

Package ‘dMod’

September 8, 2017

Type Package

Title Dynamic Modeling and Parameter Estimation in ODE Models

Version 0.4

Date 2017-09-05

Author Daniel Kaschek

Maintainer Daniel Kaschek <daniel.kaschek@gmail.com>

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 (>= 0.3),

Imports deSolve, rootSolve, ggplot2, parallel, stringr, plyr

Suggests MASS, rPython, pander, knitr, rmarkdown

RoxygenNote 5.0.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-08 21:59:18 UTC

R topics documented:

*.fn	4
+.datalist	5
+.fn	7
+.objfn	8
+.objlist	9
addReaction	10
as.data.frame.datalist	12
as.data.frame.eqnlist	12

as.eqnvec	13
as.objlist	14
as.parframe	14
as.parvec.parframe	15
attrs	16
blockdiagSymb	17
combine	18
compare	18
compile	20
confint.parframe	20
conservedQuantities	21
constraintExp2	22
constraintL2	23
controls	24
coordTransform	25
datalist	26
datapointL2	27
detectFreeCores	29
dot	29
eqnlist	30
eqnvec	32
expand.grid.alt	33
fitErrorModel	33
forcingsSymb	35
format.eqnvec	35
funC0	36
getCoefficients	37
getConditions	37
getDerivs	38
getFluxes	38
getLocalDLLs	40
getParameters	40
getReactions	41
ggopen	42
jakstat	43
lbind	43
load.parlist	44
loadDLL	44
long2wide	45
lsdMod	45
mname	46
modelName	46
mssample	47
mstrust	48
nll	52
normL2	53
nullZ	54
objframe	55

steadyStates	99
strelide	100
strpad	101
submatrix	102
subset.eqnlist	102
summary.eqnvec	103
symmetryDetection	103
theme_dMod	104
trustL1	105
wide2long	107
wide2long.data.frame	107
wide2long.list	108
wide2long.matrix	109
write.eqnlist	109
wrss	110
Xd	110
Xf	111
Xs	112
Xt	116
Y	117
%.*%	118

Index**120**

*.fn	<i>Concatenation of functions</i>
------	-----------------------------------

Description

Used to concatenate observation functions, prediction functions and parameter transformation functions.

Usage

```
## S3 method for class 'fn'
p1 * p2
```

Arguments

p1	function of class obsfn, prdfn or parfn
p2	function of class obsfn, prdfn or parfn

Value

Object of the same class as x1 and x2.

Examples

```

# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)

```

+.datalist

*Direct sum of datasets***Description**

Used to merge datasets with overlapping conditions.

Usage

```

## S3 method for class 'datalist'
data1 + data2

```

Arguments

data1	dataset of class datalist
data2	dataset of class datalist

Details

Each data list contains data frames for a number of conditions. The direct sum of datalist is meant as merging the two data lists and returning the overarching datalist.

Value

Object of class datalist for the union of conditions.

Examples

```
# Start with two data frames
mydata1 <- data.frame(
  name = "A",
  time = 0:1,
  value = 1:2,
  sigma = .1,
  compound = c("DEM", "APAP"),
  dose = "0.1"
)

mydata2 <- data.frame(
  name = "A",
  time = 0:1,
  value = 3:4,
  sigma = .1,
  compound = c("APAP", "DCF"),
  dose = "0.1"
)

# Create datalists from dataframes
data1 <- as.datalist(mydata1, split.by = c("compound", "dose"))
data2 <- as.datalist(mydata2, split.by = c("compound", "dose"))

# Direct sum of datalists
data <- data1 + data2
print(data)

# Check the condition.grid (if available)
condition.grid <- attr(data, "condition.grid")
print(condition.grid)
```

+.fn	<i>Direct sum of functions</i>
------	--------------------------------

Description

Used to add prediction function, parameter transformation functions or observation functions.

Usage

```
## S3 method for class 'fn'
x1 + x2
```

Arguments

x1	function of class obsfn, prdfn or parfn
x2	function of class obsfn, prdfn or parfn

Details

Each prediction function is associated to a number of conditions. Adding functions means merging or overwriting the set of conditions.

Value

Object of the same class as x1 and x2 which returns results for the union of conditions.

See Also

[P](#), [Y](#), [Xs](#)

Examples

```
# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
```

```

trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)

```

+.objfn

Direct sum of objective functions

Description

Direct sum of objective functions

Usage

```
## S3 method for class 'objfn'
x1 + x2
```

Arguments

x1	function of class objfn
x2	function of class objfn

Details

The objective functions are evaluated and their results are added. Sometimes, the evaluation of an objective function depends on results that have been computed internally in a preceding objective function. Therefore, environments are forwarded and all evaluations take place in the same environment. The first objective function in a sum of functions generates a new environment.

Value

Object of class objfn.

See Also

[normL2](#), [constraintL2](#), [priorL2](#), [datapointL2](#)

Examples

```
## Generate three objective functions
prior <- structure(rep(0, 5), names = letters[1:5])

obj1 <- constraintL2(mu = prior, attr.name = "center")
obj2 <- constraintL2(mu = prior + 1, attr.name = "right")
obj3 <- constraintL2(mu = prior - 1, attr.name = "left")

## Evaluate first objective function on a random vector
pouter <- prior + rnorm(length(prior))
print(obj1(pouter))

## Split into fixed and non-fixed part
fixed <- pouter[4:5]
pouter <- pouter[1:3]
print(obj1(pouter, fixed = fixed))

## Visualize the result by a parameter profile
myfit <- trust(obj1, pouter, rinit = 1, rmax = 10, fixed = fixed)
myprof <- profile(obj1, myfit$argument, "a", fixed = fixed)
plotProfile(myprof)

## Create new objective function by adding the single ones,
## then evaluate the random vector again
pouter <- prior + rnorm(length(prior))
obj <- obj1 + obj2 + obj3
print(obj(pouter))
```

+.objlist

Add two lists element by element

Description

Add two lists element by element

Usage

```
## S3 method for class 'objlist'
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.

addReaction	<i>Add reaction to reaction table</i>
-------------	---------------------------------------

Description

Add reaction to reaction table

Usage

```
addReaction(eqnlist, from, to, rate, description = names(rate))
```

Arguments

eqnlist	equation list, see eqnlist
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	character. The rate associated with the reaction. The name is employed as a description of the reaction.
description	Optional description instead of names(rate).

Value

An object of class [eqnlist](#).

Examples

```

f <- eqnlist()
f <- addReaction(f, "2*A+B", "C + 2*D", "k1*B*A^2")
f <- addReaction(f, "C + A", "B + A", "k2*C*A")

# Write your example here. You can also add more Start..End blocks if needed.
# Please mask all output such as print() with the special tag
#
# such that the test is not littered. Statements guarded by are enabled
# in the example file which is extracted from this test file. To extract the
# example run
#   extractExamples()
# on the R command line.

## Generate another equation list
eq <- eqnlist()
eq <- addReaction(eq, "A", "pA", "act_A * A * stimulus", "Phosphorylation of A")
eq <- addReaction(eq, "pA", "A", "deact_A * pA", "Deposphorylation of pA")
eq <- addReaction(eq, "2*pA", "pA_pA", "form_complex_pA * pA^2", "Complex formation of pA")
eq <- addReaction(eq, "B", "pB", "act_B * B * pA_pA", "Phosphorylation of B")
eq <- addReaction(eq, "pB", "B", "deact_B * pB", "Deposphorylation of pB")

## Extract data.frame of reactions
reactions <- getReactions(eq)
print(reactions)

## Get conserved quantities
cq <- conservedQuantities(eq$smatrix)
print(cq)

## Get fluxes
fluxes <- getFluxes(eq)
print(fluxes)

## Subsetting of equation list
subeq1 <- subset(eq, "pB" %in% Product)
print(subeq1)
subeq2 <- subset(eq, grepl("not_available", Description))
print(subeq2)

## Time derivatives of observables
observables <- eqnvec(pA_obs = "s1*pA", tA_obs = "s2*(A + pA)")
dobs <- dot(observables, eq)

## Combined equation vector for ODE and observables
f <- c(as.eqnvec(eq), dobs)
print(f)

```

```
as.data.frame.datalist
```

Coerce to a Data Frame

Description

Coerce to a Data Frame

Usage

```
## S3 method for class 'datalist'
as.data.frame(x, ...)

## S3 method for class 'prdlst'
as.data.frame(x, ..., data = NULL, errfn = NULL)
```

Arguments

x	any R object
...	not used right now
data	data list object
errfn	obsfn object, the error model function to predict sigma

Value

a data frame

```
as.data.frame.eqnlist Coerce equation list into a data frame
```

Description

Coerce equation list into a data frame

Usage

```
## S3 method for class 'eqnlist'
as.data.frame(x, ...)
```

Arguments

x	object of class eqnlist
...	other arguments

Value

a `data.frame` with columns "Description" (character), "Rate" (character), and one column per ODE state with the state names. The state columns correspond to the stoichiometric matrix.

as.eqnvec	<i>Coerce to an equation vector</i>
-----------	-------------------------------------

Description

Coerce to an equation vector

Generate equation vector object

An equation list stores an ODE in a list format. The function translates this list into the right-hand sides of the ODE.

Usage

```
as.eqnvec(x, ...)
```

```
## S3 method for class 'character'
```

```
as.eqnvec(x = NULL, names = NULL, ...)
```

```
## S3 method for class 'eqnlist'
```

```
as.eqnvec(x, ...)
```

Arguments

x	object of class <code>character</code> or <code>eqnlist</code>
...	arguments going to the corresponding methods
names	<code>character</code> , the left-hand sides of the equation

Details

If x is of class `eqnlist`, [getFluxes](#) is called and coerced into a vector of equations.

Value

object of class [eqnvec](#).

as.objlist	<i>Generate objective list from numeric vector</i>
------------	--

Description

Generate objective list from numeric vector

Usage

```
as.objlist(p)
```

Arguments

p	Named 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.objlist(p)
```

as.parframe	<i>Coerce object to a parameter frame</i>
-------------	---

Description

Coerce object to a parameter frame

Usage

```
as.parframe(x, ...)

## S3 method for class 'parlist'
as.parframe(x, sort.by = "value", ...)
```

Arguments

x	object to be coerced
...	other arguments
sort.by	character indicating by which column the returned parameter frame should be sorted. Defaults to "value".

Value

object of class `parframe`.

Examples

```
## Generate a prediction function
regfn <- c(y = "sin(a*time)")

g <- Y(regfn, parameters = "a")
x <- Xt(condition = "C1")

## Generate data
data <- datalist(
  C1 = data.frame(
    name = "y",
    time = 1:5,
    value = sin(1:5) + rnorm(5, 0, .1),
    sigma = .1
  )
)

## Initialize parameters and time
pars <- c(a = 1)
times <- seq(0, 5, .1)

plot((g*x)(times, pars), data)

## Do many fits from random positions and store them into parlist
out <- as.parlist(lapply(1:50, function(i) {
  trust(normL2(data, g*x), pars + rnorm(length(pars), 0, 1), rinit = 1, rmax = 10)
}))

summary(out)

## Reduce parlist to parframe
parframe <- as.parframe(out)
plotValues(parframe)

## Reduce parframe to best fit
bestfit <- as.parvec(parframe)
plot((g*x)(times, bestfit), data)
```

as.parvec.parframe *Select a parameter vector from a parameter frame.*

Description

Obtain a parameter vector from a parameter frame.

Usage

```
## S3 method for class 'parframe'
as.parvec(x, index = 1, ...)
```

Arguments

x	A parameter frame, e.g., the output of <code>as.parframe</code> .
index	Integer, the parameter vector with the <code>index</code> -th lowest objective value.
...	not used right now

Details

With this command, additional information included in the parameter frame as the objective value and the convergence state are removed and a parameter vector is returned. This parameter vector can be used to e.g., evaluate an objective function.

On selection, the parameters in the parameter frame are ordered such, that the parameter vector with the lowest objective value is at 'index' 1. Thus, the parameter vector with the 'index'-th lowest objective value is easily obtained.

Value

The parameter vector with the 'index'-th lowest objective value.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

attrs	<i>Select attributes.</i>
-------	---------------------------

Description

Select or discard attributes from an object.

Usage

```
attrs(x, atr = NULL, keep = TRUE)
```

Arguments

x	The object to work on
atr	An optional list of attributes which are either kept or removed. This parameter defaults to <code>dim</code> , <code>dimnames</code> , <code>names</code> , <code>col.names</code> , and <code>row.names</code> .
keep	For <code>keep = TRUE</code> , <code>atr</code> is a positive list on attributes which are kept, for <code>keep = FALSE</code> , 'atr' are removed.

Value

x with selected attributes.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

Mirjam Fehling-Kaschek, <mirjam.fehling@physik.uni-freiburg.de>

blockdiagSymb	<i>Embed two matrices into one blockdiagonal matrix</i>
---------------	---

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

Examples

```
M <- matrix(1:9, 3, 3, dimnames = list(letters[1:3], letters[1:3]))
N <- matrix(1:4, 2, 2, dimnames = list(LETTERS[1:2], LETTERS[1:2]))
blockdiagSymb(M, N)
```

combine	<i>Combine several data.frames by rowbind</i>
---------	---

Description

Combine several data.frames by rowbind

Usage

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

Arguments

... data.frames or matrices with not necessarily overlapping colnames

Details

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

Value

A data.frame

Examples

```
data1 <- data.frame(Description = "reaction 1", Rate = "k1*A", A = -1, B = 1)
data2 <- data.frame(Description = "reaction 2", Rate = "k2*B", B = -1, C = 1)
combine(data1, data2)
```

compare	<i>Compare two objects and return differences</i>
---------	---

Description

Works either on a list or on two arguments. In case of a list, comparison is done with respect to a reference entry. Besides the objects themselves also some of their attributes are compared, i.e. "equations", "parameters" and "events" and "forcings".

Usage

```
compare(vec1, ...)

## S3 method for class 'list'
compare(vec1, vec2 = NULL, reference = 1, ...)

## S3 method for class 'character'
compare(vec1, vec2 = NULL, ...)

## S3 method for class 'eqnvec'
compare(vec1, vec2 = NULL, ...)

## S3 method for class 'data.frame'
compare(vec1, vec2 = NULL, ...)
```

Arguments

vec1	object of class <code>eqnvec</code> , character or <code>data.frame</code> . Alternatively, a list of such objects.
...	arguments going to the corresponding methods
vec2	same as <code>vec1</code> . Not used if <code>vec1</code> is a list.
reference	numeric of length one, the reference entry.

Value

`data.frame` or list of `data.frames` with the differences.

Examples

```
## Compare equation vectors
eq1 <- eqnvec(a = "-k1*a + k2*b", b = "k2*a - k2*b")
eq2 <- eqnvec(a = "-k1*a", b = "k2*a - k2*b", c = "k2*b")
compare(eq1, eq2)

## Compare character vectors
c1 <- c("a", "b")
c2 <- c("b", "c")
compare(c1, c2)

## Compare data.frames
d1 <- data.frame(var = "a", time = 1, value = 1:3, method = "replace")
d2 <- data.frame(var = "a", time = 1, value = 2:4, method = "replace")
compare(d1, d2)

## Compare structures like prediction functions
fn1 <- function(x) x^2
attr(fn1, "equations") <- eq1
attr(fn1, "parameters") <- c1
attr(fn1, "events") <- d1
```

```
fn2 <- function(x) x^3
attr(fn2, "equations") <- eq2
attr(fn2, "parameters") <- c2
attr(fn2, "events") <- d2

mylist <- list(f1 = fn1, f2 = fn2)
compare(mylist)
```

compile

Compile one or more prdfn, obsfn or parfn objects

Description

Compile one or more prdfn, obsfn or parfn objects

Usage

```
compile(..., output = NULL, args = NULL, cores = 1)
```

Arguments

...	Objects of class parfn, obsfn or prdfn
output	Optional character of the file to be produced. If several objects were passed, the different C files are all compiled into one shared object file.
args	Additional arguments for the R CMD SHLIB call, e.g. -leinspline.
cores	Number of cores used for compilation when several files are compiled.

confint.parframe

Profile uncertainty extraction

Description

extract parameter uncertainties from profiles

Usage

```
## S3 method for class 'parframe'
confint(object, parm = NULL, level = 0.95, ...,
        val.column = "data")
```

Arguments

object	object of class parframe, returned from profile function.
parm	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	the confidence level required.
...	not used right now.
val.column	the value column used in the parframe, usually 'data'.

conservedQuantities *Determine conserved quantities by finding the kernel of the stoichiometric matrix*

Description

Determine conserved quantities by finding the kernel of the stoichiometric matrix

Usage

```
conservedQuantities(S)
```

Arguments

S Stoichiometric matrix

Value

Data frame with conserved quantities carrying an attribute with the number of conserved quantities.

