Package 'dMod'

June 11, 2015

Type Package

Title Dynamic Modeling and Parameter Estimation in ODE Models

Version 0.1

Date 2015-06-09

Author Daniel Kaschek

Maintainer Daniel Kaschek <daniel.kaschek@physik.uni-freiburg.de>

Description The framework provides functions to generate ODEs of reaction networks, parameter transformations, observation functions, residual functions, etc. The framework follows the paradigm that derivative information should be used for optimization whenever possible. Therefore, all major functions produce and can handle expressions for symbolic derivatives.

License GPL (>= 2)

Depends cOde, trust, ggplot2

Suggests MASS, inline, rPython, rootSolve

NeedsCompilation no

Repository CRAN

Date/Publication 2015-06-11 18:31:34

R topics documented:

`+.obj` *Add two lists element by element*

Description

Add two lists element by element

Usage

```
## S3 method for class 'obj'
out1 + out2
```

Arguments

out1	List of numerics or matrices
out2	List with the same structure as out1 (there will be no warning when mismatching)

Details

If out1 has names, out2 is assumed to share these names. Each element of the list out1 is inspected. If it has a names attributed, it is used to do a matching between out1 and out2. The same holds for the attributed dimnames. In all other cases, the "+" operator is applied the corresponding elements of out1 and out2 as they are.

Value

List of length of out1.

---

addObservable *Add observables to ODEs*

Description

Add observables to ODEs

Usage

```
addObservable(observable, f)
```

Arguments

observable	named character vector. Names correspond to observable names, the chars correspond to the observation function
f	equation list

Details

Observables are translated into an ODE and added to the list of equations

Value

An object of class eqnList, a named vector with the equations. Contains attributes "SMatrix" (the stoichiometric matrix), "species" (the state names), "rates" (the rate expressions) and "description".

Examples

```
reactions <- data.frame(Description = c("Activation", "Deactivation"),
                        Rate = c("act*A", "deact*pA"), A=c(-1,1), pA=c(1, -1))
f <- generateEquations(reactions)
myobs <- c(tA = "s1*(pA + A)", dA = "s2*(pA-A)")
f <- addObservable(myobs, f)
```

addReaction *Add reaction to reaction table*

Description

Add reaction to reaction table

Usage

```
addReaction(from, to, rate, f = NULL)
```

Arguments

from	character with the left hand side of the reaction, e.g. "2*A + B"
to	character with the right hand side of the reaction, e.g. "C + 2*D"
rate	named character. The rate associated with the reaction. The name is employed as a description of the reaction.
f	equation list, see generateEquations

Value

An object of class eqnList, a named vector with the equations. Contains attributes "SMatrix" (the stoichiometric matrix), "species" (the state names), "rates" (the rate expressions) and "description".

Examples

```
## Not run:
f <- addReaction("2*A+B", "C + 2*D", "k1*B*A^2", NULL)
f <- addReaction("C + A", "B + A", "k2*C*A", f)

## End(Not run)
```

as.obj *Generate dummy list of class* obj *from named numeric*

Description

Generate dummy list of class obj from named numeric

Usage

```
as.obj(p)
```

Arguments

p	Names numeric vector

Value

list with entries value (`0`), gradient (`rep(0, length(p))`) and hessian (`matrix(0, length(p), length(p))`) of class `obj`.

Examples

```
p <- c(A = 1, B = 2)
as.obj(p)
```

---

blockdiagSymb *Embed two matrices into one blockdiagonal matrix*

Description

Embed two matrices into one blockdiagonal matrix

Usage

```
blockdiagSymb(M, N)
```

Arguments

`M`	matrix of type character
`N`	matrix of type character

Value

Matrix of type character containing M and N as upper left and lower right block

---

combine *Combine several data.frames from csv read-in*

Description

Combine several data.frames from csv read-in

Usage

```
combine(...)
```

Arguments

`...`	data.frames from csv read-in

Details

This function is useful when separating the model into independent csv model files, e.g.~a receptor model and several downstream paths. Then, the models can be recombined into one model by `combine()`.

Value

A `data.frame` with columns "Description", "Rate" and one column for each ODE state in the model. See generateEquations.

`constraintExp2` *Compute a differentiable box prior*

Description

Compute a differentiable box prior

Usage

```
constraintExp2(p, mu, sigma = 1, k = 0.05, fixed = NULL)
```

Arguments

`p`	Named numeric, the parameter value
`mu`	Named numeric, the prior values, means of boxes
`sigma`	Named numeric, half box width
`k`	Named numeric, shape of box; if 0 a quadratic prior is obtained, the higher k the more box shape, gradient at border of the box (-sigma, sigma) is equal to sigma*k
`fixed`	Named numeric with fixed parameter values (contribute to the prior value but not to gradient and Hessian)

Value

list with entries: value (numeric, the weighted residual sum of squares), gradient (numeric, gradient) and hessian (matrix of type numeric).

