Package 'salad'

May 9, 2026

Type Package

Title Simple Automatic Differentiation

Version 1.2

Date 2024-12-18

Maintainer Hervé Perdry <herve.perdry@universite-paris-saclay.fr>

Description Handles both vector and matrices, using a flexible S4 class for automatic differentiation. The method used is forward automatic differentiation. Many functions and methods have been defined,
so that in most cases, functions written without automatic differentiation in mind can be used without change.

License MIT + file LICENSE

Encoding UTF-8

Depends methods

Imports stats

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Author Hervé Perdry [aut, cre]

Repository CRAN

Date/Publication 2024-12-18 12:30:02 UTC

Contents

apply,dual-method 2
Arithmetic 3
bind 5
c 6
colSums 7
Comparison 8

d 8
diag 9
dnorm 10
dual 11
dual-class 12
dualFun1 13
Extract 14
gradient.descent 16
ifelse 17
inversion 18
MathFun 19
matmult 22
matrix 24
optiWrap 25
outer 26
rep 28
salad 28
shape 29
Summary 30
t 32

Index **33**

`apply,dual-method` *Apply functions over array margins of dual objects*

Description

This method generalizes ‘base::apply‘ to dual objects.

Usage

```
## S4 method for signature 'dual'
apply(X, MARGIN, FUN, ..., simplify = TRUE)
```

Arguments

`X`	a dual object (with array or matrix shape)
`MARGIN`	a vector giving the subscript which the function will be applied over
`FUN`	the function to be applied
`...`	extra arguments for ‘FUN‘
`simplify`	a logical indicating whether the results should be simplified

Value

The returned value depends on the values returned by ‘FUN‘, similarly to ‘base::apply‘

See Also

apply

Examples

```
A <- matrix( c(1,2,3,4), 2, 2)
x <- dual(A)
cs <- apply(x, 2, sum)
cs
d(cs)
# prefered method for summing over the columns
colSums(x)
```

Arithmetic *Arithmetic Operators*

Description

Arithmetic operators for objects of class 'dual'

Usage

```
## S4 method for signature 'dual,dual'
e1 + e2

## S4 method for signature 'dual,numericOrArray'
e1 + e2

## S4 method for signature 'numericOrArray,dual'
e1 + e2

## S4 method for signature 'dual,missing'
e1 + e2

## S4 method for signature 'dual,dual'
e1 - e2

## S4 method for signature 'dual,missing'
e1 - e2

## S4 method for signature 'dual,numericOrArray'
e1 - e2

## S4 method for signature 'numericOrArray,dual'
e1 - e2

## S4 method for signature 'dual,dual'
```

```
e1 * e2

## S4 method for signature 'dual,numeric'
e1 * e2

## S4 method for signature 'numeric,dual'
e1 * e2

## S4 method for signature 'dual,numeric'
e1 / e2

## S4 method for signature 'numeric,dual'
e1 / e2

## S4 method for signature 'dual,dual'
e1 / e2

## S4 method for signature 'dual,numeric'
e1 ^ e2

## S4 method for signature 'numeric,dual'
e1 ^ e2

## S4 method for signature 'dual,dual'
e1 ^ e2
```

Arguments

e1	object of class 'dual' or 'numeric'
e2	object of class 'dual' or 'numeric'

Details

The usual operations are performed, with appropriate propagation of the derivatives

Value

An object of class 'dual'.

Examples

```
x <- dual( c(1,2) )
a <- 2 * x + 3
a
d(a)
b <- x[1] + 3*x[2]
b
d(b)
```

bind *Binding methods for dual objects*

Description

Methods allowing to use ‘cbind‘ and ‘rbind‘ with dual objects.