Author(s)

Malenke Mader, <Malenka.Mader@fdm.uni-freiburg.de>

Examples

```
# Write your example here. You can also add more Start..End blocks if needed.
# Please mask all output such as print() with the special tag
#
# such that the test is not littered. Statements guarded by are enabled
# in the example file which is extracted from this test file. To extract the
# example run
#   extractExamples()
# on the R command line.

## Generate another equation list
eq <- eqnlist()
```

```

eq <- addReaction(eq, "A", "pA", "act_A * A * stimulus", "Phosphorylation of A")
eq <- addReaction(eq, "pA", "A", "deact_A * pA", "Deposphorylation of pA")
eq <- addReaction(eq, "2*pA", "pA_pA", "form_complex_pA * pA^2", "Complex formation of pA")
eq <- addReaction(eq, "B", "pB", "act_B * B * pA_pA", "Phosphorylation of B")
eq <- addReaction(eq, "pB", "B", "deact_B * pB", "Deposphorylation of pB")

## Extract data.frame of reactions
reactions <- getReactions(eq)
print(reactions)

## Get conserved quantities
cq <- conservedQuantities(eq$smatrix)
print(cq)

## Get fluxes
fluxes <- getFluxes(eq)
print(fluxes)

## Subsetting of equation list
subeq1 <- subset(eq, "pB" %in% Product)
print(subeq1)
subeq2 <- subset(eq, grepl("not_available", Description))
print(subeq2)

## Time derivatives of observables
observables <- eqnvec(pA_obs = "s1*pA", tA_obs = "s2*(A + pA)")
dobs <- dot(observables, eq)

## Combined equation vector for ODE and observables
f <- c(as.eqnvec(eq), dobs)
print(f)

```

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). Object of class `objlist`.

constraintL2	<i>Soft L2 constraint on parameters</i>
--------------	---

Description

Soft L2 constraint on parameters

Usage

```
constraintL2(mu, sigma = 1, attr.name = "prior", condition = NULL)
```

Arguments

mu	named numeric, the prior values
sigma	named numeric of length of mu or numeric of length one or character of length of mu or character of length one
attr.name	character. The constraint value is additionally returned in an attributed with this name
condition	character, the condition for which the constraint should apply. If NULL, applies to any condition.

Details

If sigma is numeric, the function computes the constraint value

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

and its derivatives with respect to p. If sigma is a character, the function computes

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

and its derivatives with respect to p and sigma. Sigma parameters being passed to the function are ALWAYS assumed to be on a log scale, i.e. internally sigma parameters are converted by `exp()`.

Value

object of class objfn

See Also

[wrss](#)

Examples

```
mu <- c(A = 0, B = 0)
sigma <- c(A = 0.1, B = 1)
myfn <- constraintL2(mu, sigma)
myfn(pars = c(A = 1, B = -1))

# Introduce sigma parameter but fix them (sigma parameters
# are assumed to be passed on log scale)
mu <- c(A = 0, B = 0)
sigma <- paste("sigma", names(mu), sep = "_")
myfn <- constraintL2(mu, sigma)
pars <- c(A = .8, B = -.3, sigma_A = -1, sigma_B = 1)
myfn(pars = pars[c(1, 3)], fixed = pars[c(2, 4)])

# Assume same sigma parameter for both A and B
# sigma is assumed to be passed on log scale
mu <- c(A = 0, B = 0)
myfn <- constraintL2(mu, sigma = "sigma")
pars <- c(A = .8, B = -.3, sigma = 0)
myfn(pars = pars)
```

controls

List, get and set controls for different functions

Description

Applies to objects of class objfn, parfn, prdfn and obsfn. Allows to manipulate different arguments that have been set when creating the objects.

Usage

```
controls(x, ...)

## S3 method for class 'objfn'
controls(x, name = NULL, ...)

## S3 method for class 'fn'
controls(x, condition = NULL, name = NULL, ...)
```



```

controls(x, ...) <- value

## S3 replacement method for class 'objfn'
controls(x, name, ...) <- value

## S3 replacement method for class 'fn'
controls(x, condition = NULL, name, ...) <- value

```

Arguments

x	function
...	arguments going to the appropriate S3 methods
name	character, the name of the control
condition	character, the condition name
value	the new value

Details

If called without further arguments, `controls(x)` lists the available controls within an object. Calling `controls()` with `name` and `condition` returns the control value. The value can be overwritten. If a list or data.frame is returned, elements of those can be manipulated by the `$-` or `[]`-operator.

Value

Either a print-out or the values of the control.

Examples

```

## parfn with condition
p <- P(eqnvec(x = "-a*x"), method = "implicit", condition = "C1")
controls(p)
controls(p, "C1", "keep.root")
controls(p, "C1", "keep.root") <- FALSE

## obsfn with NULL condition
g <- Y(g = eqnvec(y = "s*x"), f = NULL, states = "x", parameters = "s")
controls(g)
controls(g, NULL, "attach.input")
controls(g, NULL, "attach.input") <- FALSE

```

 coordTransform

Coordinate transformation for data frames

Description

Applies a symbolically defined transformation to the value column of a data frame. Additionally, if a sigma column is present, those values are transformed according to Gaussian error propagation.

Usage

```
coordTransform(data, transformations)
```

Arguments

`data` data frame with at least columns "name" (character) and "value" (numeric). Can optionally contain a column "sigma" (numeric).

`transformations` character (the transformation) or named list of characters. In this case, the list names must be a subset of those contained in the "name" column.

Value

The data frame with the transformed values and sigma uncertainties.

datalist	<i>Generate a datalist object</i>
----------	-----------------------------------

Description

The datalist object stores time-course data in a list of data.frames. The names of the list serve as identifiers, e.g. of an experimental condition, etc.

Usage

```
datalist(...)

as.datalist(x, ...)

## S3 method for class 'data.frame'
as.datalist(x, split.by = NULL, ...)

## S3 method for class 'list'
as.datalist(x, names = NULL, ...)
```

Arguments

`...` data.frame objects to be coerced into a list and additional arguments

`x` object of class `data.frame` or `list`

`split.by` vector of columns names which yield a unique identifier (conditions). If `NULL`, all columns except for the expected standard columns "name", "time", "value" and "sigma" will be selected.

`names` optional names vector, otherwise names are taken from `mylist`

Details

Datalists can be plotted, see [plotData](#) and merged, see [sumdatalist](#). They are the basic structure when combining model prediction and data via the [normL2](#) objective function.

Value

Object of class `datalist`.

Object of class `datalist`

Examples

```
## Generate datalist from scratch
mydata1 <- data.frame(name = "A",
                      time = 0:5,
                      value = 0:5,
                      sigma = .1)

mydata2 <- data.frame(name = "A",
                      time = 0:5,
                      value = sin(0:5),
                      sigma = .1)

data <- datalist(C1 = mydata1, C2 = mydata2)
print(data)
plot(data)

## Generate datalist from single data.frame
times <- seq(0, 2*pi, length.out = 20)
mydata <- data.frame(name = "A",
                     time = times,
                     value = c(sin(times), 1.5 * sin(times)),
                     sigma = .1,
                     stage = rep(c("upper", "lower"), each = 10),
                     phase = rep(c("first", "second"), each = 20))

data <- as.datalist(mydata, split.by = c("stage", "phase"))
print(data)
plot(data)

condition.grid <- attr(data, "condition.grid")
print(condition.grid)
```

 datapointL2

L2 objective function for validation data point

Description

L2 objective function for validation data point

Usage

```
datapointL2(name, time, value, sigma = 1, attr.name = "validation",
            condition)
```

Arguments

name	character, the name of the prediction, e.g. a state name.
time	numeric, the time-point associated to the prediction
value	character, the name of the parameter which contains the prediction value.
sigma	numeric, the uncertainty of the introduced test data point
attr.name	character. The constraint value is additionally returned in an attributed with this name
condition	character, the condition for which the prediction is made.

Details

Computes the constraint value

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

and its derivatives with respect to p.

Value

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

See Also

[wrss](#), [constraintL2](#)

Examples

```
prediction <- list(a = 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$a, "deriv") <- derivs
p0 <- c(A = 1, k1 = 2)

vali <- datapointL2(name = "A", time = 0, value = "newpoint", sigma = 1, condition = "a")
vali(pars = c(p0, newpoint = 1), env = .GlobalEnv)
```

detectFreeCores	<i>Detect number of free cores (on UNIX)</i>
-----------------	--

Description

Read /proc/loadavg and subtract from the number of cores

Usage

```
detectFreeCores(machine = NULL)
```

Arguments

machine	character, e.g. "user@localhost".
---------	-----------------------------------

dot	<i>Symbolic time derivative of equation vector given an equation list</i>
-----	---

Description

The time evolution of the internal states is defined in the equation list. Time derivatives of observation functions are expressed in terms of the rates of the internal states.

Usage

```
dot(observable, eqnlist)
```

Arguments

observable	named character vector or object of type eqnvec
eqnlist	equation list

Details

Observables are translated into an ODE

Value

An object of class [eqnvec](#)

Examples

```

# Write your example here. You can also add more Start..End blocks if needed.
# Please mask all output such as print() with the special tag
#
# such that the test is not littered. Statements guarded by are enabled
# in the example file which is extracted from this test file. To extract the
# example run
#   extractExamples()
# on the R command line.

## Generate another equation list
eq <- eqnlist()
eq <- addReaction(eq, "A", "pA", "act_A * A * stimulus", "Phosphorylation of A")
eq <- addReaction(eq, "pA", "A", "deact_A * pA", "Deposphorylation of pA")
eq <- addReaction(eq, "2*pA", "pA_pA", "form_complex_pA * pA^2", "Complex formation of pA")
eq <- addReaction(eq, "B", "pB", "act_B * B * pA_pA", "Phosphorylation of B")
eq <- addReaction(eq, "pB", "B", "deact_B * pB", "Deposphorylation of pB")

## Extract data.frame of reactions
reactions <- getReactions(eq)
print(reactions)

## Get conserved quantities
cq <- conservedQuantities(eq$smatrix)
print(cq)

## Get fluxes
fluxes <- getFluxes(eq)
print(fluxes)

## Subsetting of equation list
subeq1 <- subset(eq, "pB" %in% Product)
print(subeq1)
subeq2 <- subset(eq, grepl("not_available", Description))
print(subeq2)

## Time derivatives of observables
observables <- eqnvec(pA_obs = "s1*pA", tA_obs = "s2*(A + pA)")
dobs <- dot(observables, eq)

## Combined equation vector for ODE and observables
f <- c(as.eqnvec(eq), dobs)
print(f)

```

Description

The eqnlist object stores an ODE as a list of stoichiometric matrix, rate expressions, state names and compartment volumes.

Translates a reaction network, e.g. defined by a data.frame, into an equation list object.

Usage

```
eqnlist(smatrix = NULL, states = colnames(smatrix), rates = NULL,
        volumes = NULL, description = NULL)
```

```
as.eqnlist(data, volumes)
```

```
## S3 method for class 'data.frame'
as.eqnlist(data, volumes = NULL)
```

```
is.eqnlist(x)
```

Arguments

smatrix	Matrix of class numeric. The stoichiometric matrix, one row per reaction/process and one column per state.
states	Character vector. Names of the states.
rates	Character vector. The rate expressions.
volumes	Named character, volume parameters for states. Names must be a subset of the states. Values can be either characters, e.g. "V1", or numeric values for the volume. If volumes is not NULL, missing entries are treated as 1.
description	Character vector. Description of the single processes.
data	data.frame with columns Description, Rate, and one column for each state reflecting the stoichiometric matrix
x	object of class eqnlist
...	additional arguments to be passed to or from methods.

Details

If data is a data.frame, it must contain columns "Description" (character), "Rate" (character), and one column per ODE state with the state names. The state columns correspond to the stoichiometric matrix.

Value

An object of class eqnlist, basically a list.

Object of class [eqnlist](#)

Examples

```
# Generate eqnlist from the constructor
S <- matrix(c(-1, 1, 1, -1),
            nrow = 2, ncol = 2,
            dimnames = list(NULL, c("A", "B")))

rates <- c("k1*A", "k2*B")
description <- c("forward", "backward")

f <- eqnlist(smatrix = S, rates = rates, description = description)
print(f)

# Convert to data.frame
fdata <- as.data.frame(f)
print(fdata)

# Generate eqnlist from data.frame and add volume parameter
f <- as.eqnlist(fdata, volumes = c(A = "Vcyt", B = "Vnuc"))
print(f)
print(as.eqnvec(f))
print(as.eqnvec(f, type = "amount"))
```

eqnvec

Generate equation vector object

Description

The eqnvec object stores explicit algebraic equations, like the right-hand sides of an ODE, observation functions or parameter transformations as named character vectors.

Usage

```
eqnvec(...)
```

```
is.eqnvec(x)
```

Arguments

... mathematical expressions as characters to be coerced, the right-hand sides of the equations

x object of any class

Value

object of class eqnvec, basically a named character.

See Also[eqnlist](#)**Examples**

```
v <- eqnvec(y = "2*x + offset")
print(v)
is.eqnvec(v)
```

expand.grid.alt	<i>Alternative version of expand.grid</i>
-----------------	---

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 of combinations of elements of seq1 and seq2

fitErrorModel	<i>Fit an error model</i>
---------------	---------------------------

Description

Fit an error model to reduced replicate data, see [reduceReplicates](#).

Usage

```
fitErrorModel(data, factors, errorModel = "exp(s0)+exp(srel)*x^2",
  par = c(s0 = 1, srel = 0.1), plotting = TRUE, blather = FALSE, ...)
```

Arguments

data	Reduced replicate data, see reduceReplicates . Need columns "value", "sigma", "n".
factors	'data' is pooled with respect to the columns named here, see Details.
errorModel	Character vector defining the error model in terms of the variance. Use x to reference the independent variable, see Details.
par	Initial values for the parameters of the error model.
plotting	If TRUE, a plot of the pooled variance together with the fit of the error model is shown.
blather	If TRUE, additional information is returned, such as fit parameters and sigmaLS (original sigma given in input data).
...	Parameters handed to the optimizer optim .

Details

The variance estimator using $n - 1$ data points is χ^2 distributed with $n - 1$ degrees of freedom. Given replicates for consecutive time points, the sample variance can be assumed a function of the sample mean. By defining an error model which must hold for all time points, a maximum likelihood estimator for the parameters of the error model can be derived. The parameter 'errorModel' takes the error model as a character vector, where the mean (independent variable) is referred to as x.