`constraintL2` *Soft L2 constraint on parameters*

Description

Soft L2 constraint on parameters

Usage

```
constraintL2(p, mu, sigma = 1, fixed = NULL)
```

Arguments

`p`	Namec numeric, the parameter value
`mu`	Named numeric, the prior values
`sigma`	Named numeric of length of mu or numeric of length one.
`fixed`	Named numeric with fixed parameter values (contribute to the prior value but not to gradient and Hessian)

Details

Computes the constraint value

$$\frac{1}{2}\left(\frac{p-\mu}{\sigma}\right)^2$$

and its derivatives with respect to p.

Value

List of class `obj`, i.e. objective value, gradient and Hessian as list.

See Also

wrss

Examples

```
p <- c(A = 1, B = 2, C = 3)
mu <- c(A = 0, B = 0)
sigma <- c(A = 0.1, B = 1)
constraintL2(p, mu, sigma)
```

`datapointL2` *L2 objective function for validation data point*

Description

L2 objective function for validation data point

Usage

```
datapointL2(p, prediction, mu, time = 0, sigma = 1, fixed = NULL)
```

Arguments

`p`	Namec numeric, the parameter values
`prediction`	Matrix with first column "time" and one column per predicted state. Can have an attribute `deriv`, the matrix of sensitivities. If present, derivatives of the objective function with respect to the parameters are returned.
`mu`	Named character of length one. Has the structure `mu = c(parname = statename)`, where `statename` is one of the column names of `prediction` and `parname` is one of the names of `p`, allowing to treat the validation data point as a parameter.
`time`	Numeric of length one. An existing time point in `prediction`.
`sigma`	Numeric of length one. The uncertainty assumed for the validation data point.
`fixed`	Named numeric with fixed parameter values (contribute to the prior value but not to gradient and Hessian)

Details

Computes the constraint value

$$\left(\frac{x(t) - \mu}{\sigma}\right)^2$$

and its derivatives with respect to p.

Value

List of class `obj`, i.e. objective value, gradient and Hessian as list.

See Also

wrss, constraintL2

Examples

```
## Not run:
prediction <- matrix(c(0, 1), nrow = 1, dimnames = list(NULL, c("time", "A")))
derivs <- matrix(c(0, 1, 0.1), nrow = 1, dimnames = list(NULL, c("time", "A.A", "A.k1")))
attr(prediction, "deriv") <- derivs
p0 <- c(A = 1, k1 = 2)
mu <- c(newpoint = "A")
timepoint <- 0

datapointL2(p = c(p, newpoint = 2), prediction, mu, timepoint)
datapointL2(p = c(p, newpoint = 1), prediction, mu, timepoint)
datapointL2(p = c(p, newpoint = 0), prediction, mu, timepoint)

## End(Not run)
```

---

expand.grid.alt *Alternative version of expand.grid*

Description

Alternative version of expand.grid

Usage

```
expand.grid.alt(seq1, seq2)
```

Arguments

seq1	Vector, numeric or character
seq2	Vector, numeric or character

Value

Matrix ob combinations of elemens of seq1 and seq2

---

forcingsSymb *Return some useful forcing functions as strings*

Description

Return some useful forcing functions as strings

Usage

```
forcingsSymb(type = c("Gauss", "Fermi", "1-Fermi", "MM", "Signal"),
  parameters = NULL)
```

Arguments

type	Which function to be returned
parameters	Named vector, character or numeric. Replace parameters by the corresponding valus in parameters.

Value

String with the function

---

funC0 *Evaluation of algebraic expressions defined by characters*

Description

Evaluation of algebraic expressions defined by characters

Usage

```
funC0(x, compile = TRUE)
```

Arguments

x	Name character vector, the algebraic expressions
compile	Logical. The function is either compiled (requires the inline package) or evaluated in raw R.

Value

A prediction function f(mylist) where mylist is a list of numeric vectors that can be coerced into a matrix. The names correspond to the symbols used in the algebraic expressions. The function f returns a matrix.