Usage

```
## S4 method for signature 'dual,dual'
rbind2(x,y,...)

## S4 method for signature 'dual,numericOrArray'
rbind2(x,y,...)

## S4 method for signature 'numericOrArray,dual'
rbind2(x,y,...)

## S4 method for signature 'dual,missing'
rbind2(x,y,...)

## S4 method for signature 'dual,dual'
cbind2(x,y,...)

## S4 method for signature 'dual,numericOrArray'
cbind2(x,y,...)

## S4 method for signature 'numericOrArray,dual'
cbind2(x,y,...)

## S4 method for signature 'dual,missing'
cbind2(x,y,...)
```

Arguments

x, y	dual or numeric objects
...	extra parameters (ignored)

Value

A dual matrix combining the arguments.

Examples

```
x <- dual( c(1, 3) )
y <- cbind(x, 2*x+1, 3*x+2, c(0,1))
y
```

```
d(y, "x1")
```

---

c *Concatenation methods*

Description

Methods have been defined in order to allow the concatenation of ‘dual‘ objects together and with constant objects.

Usage

```
## S4 method for signature 'numericOrArray'
c(x, ...)
```

Arguments

x	first object to concatenate
...	other objects

Value

an object of class dual.

Examples

```
x <- dual( 1 )
# concatenation with a constant
x <- c(x, 2)
x
d(x)
# concatenation of dual objects
x1 <- sum(x)
x2 <- sum(x**2)
y <- c(a = x1, b = x2)  # you can use named arguments
y
d(y)
```

colSums *Row and column sums and means*

Description

Method extending to dual matrices the corresponding methods for dual matrices.

Usage

```
rowSums.dual(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'dual'
rowSums(x, na.rm = FALSE, dims = 1, ...)

colSums.dual(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'dual'
colSums(x, na.rm = FALSE, dims = 1, ...)

rowMeans.dual(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'dual'
rowMeans(x, na.rm = FALSE, dims = 1, ...)

colMeans.dual(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'dual'
colMeans(x, na.rm = FALSE, dims = 1, ...)
```

Arguments

x	a dual matrix or array
na.rm	if 'TRUE', missing values are removed
dims	which dimensions are regarded as rows and cols
...	extra parameters (ignored)

Value

a dual object (usually a dual vector).

Examples

```
x <- dual( c(1,2) )
x <- cbind(x, 2*x+1)
rowSums(x)
d(rowSums(x), "x1")
```

Comparison *Comparison Operators*

Description

Comparison operators for objects of class 'dual'

Usage

```
## S4 method for signature 'dual,ANY'
Compare(e1, e2)
```

Arguments

e1	object of class 'dual' or 'numeric'
e2	object of class 'dual' or 'numeric'

Details

usual comparison operators, ignoring derivatives valuesa

Value

a logical vector

d *get list of derivatives*

Description

Get value, differential of a dual object, and the names of associated variables.

Usage

```
d(x, varnames)

value(x)

## S3 method for class 'dual'
value(x)

## S3 method for class 'numeric'
value(x)

varnames(x)
```

```
## S3 method for class 'dual'
varnames(x)

## S3 method for class 'numeric'
varnames(x)
```

Arguments

x	a dual (or numeric) oject
varnames	(optional) a vector or varnames to take derivatives along

Details

If ‘varnames‘ is provided to the function ‘d‘, a list of derivatives along the given variables will be sent back. In general, it sends back the derivatives along all associated variables.

The ‘varnames‘ function sends back the names of all variables for which a derivative is defined.

Value

A named list of derivatives.

Examples

```
x <- dual(c(3,2))
varnames(x^2)
x**2
value(x**2)
d(x**2)
d(x**2, "x1")
# you can use these methods with a numerical constant
value(1)
varnames(1)
d(1, "x1")
```

---

diag *Matrix diagonals*

Description

Methods extending to dual objects the corresponding methods for numeric objects.