It is desirable to estimate the variance from many replicates. The parameter 'data' must provide one or more columns which define the pooling of data. In case more than one column is announced by 'factors', all combinations are constructed. If, e.g., 'factors = c("condition", "name")' is used, where "condition" is "a", "b", "c" and repeating and "name" is "d", "e" and repeating, the effective conditions used for pooling are "a d", "b e", "c d", "a e", "b d", and "c e".

By default, a plot of the pooled data, sigma and its confidence bound at 68% and 95% is shown.

Value

Returned by default is a data frame with columns as in 'data', but with the sigma values replaced by the derived values, obtained by evaluating the error model with the fit parameters.

If the blather = TRUE option is chosen, fit values of the parameters of the error model are appended, with the column names equal to the parameter names. The error model is appended as the attribute "errorModel". Confidence bounds for sigma at confidence level 68% and 95% are calculated, their values come next in the returned data frame. Finally, the effective conditions are appended to easily check how the pooling was done.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

forcingsSymb	<i>Return some useful forcing functions as strings</i>
--------------	--

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 value in parameters.

Value

String with the function

format.eqnvec	<i>Encode equation vector in format with sufficient spaces</i>
---------------	--

Description

Encode equation vector in format with sufficient spaces

Usage

```
## S3 method for class 'eqnvec'
format(x, ...)
```

Arguments

x	object of class eqnvec . Alternatively, a named parsable character vector.
...	additional arguments

Value

named character

funC0

*Evaluation of algebraic expressions defined by characters***Description**

Evaluation of algebraic expressions defined by characters

Usage

```
funC0(x, variables = getSymbols(x, exclude = parameters), parameters = NULL,
      compile = FALSE, modelname = NULL, verbose = FALSE, convenient = TRUE,
      warnings = TRUE)
```

Arguments

x	Object of class eqnvec or a named character vector with the algebraic expressions
variables	character vector, the symbols that should be treated as variables
parameters	character vector, the symbols that should be treated as parameters
compile	Logical. Directly compile the file. If FALSE and modelname is available, the C file is written but not compiled. If modelname is not available, an R function is generated and returned.
modelname	file name of the generated C file.
verbose	Print compiler output to R command line.
convenient	logical, if TRUE return a function with argument . . . to pass all variables/parameters as named arguments
warnings	logical. Suppress warnings about missing variables/parameters that are automatically replaced by zero values.

Value

Either a prediction function $f(\dots, \text{attach.input} = \text{FALSE})$ where the variables/parameters are passed as named arguments or a prediction function $f(M, p, \text{attach.input} = \text{FALSE})$ where M is the matrix of variable values (columns with colnames correspond to different variables) and p is the vector of parameter values. The argument attach.input determines whether M is attached to the output. The function f returns a matrix.

Examples

```
library(ggplot2)
myfun <- funC0(c(y = "a*x^4 + b*x^2 + c"))
out <- myfun(a = -1, b = 2, c = 3, x = seq(-2, 2, .1), attach.input = TRUE)
qplot(x = x, y = y, data = as.data.frame(out), geom = "line")
```

getCoefficients	<i>Get coefficients from a character</i>
-----------------	--

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

getConditions	<i>Extract the conditions of an object</i>
---------------	--

Description

Extract the conditions of an object

Usage

```
getConditions(x, ...)  
  
## S3 method for class 'list'  
getConditions(x, ...)  
  
## S3 method for class 'fn'  
getConditions(x, ...)
```

Arguments

x	object from which the conditions should be extracted
...	additional arguments (not used right now)

Value

The conditions in a format that depends on the class of x.

getDerivs	<i>Extract the derivatives of an object</i>
-----------	---

Description

Extract the derivatives of an object

Usage

```
getDerivs(x, ...)  
  
## S3 method for class 'parvec'  
getDerivs(x, ...)  
  
## S3 method for class 'prdframe'  
getDerivs(x, ...)  
  
## S3 method for class 'prdlist'  
getDerivs(x, ...)  
  
## S3 method for class 'list'  
getDerivs(x, ...)  
  
## S3 method for class 'objlist'  
getDerivs(x, ...)
```

Arguments

x	object from which the derivatives should be extracted
...	additional arguments (not used right now)

Value

The derivatives in a format that depends on the class of x. This is parvec -> matrix, prdframe -> prdframe, prdlist -> prdlist, objlist -> named numeric.

getFluxes	<i>Generate list of fluxes from equation list</i>
-----------	---

Description

Generate list of fluxes from equation list

Usage

```
getFluxes(eqnlist, type = c("conc", "amount"))
```

Arguments

eqnlist object of class [eqnlist](#).
 type "conc." or "amount" for fluxes in units of concentrations or number of molecules.

Value

list of named characters, the in- and out-fluxes for each state.

Examples

```
# Write your example here. You can also add more Start..End blocks if needed.
# Please mask all output such as print() with the special tag
#
# such that the test is not littered. Statements guarded by are enabled
# in the example file which is extracted from this test file. To extract the
# example run
#   extractExamples()
# on the R command line.

## Generate another equation list
eq <- eqnlist()
eq <- addReaction(eq, "A", "pA", "act_A * A * stimulus", "Phosphorylation of A")
eq <- addReaction(eq, "pA", "A", "deact_A * pA", "Deposphorylation of pA")
eq <- addReaction(eq, "2*pA", "pA_pA", "form_complex_pA * pA^2", "Complex formation of pA")
eq <- addReaction(eq, "B", "pB", "act_B * B * pA_pA", "Phosphorylation of B")
eq <- addReaction(eq, "pB", "B", "deact_B * pB", "Deposphorylation of pB")

## Extract data.frame of reactions
reactions <- getReactions(eq)
print(reactions)

## Get conserved quantities
cq <- conservedQuantities(eq$smatrix)
print(cq)

## Get fluxes
fluxes <- getFluxes(eq)
print(fluxes)

## Subsetting of equation list
subeq1 <- subset(eq, "pB" %in% Product)
print(subeq1)
subeq2 <- subset(eq, grepl("not_available", Description))
print(subeq2)

## Time derivatives of observables
observables <- eqnvec(pA_obs = "s1*pA", tA_obs = "s2*(A + pA)")
dobs <- dot(observables, eq)

## Combined equation vector for ODE and observables
f <- c(as.eqnvec(eq), dobs)
```

```
print(f)
```

getLocalDLLs	<i>Determine loaded DLLs available in working directory</i>
--------------	---

Description

Determine loaded DLLs available in working directory

Usage

```
getLocalDLLs()
```

Value

Character vector with the names of the loaded DLLs available in the working directory

getParameters	<i>Extract the parameters of an object</i>
---------------	--

Description

Extract the parameters of an object

Usage

```
getParameters(..., conditions = NULL)
```

```
## S3 method for class 'odemodel'
getParameters(x, conditions = NULL)
```

```
## S3 method for class 'fn'
getParameters(x, conditions = NULL)
```

```
## S3 method for class 'parvec'
getParameters(x, conditions = NULL)
```

```
## S3 method for class 'prdfame'
getParameters(x, conditions = NULL)
```

```
## S3 method for class 'prdlist'
getParameters(x, conditions = NULL)
```


Arguments

... objects from which the parameters should be extracted
 conditions character vector specifying the conditions to which getParameters is restricted
 x object from which the parameters are extracted

Value

The parameters in a format that depends on the class of x.

getReactions	<i>Generate a table of reactions (data.frame) from an equation list</i>
--------------	---

Description

Generate a table of reactions (data.frame) from an equation list

Usage

```
getReactions(eqnlist)
```

Arguments

eqnlist object of class [eqnlist](#)

Value

data.frame with educts, products, rate and description. The first column is a check if the reactions comply with reaction kinetics.

Examples

```
# Write your example here. You can also add more Start..End blocks if needed.
# Please mask all output such as print() with the special tag
#
# such that the test is not littered. Statements guarded by are enabled
# in the example file which is extracted from this test file. To extract the
# example run
#   extractExamples()
# on the R command line.

## Generate another equation list
eq <- eqnlist()
eq <- addReaction(eq, "A", "pA", "act_A * A * stimulus", "Phosphorylation of A")
eq <- addReaction(eq, "pA", "A", "deact_A * pA", "Deposphorylation of pA")
eq <- addReaction(eq, "2*pA", "pA_pA", "form_complex_pA * pA^2", "Complex formation of pA")
eq <- addReaction(eq, "B", "pB", "act_B * B * pA_pA", "Phosphorylation of B")
eq <- addReaction(eq, "pB", "B", "deact_B * pB", "Deposphorylation of pB")
```

```
## Extract data.frame of reactions
reactions <- getReactions(eq)
print(reactions)

## Get conserved quantities
cq <- conservedQuantities(eq$smatrix)
print(cq)

## Get fluxes
fluxes <- getFluxes(eq)
print(fluxes)

## Subsetting of equation list
subeq1 <- subset(eq, "pB" %in% Product)
print(subeq1)
subeq2 <- subset(eq, grepl("not_available", Description))
print(subeq2)

## Time derivatives of observables
observables <- eqnvec(pA_obs = "s1*pA", tA_obs = "s2*(A + pA)")
dobs <- dot(observables, eq)

## Combined equation vector for ODE and observables
f <- c(as.eqnvec(eq), dobs)
print(f)
```

ggopen

Open last plot in external pdf viewer

Description

Convenience function to show last plot in an external viewer.

Usage

```
ggopen(plot = last_plot(), command = "xdg-open", ...)
```

Arguments

plot	ggplot2 plot object.
command	character, indicating which pdf viewer is started.
...	arguments going to ggsave.

jakstat	<i>Time-course data for the JAK-STAT cell signaling pathway</i>
---------	---

Description

Phosphorylated Epo receptor (pEpoR), phosphorylated STAT in the cytoplasm (tpSTAT) and total STAT (tSTAT) in the cytoplasm have been measured at times 0, ..., 60.

lbind	<i>Bind named list of data.frames into one data.frame</i>
-------	---

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 is augmented 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 original columns augmented by a "condition" column.

load.parlist	<i>Construct fitlist from temporary files.</i>
--------------	--

Description

An aborted `mstrust` leaves behind results of already completed fits. This command loads these fits into a fitlist.

Usage

```
load.parlist(folder)
```

Arguments

folder	Path to the folder where the fit has left its results.
--------	--

Details

The command `mstrust` saves each completed fit along the multi-start sequence such that the results can be resurrected on abortion. This command loads a fitlist from these intermediate results.

Value

An object of class `parlist`.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

See Also

[mstrust](#)

loadDLL	<i>Load shared object for a dMod object</i>
---------	---

Description

Usually when restarting the R session, although all objects are saved in the workspace, the dynamic libraries are not linked any more. `loadDLL` is a wrapper for `dyn.load` that uses the "modelName" attribute of `dMod` objects like prediction functions, observation functions, etc. to load the corresponding shared object.

Usage

```
loadDLL(...)
```

Arguments

... objects of class prdfn, obsfn, parfn, objfn, ...

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

lsdMod *Print list of dMod objects in .GlobalEnv*

Description

Lists the objects for a set of classes.

Usage

```
lsdMod(classlist = c("odemodel", "parfn", "prdfn", "obsfn", "objfn",
  "datalist"), envir = .GlobalEnv)
```

Arguments

classlist List of object classes to print.
 envir Alternative environment to search for objects.

Examples

```
## Not run:
lsdMod()
lsdMod(classlist = "prdfn", envir = environment(obj))

## End(Not run)
```

mname	<i>Get modelname from single object (used internally)</i>
-------	---

Description

Get modelname from single object (used internally)

Usage

```
mname(x, conditions = NULL)

## S3 method for class 'NULL'
mname(x, conditions = NULL)

## S3 method for class 'character'
mname(x, conditions = NULL)

## S3 method for class 'objfn'
mname(x, conditions = NULL)

## S3 method for class 'fn'
mname(x, conditions = NULL)
```

Arguments

x	dMod object
conditions	character vector of conditions

modelname	<i>Get and set modelname</i>
-----------	------------------------------

Description

The modelname attribute refers to the name of a C file associated with a dMod function object like prediction-, parameter transformation- or objective functions.

Usage

```
modelname(..., conditions = NULL)

modelname(x, ...) <- value

## S3 replacement method for class 'fn'
modelname(x, conditions = NULL, ...) <- value

## S3 replacement method for class 'objfn'
modelname(x, conditions = NULL, ...) <- value
```

Arguments

...	objects of type prdfn, parfn, objfn
conditions	character vector of conditions
x	dMod object for which the model name should be set
value	character, the new modelname (does not change the C file)

Value

character vector of model names, corresponding to C files in the local directory.

mssample	<i>Generate sample for multi-start fit</i>
----------	--

Description

Generate sample for multi-start fit

Usage

```
mssample(center, samplefun = "rnorm", fits = 20, ...)
```

Arguments

center	named numeric, the center around we sample
samplefun	character, indicating the random number generator, defaults to "rnorm".
fits	length of the sample
...	arguments going to samplefun

Value

matrix with the parameter samples

mstrust

Non-Linear Optimization, multi start

Description

Wrapper around [trust](#) allowing for multiple fits from randomly chosen initial values.

Usage

```
mstrust(objfun, center, studyname, rinit = 0.1, rmax = 10, fits = 20,
        cores = 1, samplefun = "rnorm", resultPath = ".", stats = FALSE, ...)
```

Arguments

objfun	Objective function, see trust .
center	Parameter values around which the initial values for each fit are randomly sampled. The initial values handed to trust are the sum of center and the output of 'samplefun', center + 'samplefun'. See trust , parinit.
studyname	The names of the study or fit. This name is used to determine filenames for interim and final results. See Details.
rinit	Starting trust region radius, see trust .
rmax	Maximum allowed trust region radius, see trust .
fits	Number of fits (jobs).
cores	Number of cores for job parallelization.
samplefun	Function to sample random initial values. It is assumed, that 'samplefun' has a named parameter "n" which defines how many random numbers are to be returned, such as for rnorm or runif . By default rnorm is used. Parameters for samplefun are simply appended as named parameters to the mstrust call and automatically handed to samplefun by matching parameter names.
resultPath	character indicating the folder where the results should be stored. Defaults to ".".
stats	If true, the same summary statistic as written to the logfile is printed to command line on mstrust completion.
...	Additional parameters handed to trust (), samplefun (), or the objective function by matching parameter names. All unmatched parameters are handed to the objective function objfun (). The log file starts with a table telling which parameter was assigned to which function.