Examples

```
## Not run:
myfun <- funC0(c(x = "x", y = "a*x^4 + b*x^2 + c"))
out <- myfun(list(a = -1, b = 2, c = 3, x = seq(-2, 2, .1)))
plot(out[, 1], out[, 2])

## End(Not run)
```

generateEquations *Generate equations from data.frame(s)*

Description

Generate equations from data.frame(s)

Usage

```
generateEquations(..., volumes = NULL)
```

Arguments

...	one or more data.frames with columns "Description" (character), "Rate" (character), and one column per ODE state with the state names. The state columns correspond to the stoichiometric matrix.
volumes	Named character, volume parameters for species. Names must be a subset of the species. Values can be either characters, e.g. "V1", or numeric values for the volume. If volumes is not NULL, missing species are treated like 1.

Details

This function is supposed to translate a reaction network as being defined in a csv file into the raw equations.

Value

An object of class eqnList, a named vector with the equations. Contains attributes "SMatrix" (the stoichiometric matrix), "species" (the state names), "rates" (the rate expressions) and "description".

Examples

```
#######################################
# Example 1
#######################################
reactions <- data.frame(Description = c("Activation", "Deactivation"),
                        Rate = c("act*A", "deact*pA"), A=c(-1,1), pA=c(1, -1) )
f <- generateEquations(reactions)
f

#######################################
# Example 2
#######################################
reactions <- data.frame(Description = c("Activation", "Deactivation", "Production", "Degradation"),
                        Rate = c("act*A", "deact*pA", "prod", "degrad*pA"),
                        A=c(-1,1, 1, NA),
                        pA=c(1, -1, NA, -1))
volumes <- c(A = "V1", pA = "V2")
f <- generateEquations(reactions, volumes = volumes)
```

```
f[1:length(f)]
```

`generateModel` *Generate the model objects for use in Xs (models with sensitivities)*

Description

Generate the model objects for use in Xs (models with sensitivities)

Usage

```
generateModel(f, forcings = NULL, fixed = NULL, modelname = "f", ...)
```

Arguments

`f`	Named character vector with the ODE
`forcings`	Character vector with the names of the forcings
`fixed`	Character vector with the names of parameters (initial values and dynamic) for which no sensitivities are required (will speed up the integration).
`modelname`	Character, the name of the C file being generated.
`...`	Further arguments being passed to funC.

Value

list with `func` (ODE object) and `extended` (ODE+Sensitivities object)

`getCoefficients` *Get coefficients from a character*

Description

Get coefficients from a character

Usage

```
getCoefficients(char, symbol)
```

Arguments

`char`	character, e.g. "2*x + y"
`symbol`	single character, e.g. "x" or "y"

Value

numeric vector with the coefficients

Examples

```
getCoefficients("2*x + x + y", "x")
```

---

lbind *Bind named list of data.frames into one data.frame*

Description

Bind named list of data.frames into one data.frame

Usage

```
lbind(mylist)
```

Arguments

mylist	A named list of data.frame. The data.frames are expected to have the same structure.

Details

Each data.frame ist augented by a "condition" column containing the name attributed of the list entry. Subsequently, the augmented data.frames are bound together by `rbind`.

Value

data.frame with the originial columns augmented by a "condition" column.

---

loadTemplate *Load a template file in the editor*

Description

Load a template file in the editor

Usage

```
loadTemplate(i = 1)
```

Arguments

i	Integer, choose a template to be loaded

Details

Possible templates are: i = 1: Do parameter estimation in a dynamic model with fixed forcings

long2wide *Translate long to wide format (inverse of wide2long.matrix)*

Description

Translate long to wide format (inverse of wide2long.matrix)

Usage

```
long2wide(out)
```

Arguments

out	data.frame in long format

Value

data.frame in wide format

P *Parameter transformation*

Description

Parameter transformation

Usage

```
P(trafo, parameters = NULL, compile = FALSE)
```

Arguments

trafo	Named character vector. Names correspond to the parameters being fed into the model (the inner parameters). The elements of tafo are equations that express the inner parameters in terms of other parameters (the outer parameters)
parameters	Character vector. Optional. If given, the generated parameter transformation returns values for each element in parameters. If elements of parameters are not in names(trafo) the identity transformation is assumed.
compile	Logical, compile the function (see funC0)

Value

a function p2p(p, fixed = NULL, deriv = TRUE) representing the parameter transformation. Here, p is a named numeric vector with the values of the outer parameters, fixed is a named numeric vector with values of the outer parameters being considered as fixed (no derivatives returned) and deriv is a logical determining whether the Jacobian of the parameter transformation is returned as attribute "deriv".