Usage

```
diag.dual(x, nrow, ncol, names = TRUE)

## S4 method for signature 'dual'
diag(x = 1, nrow, ncol, names = TRUE)

## S4 replacement method for signature 'dual,dual'
diag(x) <- value

## S4 replacement method for signature 'dual,numericOrArray'
diag(x) <- value
```

Arguments

x	a dual object
nrow, ncol	(optional) dimensions of result
names	if 'TRUE', pass names along
value	replacement value

Value

A dual object, similarly to 'base::diag'

Examples

```
x <- dual( c(1,2) )
diag(x)
d(diag(x), "x1")
y <- matrix(x, 2, 2)
diag(y) <- 2*diag(y)
y
d(y)
diag(y)
```

---

dnorm *Normal distribution*

Description

Density for the normal distribution, accepting objects of class 'dual'

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)

dnorm.dual(x, mean = 0, sd = 1, log = FALSE)
```

Arguments

x	vector of values
mean	vector of means
sd	vector of standard deviations
log	logical. If TRUE, log of densities are returned

Details

'dnorm.dual' will make straightfoward a computation (in R), that works both with numeric or dual objects. 'dnorm' will call 'dnorm.dual' if any of the objects is of class dual, or 'stats::dnorm' is all objects are of class numeric. As 'stats::dnorm' is in written in C it is factor.

If you care for performance, use 'stats::dnorm' directly for non dual numbers, and 'dnorm.dual' for dual numbers.

Value

a dual object.

Examples

```
x <- dual(0)
dnx <- dnorm(x)
dnx
d(dnx)
```

---

dual *Dual objects*

Description

Create a dual object

Usage

```
dual(x, varnames, dx, constant = FALSE)
```

Arguments

x	a numeric object (vector, matrix, or array)
varnames	(optional) the name of the variables in x
dx	(optional) a list of derivatives for the elements of x
constant	if 'TRUE', then a constant is returned.

Details

The basic usage is `dual(x)` which will create an object of class 'dual' with unit derivatives in each of its components. The variable names will be derived from the names of `x`, or generated in the form `x1`, `x2`, etc.

Another possible usage is `dual(x, varnames = c('x1', 'x2'), constant = TRUE)` which returns an object with null derivatives in `x1` and `x2`.

Finally, a list of derivatives can be defined using option `dx`.

Value

an object of class 'dual'

Examples

```
# simple usage
x <- dual( c(1,2) )
x
d(x)
x <- dual(matrix(c(1,2,3,4), 2, 2))
x
d(x, "x1.1")

# using an object with names
x <- dual( c(a = 1, b = 2) )
x
d(x)

# generate a constant
x <- dual(1, varnames = c("x1", "x2"), constant = TRUE)

# specify dx
x <- dual(c(1,2), dx = list(x1 = c(1,1)))
x
d(x)
# this is equivalent to :
x <- dual(1)
x <- c(x, x + 1)
x
d(x)
```

`dual-class` *dual class*

Description

An S4 class for forward differentiation of vector and matrix computations.

Details

A dual object can be either a vector or a matrix. It can contain derivatives with respect to several variables. The derivatives will have the same shape as the value.

The shape of an object can be changed using 'dim<-'. Note that by default 'as.matrix' and 'as.vector' will send back a regular vector/matrix object, dropping the derivatives. See 'salad' to change this behaviour if needed (this is not the recommended solution).

Many methods and functions have been redefined in the package, in order to allow to apply existing code to 'dual' objects, with no or little change.

Slots

`x` the value of the object. Use the function 'value' to access this slot.

`d` a (named) list of derivatives. Use the function 'd' to access this slot.

See Also

`value`, `d`, `dual`, `salad`.

Examples

```
# creating a vector of length 4
x <- dual( c(1,2,1,0) )
x
d(x)
# turning x into a matrix
dim(x) <- c(2,2)
x
d(x)
# and back into a vector
dim(x) <- NULL
x
# weighted sum of the elements of x
S <- sum(1:4 * x)
S
d(S)
```

---

dualFun1 *Defining in-house derivatives*

Description

Defining the differential of a univariate function

Usage

```
dualFun1(f, df)
```

Arguments

f	a function with a unique argument
df	the differential of f

Details

This function returns a new function that can be applied to a dual object. This allows to extend the package by defining functions it is currenlty unable to derive. It can also gain some time for intensively used functions (see examples below).

Value

Returns a function.

Examples