Details

By running multiple fits starting at randomly chosen initial parameters, the chisquare landscape can be explored using a deterministic optimizer. Here, [trust](#) is used for optimization. The standard procedure to obtain random initial values is to sample random variables from a uniform distribution


```

trafo0)

conditions <- c("a", "b")

trafo <- list()
trafo$a <- replaceSymbols("s1", "s1_a", trafo0)
trafo$b <- replaceSymbols("s1", "s1_b", trafo0)

events <- list()
events$a <- data.frame(var = "A", time = 5, value = 1, method = "add")
events$b <- data.frame(var = "A", time = 5, value = 2, method = "add")

## Model definition (generate compiled objects
## and functions from above information)

# ODE model
model <- odemodel(f)

# Observation function
g <- Y(observables, f, compile = TRUE, modelname = "obsfn")

# Parameter transformation for steady states
pSS <- P(f, method = "implicit", condition = NULL)

# Condition-specific transformation and prediction functions
p0 <- x <- NULL
for (C in conditions) {
  p0 <- p0 + P(trafo[[C]], condition = C)
  x <- x + Xs(model, events = events[[C]], condition = C)
}

## Process data

# Data
data <- datalist(
  a = data.frame(time = c(0, 2, 7),
                 name = "Bobs",
                 value = c(.3, .3, .3),
                 sigma = c(.03, .03, .03)),
  b = data.frame(time = c(0, 2, 7),
                 name = "Bobs",
                 value = c(.1, .1, .2),
                 sigma = c(.01, .01, .02))
)

## Evaluate model/data

# Initialize times and parameters
times <- seq(0, 10, .1)
outerpars <- attr(p0, "parameters")
pouter <- structure(rnorm(length(outerpars)),
                   names = outerpars)

```

```

# Plot prediction
plot((g*x*p0)(times, pouter))
plot((g*x*pSS*p0)(times, pouter))
plotFluxes(pouter, g*x*p0, times, getFluxes(r)$B)

# Fit data with L2 constraint
constr <- constraintL2(mu = 0*pouter, sigma = 5)
myfit <- trust(normL2(data, g*x*p0) + constr,
               pouter, rinit = 1, rmax = 10)
plot((g*x*p0)(times, myfit$argument), data)

# List of fits
obj <- normL2(data, g*x*p0) + constr
mylist <- mstrust(normL2(data, g*x*p0) + constr,
                 pouter, fits = 10, cores = 4, sd = 4)
plot(mylist)
pars <- as.parframe(mylist)
plotValues(pars)

bestfit <- as.parvec(pars)

# Compute profiles of the fit
profile <- profile(normL2(data, g*x*p0) + constr,
                  bestfit, "k1", limits = c(-5, 5))
plotProfile(profile)

# Add a validation objective function
vali <- datapointL2("A", 2, "mypoint", .1, condition = "a")
# Get validation point parameters
mypoint <- trust(normL2(data, g*x*p0) + constr + vali,
                 c(mypoint = 0), rinit = 1, rmax = 10,
                 fixed = bestfit)$argument
# Compute profile and plot
profile <- profile(normL2(data, g*x*p0) + constr + vali,
                  c(bestfit, mypoint), "mypoint")
plotProfile(profile)

## Using the controls() function to manipulate objects -----

# List the available controls of an object
controls(x)

# List a specific control
controls(x, condition = "a", name = "optionsSens")

# Change specific control
controls(x, condition = "a", name = "optionsSens") <-
  list(method = "lsoda", rtol = 1e-8, atol = 1e-8)

# Almost every function (objfn, parfn, prdfn, obsfn) saves the arguments
# by which it was invoked in a list named "controls", which can be manipulated

```

```

# by the controls() function.

## New example: condition-specific observation parameters -----

# Condition-specific observation parameters
f <- eqnvec(A = "-k1*A", B = "k1*A - k2*B")
observables <- eqnvec(Bobs = "s1*B")
conditions <- c("scale1", "scale2")

dynpars <- getSymbols(c(names(f), f))
obsvars <- setdiff(getSymbols(observables), dynpars)
trafo0 <- structure(c(dynpars, obsvars), names = c(dynpars, obsvars))

obsvars_all <- outer(obsvars, conditions, paste, sep = "_")
trafo_all <- structure(c(dynpars, obsvars_all),
                      names = c(dynpars, obsvars_all))
trafo_all <- replaceSymbols(dynpars,
                           paste0("exp(", dynpars, ")"),
                           trafo_all)
trafo_all <- replaceSymbols(obsvars_all,
                           paste0("exp(", obsvars_all, ")"),
                           trafo_all)

pObs <- NULL
for (C in conditions) {
  pObs <- pObs + P(replaceSymbols(obsvars,
                                  paste(obsvars, C, sep = "_"),
                                  trafo0),
                  condition = C)
}

x <- Xs(model)
p <- P(trafo_all)
y <- g*pObs*x*p

times <- seq(0, 10, .1)
outerpars <- attr(p, "parameters")
pouter <- structure(rnorm(length(outerpars)), names = outerpars)

# Plot prediction
plot((g*pObs*x*p)(times, pouter))

## End(Not run)

```

Description

Compute the negative log-likelihood

Usage

```
nll(nout)
```

Arguments

nout data.frame (result of [res](#)) or object of class [objframe](#).

Value

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

normL2	<i>L2 norm between data and model prediction</i>
--------	--

Description

For parameter estimation and optimization, an objective function is needed. normL2 returns an objective function for the L2 norm of data and model prediction. The resulting objective function can be used for optimization with the trust optimizer, see [mstrust](#).

Usage

```
normL2(data, x, errmodel = NULL, times = NULL, attr.name = "data")
```

Arguments

data	object of class datalist
x	object of class prdfn
errmodel	object of class obsfn
times	numeric vector, additional time points where the prediction function is evaluated. If NULL, time points are extracted from the datalist solely. If the prediction function makes use of events, hand over event times here.
attr.name	character. The constraint value is additionally returned in an attributed with this name

Details

Objective functions can be combined by the "+" operator, see [sumobjfn](#).

Value

Object of class `obsfn`, i.e. a function `obj(..., fixed, deriv, conditions, env)` that returns an objective list, [objlist](#).

Examples

```
## Generate a prediction function

times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid, condition = "C1")

pars <- structure(rep(0, nrow(grid)), names = row.names(grid))

## Simulate data
data.list <- lapply(1:3, function(i) {
  prediction <- x(times, pars + rnorm(length(pars), 0, 1))
  cbind(wide2long(prediction), sigma = 1)
})

data <- as.datalist(do.call(rbind, data.list))

## Generate objective function based on data and model
## Then fit the data and plot the result
obj <- normL2(data, x)
myfit <- trust(obj, pars, rinit = 1, rmax = 10)
plot(x(times, myfit$argument), data)
```

nullZ

Find integer-null space of matrix A

Description

Find integer-null space of matrix A

Usage

```
nullZ(A, tol = sqrt(.Machine$double.eps))
```

Arguments

A matrix for which the null space is searched
tol tolerance to find pivots in `rref`-function below

Value

null space of A with only integers in it

Author(s)

Malenka Mader, <Malenka.Mader@fdm.uni-freiburg.de>

objframe

Objective frame

Description

An objective frame is supposed to store the residuals of a model prediction with respect to a data frame.

Usage

```
objframe(mydata, deriv = NULL, deriv.err = NULL)
```

Arguments

mydata	data.frame as being generated by res .
deriv	matrix of the derivatives of the residuals with respect to parameters.
deriv.err	matrix of the derivatives of the error model.

Value

An object of class objframe, i.e. a data frame with attribute "deriv".

objlist

Generate objective list

Description

An objective list contains an objective value, a gradient, and a Hessian matrix.

Objective lists can contain additional numeric attributes that are preserved or combined with the corresponding attributes of another objective list when both are added by the "+" operator, see [sumobjlist](#).

Objective lists are returned by objective functions as being generated by [normL2](#), [constraintL2](#), [priorL2](#) and [datapointL2](#).

Usage

```
objlist(value, gradient, hessian)
```

Arguments

value	numeric of length 1
gradient	named numeric
hessian	matrix with rownames and colnames according to gradient names

Value

Object of class `objlist`

obsfn	<i>Observation function</i>
-------	-----------------------------

Description

An observation function is a function that is concatenated with a prediction function via [prodfn](#) to yield a new prediction function, see [prdfn](#). Observation functions are generated by [Y](#). Handling of the conditions is then organized by the `obsfn` object.

Usage

```
obsfn(X2Y, parameters = NULL, condition = NULL)
```

Arguments

X2Y	the low-level observation function generated e.g. by Y .
parameters	character vector with parameter names
condition	character, the condition name

Details

Observation functions can be "added" by the "+" operator, see [sumfn](#). Thereby, observations for different conditions are merged or, overwritten. Observation functions can also be concatenated with other functions, e.g. observation functions ([obsfn](#)) or prediction functions ([prdfn](#)) by the "*" operator, see [prodfn](#).

Value

Object of class `obsfn`, i.e. a function `x(..., fixed, deriv, conditions, env)` which returns a [prdfn](#). The arguments `out` (prediction) and `pars` (parameter values) should be passed via the `...` argument.

Examples

```

# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)

```

odemodel

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

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

Usage

```

odemodel(f, deriv = TRUE, forcings = NULL, fixed = NULL,
  modelname = "odemodel", solver = c("deSolve", "Sundials"),
  gridpoints = NULL, verbose = FALSE, ...)

```

Arguments

f	Something that can be converted to <code>eqnvec</code> , e.g. a named character vector with the ODE
deriv	logical, generate sensitivities or not
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.
solver	Solver for which the equations are prepared.
gridpoints	Integer, the minimum number of time points where the ODE is evaluated internally
verbose	Print compiler output to R command line.
...	Further arguments being passed to <code>funC</code> .

Value

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

Examples

```
## Not run:

## Generate a compiled ODE model from an equation vector
## The model will not return sensitivities for "switch"
## Files will be generated in your working directory!

f <- eqnvec(A = "-k*A + switch*F")
model <- odemodel(f, forcings = "F", fixed = "switch")
print(model)

## Generate the same model from an equation list
f <- addReaction(NULL, from = "", to = "A", rate = "switch*F", description = "production")
f <- addReaction(f, from = "A", to = "", rate = "k*A", description = "degradation")
print(f)

model <- odemodel(f, forcings = "F", fixed = "switch")
print(model)

## End(Not run)
```

P *Generate a parameter transformation function*

Description

Generate parameter transformation function from a named character vector or object of class `eqnvec`. This is a wrapper function for `Pexpl` and `Pimpl`. See for more details there.

Usage

```
P(trafo = NULL, parameters = NULL, condition = NULL,
  attach.input = FALSE, keep.root = TRUE, compile = FALSE,
  modelname = NULL, method = c("explicit", "implicit"), verbose = FALSE)
```

Arguments

<code>trafo</code>	object of class <code>eqnvec</code> or named character
<code>parameters</code>	character vector
<code>condition</code>	character, the condition for which the transformation is generated
<code>attach.input</code>	attach those incoming parameters to output which are not overwritten by the parameter transformation.
<code>keep.root</code>	logical, applies for <code>method = "implicit"</code> . The root of the last evaluation of the parameter transformation function is saved as guess for the next evaluation.
<code>compile</code>	logical
<code>modelname</code>	character
<code>method</code>	character, either <code>"explicit"</code> or <code>"implicit"</code>
<code>verbose</code>	Print out information during compilation

Value

An object of class `parfn`.

`parfn` *Parameter transformation function*

Description

Generate functions that transform one parameter vector into another by means of a transformation, pushing forward the jacobian matrix of the original parameter. Usually, this function is called internally, e.g. by `P`. However, you can use it to add your own specialized parameter transformations to the general framework.

Usage

```
parfn(p2p, parameters = NULL, condition = NULL)
```

Arguments

p2p	a transformation function for one condition, i.e. a function <code>p2p(p, fixed, deriv)</code> which translates a parameter vector <code>p</code> and a vector of fixed parameter values <code>fixed</code> into a new parameter vector. If <code>deriv = TRUE</code> , the function should return an attribute <code>deriv</code> with the Jacobian matrix of the parameter transformation.
parameters	character vector, the parameters accepted by the function
condition	character, the condition for which the transformation is defined

Value

object of class `parfn`, i.e. a function `p(..., fixed, deriv, conditions, env)`. The argument `pars` should be passed via the `...` argument.

Contains attributes "mappings", a list of `p2p` functions, "parameters", the union of parameters accepted by the mappings and "conditions", the total set of conditions.

See Also

[sumfn](#), [P](#)

Examples

```
# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
```

```

# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)

```

parframe

Generate a parameter frame

Description

A parameter frame is a data.frame where the rows correspond to different parameter specifications. The columns are divided into three parts. (1) the meta-information columns (e.g. index, value, constraint, etc.), (2) the attributes of an objective function (e.g. data contribution and prior contribution) and (3) the parameters.

Usage

```
parframe(x = NULL, parameters = colnames(x), metanames = NULL,
  obj.attributes = NULL)
```

```
## S3 method for class 'parframe'
x[i = NULL, j = NULL, drop = FALSE]
```

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

Arguments

x	data.frame.
parameters	character vector, the names of the parameter columns.
metanames	character vector, the names of the meta-information columns.
obj.attributes	character vector, the names of the objective function attributes.
i	row index in any format
j	column index in any format
drop	logical. If TRUE the result is coerced to the lowest possible dimension
...	additional arguments

Details

Parameter frames can be subsetted either by [,] or by subset. If [, index] is used, the names of the removed columns will also be removed from the corresponding attributes, i.e. metanames, obj.attributes and parameters.

Value

An object of class parframe, i.e. a data.frame with attributes for the different names. Inherits from data.frame.

See Also

[profile](#), [mstrust](#)

Examples

```
## Generate a prediction function
regfn <- c(y = "sin(a*time)")

g <- Y(regfn, parameters = "a")
x <- Xt(condition = "C1")

## Generate data
data <- datalist(
  C1 = data.frame(
    name = "y",
    time = 1:5,
    value = sin(1:5) + rnorm(5, 0, .1),
    sigma = .1
  )
)

## Initialize parameters and time
pars <- c(a = 1)
times <- seq(0, 5, .1)

plot((g*x)(times, pars), data)

## Do many fits from random positions and store them into parlist
out <- as.parlist(lapply(1:50, function(i) {
  trust(normL2(data, g*x), pars + rnorm(length(pars), 0, 1), rinit = 1, rmax = 10)
}))

summary(out)

## Reduce parlist to parframe
parframe <- as.parframe(out)
plotValues(parframe)

## Reduce parframe to best fit
bestfit <- as.parvec(parframe)
plot((g*x)(times, bestfit), data)
```

parlist	<i>Parameter list</i>
---------	-----------------------

Description

The special use of a parameter list is to save the outcome of multiple optimization runs provided by [mstrust](#), into one list.

Parameter list

Fitlists carry an fit index which must be held unique on merging multiple fitlists.

Usage

```
parlist(...)

as.parlist(x = NULL)

## S3 method for class 'parlist'
summary(object, ...)

## S3 method for class 'parlist'
c(...)
```

Arguments

...	Objects to be coerced to parameter list.
x	list of lists, as returned by trust
object	a parlist

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

See Also

[load.parlist](#), [plot.parlist](#)

Examples

```
## Generate a prediction function
regfn <- c(y = "sin(a*time)")

g <- Y(regfn, parameters = "a")
x <- Xt(condition = "C1")

## Generate data
data <- datalist(
  C1 = data.frame(
```

```

    name = "y",
    time = 1:5,
    value = sin(1:5) + rnorm(5, 0, .1),
    sigma = .1
  )
)

## Initialize parameters and time
pars <- c(a = 1)
times <- seq(0, 5, .1)

plot((g*x)(times, pars), data)

## Do many fits from random positions and store them into parlist
out <- as.parlist(lapply(1:50, function(i) {
  trust(normL2(data, g*x), pars + rnorm(length(pars), 0, 1), rinit = 1, rmax = 10)
}))

summary(out)

## Reduce parlist to parframe
parframe <- as.parframe(out)
plotValues(parframe)

## Reduce parframe to best fit
bestfit <- as.parvec(parframe)
plot((g*x)(times, bestfit), data)

```

parvec

Parameter vector

Description

A parameter vector is a named numeric vector (the parameter values) together with a "deriv" attribute (the Jacobian of a parameter transformation by which the parameter vector was generated).

Dispatch as.parvec.

Parameter vector

Usage

```
parvec(..., deriv = NULL)
```

```
as.parvec(x, ...)
```

```
## S3 method for class 'numeric'
as.parvec(x, names = NULL, deriv = NULL, ...)
```

```
## S3 method for class 'parvec'
x[... , drop = FALSE]
```



```
## S3 method for class 'parvec'
c(...)
```

Arguments

...	objects to be concatenated
deriv	matrix with rownames (according to names of ...) and colnames according to the names of the parameter by which the parameter vector was generated.
x	numeric or named numeric, the parameter values
names	optional character vector, the parameter names. Otherwise, names are taken from x.
drop	logical, drop empty columns in Jacobian after subsetting. ATTENTION: Be careful with this option. The default behavior is to keep the columns in the Jacobian. This can lead to unintended results when subsetting the parvec and using it e.g. in another parameter transformation.