See Also

Pi for implicit parameter transformations and concatenation for the concatenation of parameter transformations

Examples

```
## Not run:
logtrafo <- c(k1 = "exp(logk1)", k2 = "exp(logk2)", A = "exp(logA)", B = "exp(logB)")
P.log <- P(logtrafo)

p.outerValue <- c(logk1 = 1, logk2 = -1, logA = 0, logB = 0)
(P.log)(p.outerValue)

## End(Not run)
```

`Pi` *Parameter transformation (implicit)*

Description

Parameter transformation (implicit)

Usage

```
Pi(trafo, parameters = NULL, compile = FALSE)
```

Arguments

`trafo`	Named character vector defining the equations to be set to zero. Names correspond to dependent variables.
`parameters`	Character vector, the independent variables.
`compile`	Logical, compile the function (see funC0)

Details

Usually, the equations contain the dependent variables, the independent variables and other parameters. The argument `p` of `p2p` must provide values for the independent variables and the parameters but ALSO FOR THE DEPENDENT VARIABLES. Those serve as initial guess for the dependent variables. The dependent variables are then numerically computed by multiroot. The Jacobian of the solution with respect to dependent variables and parameters is computed by the implicit function theorem. The function `p2p` returns all parameters as they are with corresponding 1-entries in the Jacobian. #'

Value

a function p2p(p, fixed = NULL, deriv = TRUE) representing the parameter transformation. Here, p is a named numeric vector with the values of the outer parameters, fixed is a named numeric vector with values of the outer parameters being considered as fixed (no derivatives returned) and deriv is a logical determining whether the Jacobian of the parameter transformation is returned as attribute "deriv".

See Also

P for explicit parameter transformations and concatenation for the concatenation of parameter transformations

Examples

```
## Not run:
########################################################################
## Example 1: Steady-state trafo
########################################################################
f <- c(A = "-k1*A + k2*B",
       B = "k1*A - k2*B")
P.steadyState <- Pi(f, "A")

p.outerValues <- c(k1 = 1, k2 = 0.1, A = 10, B = 1)
P.steadyState(p.outerValues)

########################################################################
## Example 2: Steady-state trafo combined with log-transform
########################################################################
f <- c(A = "-k1*A + k2*B",
       B = "k1*A - k2*B")
P.steadyState <- Pi(f, "A")

logtrafo <- c(k1 = "exp(logk1)", k2 = "exp(logk2)", A = "exp(logA)", B = "exp(logB)")
P.log <- P(logtrafo)

p.outerValue <- c(logk1 = 1, logk2 = -1, logA = 0, logB = 0)
(P.log)(p.outerValue)
(P.steadyState %o% P.log)(p.outerValue)

## End(Not run)
```

---

plotCombined *Plot a list of model predictions and a list of data points in a combined plot*

Description

Plot a list of model predictions and a list of data points in a combined plot

Usage

```
plotCombined(prediction, data, ...)
```

Arguments

prediction	Named list of matrices or data.frames, usually the output of a prediction function as generated by Xs.
data	Named list of data.frames as being used in res, i.e. with columns name, time, value and sigma.
...	Further arguments going to subset.

Details

The data.frame being plotted has columns time, value, sigma, name and condition.

Value

A plot object of class ggplot.

---

plotData *Plot a list data points*

Description

Plot a list data points

Usage

```
plotData(data, ...)
```

Arguments

data	Named list of data.frames as being used in res, i.e. with columns name, time, value and sigma.
...	Further arguments going to subset.

Details

The data.frame being plotted has columns time, value, sigma, name and condition.

Value

A plot object of class ggplot.

plotPaths *Profile likelihood: plot of the parameter paths.*

Description

Profile likelihood: plot of the parameter paths.

Usage

```
plotPaths(..., whichPar = NULL, sort = FALSE)
```

Arguments

...	Lists of profiles as being returned by profile.
whichPar	Character, the name of the parameter taken as a reference (x-axis)
sort	Logical. If paths from different parameter profiles are plotted together, possible combinations are either sorted or all combinations are taken as they are.

Value

A plot object of class ggplot.

plotPrediction *Plot a list of model predictions*

Description

Plot a list of model predictions

Usage

```
plotPrediction(prediction, ...)
```

Arguments

prediction	Named list of matrices or data.frames, usually the output of a prediction function as generated by Xs.
...	Further arguments going to subset.

Details

The data.frame being plotted has columns time, value, name and condition.

Value

A plot object of class ggplot.

plotProfile *Profile likelihood plot*

Description

Profile likelihood plot

Usage

```
plotProfile(..., maxvalue = 5)
```

Arguments

...	Lists of profiles as being returned by profile.
maxvalue	Numeric, the value where profiles are cut off.