```
# using salad do compute the differential of a quadratic function
f <- function(x) x**2 + x + 1
x <- dual(4)
f(x)
d(f(x))

# using `dualFun1` to define the differential of f saves time
f1 <- dualFun1(f, \(x) 2*x + 1)
f1(x)
d(f1(x))
system.time( for(i in 1:500) f(x) )
system.time( for(i in 1:500) f1(x) )
```

Extract *Extract or replace parts of an object*

Description

Methods for extraction or replacements of parts of dual objects.

Usage

```
## S4 replacement method for signature 'dual,index,index,dual'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,missing,index,dual'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,index,missing,dual'
x[i, j, ...] <- value
```

```
## S4 replacement method for signature 'dual,missing,missing,dual'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,index,index,logicalOrNumericOrArray'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,missing,index,logicalOrNumericOrArray'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,index,missing,logicalOrNumericOrArray'
x[i, j, ...] <- value

## S4 replacement method for signature 'dual,missing,missing,logicalOrNumericOrArray'
x[i, j, ...] <- value

## S4 method for signature 'dual,index,index'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'dual,missing,index'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'dual,index,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'dual,missing,missing'
x[i, j, ..., drop = TRUE]
```

Arguments

x	dual object
i, j	indices of elements to extract or replace
...	supplementary indices (for arrays)
value	replacement value
drop	for dual matrices or array.

Value

returns a dual object (the semantic is the same as base extraction and replacement methods).

Examples

```
x <- c(1, 2, 3)
x[2] <- dual(4)
x
d(x)
```

`gradient.descent` *Gradient descent*

Description

A simple implementation of the gradient descent algorithm

Usage

```
gradient.descent(
  par,
  fn,
  ...,
  step = 0.1,
  maxit = 100,
  reltol = sqrt(.Machine$double.eps),
  trace = FALSE
)
```

Arguments

`par`	Initial value
`fn`	A function to be minimized (or maximized if 'step' < 0)
`...`	Further arguments to be passed to 'fn'
`step`	Step size. Use a negative value to perform a gradient ascent.
`maxit`	Maximum number of iterations
`reltol`	Relative convergence tolerance
`trace`	If 'TRUE', keep trace of the visited points

Details

First note that this is not an efficient optimisation method. It is included in the package as a demonstration only.

The function iterates $x_{n+1} = x_n - step \times gradf(x_n)$ until convergence. The gradient is computed using automatic differentiation.

The convergence criterion is as in `optim` $\frac{|f(x_{n+1}) - f(x_n)|}{|f(x[n])| + reltol} < reltol$.

Value

a list with components: 'par' is the final value of the parameter, 'value' is the value of 'f' at 'par', 'counts' is the number of iterations performed, 'convergence' is '0' is the convergence criterion was met. If 'trace' is 'TRUE', an extra component 'trace' is included, which is a matrix giving the successive values of x_n.

Examples

```
f <- function(x) (x[1] - x[2])**4 + (x[1] + 2*x[2])**2 + x[1] + x[2]

X <- seq(-1, .5, by = 0.01)
Y <- seq(-0.5, 0.5, by = 0.01)
Z <- matrix(NA_real_, nrow = length(X), ncol = length(Y))
for(i in seq_along(X)) for(j in seq_along(Y)) Z[i,j] <- f(c(X[i],Y[j]))

par(mfrow = c(2,2), mai = c(1,1,1,1)/3)
contour(X,Y,Z, levels = c(-0.2, 0, 0.3, 2**(0:6)), main = "step = 0.01")
gd1 <- gradient.descent(c(0,0), f, step = 0.01, trace = TRUE)
lines(t(gd1$trace), type = "o", col = "red")

contour(X,Y,Z, levels = c(-0.2, 0, 0.3, 2**(0:6)))
gd2 <- gradient.descent(c(0,0), f, step = 0.1, trace = TRUE)
lines(t(gd2$trace), type = "o", col = "red")

contour(X,Y,Z, levels = c(-0.2, 0, 0.3, 2**(0:6)))
gd3 <- gradient.descent(c(0,0), f, step = 0.18, trace = TRUE)
lines(t(gd3$trace), type = "o", col = "red")

contour(X,Y,Z, levels = c(-0.2, 0, 0.3, 2**(0:6)))
gd4 <- gradient.descent(c(0,0), f, step = 0.2, trace = TRUE)
lines(t(gd4$trace), type = "o", col = "red")
```

---

ifelse *Conditionnal Element Selection*

Description

'ifelse' methods extend 'base::ifelse' to allow using dual objects for 'yes' or 'no' arguments.

Usage