Value

An object of class parvec, i.e. a named numeric vector with attribute "deriv".

Examples

```
# Generate a parameter vector
v <- parvec(a = 2, b = 3)
print(v)
print(getDerivs(v))

# Parameter vector from a named numeric
M <- matrix(c(1, 1, 0, 1),
            nrow = 2, ncol = 2,
            dimnames = list(c("a", "b"), c("A", "B")))
v <- as.parvec(x = c(a = 2, b = 3), deriv = M)
print(v)
print(getDerivs(v))

# Subsetting of parameter vectors
# Case 1: Dependencies in the Jacobian are maintained
w <- v[1]
print(w)
print(getDerivs(w))

# Case 2: Dependencies are dropped
w <- v[1, drop = TRUE]
print(w)
print(getDerivs(w))

# Concatenating parameter vectors
w <- parvec(c = 4, d = 5)
```

```
print(c(v, w))
print(getDerivs(c(v, w)))
```

Pexpl

Parameter transformation

Description

Parameter transformation

Usage

```
Pexpl(trafo, parameters = NULL, attach.input = FALSE, condition = NULL,
      compile = FALSE, modelName = NULL, verbose = FALSE)
```

Arguments

trafo	Named character vector. Names correspond to the parameters being fed into the model (the inner parameters). The elements of trafo 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.
attach.input	attach those incoming parameters to output which are not overwritten by the parameter transformation.
condition	character, the condition for which the transformation is generated
compile	Logical, compile the function (see funC0)
modelName	Character, used if compile = TRUE, sets a fixed filename for the C file.
verbose	Print compiler output to R command line.

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

[Pimpl](#) for implicit parameter transformations

Examples

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

pars <- c(logk1 = 1, logk2 = -1, logA = 0, logB = 0)
out <- p_log(pars)
getDerivs(out)
```

Pimpl

*Parameter transformation (implicit)***Description**

Parameter transformation (implicit)

Usage

```
Pimpl(trafo, parameters = NULL, condition = NULL, keep.root = TRUE,
      positive = TRUE, compile = FALSE, modelname = NULL, verbose = 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.
condition	character, the condition for which the transformation is generated
keep.root	logical, applies for method = "implicit". The root of the last evaluation of the parameter transformation function is saved as guess for the next evaluation.
positive	logical, returns projection to the (semi)positive range. Comes with a warning if the steady state has been found to be negative.
compile	Logical, compile the function (see funC0)
modelname	Character, used if compile = TRUE, sets a fixed filename for the C file.
verbose	Print compiler output to R command line.

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

[Pexpl](#) for explicit parameter transformations

Examples

```
#####
## Example 1: Steady-state trafo
#####
f <- c(A = "-k1*A + k2*B",
      B = "k1*A - k2*B")
P.steadyState <- Pimpl(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 <- Pimpl(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 * P.log)(p.outerValue)

#####
## Example 3: Steady-states with conserved quantities
#####
f <- c(A = "-k1*A + k2*B", B = "k1*A - k2*B")
replacement <- c(B = "A + B - total")
f[names(replacement)] <- replacement

pSS <- Pimpl(f, "total")
pSS(c(k1 = 1, k2 = 2, A = 5, B = 5, total = 3))
```

plot.datalist *Plot a list data points*

Description

Plot a list data points

Usage

```
## S3 method for class 'datalist'
plot(x, ..., scales = "free", facet = "wrap")
```

Arguments

x	Named list of data.frames as being used in res , i.e. with columns name, time, value and sigma.
...	Further arguments going to subset.
scales	The scales argument of facet_wrap or facet_grid, i.e. "free", "fixed", "free_x" or "free_y"
facet	Either "wrap" or "grid"

Details

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

Value

A plot object of class ggplot.

plot.parlist *Plot a parameter list.*

Description

Plot a parameter list.

Usage

```
## S3 method for class 'parlist'
plot(x, path = FALSE, ...)
```

Arguments

x	fitlist obtained from mstrust
path	print path of parameters from initials to convergence. For this option to be TRUE <code>mstrust</code> must have had the option 'blather'.
...	additional arguments

Details

If path=TRUE:

Author(s)

Malenka Mader, <Malenka.Mader@fdm.uni-freiburg.de>

plotCombined	<i>Plot a list of model predictions and a list of data points in a combined plot</i>
--------------	--

Description

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

Usage

```
plotCombined(prediction, data = NULL, ..., scales = "free",
             facet = "wrap", transform = NULL)
```

```
## S3 method for class 'prdlist'
plot(x, data = NULL, ..., scales = "free",
     facet = "wrap", transform = NULL)
```

```
## S3 method for class 'prdfame'
plot(x, data = NULL, ..., scales = "free",
     facet = "wrap", transform = NULL)
```

Arguments

prediction	Named list of matrices or data.frames, usually the output of a prediction function as generated by <code>Xs</code> .
data	Named list of data.frames as being used in <code>res</code> , i.e. with columns name, time, value and sigma.
...	Further arguments going to subset.
scales	The scales argument of <code>facet_wrap</code> or <code>facet_grid</code> , i.e. "free", "fixed", "free_x" or "free_y"
facet	Either "wrap" or "grid"
transform	list of transformation for the states, see <code>coordTransform</code> .
x	prediction

Details

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

Value

A plot object of class ggplot.

Examples

```
## Observation function
fn <- eqnvec(
  sine = "1 + sin(6.28*omega*time)",
  cosine = "cos(6.28*omega*time)"
)
g <- Y(fn, parameters = "omega")

## Prediction function for time
x <- Xt()

## Parameter transformations to split conditions
p <- NULL
for (i in 1:3) {
  p <- p + P(trafo = c(omega = paste0("omega_", i)), condition = paste0("frequency_", i))
}

## Evaluate prediction
times <- seq(0, 1, .01)
pars <- structure(seq(1, 2, length.out = 3), names = attr(p, "parameters"))

prediction <- (g*x*p)(times, pars)

## Plotting prediction
# plot(prediction)
plotPrediction(prediction)
plotPrediction(prediction, scales = "fixed")
plotPrediction(prediction, facet = "grid")
plotPrediction(prediction,
  scales = "fixed",
  transform = list(sine = "x^2", cosine = "x - 1"))

## Simulate data
dataset <- wide2long(prediction)
dataset <- dataset[seq(1, nrow(dataset), 5),]
dataset$value <- dataset$value + rnorm(nrow(dataset), 0, .1)
dataset$sigma <- 0.1
data <- as.datalist(dataset, split.by = "condition")

## Plotting data
# plot(data)
plotData(data)

## Plotting data and prediction with subsetting
```

```
# plot(prediction, data)
plotCombined(prediction, data)
plotCombined(prediction, data,
              time <= 0.5 & condition == "frequency_1")
plotCombined(prediction, data,
              time <= 0.5 & condition != "frequency_1",
              facet = "grid")
```

plotData

Plot a list data points

Description

Plot a list data points

Usage

```
plotData(data, ..., scales = "free", facet = "wrap", transform = NULL)
```

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.
scales	The scales argument of <code>facet_wrap</code> or <code>facet_grid</code> , i.e. "free", "fixed", "free_x" or "free_y"
facet	Either "wrap" or "grid"
transform	list of transformation for the states, see coordTransform .

Details

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

Value

A plot object of class `ggplot`.

Examples

```
## Observation function
fn <- eqnvec(
  sine = "1 + sin(6.28*omega*time)",
  cosine = "cos(6.28*omega*time)"
)
g <- Y(fn, parameters = "omega")

## Prediction function for time
x <- Xt()
```



```

## Parameter transformations to split conditions
p <- NULL
for (i in 1:3) {
  p <- p + P(trafo = c(omega = paste0("omega_", i)), condition = paste0("frequency_", i))
}

## Evaluate prediction
times <- seq(0, 1, .01)
pars <- structure(seq(1, 2, length.out = 3), names = attr(p, "parameters"))

prediction <- (g*x*p)(times, pars)

## Plotting prediction
# plot(prediction)
plotPrediction(prediction)
plotPrediction(prediction, scales = "fixed")
plotPrediction(prediction, facet = "grid")
plotPrediction(prediction,
               scales = "fixed",
               transform = list(sine = "x^2", cosine = "x - 1"))

## Simulate data
dataset <- wide2long(prediction)
dataset <- dataset[seq(1, nrow(dataset), 5),]
dataset$value <- dataset$value + rnorm(nrow(dataset), 0, .1)
dataset$sigma <- 0.1
data <- as.datalist(dataset, split.by = "condition")

## Plotting data
# plot(data)
plotData(data)

## Plotting data and prediction with subsetting
# plot(prediction, data)
plotCombined(prediction, data)
plotCombined(prediction, data,
             time <= 0.5 & condition == "frequency_1")
plotCombined(prediction, data,
             time <= 0.5 & condition != "frequency_1",
             facet = "grid")

```

plotFluxes

Plot Fluxes given a list of flux Equations

Description

Plot Fluxes given a list of flux Equations

Usage

```
plotFluxes(pouter, x, times, fluxEquations, nameFlux = "Fluxes:",
           fixed = NULL)
```

Arguments

pouter	parameters
x	The model prediction function <code>x(times, pouter, fixed, ...)</code>
times	Numeric vector of time points for the model prediction
fluxEquations	list of chars containing expressions for the fluxes, if names are given, they are shown in the legend. Easy to obtain via subset.eqnlist , see Examples.
nameFlux	character, name of the legend.
fixed	Named numeric vector with fixed parameters

Value

A plot object of class `ggplot`.

Examples

```
## Not run:
plotFluxes(bestfit, x, times, attr(subset(f, "B"%in%Product), "rates"), nameFlux = "B production")

## End(Not run)
```

plotPars

Plot parameter values for a fitlist

Description

Plot parameter values for a fitlist

Usage

```
plotPars(x, tol = 1, ...)
```

Arguments

x	parameter frame as obtained by <code>as.parframe(mstrust)</code>
tol	maximal allowed difference between neighboring objective values to be recognized as one.
...	arguments for subsetting of x

plotPaths	<i>Profile likelihood: plot of the parameter paths.</i>
-----------	---

Description

Profile likelihood: plot of the parameter paths.

Usage

```
plotPaths(profs, ..., whichPar = NULL, sort = FALSE, relative = TRUE,
          scales = "fixed")
```

Arguments

profs	profile or list of profiles as being returned by profile
...	arguments going to subset
whichPar	Character or index vector, indicating the parameters that are taken as possible 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.
relative	logical indicating whether the origin should be shifted.
scales	character, either "free" or "fixed".

Details

See [profile](#) for examples.

Value

A plot object of class ggplot.

plotPrediction	<i>Plot a list of model predictions</i>
----------------	---

Description

Plot a list of model predictions

Usage

```
plotPrediction(prediction, ..., scales = "free", facet = "wrap",
              transform = NULL)
```

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.
scales	The scales argument of facet_wrap or facet_grid, i.e. "free", "fixed", "free_x" or "free_y"
facet	Either "wrap" or "grid"
transform	list of transformation for the states, see coordTransform .

Details

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

Value

A plot object of class ggplot.

Examples

```
## Observation function
fn <- eqnvec(
  sine = "1 + sin(6.28*omega*time)",
  cosine = "cos(6.28*omega*time)"
)
g <- Y(fn, parameters = "omega")

## Prediction function for time
x <- Xt()

## Parameter transformations to split conditions
p <- NULL
for (i in 1:3) {
  p <- p + P(trafo = c(omega = paste0("omega_", i)), condition = paste0("frequency_", i))
}

## Evaluate prediction
times <- seq(0, 1, .01)
pars <- structure(seq(1, 2, length.out = 3), names = attr(p, "parameters"))

prediction <- (g*x*p)(times, pars)

## Plotting prediction
# plot(prediction)
plotPrediction(prediction)
plotPrediction(prediction, scales = "fixed")
plotPrediction(prediction, facet = "grid")
plotPrediction(prediction,
  scales = "fixed",
  transform = list(sine = "x^2", cosine = "x - 1"))
```

```

## Simulate data
dataset <- wide2long(prediction)
dataset <- dataset[seq(1, nrow(dataset), 5),]
dataset$value <- dataset$value + rnorm(nrow(dataset), 0, .1)
dataset$sigma <- 0.1
data <- as.datalist(dataset, split.by = "condition")

## Plotting data
# plot(data)
plotData(data)

## Plotting data and prediction with subsetting
# plot(prediction, data)
plotCombined(prediction, data)
plotCombined(prediction, data,
             time <= 0.5 & condition == "frequency_1")
plotCombined(prediction, data,
             time <= 0.5 & condition != "frequency_1",
             facet = "grid")

```

plotProfile

Profile likelihood plot

Description

Profile likelihood plot

Usage

```
plotProfile(profs, ..., maxvalue = 5, parlist = NULL)
```

Arguments

profs	Lists of profiles as being returned by profile .
...	logical going to subset before plotting.
maxvalue	Numeric, the value where profiles are cut off.
parlist	Matrix or data.frame with columns for the parameters to be added to the plot as points. If a "value" column is contained, deltas are calculated with respect to lowest chisquare of profiles.

Details

See [profile](#) for examples.

Value

A plot object of class ggplot.

plotResiduals	<i>Plot residuals for a fitlist</i>
---------------	-------------------------------------

Description

Plot residuals for a fitlist

Usage

```
plotResiduals(parframe, x, data, split = "condition", errmodel = NULL, ...)
```

Arguments

parframe	Object of class parframe, e.g. returned by mstrust
x	Prediction function returning named list of data.frames with names as data.
data	Named list of data.frames, i.e. with columns name, time, value and sigma.
split	List of characters specifying how to summarise the residuals by $\sqrt{\text{res}_i^2}$, split[1] used for x-axis, split[2] for grouping (color), and any additional for facet_wrap()
errmodel	object of type prdfn, the error model function.
...	Additional arguments for x

Value

A plot object of class ggplot with data.frame as attribute attr(P, "out").

Examples

```
## Not run:
# time axis:
plotResiduals(myfitlist, g*x*p, data,
  c("time", "index", "condition", "name"),
  conditions = myconditions[1:4])
# condition axis (residuals summed over time for each observable and condition):
plotResiduals(myfitlist, g*x*p, data, c("condition", "name", "index"))

## End(Not run)
```

plotValues	<i>Plotting objective values of a collection of fits</i>
------------	--

Description

Plotting objective values of a collection of fits

Usage

```
plotValues(x, tol = 1, ...)
```

Arguments

x	data.frame with columns "value", "converged" and "iterations", e.g. a parframe .
tol	maximal allowed difference between neighboring objective values to be recognized as one.
...	arguments for subsetting of x

prdfn	<i>Prediction function</i>
-------	----------------------------

Description

A prediction function is a function `x(..., fixed, deriv, conditions)`. Prediction functions are generated by [Xs](#), [Xf](#) or [Xd](#). For an example see the last one.

Usage

```
prdfn(P2X, parameters = NULL, condition = NULL)
```

Arguments

P2X	transformation function as being produced by Xs .
parameters	character vector with parameter names
condition	character, the condition name

Details

Prediction functions can be "added" by the "+" operator, see [sumfn](#). Thereby, predictions for different conditions are merged or overwritten. Prediction functions can also be concatenated with other functions, e.g. observation functions ([obsfn](#)) or parameter transformation functions ([parfn](#)) by the "*" operator, see [prodfn](#).

Value

Object of class `prdfn`, i.e. a function `x(..., fixed, deriv, conditions, env)` which returns a [prdfn](#). The arguments `times` and `pars` (parameter values) should be passed via the `...` argument, in this order.

