summary.mkinfit, 17, 46, 52, 91
summary.nlme.mmkin, 17, 71, 93, 93
summary.saem.mmkin, 17, 86, 95, 95
synthetic_data_for_UBA_2014, 97
test_data_from_UBA_2014, 100
tikz, 80, 82
transform_odeparms, 51, 101

update.mkinfit, 37, 104
update.nlme.mmkin(nlme.mmkin), 69