Value

A plot object of class ggplot.

print.eqnList *Print function for eqnList*

Description

Print function for eqnList

Usage

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

Arguments

x	equation list
...	Argument not used right now

print0 *Print without attributes*

Description

Print without attributes

Usage

```
print0(x, list_attributes = TRUE)
```

Arguments

x	The object to be printed out
list_attributes	prints a list of attribute names if TRUE (default=TRUE)

Details

To suppress the printout of attributes like "deriv".

priorL2 *L2 objective function for prior value*

Description

As a prior function, it returns derivatives with respect to the penalty parameter in addition to parameter derivatives.

Usage

```
priorL2(p, mu, lambda = "lambda", fixed = NULL)
```

Arguments

p	Namec numeric, the parameter value
mu	Named numeric, the prior values
lambda	Character of length one. The name of the penalty paramter in p.
fixed	Named numeric with fixed parameter values (contribute to the prior value but not to gradient and Hessian)

Details

Computes the constraint value

$$\lambda \|p - \mu\|^2$$

and its derivatives with respect to p and lambda.

Value

List of class obj, i.e. objective value, gradient and Hessian as list.

See Also

wrss, constraintExp2

Examples

```
p <- c(A = 1, B = 2, C = 3, lambda = 1)
mu <- c(A = 0, B = 0)
priorL2(p, mu, lambda = "lambda")
```

---

profile *Profile-likelihood (PL) computation*

Description

Profile-likelihood (PL) computation

Usage

```
profile(obj, pars, whichPar, alpha = 0.05, limits = c(lower = -Inf, upper =
  Inf), stepControl = NULL, algoControl = NULL, optControl = NULL,
  verbose = FALSE, ...)
```

Arguments

obj	Objective function obj(pars, fixed, ...) returning a list with "value", "gradient" and "hessian".
pars	Parameter vector corresponding to the log-liklihood optimum.
whichPar	Numeric or character. The parameter for which the profile is computed.
alpha	Numeric, the significance level based on the chisquare distribution with df=1
limits	Numeric vector of length 2, the lower and upper deviance from the original value of pars[whichPar]
stepControl	List of arguments controlling the step adaption. Defaults to list(stepsize = 1e-4, min = 0, max = In
algoControl	List of arguments controlling the fast PL algorithm. defaults to list(gamma = 1, W = c("hessian", "i
optControl	List of arguments controlling the trust() optimizer. Defaults to list(rinit = .1, rmax = 10, iterli See trust for more details.
verbose	Logical, print verbose messages.
...	Arguments going to obj()

Details

Computation of the profile likelihood is based on the method of Lagrangian multipliers and Euler integration of the corresponding differential equation of the profile likelihood paths.

`algoControl`: Since the Hessian which is needed for the differential equation is frequently misspecified, the error in integration needs to be compensated by a correction factor `gamma`. Instead of the Hessian, an identity matrix can be used. To guarantee that the profile likelihood path stays on the true path, each point proposed by the differential equation can be used as starting point for an optimization run when `reoptimize = TRUE`. The correction factor `gamma` is adapted based on the amount of actual correction. If this exceeds the value `correction`, `gamma` is reduced. In some cases, the Hessian becomes singular. This leads to problems when inverting the Hessian. To avoid this problem, the matrix `reg*Id` is added to the Hessian.

`stepControl`: The Euler integration starts with `stepsize`. In each step the predicted change of the objective function is compared with the actual change. If this is larger than `atol`, the stepsize is reduced. For small deviations, either compared the abolute tolerance `atol` or the relative tolerance `rtol`, the stepsize may be increased. `max` and `min` are upper and lower bounds for `stepsize`. `limit` is the maximum number of steps that are take for the profile computation.

Value

Named list of length one. The name is the parameter name. The list enty is a matrix with columns "value" (the objective value), "constraint" (deviation of the profiled paramter from the original value), "stepsize" (the stepsize take for the iteration), "gamma" (the gamma value employed for the iteration), one column per parameter (the profile paths).