Examples

```
# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)
```


Description

A prediction frame is used to store a model prediction in a matrix. The columns of the matrix are "time" and one column per state. The prediction frame has attributes "deriv", the matrix of sensitivities with respect to "outer parameters" (see [P](#)), an attribute "sensitivities", the matrix of sensitivities with respect to the "inner parameters" (the model parameters, left-hand-side of the parameter transformation) and an attributes "parameters", the parameter vector of inner parameters to produce the prediction frame.

Prediction frames are usually the constituents of prediction lists ([prdlst](#)). They are produced by [Xs](#), [Xd](#) or [Xf](#). When you define your own prediction functions, see P2X in [prdfn](#), the result should be returned as a prediction frame.

Usage

```
prdframe(prediction = NULL, deriv = NULL, sensitivities = NULL,
          parameters = NULL)
```

Arguments

prediction	matrix of model prediction
deriv	matrix of sensitivities wrt outer parameters
sensitivities	matrix of sensitivitie wrt inner parameters
parameters	names of the outer paramters

Value

Object of class prdframe, i.e. a matrix with other matrices and vectors as attributes.

prdlst	<i>Prediction list</i>
--------	------------------------

Description

A prediction list is used to store a list of model predictions from different prediction functions or the same prediction function with different parameter specifications. Each entry of the list is a [prdframe](#).

Usage

```
prdlst(...)

as.prdlst(x, ...)

## S3 method for class 'list'
as.prdlst(x = NULL, names = NULL, ...)
```

Arguments

...	objects of class <code>prdfn</code> conditions.
x	list of prediction frames
names	character vector, the list names, e.g. the names of the experimental

predict.prdfn

Model Predictions

Description

Make a model prediction for times and a parameter frame. The function is a generalization of the standard prediction by a prediction function object in that it allows to pass a parameter frame instead of a single parameter vector.

Usage

```
## S3 method for class 'prdfn'
predict(object, ..., times, pars, data = NULL)
```

Arguments

object	prediction function
...	Further arguments go to the prediction function
times	numeric vector of time points
pars	parameter frame, e.g. output from <code>mstrust</code> or <code>profile</code>
data	data list object. If data is passed, its <code>condition.grid</code> attribute is used to augment the output dataframe by additional columns. "data" itself is returned as an attribute.

Value

A data frame

print.eqnlist *Print or pander equation list*

Description

Print or pander equation list

Usage

```
## S3 method for class 'eqnlist'  
print(x, pander = FALSE, ...)
```

Arguments

x	object of class eqnlist
pander	logical, use pander for output (used with R markdown)
...	additional arguments

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>
Daniel Kaschek, <daniel.kaschek@physik.uni-freiburg.de>

print.eqnvec *Print equation vector*

Description

Print equation vector

Usage

```
## S3 method for class 'eqnvec'  
print(x, width = 140, pander = FALSE, ...)
```

Arguments

x	object of class eqnvec .
width	numeric, width of the print-out
pander	logical, use pander for output (used with R markdown)
...	not used right now

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

print.parfn

Pretty printing parameter transformations

Description

Pretty printing parameter transformations

Usage

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

Arguments

x	prediction function
...	additional arguments

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

print.parvec

Pretty printing for a parameter vector

Description

Pretty printing for a parameter vector

Usage

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

Arguments

x	object of class parvec
...	not used yet.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

print0 *Print object and its "default" attributes only.*

Description

Print object and its "default" attributes only.

Usage

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

Arguments

x	Object to be printed
list_attributes	Prints the names of all attribute of x, defaults to TRUE

Details

Before the 'x' is printed by print.default, all its arguments not in the default list of `attrs` are removed.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>
 Mirjam Fehling-Kaschek, <mirjam.fehling@physik.uni-freiburg.de>

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(mu, lambda = "lambda", attr.name = "prior", condition = NULL)
```

Arguments

mu	Named numeric, the prior values
lambda	Character of length one. The name of the penalty parameter in p.
attr.name	character. The constraint value is additionally returned in an attributed with this name
condition	character, the condition for which the constraint should apply. If NULL, applies to any condition.

Details

Computes the constraint value

$$e^{\lambda} \|p - \mu\|^2$$

and its derivatives with respect to p and lambda.

Value

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

See Also

[wrss](#)

Examples

```
p <- c(A = 1, B = 2, C = 3, lambda = 0)
mu <- c(A = 0, B = 0)
obj <- priorL2(mu = mu, lambda = "lambda")
obj(pars = p + rnorm(length(p), 0, .1))
```

profile

Profile-likelihood (PL) computation

Description

Profile-likelihood (PL) computation

Usage

```
profile(obj, pars, whichPar, alpha = 0.05, limits = c(lower = -Inf, upper =
  Inf), method = c("integrate", "optimize"), stepControl = NULL,
  algoControl = NULL, optControl = NULL, verbose = FALSE, cores = 1,
  ...)
```

Arguments

obj	Objective function <code>obj(pars, fixed, ...)</code> returning a list with "value", "gradient" and "hessian". If attribute "valueData" and/or "valuePrior" are returned they are attached to the return value.
pars	Parameter vector corresponding to the log-likelihood optimum.
whichPar	Numeric or character vector. The parameters for which the profile is computed.
alpha	Numeric, the significance level based on the chisquare distribution with <code>df=1</code>
limits	Numeric vector of length 2, the lower and upper deviance from the original value of <code>pars[whichPar]</code>
method	Character, either "integrate" or "optimize". This is a short-cut for setting <code>stepControl</code> , <code>algoControl</code> and <code>optControl</code> by hand.

stepControl	List of arguments controlling the step adaption. Defaults to integration set-up, i.e. <code>list(stepsize = 1e-4, min = 1e-4, max = Inf, atol = 1e-2, rtol = 1e-2, limit = 100)</code>
algoControl	List of arguments controlling the fast PL algorithm. defaults to <code>list(gamma = 1, W = "hessian", reoptimize = TRUE)</code>
optControl	List of arguments controlling the <code>trust()</code> optimizer. Defaults to <code>list(rinit = .1, rmax = 10, iterlim = 100)</code> . See trust for more details.
verbose	Logical, print verbose messages.
cores	number of cores used when computing profiles for several parameters.
...	Arguments going to <code>obj()</code>

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 pseudoinverse is computed by removing all singular values lower than `reg`.

`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 absolute 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. `stop` is a character, usually "value" or "data", for which the significance level `alpha` is evaluated.

Value

Named list of length one. The name is the parameter name. The list entry is a matrix with columns "value" (the objective value), "constraint" (deviation of the profiled parameter from the original value), "stepsize" (the stepsize take for the iteration), "gamma" (the gamma value employed for the iteration), "valueData" and "valuePrior" (if specified in `obj`), one column per parameter (the profile paths).

Examples

```
## Not run:

## Parameter transformation
trafo <- eqnvec(a = "exp(loga)",
               b = "exp(logb)",
               c = "exp(loga)*exp(logb)*exp(logc)")
p <- P(trafo)

## Objective function
obj1 <- constraintL2(mu = c(a = .1, b = 1, c = 10), sigma = .6)
```

```

obj2 <- constraintL2(mu = c(loga = 0, logb = 0), sigma = 10)
obj <- obj1*p + obj2

## Initialize parameters and obtain fit
pars <- c(loga = 1, logb = 1, logc = 1)
myfit <- trust(obj, pars, rinit = 1, rmax = 10)
myfit.fixed <- trust(obj, pars[-1], rinit = 1, rmax = 10, fixed = pars[1])

## Compute profiles by integration method
profiles.approx <- do.call(
  rbind,
  lapply(1:3, function(i) {
    profile(obj, myfit$argument, whichPar = i, limits = c(-3, 3),
            method = "integrate")
  })
)

## Compute profiles by repeated optimization
profiles.exact <- do.call(
  rbind,
  lapply(1:3, function(i) {
    profile(obj, myfit$argument, whichPar = i, limits = c(-3, 3),
            method = "optimize")
  })
)

## Compute profiles for fit with fixed element by integration method
profiles.approx.fixed <- do.call(
  rbind,
  lapply(1:2, function(i) {
    profile(obj, myfit.fixed$argument, whichPar = i, limits = c(-3, 3),
            method = "integrate",
            fixed = pars[1])
  })
)

## Plotting
plotProfile(profiles.approx)
plotProfile(list(profiles.approx, profiles.exact))
plotProfile(list(profiles.approx, profiles.approx.fixed))

plotPaths(profiles.approx, sort = TRUE)
plotPaths(profiles.approx, whichPar = "logc")
plotPaths(list(profiles.approx, profiles.approx.fixed), whichPar = "logc")

## End(Not run)

```


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.

`python.version.request`

Check if rPython comes with the correct Python version

Description

rPython is linked against a certain Python version found on the system. If Python code called from R requires a specific Python version, the rPython package needs to be reinstalled. This functions helps to do this in one line.

Usage

```
python.version.request(version)
```

Arguments

version	character indicating the requested Python version
---------	---

Value

TRUE if rPython is linked against the requested version. Otherwise, the user is asked if rPython should be reinstalled with the correctly linked Python version.

python.version.rpython

Get the Python version to which rPython is linked

Description

Get the Python version to which rPython is linked

Usage

python.version.rpython()

Value

The Python version and additional information

python.version.sys

Check which Python versions are installed on the system

Description

Check which Python versions are installed on the system

Usage

python.version.sys(version = NULL)

Arguments

version NULL or character. Check for specific version

Value

Character vector with the python versions and where they are located.

reduceReplicates	<i>Reduce replicated measurements to mean and standard deviation</i>
------------------	--

Description

Obtain the mean and standard deviation from replicates per condition.

Usage

```
reduceReplicates(file, select = "condition", datatrans = NULL)
```

Arguments

<code>file</code>	Data file of csv. See Format for details.
<code>select</code>	Names of the columns in the data file used to define conditions, see Details.
<code>datatrans</code>	Character vector describing a function to transform data. Use <code>x</code> to refer to data.

Format

The following columns are mandatory for the data file.

name Name of the observed species.

time Measurement time point.

value Measurement value.

condition The condition under which the observation was made.

In addition to these columns, any number of columns can follow to allow a fine grained definition of conditions. The values of all columns named in 'select' are then merged to get the set of conditions.

Details

Experiments are usually repeated multiple times possibly under different conditions leading to replicated measurements. The column "Condition" in the data allows to group the data by their condition. However, sometimes, a more fine grained grouping is desirable. In this case, any number of additional columns can be appended to the data. These columns are referred to as "condition identifier". Which of the condition identifiers are used to do the grouping is user defined by announcing the 'select'. The mandatory column "Condition" is always used. The total set of different conditions is thus defined by all combinations of values occurring in the selected condition identifiers. The replicates of each condition are then reduced to mean and variance. New condition names are derived by merging all conditions which were used in mean and std.

Value

A data frame of the following variables

time Measurement time point.

name Name of the observed species.

value Mean of replicates.

sigma Standard error of the mean, NA for single measurements.

n The number of replicates reduced.

condition The condition for which the value and sigma were calculated. If more than one column were used to define the condition, this variable holds the effective condition which is the combination of all applied single conditions.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

 repar

Reparameterization

Description

Reparameterization

Usage

```
repar(expr, trafo = NULL, ...)
```

Arguments

expr	character of the form "lhs ~ rhs" where rhs reparameterizes lhs. Both lhs and rhs can contain a number of symbols whose values need to be passed by the ... argument.
trafo	character or equation vector or list thereof. The object where the replacement takes place in
...	pass symbols as named arguments

Details

Left and right-hand side of expr are searched for symbols. If separated by "_", symbols are recognized as such, e.g. in Delta_x where the symbols are "Delta" and "x". Each symbol for which values (character or numbers) are passed by the ... argument is replaced.

Value

an equation vector with the reparameterization.

Examples

```

innerpars <- letters[1:3]
constraints <- c(a = "b + c")
mycondition <- "cond1"

trafo <- repara("x ~ x", x = innerpars)
trafo <- repara("x ~ y", trafo, x = names(constraints), y = constraints)
trafo <- repara("x ~ exp(x)", trafo, x = innerpars)
trafo <- repara("x ~ x + Delta_x_condition", trafo, x = innerpars, condition = mycondition)

```

repWithNames	<i>Named repetitions</i>
--------------	--------------------------

Description

Wrapper on rep() to input names instead of length.

Usage

```
repWithNames(x, names)
```

Arguments

x	Value to be repeated.
names	List of names.

res	<i>Compare data and model prediction by computing residuals</i>
-----	---

Description

Compare data and model prediction by computing residuals

Usage

```
res(data, out, err = NULL)
```

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 subset of those contained in the sensitivity equations). If "deriv" is given, also "parameters" needs to be given.
err	output of the error model function

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 divided 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"))
```

rref *Transform matrix A into reduced row echelon form this function is written along the lines of the rref-matlab function.*

Description

Transform matrix A into reduced row echelon form this function is written along the lines of the rref-matlab function.

Usage

```
rref(A, tol = sqrt(.Machine$double.eps), verbose = FALSE,
     fractions = FALSE)
```

Arguments

A	matrix for which the reduced row echelon form is searched
tol	tolerance to find pivots
verbose	logical, print verbose information
fractions	logical, not used right now.

Value

a list of two entries is returned; `ret[[1]]` is the reduced row echelon form of A, `ret[[2]]` is the index of columns in which a pivot was found

Author(s)

Malenka Mader, <Malenka.Mader@fdm.uni-freiburg.de>

runbg	<i>Run an R expression in the background (only on UNIX)</i>
-------	---

Description

Generate an R code of the expression that is copied via scp to any machine (ssh-key needed). Then collect the results.

Usage

```
runbg(..., machine = "localhost", filename = NULL, input = ls(.GlobalEnv),
       compile = FALSE, wait = FALSE)
```

Arguments

...	Some R code
machine	Character vector, e.g. "localhost" or "knecht1.fdm.uni-freiburg.de" or <code>c(localhost, localhost)</code> .
filename	Character, defining the filename of the temporary file. Random file name is chosen if NULL.
input	Character vector, the objects in the workspace that are stored into an R data file and copied to the remote machine.
compile	Logical. If TRUE, C files are copied and compiled on the remote machine. Otherwise, the .so files are copied.
wait	Logical. Wait until executed. If TRUE, the code checks if the result file is already present in which case it is loaded. If not present, <code>runbg()</code> starts, produces the result and loads it as <code>.runbgOutput</code> directly into the workspace. If <code>wait = FALSE</code> , <code>runbg()</code> starts in the background and the result is only loaded into the workspace when the <code>get()</code> function is called, see Value section.

Details

runbg() generates a workspace from the input argument and copies the workspace and all C files or .so files to the remote machines via scp. This will only work if *an ssh-key had been generated and added to the authorized keys on the remote machine*. The code snippet, i.e. the ... argument, can include several intermediate results but only the last call which is not redirected into a variable is returned via the variable .runbgOutput, see example below.

Value

List of functions check, get() and purge(). check() checks, if the result is ready. get() copies the result file to the working directory and loads it into the workspace as an object called .runbgOutput. This object is a list named according to the machines that contains the results returned by each machine. purge() deletes the temporary folder from the working directory and the remote machines.

Examples

```
## Not run:
out_job1 <- runbg({
  M <- matrix(rnorm(1e2), 10, 10)
  solve(M)
}, machine = c("localhost", "localhost"), filename = "job1")
out_job1$check()
out_job1$get()
result <- .runbgOutput
print(result)
out_job1$purge()

## End(Not run)
```

runbgInstall

Remote install dMod to a ssh-reachable host

Description

Install your local dMod version to a remote host via ssh.