Examples

```
## Not run:
## ----------------------
## Example 1
## ----------------------
trafo <- c(a = "exp(loga)", b = "exp(logb)",c = "exp(loga)*exp(logb)*exp(logc)")
p <- P(trafo)
obj <- function(pOuter, fixed = NULL)
   constraintL2(p(pOuter, fixed), c(a =.1, b = 1, c = 10), 1)

ini <- c(loga = 1, logb = 1, logc = 1)
myfit <- trust(obj, ini, rinit=1, rmax=10)
profiles <- sapply(1:3, function(i)
   profile(obj, myfit$argument, whichPar = i, limits = c(-5, 5),
                algoControl=list(gamma=1, reoptimize=FALSE), verbose=TRUE))
plotProfile(profiles)
plotPaths(profiles)

## ----------------------------
## Example 2
## ----------------------------
trafo <- c(a = "exp(loga)", b = "exp(logb)",c = "exp(loga)*exp(logb)*exp(logc)")
p <- P(trafo)
obj <- function(pOuter, fixed = NULL, sigma)
  constraintL2(p(pOuter, fixed), c(a =.1, b = 1, c = 10), 1) +
```

```
  constraintL2(pOuter, mu = c(loga = 0, logb = 0), sigma = sigma, fixed = fixed)


ini <- c(loga = 1, logb = 1, logc = 1)
myfit <- trust(obj, ini[-1], rinit=1, rmax=10, fixed = ini[1], sigma = 10)
profiles.approx <- sapply(1:2, function(i)
  profile(obj, myfit$argument, whichPar = i, limits = c(-10, 10),
               algoControl=list(gamma=1, reoptimize=FALSE),
               verbose=TRUE, fixed = ini[1], sigma = 10))
profiles.exact  <- sapply(1:2, function(i)
  profile(obj, myfit$argument, whichPar = i, limits = c(-10, 10),
               algoControl=list(gamma=0, reoptimize=TRUE),
               verbose=TRUE, fixed = ini[1], sigma = 10))

plotProfile(profiles.approx, profiles.exact)

## End(Not run)
```

`progressBar` *Progress bar*

Description

Progress bar

Usage

```
progressBar(percentage, size = 50, number = TRUE)
```

Arguments

`percentage`	Numeric between 0 and 100
`size`	Integer, the size of the bar print-out
`number`	Logical, Indicates whether the percentage should be printed out.

`res` *Compare data and model prediction by computing residuals*

Description

Compare data and model prediction by computing residuals

Usage

```
res(data, out)
```

Arguments

data	data.frame with name (factor), time (numeric), value (numeric) and sigma (numeric)
out	output of ode(), optionally augmented with attributes "deriv" (output of ode() for the sensitivity equations) and "parameters" (character vector of parameter names, a subsest of those contained in the sensitivity equations). If "deriv" is given, also "parameters" needs to be given.

Value

data.frame with the original data augmented by columns "prediction" (numeric, the model prediction), "residual" (numeric, difference between prediction and data value), "weighted.residual" (numeric, residual devided by sigma). If "deriv" was given, the returned data.frame has an attribute "deriv" (data.frame with the derivatives of the residuals with respect to the parameters).

---

resolveRecurrence *Place top elements into bottom elemens*

Description

Place top elements into bottom elemens

Usage

```
resolveRecurrence(variables)
```

Arguments

variables	named character vector

Details

If the names of top vector elements occur in the bottom of the vector, they are replaced by the character of the top entry. Useful for steady state conditions.

Value

named character vector of the same length as `variables`

Examples

```
resolveRecurrence(c(A = "k1*B/k2", C = "A*k3+k4", D="A*C*k5"))
```

subset.eqnList *subset of an equation list*

Description

subset of an equation list

Usage

```
## S3 method for class 'eqnList'
subset(x, ...)
```

Arguments

x	the equation list
...	logical expression for subsetting

Details

The argument ... can contain "Educt", "Product", "Rate" and "Description". The "

Value

An object of class eqnList, a named vector with the equations. Contains attributes "SMatrix" (the stoichiometric matrix), "species" (the state names), "rates" (the rate expressions) and "description".

Examples

```
reactions <- data.frame(Description = c("Activation", "Deactivation"),
                        Rate = c("act*A", "deact*pA"), A=c(-1,1), pA=c(1, -1) )
f <- generateEquations(reactions)
subset(f, "A"%in%Educt)
subset(f, "pA"%in%Product)
subset(f, grepl("act", Rate))
```

symmetryDetection *Search for symmetries in the loaded model*

Description

Search for symmetries in the loaded model

Usage

```
symmetryDetection(f, obsvect = NULL, prediction = NULL, initial = NULL,
  ansatz = "uni", pMax = 2, inputs = c(), fixed = c(), cores = 1,
  allTrafos = FALSE)
```