Usage

```
runbgInstall(sshtarget, source = NULL, type = "local")
```

Arguments

sshtarget	The ssh host url.
source	If type = local, source must point to the source directory of your dMod version. This is most probably you local dMod git repository.
type	Which dMod to install. At the moment, only your local version is supported.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

runbg_bwfor

Run an R expression on the bwForCluster

Description

Generate an R code of the expression that is copied via scp to the bwForCluster (ssh-key needed). Then collect the results.

Usage

```
runbg_bwfor(..., machine, filename = NULL, nodes = 1, cores = 1,
  walltime = "01:00:00", input = ls(.GlobalEnv), compile = TRUE)
```

Arguments

...	Some R code
machine	e.g. fr_dk846@bwfor.cluster.uni-mannheim.de
filename	Character, defining the filename of the temporary file. Random file name is chosen if NULL.
nodes	Number of nodes, e.g. 10
cores	Number of cores, e.g. 16
walltime	estimated runtime in the format hh:mm:ss, e.g. 01:30:00. Jobs with a walltime up to 30 min are sent to a quick queue. When the walltime is exceeded, all jobs are automatically killed by the queue.
input	Character vector, the objects in the workspace that are stored into an R data file and copied to the remote machine.
compile	Logical. If TRUE, C files are copied and compiled on the remote machine. Otherwise, the .so files are copied.

Details

runbg() generates a workspace from the input argument and copies the workspace and all C files or .so files to the remote machines via scp. This will only work if *an ssh-key had been generated and added to the authorized keys on the remote machine*. The code snippet, i.e. the ... argument, can include several intermediate results but only the last call which is not redirected into a variable is returned via the variable .runbgOutput, see example below.

Value

List of functions check(), get() and purge(). check() checks, if the result is ready. get() copies the result file to the working directory and loads it into the workspace as an object called .runbgOutput. This object is a list named according to the machines that contains the results returned by each machine. purge() deletes the temporary folder from the working directory and the remote machines.

Examples

```
## Not run:
out_job1 <- runbg({
  mstrust(obj, center, fits = 10, cores = 2)
},
machine = "bwfor", nodes = 2, cores = "2:best",
walltime = "00:01:00",
filename = "job1")
out_job1$check()
out_job1$get()
out_job1$purge()
result <- .runbgOutput
print(result)

## End(Not run)
```

scale_color_dMod

Standard dMod color palette

Description

Standard dMod color palette

Usage

```
scale_color_dMod(...)
```

Arguments

... arguments goint to `codescale_color_manual()`

Examples

```
library(ggplot2)
times <- seq(0, 2*pi, 0.1)
values <- sin(times)
data <- data.frame(
  time = times,
  value = c(values, 1.2*values, 1.4*values, 1.6*values),
  group = rep(c("C1", "C2", "C3", "C4"), each = length(times))
)
qplot(time, value, data = data, color = group, geom = "line") +
  theme_dMod() + scale_color_dMod()
```

scale_fill_dMod	<i>Standard dMod color scheme</i>
-----------------	-----------------------------------

Description

Standard dMod color scheme

Usage

```
scale_fill_dMod(...)
```

Arguments

... arguments goint to codescale_color_manual()

stat.parlist	<i>Gather statistics of a fitlist</i>
--------------	---------------------------------------

Description

Gather statistics of a fitlist

Usage

```
stat.parlist(x)
```

Arguments

x The fitlist

steadyStates	<i>Calculate analytical steady states</i>
--------------	---

Description

Calculate analytical steady states

Usage

```
steadyStates(model, file = NULL, smatrix = NULL, states = NULL,
  rates = NULL, forcings = NULL, givenCQs = NULL, neglect = NULL,
  sparsifyLevel = 2, outputFormat = "R")
```

Arguments

model	Either name of the csv-file or the eqnlist of the model. If NULL, specify smatrix, states and rates by hand.
file	Name of the file to which the steady-state equations are saved. Read this file with readRDS .
smatrix	Numeric matrix, stiochiometry matrix of the system
states	Character vector, state vector of the system
rates	Character vector, flux vector of the system
forcings	Character vector with the names of the forcings
givenCQs	Character vector with conserved quantities. Use the format c("A + pA = totA", "B + pB = totB"). If NULL, conserved quantities are automatically calculated.
neglect	Character vector with names of states and parameters that must not be used for solving the steady-state equations
sparsifyLevel	numeric, Upper bound for length of linear combinations used for simplifying the stoichiometric matrix
outputFormat	Define the output format. By default "R" generating dMod compatible output. To obtain an output appropriate for d2d [1] "M" must be selected.

Value

Character vector of steady-state equations.

Author(s)

Marcus Rosenblatt, <marcus.rosenblatt@fdm.uni-freiburg.de>

References

[1] <https://bitbucket.org/d2d-development/d2d-software/wiki/Home>

strelide

Elide character vector

Description

Elide character vector

Usage

```
strelide(string, width, where = "right", force = FALSE)
```

Arguments

string	String subject to eliding
width	Width including eliding ... of return string
where	Eliding can happen at 'left', 'middel', or 'right'. Defaults to 'right'.
force	Elide, even is <string> is shorter than <width>. Default to 'FALSE'.

Details

Elide a string to <width>. Eliding can happen at 'left', 'middle', or 'right'. # If forcing = FALSE, which is the default, strings shorter than <width> are returned unaltered; forcing = TRUE inserts eliding symbols (...) in any case.

Value

Elided string of length <width>.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

strpad	<i>Pad string to desired width</i>
--------	------------------------------------

Description

Pad string to desired width

Usage

```
strpad(string, width, where = "right", padding = " ", autoelide = FALSE)
```

Arguments

string	String to pad
width	Desired width of padded string
where	Padding can be inserted to the right or left of <string>. Default to 'right'.
padding	A single character with which the padding space is filled. Defaults to blank ' ' yielding invisible padding.
autoelide	If TRUE, <string> is elided if it is wider than <width>. The position of eliding follows <where>. Defaults to FALSE.

Value

Padded string of length <width>.

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

submatrix	<i>Submatrix of a matrix returning ALWAYS a matrix</i>
-----------	--

Description

Submatrix of a matrix returning ALWAYS a matrix

Usage

```
submatrix(M, rows = 1:nrow(M), cols = 1:ncol(M))
```

Arguments

M	matrix
rows	Index vector
cols	Index vector

Value

The matrix M[rows, cols], keeping/adjusting attributes like ncol nrow and dimnames.

subset.eqnlist	<i>subset of an equation list</i>
----------------	-----------------------------------

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](#)

Examples

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

summary.eqnvec	<i>Summary of an equation vector</i>
----------------	--------------------------------------

Description

Summary of an equation vector

Usage

```
## S3 method for class 'eqnvec'
summary(object, ...)
```

Arguments

object	of class eqnvec .
...	additional arguments

Author(s)

Wolfgang Mader, <Wolfgang.Mader@fdm.uni-freiburg.de>

symmetryDetection	<i>Search for symmetries in the loaded model</i>
-------------------	--

Description

Search for symmetries in the loaded model

Usage

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

Arguments

f	object containing the ODE for which <code>as.eqnvec()</code> is defined
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 consider fixed
cores	maximal number of cores used for the analysis
allTrafos	do not remove transformations with a common parameter factor

Examples

```
## Not run:
eq <- NULL
eq <- addReaction(eq, "A", "B", "k1*A")
eq <- addReaction(eq, "B", "A", "k2*B")

observables <- eqnvec(Aobs = "alpha * A")

symmetryDetection(eq, observables)

## End(Not run)
```

 theme_dMod

Standard plotting theme of dMod

Description

Standard plotting theme of dMod

Usage

```
theme_dMod(base_size = 11, base_family = "")
```

Arguments

base_size	numeric, font-size
base_family	character, font-name

trustL1

*Non-Linear Optimization***Description**

This function carries out a minimization or maximization of a function using a trust region algorithm. See the references for details.

Usage

```
trustL1(objfun, parinit, mu = 0 * parinit, one.sided = FALSE, lambda = 1,
        rinit, rmax, parscale, iterlim = 100, fterm = sqrt(.Machine$double.eps),
        mterm = sqrt(.Machine$double.eps), minimize = TRUE, blather = FALSE,
        blather2 = FALSE, ...)
```

```
trust(objfun, parinit, rinit, rmax, parscale, iterlim = 100,
      fterm = sqrt(.Machine$double.eps), mterm = sqrt(.Machine$double.eps),
      minimize = TRUE, blather = FALSE, parupper = Inf, parlower = -Inf,
      ...)
```

Arguments

objfun	an R function that computes value, gradient, and Hessian of the function to be minimized or maximized and returns them as a list with components value, gradient, and hessian. Its first argument should be a vector of the length of parinit followed by any other arguments specified by the ... argument.
parinit	starting parameter values for the optimization. Must be feasible (in the domain).
mu	named numeric value. The reference value for L1 penalized parameters.
one.sided	logical. One-sided penalization.
lambda	strength of the L1 penalty
rinit	starting trust region radius. The trust region radius (see details below) is adjusted as the algorithm proceeds. A bad initial value wastes a few steps while the radius is adjusted, but does not keep the algorithm from working properly.
rmax	maximum allowed trust region radius. This may be set very large. If set small, the algorithm traces a steepest descent path (steepest ascent, when minimize = FALSE).
parscale	an estimate of the size of each parameter at the minimum. The algorithm operates as if optimizing function(x, ...) objfun(x / parscale, ...). May be missing in which case no rescaling is done. See also the details section below.
iterlim	a positive integer specifying the maximum number of iterations to be performed before the program is terminated.
fterm	a positive scalar giving the tolerance at which the difference in objective function values in a step is considered close enough to zero to terminate the algorithm.

mterm	a positive scalar giving the tolerance at which the two-term Taylor-series approximation to the difference in objective function values in a step is considered close enough to zero to terminate the algorithm.
minimize	If TRUE minimize. If FALSE maximize.
blather	If TRUE return extra info.
blather2	even more information
...	additional argument to objfun
parupper	named numeric vector of upper bounds.
parlower	named numeric vector of lower bounds.

Details

See Fletcher (1987, Section 5.1) or Nocedal and Wright (1999, Section 4.2) for detailed expositions.

Value

A list containing the following components:

- value: the value returned by objfun at the final iterate.
- gradient: the gradient returned by objfun at the final iterate.
- hessian: the Hessian returned by objfun at the final iterate.
- argument: the final iterate
- converged: if TRUE the final iterate was deemed optimal by the specified termination criteria.
- iterations: number of trust region subproblems done (including those whose solutions are not accepted).
- argpath: (if blather == TRUE) the sequence of iterates, not including the final iterate.
- argtry: (if blather == TRUE) the sequence of solutions of the trust region subproblem.
- steptype: (if blather == TRUE) the sequence of cases that arise in solutions of the trust region subproblem. "Newton" means the Newton step solves the subproblem (lies within the trust region). Other values mean the subproblem solution is constrained. "easy-easy" means the eigenvectors corresponding to the minimal eigenvalue of the rescaled Hessian are not all orthogonal to the gradient. The other cases are rarely seen. "hard-hard" means the Lagrange multiplier for the trust region constraint is minus the minimal eigenvalue of the rescaled Hessian; "hard-easy" means it isn't.
- accept: (if blather == TRUE) indicates which of the sequence of solutions of the trust region subproblem were accepted as the next iterate. (When not accepted the trust region radius is reduced, and the previous iterate is kept.)
- r: (if blather == TRUE) the sequence of trust region radii.
- rho: (if blather == TRUE) the sequence of ratios of actual over predicted decrease in the objective function in the trust region subproblem, where predicted means the predicted decrease in the two-term Taylor series model used in the subproblem.
- valpath: (if blather == TRUE) the sequence of objective function values at the iterates.
- valtry: (if blather == TRUE) the sequence of objective function values at the solutions of the trust region subproblem.

- `preddiff`: (if `blather == TRUE`) the sequence of predicted differences using the two-term Taylor-series model between the function values at the current iterate and at the solution of the trust region subproblem.
- `stepnorm`: (if `blather == TRUE`) the sequence of norms of steps, that is distance between current iterate and proposed new iterate found in the trust region subproblem.

<code>wide2long</code>	<i>Translate wide output format (e.g. from ode) into long format</i>
------------------------	--

Description

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

Usage

```
wide2long(out, keep = 1, na.rm = FALSE)
```

Arguments

<code>out</code>	data.frame or matrix or list of matrices in wide format
<code>keep</code>	Index vector, the columns to keep
<code>na.rm</code>	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)`)

<code>wide2long.data.frame</code>	<i>Translate wide output format (e.g. from ode) into long format</i>
-----------------------------------	--

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)`)

<code>wide2long.list</code>	<i>Translate wide output format (e.g. from ode) into long format</i>
-----------------------------	--

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	<i>Translate wide output format (e.g. from ode) into long format</i>
------------------	--

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)`)

write.eqnlist	<i>Write equation list into a csv file</i>
---------------	--

Description

Write equation list into a csv file

Usage

```
write.eqnlist(eqnlist, ...)
```

Arguments

eqnlist	object of class <code>eqnlist</code>
...	Arguments going to <code>write.csv</code>

 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](#)) or object of class [objframe](#).

Value

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

 Xd

Model prediction function from data.frame

Description

Model prediction function from data.frame

Usage

```
Xd(data, condition = NULL)
```

Arguments

data data.frame with columns "name", "time", and row names that are taken as parameter names. The data frame can contain a column "value" to initialize the parameters.

condition either NULL (generic prediction for any condition) or a character, denoting the condition for which the function makes a prediction.

Value

Object of class [prdfn](#), i.e. a function `x(times pars, deriv = TRUE, conditions = NULL)`, see also [Xs](#). Attributes are "parameters", the parameter names (row names of the data frame), and possibly "pouter", a named numeric vector which is generated from `data$value`.