Arguments

f	eqnList object or vector containing ODEs
obsvect	vector of observation functions
prediction	vector containing prediction to be tested
initial	vector containing initial values
ansatz	type of infinitesimal ansatz used for the analysis (uni, par, multi)
pMax	maximal degree of infinitesimal ansatz
inputs	specify the input variables
fixed	variables to concider fixed
cores	maximal number of cores used for the analysis
allTrafos	do not remove transformations with a common parameter factor

---

variableTransformation

Do a variable transformation in the ODE

Description

Do a variable transformation in the ODE

Usage

```
variableTransformation(observables, f = NULL, dynvar = NULL, stoi = NULL,
  flows = NULL, conserved = TRUE)
```

Arguments

observables	Named character vector. The names are the new variable names, the vector entries define the new variables in terms of the old ones.
f	An object of class eqnList, see generateEquations.
dynvar	Character vector with the old variable names
stoi	The stoichiometric matrix
flows	Character vector with the rate expressions
conserved	Logical. If true, the conserved quantities derived from the stoichiometric matrix are automatically used for extending the vector of observables. See details.

Details

Usually, the function is called by either using the f argument and leaving the other arguments NULL or by leaving f NULL and defining the ODE by the arguments dynvar, stoi and flows. The observables vector can have less entries than the vector dynvar. In this case, the observables are automatically extended by old variables to generate a full rank variable transformation. If conserved is TRUE, the conserved quantities are preferentially used to extend the observables vector. Consequently, the transformed equations will return a certain number of zero-equation.

Value

Named character vector with the ODE expressed in the new variables. In addition, attributes "variables" (the variable transformation) and "inverse" (the inverse transformation) are returned.

`wide2long` *Translate wide output format (e.g. from ode) into long format*

Description

Translate wide output format (e.g. from ode) into long format

Usage

```
wide2long(out, keep, na.rm)
```

Arguments

`out`	data.frame or matrix or list of matrices in wide format
`keep`	Index vector, the columns to keep
`na.rm`	Logical, if `TRUE`, missing values are removed in the long format.

Details

The function assumes that out[,1] represents a time-like vector whereas out[,-1] represents the values. Useful for plotting with ggplot. If a list is supplied, the names of the list are added as extra column names "condition"

Value

data.frame in long format, i.e. columns "time" (out[,1]), "name" (colnames(out[,-1])), "value" (out[,-1]) and, if out was a list, "condition" (names(out))

`wide2long.data.frame` *Translate wide output format (e.g. from ode) into long format*

Description

Translate wide output format (e.g. from ode) into long format

Usage

```
## S3 method for class 'data.frame'
wide2long(out, keep = 1, na.rm = FALSE)
```

Arguments

out	data.frame or matrix or list of matrices in wide format
keep	Index vector, the columns to keep
na.rm	Logical, if TRUE, missing values are removed in the long format.

Details

The function assumes that out[,1] represents a time-like vector whereas out[,-1] represents the values. Useful for plotting with ggplot. If a list is supplied, the names of the list are added as extra column names "condition"

Value

data.frame in long format, i.e. columns "time" (out[,1]), "name" (colnames(out[,-1])), "value" (out[,-1]) and, if out was a list, "condition" (names(out))

---

wide2long.list *Translate wide output format (e.g. from ode) into long format*

Description

Translate wide output format (e.g. from ode) into long format

Usage

```
## S3 method for class 'list'
wide2long(out, keep = 1, na.rm = FALSE)
```

Arguments

out	list of matrices in wide format
keep	Index vector, the columns to keep
na.rm	Logical, if TRUE, missing values are removed in the long format.

Details

The function assumes that out[,1] represents a time-like vector whereas out[,-1] represents the values. Useful for plotting with ggplot. If a list is supplied, the names of the list are added as extra column names "condition"

Value

data.frame in long format, i.e. columns "time" (out[,1]), "name" (colnames(out[,-1])), "value" (out[,-1]) and, if out was a list, "condition" (names(out))

wide2long.matrix *Translate wide output format (e.g. from ode) into long format*

Description

Translate wide output format (e.g. from ode) into long format

Usage

```
## S3 method for class 'matrix'
wide2long(out, keep = 1, na.rm = FALSE)
```

Arguments

out	data.frame or matrix or list of matrices in wide format
keep	Index vector, the columns to keep
na.rm	Logical, if TRUE, missing values are removed in the long format.

Details

The function assumes that out[,1] represents a time-like vector whereas out[,-1] represents the values. Useful for plotting with ggplot. If a list is supplied, the names of the list are added as extra column names "condition"

Value

data.frame in long format, i.e. columns "time" (out[,1]), "name" (colnames(out[,-1])), "value" (out[,-1]) and, if out was a list, "condition" (names(out))

wrss *Compute the weighted residual sum of squares*

Description

Compute the weighted residual sum of squares

Usage

```
wrss(nout)
```

Arguments

nout	data.frame (result of res)

Value

list with entries value (numeric, the weighted residual sum of squares), gradient (numeric, gradient) and hessian (matrix of type numeric).

Xf *Model evaluation without sensitivities.*

Description

Interface to get an ODE into a model function `x(times, pars, forcings, events)` returning ODE output. It is a reduced version of Xs, missing the sensitivities.

Usage

```
Xf(func, forcings = NULL, events = NULL, optionsOde = list(method =
  "lsoda"))
```

Arguments

`func`	return value from `funC(f)` where `f` defines the ODE.
`forcings`	data.frame with columns name (factor), time (numeric) and value (numeric). The ODE forcings.
`events`	data.frame of events with columns "var" (character, the name of the state to be affected), "time" (numeric, time point), "value" (numeric, value), "method" (character, either "replace", "add" or "multiply"). See events.
`optionsOde`	list with arguments to be passed to odeC() for the ODE integration.

Details

Can be used to integrate additional quantities, e.g. fluxes, by adding them to `f`. All quantities that are not initialised by pars in `x(times, pars, forcings, events)` are initialized at 0.

Xs *Model evaluation.*

Description

Interface to combine an ODE and its sensitivity equations into one model function `x(times, pars, forcings, events, de` returning ODE output and sensitivities.

Usage

```
Xs(func, extended, forcings = NULL, events = NULL,
  optionsOde = list(method = "lsoda"), optionsSens = list(method =
  "lsodes"))
```

Arguments

func	return value from funC(f) where f defines the ODE.
extended	return value from funC(c(f, sensitivitiesSymb(f))).
forcings	data.frame with columns name (factor), time (numeric) and value (numeric). The ODE forcings.
events	data.frame of events with columns "var" (character, the name of the state to be affected), "time" (numeric, time point), "value" (numeric, value), "method" (character, either "replace", "add" or "multiply"). See events.
optionsOde	list with arguments to be passed to odeC() for the ODE integration.
optionsSens	list with arguments to be passed to odeC() for integration of the extended system

Value

A model prediction function x(times, pars, forcings, events, deriv = TRUE) representing the model evaluation. The result of x(times, pars, forcings, events, deriv = TRUE) contains attributes "sensitivities" and "deriv" with the sensitivities if deriv=TRUE. If deriv=FALSE, sensitivities are not computed (saving time). If pars is the result of p(pouter) (see P), the Jacobian of the parameter transformation and the sensitivities of the ODE are multiplied according to the chain rule for differentiation. The result is saved in the attributed "deriv", i.e. in this case the attibutes "deriv" and "sensitivities" do not coincide.

Y *Observation functions.*

Description

Creates a function y(out, pars) that evaluates an observation function and its derivatives based on the output of a model function x(times, pars), see Xf and Xs.

Usage

```
Y(g, f, compile = FALSE)
```

Arguments

g	Named character vector defining the observation function
f	Named character, the underlying ODE
compile	Logical, compile the function (see funC0)

Value

a function y(out, pars, attach=FALSE) representing the evaluation of the observation function. If out has the attribute "sensitivities", the result of y(out, pars), will have an attributed "deriv" which reflec the sensitivities of the observation with respect to the parameters. If pars is the result of a parameter transformation p(pars) (see P), the Jacobian of the parameter transformation and the sensitivities of the observation function are multiplied according to the chain rule for differentiation. If attach = TRUE, the original argument out will be attached to the evaluated observations.

%o% *Concatenation of parameter transformations*

Description

Concatenation of parameter transformations

Usage

```
p1 %o% p2
```

Arguments

p1	Return value of P or Pi
p2	Return value of P or Pi

Value

A function `p2p(p, fixed = NULL, deriv = TRUE)`, the concatenation of `p1` and `p2`.

Examples

```
## Not run:
#' ########################################################################
## Example: Steady-state trafo combined with log-transform
########################################################################
f <- c(A = "-k1*A + k2*B",
       B = "k1*A - k2*B")
P.steadyState <- Pi(f, "A")

logtrafo <- c(k1 = "exp(logk1)", k2 = "exp(logk2)", A = "exp(logA)", B = "exp(logB)")
P.log <- P(logtrafo)

p.outerValue <- c(logk1 = 1, logk2 = -1, logA = 0, logB = 0)
(P.log)(p.outerValue)
(P.steadyState %o% P.log)(p.outerValue)

## End(Not run)
```

Index