Examples

```
# Generate a data.frame and corresponding prediction function
timesD <- seq(0, 2*pi, 0.5)
mydata <- data.frame(name = "A", time = timesD, value = sin(timesD),
                    row.names = paste0("par", 1:length(timesD)))
x <- Xd(mydata)

# Evaluate the prediction function at different time points
times <- seq(0, 2*pi, 0.01)
pouter <- structure(mydata$value, names = rownames(mydata))
prediction <- x(times, pouter)
plot(prediction)
```

Xf

Model prediction function for ODE models 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(odemodel, forcings = NULL, events = NULL, condition = NULL,
   optionsOde = list(method = "lsoda"))
```

Arguments

<code>odemodel</code>	Object of class odemodel .
<code>forcings,</code>	see Xs
<code>events,</code>	see Xs
<code>condition</code>	either NULL (generic prediction for any condition) or a character, denoting the condition for which the function makes a prediction.
<code>optionsOde</code>	list with arguments to be passed to <code>odeC()</code> 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(..., forcings, events)` are initialized with 0. For more details and the return value see [Xs](#).

Xs

*Model prediction function for ODE models.***Description**

Interface to combine an ODE and its sensitivity equations into one model function `x(times, pars, deriv = TRUE)` returning ODE output and sensitivities.

Usage

```
Xs(odemodel, forcings = NULL, events = NULL, names = NULL,
   condition = NULL, optionsOde = list(method = "lsoda"),
   optionsSens = list(method = "lsodes"))
```

Arguments

<code>odemodel</code>	object of class <code>odemodel</code>
<code>forcings</code>	<code>data.frame</code> with columns <code>name</code> (factor), <code>time</code> (numeric) and <code>value</code> (numeric). The ODE forcings.
<code>events</code>	<code>data.frame</code> of events with columns <code>"var"</code> (character, the name of the state to be affected), <code>"time"</code> (numeric, time point), <code>"value"</code> (numeric, value), <code>"method"</code> (character, either <code>"replace"</code> , <code>"add"</code> or <code>"multiply"</code>). See events . Within <code>Xs()</code> a <code>data.frame</code> of additional events is generated to reset the sensitivities appropriately, depending on the event method. ATTENTION: The additional events are not dynamically recalculated. If you call the prediction function with alternative events, the prediction is fine but the sensitivities can be wrong.
<code>names</code>	character vector with the states to be returned. If <code>NULL</code> , all states are returned.
<code>condition</code>	either <code>NULL</code> (generic prediction for any condition) or a character, denoting the condition for which the function makes a prediction.
<code>optionsOde</code>	list with arguments to be passed to <code>odeC()</code> for the ODE integration.
<code>optionsSens</code>	list with arguments to be passed to <code>odeC()</code> for integration of the extended system

Value

Object of class `prdfn`. If the function is called with parameters that result from a parameter transformation (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 attributes `"deriv"` and `"sensitivities"` do not coincide.

Examples

```
## Not run:
```

```
#####
## Walk through the most frequently used functions in
```



```

## connection with ODE models
#####

library(deSolve)

## Model definition (text-based, scripting part)
r <- NULL
r <- addReaction(r, "A", "B", "k1*A", "translation")
r <- addReaction(r, "B", "", "k2*B", "degradation")

f <- as.eqnvec(r)
observables <- eqnvec(Bobs = "s1*B")

innerpars <- getSymbols(c(names(f), f, observables))
trafo0 <- structure(innerpars, names = innerpars)
trafo0 <- replaceSymbols(innerpars,
                        paste0("exp(", innerpars, ")"),
                        trafo0)

conditions <- c("a", "b")

trafo <- list()
trafo$a <- replaceSymbols("s1", "s1_a", trafo0)
trafo$b <- replaceSymbols("s1", "s1_b", trafo0)

events <- list()
events$a <- data.frame(var = "A", time = 5, value = 1, method = "add")
events$b <- data.frame(var = "A", time = 5, value = 2, method = "add")

## Model definition (generate compiled objects
## and functions from above information)

# ODE model
model <- odemodel(f)

# Observation function
g <- Y(observables, f, compile = TRUE, modelname = "obsfn")

# Parameter transformation for steady states
pSS <- P(f, method = "implicit", condition = NULL)

# Condition-specific transformation and prediction functions
p0 <- x <- NULL
for (C in conditions) {
  p0 <- p0 + P(trafo[[C]], condition = C)
  x <- x + Xs(model, events[[C]], condition = C)
}

## Process data

# Data
data <- datalist(

```

```

a = data.frame(time = c(0, 2, 7),
               name = "Bobs",
               value = c(.3, .3, .3),
               sigma = c(.03, .03, .03)),
b = data.frame(time = c(0, 2, 7),
               name = "Bobs",
               value = c(.1, .1, .2),
               sigma = c(.01, .01, .02))
)

## Evaluate model/data

# Initialize times and parameters
times <- seq(0, 10, .1)
outerpars <- attr(p0, "parameters")
pouter <- structure(rnorm(length(outerpars)),
                  names = outerpars)

# Plot prediction
plot((g*x*p0)(times, pouter))
plot((g*x*pSS*p0)(times, pouter))
plotFluxes(pouter, g*x*p0, times, getFluxes(r)$B)

# Fit data with L2 constraint
constr <- constraintL2(mu = 0*pouter, sigma = 5)
myfit <- trust(normL2(data, g*x*p0) + constr,
              pouter, rinit = 1, rmax = 10)
plot((g*x*p0)(times, myfit$argument), data)

# List of fits
obj <- normL2(data, g*x*p0) + constr
mylist <- mstrust(normL2(data, g*x*p0) + constr,
                 pouter, fits = 10, cores = 4, sd = 4)
plot(mylist)
pars <- as.parframe(mylist)
plotValues(pars)

bestfit <- as.parvec(pars)

# Compute profiles of the fit
profile <- profile(normL2(data, g*x*p0) + constr,
                  bestfit, "k1", limits = c(-5, 5))
plotProfile(profile)

# Add a validation objective function
vali <- datapointL2("A", 2, "mypoint", .1, condition = "a")
# Get validation point parameters
mypoint <- trust(normL2(data, g*x*p0) + constr + vali,
                 c(mypoint = 0), rinit = 1, rmax = 10,
                 fixed = bestfit)$argument
# Compute profile and plot
profile <- profile(normL2(data, g*x*p0) + constr + vali,
                  c(bestfit, mypoint), "mypoint")

```

```

plotProfile(profile)

## Using the controls() function to manipulate objects -----

# List the available controls of an object
controls(x)

# List a specific control
controls(x, condition = "a", name = "optionsSens")

# Change specific control
controls(x, condition = "a", name = "optionsSens") <-
  list(method = "lsoda", rtol = 1e-8, atol = 1e-8)

# Almost every function (objfn, parfn, prdfn, obsfn) saves the arguments
# by which it was invoked in a list named "controls", which can be manipulated
# by the controls() function.

## New example: condition-specific observation parameters -----

# Condition-specific observation parameters
f <- eqnvec(A = "-k1*A", B = "k1*A - k2*B")
observables <- eqnvec(Bobs = "s1*B")
conditions <- c("scale1", "scale2")

dynpars <- getSymbols(c(names(f), f))
obsvars <- setdiff(getSymbols(observables), dynpars)
trafo0 <- structure(c(dynpars, obsvars), names = c(dynpars, obsvars))

obsvars_all <- outer(obsvars, conditions, paste, sep = "_")
trafo_all <- structure(c(dynpars, obsvars_all),
  names = c(dynpars, obsvars_all))
trafo_all <- replaceSymbols(dynpars,
  paste0("exp(", dynpars, ")"),
  trafo_all)
trafo_all <- replaceSymbols(obsvars_all,
  paste0("exp(", obsvars_all, ")"),
  trafo_all)

pObs <- NULL
for (C in conditions) {
  pObs <- pObs + P(replaceSymbols(obsvars,
    paste(obsvars, C, sep = "_"),
    trafo0),
    condition = C)
}

x <- Xs(model)
p <- P(trafo_all)
y <- g*pObs*x*p

```

```

times <- seq(0, 10, .1)
outerpars <- attr(p, "parameters")
pouter <- structure(rnorm(length(outerpars)), names = outerpars)

# Plot prediction
plot((g*pObs*x*p)(times, pouter))

## End(Not run)

```

Xt	<i>Generate a prediction function that returns times</i>
----	--

Description

Function to deal with non-ODE models within the framework of dMod. See example.

Usage

```
Xt(condition = NULL)
```

Arguments

`condition` either NULL (generic prediction for any condition) or a character, denoting the condition for which the function makes a prediction.

Value

Object of class [prdfn](#).

Examples

```

x <- Xt()
g <- Y(c(y = "a*time^2+b"), f = NULL, parameters = c("a", "b"))

times <- seq(-1, 1, by = .05)
pars <- c(a = .1, b = 1)

plot((g*x)(times, pars))

```

 Y *Observation functions.*

Description

Creates an object of type `obsfn` that evaluates an observation function and its derivatives based on the output of a model prediction function, see `prdfn`, as e.g. produced by `Xs`.

Usage

```
Y(g, f = NULL, states = NULL, parameters = NULL, condition = NULL,
  attach.input = TRUE, compile = FALSE, modelname = NULL,
  verbose = FALSE)
```

Arguments

<code>g</code>	Named character vector or equation vector defining the observation function
<code>f</code>	Named character of equations or object that can be converted to <code>eqnvec</code> or object of class <code>fn</code> . If <code>f</code> is provided, <code>states</code> and <code>parameters</code> are guessed from <code>f</code> .
<code>states</code>	character vector, alternative definition of "states", usually the names of <code>f</code> . If both, <code>f</code> and <code>states</code> are provided, the <code>states</code> argument overwrites the states derived from <code>f</code> .
<code>parameters</code>	character vector, alternative definition of the "parameters", usually the symbols contained in "g" and "f" except for <code>states</code> and the code word <code>time</code> . If both, <code>f</code> and <code>parameters</code> are provided, the <code>parameters</code> argument overwrites the parameters derived from <code>f</code> and <code>g</code> .
<code>condition</code>	either <code>NULL</code> (generic prediction for any condition) or a character, denoting the condition for which the function makes a prediction.
<code>attach.input</code>	logical, indicating whether the original input should be returned with the output.
<code>compile</code>	Logical, compile the function (see <code>funC0</code>)
<code>modelname</code>	Character, used if <code>compile = TRUE</code> , sets a fixed filename for the C file.
<code>verbose</code>	Print compiler output to R command line.

Value

Object of class `obsfn`, i.e. a function `y(..., deriv = TRUE, conditions = NULL)` representing the evaluation of the observation function. Arguments `out` (model prediction) and `pars` (parameter values) should be passed by the `...` argument. If `out` has the attribute "sensitivities", the result of `y(out, pars)`, will have an attributed "deriv" which reflects 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.

Examples

```

# Define a time grid on which to make a prediction by piece-wise linear function.
# Then define a (generic) prediction function based on this grid.
times <- 0:5
grid <- data.frame(name = "A", time = times, row.names = paste0("p", times))
x <- Xd(grid)

# Define an observable and an observation function
observables <- eqnvec(Aobs = "s*A")
g <- Y(g = observables, f = NULL, states = "A", parameters = "s")

# Collect parameters and define an overarching parameter transformation
# for two "experimental conditions".
dynpars <- attr(x, "parameters")
obsvars <- attr(g, "parameters")
innerpars <- c(dynpars, obsvars)

trafo <- structure(innerpars, names = innerpars)
trafo_C1 <- replaceSymbols(innerpars, paste(innerpars, "C1", sep = "_"), trafo)
trafo_C2 <- replaceSymbols(innerpars, paste(innerpars, "C2", sep = "_"), trafo)

p <- NULL
p <- p + P(trafo = trafo_C1, condition = "C1")
p <- p + P(trafo = trafo_C2, condition = "C2")

# Collect outer (overarching) parameters and
# initialize with random values
outerpars <- attr(p, "parameters")
pars <- structure(runif(length(outerpars), 0, 1), names = outerpars)

# Predict internal/unobserved states
out1 <- (x*p)(times, pars)
plot(out1)

# Predict observed states in addition to unobserved
out2 <- (g*x*p)(times, pars)
plot(out2)

```

%.*%

Multiplication of objective functions with scalars

Description

The %.*% operator allows to multiply objects of class objlist or objfn with a scalar.

Usage

```
x1 %.*% x2
```

%.%*

119

Arguments

- x1 object of class objfn or objlist.
- x2 numeric of length one.

Value

An objective function or objlist object.

Index

*Topic **data**

- jakstat, 43
- *.fn, 4
- +.datalist, 5
- +.fn, 7
- +.objfn, 8
- +.objlist, 9
- [.parframe (parframe), 61
- [.parvec (parvec), 64
- %.%, 118

- addReaction, 10
- as.data.frame.datalist, 12
- as.data.frame.eqnlist, 12
- as.data.frame.prdlist
 (as.data.frame.datalist), 12
- as.datalist (datalist), 26
- as.eqnlist (eqnlist), 30
- as.eqnvec, 13
- as.objlist, 14
- as.parframe, 14, 16, 49
- as.parlist (parlist), 63
- as.parvec (parvec), 64
- as.parvec.parframe, 15
- as.prdlist (prdlist), 81
- attrs, 16, 85

- blockdiagSymb, 17

- c.parlist (parlist), 63
- c.parvec (parvec), 64
- combine, 18
- compare, 18
- compile, 20
- confint.parframe, 20
- conservedQuantities, 21
- constraintExp2, 22
- constraintL2, 9, 23, 28, 55
- controls, 24
- controls<- (controls), 24

- coordTransform, 25, 70, 72, 76

- datalist, 26, 27, 53
- datapointL2, 9, 27, 55
- detectFreeCores, 29
- dot, 29

- eqnlist, 10, 12, 30, 31, 33, 39, 41, 83, 102,
 109
- eqnvec, 13, 19, 29, 32, 35, 58, 59, 83, 103
- events, 112
- expand.grid.alt, 33

- fitErrorModel, 33
- forcingsSymb, 35
- format.eqnvec, 35
- funC0, 36, 66, 67, 117

- getCoefficients, 37
- getConditions, 37
- getDerivs, 38
- getFluxes, 13, 38
- getLocalDLLs, 40
- getParameters, 40
- getReactions, 41
- ggopen, 42

- is.eqnlist (eqnlist), 30
- is.eqnvec (eqnvec), 32

- jakstat, 43

- lbind, 43
- load.parlist, 44, 49, 63
- loadDLL, 44
- long2wide, 45
- lsdMod, 45

- mname, 46
- modelName, 46
- modelName<- (modelName), 46

- mssample, 47
- mstrust, 44, 48, 53, 62, 63, 70, 78, 82
- multiroot, 67
- nll, 52
- normL2, 9, 27, 53, 55
- nullZ, 54
- objframe, 53, 55, 110
- objlist, 54, 55
- obsfn, 53, 56, 56, 79, 117
- odemodel, 57, 111, 112
- optim, 34
- P, 7, 59, 59, 60, 81, 112, 117
- parfn, 59, 59, 79
- parframe, 15, 61, 79
- parlist, 63
- parvec, 64
- Pexpl, 59, 66, 68
- Pimpl, 59, 66, 67
- plot.datalist, 69
- plot.parlist, 63, 69
- plot.prdframe (plotCombined), 70
- plot.prdlist (plotCombined), 70
- plotCombined, 70
- plotData, 27, 72
- plotFluxes, 73
- plotPars, 74
- plotPaths, 75
- plotPrediction, 75
- plotProfile, 77
- plotResiduals, 78
- plotValues, 79
- prdfn, 53, 56, 79, 81, 110, 112, 116, 117
- prdfn, 80, 81, 82
- prdfn, 56, 80, 81, 81
- predict.prdfn, 82
- print.eqnlist, 83
- print.eqnvec, 83
- print.parfn, 84
- print.parvec, 84
- print0, 85
- priorL2, 9, 55, 85
- prodfn, 56, 79
- prodfn (* . fn), 4
- profile, 21, 62, 75, 77, 82, 86
- progressBar, 88
- python.version.request, 89
- python.version.rpython, 90
- python.version.sys, 90
- readRDS, 100
- reduceReplicates, 33, 34, 91
- repar, 92
- repWithNames, 93
- res, 53, 55, 69, 70, 72, 93, 110
- resolveRecurrence, 94
- rnorm, 48, 49
- rref, 94
- runbg, 95
- runbg_bwfor, 97
- runbgInstall, 96
- runif, 48, 49
- scale_color_dMod, 98
- scale_fill_dMod, 99
- stat.parlist, 99
- steadyStates, 99
- strelide, 100
- strpad, 101
- submatrix, 102
- subset.eqnlist, 74, 102
- subset.parframe (parframe), 61
- sumdatalist, 27
- sumdatalist (+ .datalist), 5
- sumfn, 56, 60, 79
- sumfn (+ . fn), 7
- summary.eqnvec, 103
- summary.parlist (parlist), 63
- sumobjfn, 53
- sumobjfn (+ .objfn), 8
- sumobjlist, 55
- sumobjlist (+ .objlist), 9
- symmetryDetection, 103
- theme_dMod, 104
- trust, 48, 49, 87
- trust (trustL1), 105
- trustL1, 105
- wide2long, 107
- wide2long.data.frame, 107
- wide2long.list, 108
- wide2long.matrix, 109
- write.csv, 109
- write.eqnlist, 109
- wrss, 24, 28, 86, 110

Xd, [79](#), [81](#), [110](#)

Xf, [79](#), [81](#), [111](#)

Xs, [7](#), [70](#), [76](#), [79](#), [81](#), [110](#), [111](#), [112](#), [117](#)

Xt, [116](#)

Y, [7](#), [56](#), [117](#)