```
ifelse(test, yes, no)
```

Arguments

test	an object which can be coerced to logical mode.
yes	return values for true elements of 'test'.
no	return values for false elements of 'test'.

Value

A dual object (dual vector).

Examples

```
x <- dual(c(1,2,4,6))
y <- ifelse(x > 2, x, x/2)
y
d(y)
```

`inversion` *Determinant and matrix inversion for dual matrices*

Description

Methods extending to dual matrices the corresponding methods for numeric matrices.

Usage

```
det.dual(x, ...)

## S4 method for signature 'dual'
det(x, ...)

## S3 method for class 'dual'
determinant(x, logarithm = TRUE, ...)

## S4 method for signature 'dual,dual'
solve(a, b, ...)

## S4 method for signature 'dual,missing'
solve(a, b, ...)

## S4 method for signature 'numericOrArray,dual'
solve(a, b, ...)

## S4 method for signature 'dual,numericOrArray'
solve(a, b, ...)
```

Arguments

`x`	a dual matrix
`...`	extra parameters (ignored)
`logarithm`	if 'TRUE', get logarithm of modulus of determinant
`a, b`	dual or numerical arguments for 'solve'

Value

'det' returns a dual scalar, 'determinant' a list with components 'modulus' (which is a dual object) and 'sign', and 'solve' returns a dual object (vector or matrix).

Examples

```
x <- dual( matrix(c(1,2,1,3), 2, 2) )
det(x)
d(det(x), "x1.1")
solve(x)
d(solve(x), "x1.1")
```

MathFun *Mathematical functions*

Description

various mathematical functions and methods

Usage

```
## S3 method for class 'dual'
exp(x)

## S3 method for class 'dual'
expm1(x)

logNeper(x)

## S3 method for class 'dual'
log(x, base = exp(1))

## S3 method for class 'dual'
log10(x)

## S3 method for class 'dual'
log2(x)

## S3 method for class 'dual'
log1p(x)

## S3 method for class 'dual'
sqrt(x)

## S3 method for class 'dual'
cos(x)

## S3 method for class 'dual'
sin(x)

## S3 method for class 'dual'
```

```
tan(x)

## S3 method for class 'dual'
cospi(x)

## S3 method for class 'dual'
sinpi(x)

## S3 method for class 'dual'
tanpi(x)

## S3 method for class 'dual'
acos(x)

## S3 method for class 'dual'
asin(x)

## S3 method for class 'dual'
atan(x)

## S4 method for signature 'dual,dual'
atan2(y, x)

## S4 method for signature 'dual,numericOrArray'
atan2(y, x)

## S4 method for signature 'numericOrArray,dual'
atan2(y, x)

## S3 method for class 'dual'
cosh(x)

## S3 method for class 'dual'
sinh(x)

## S3 method for class 'dual'
tanh(x)

## S3 method for class 'dual'
acosh(x)

## S3 method for class 'dual'
asinh(x)

## S3 method for class 'dual'
atanh(x)

## S3 method for class 'dual'
```

```
abs(x)

## S3 method for class 'dual'
sign(x)

## S3 method for class 'dual'
ceiling(x)

## S3 method for class 'dual'
floor(x)

## S3 method for class 'dual'
trunc(x, ...)

## S3 method for class 'dual'
gamma(x)

## S3 method for class 'dual'
lgamma(x)

## S3 method for class 'dual'
digamma(x)

## S3 method for class 'dual'
trigamma(x)

psigamma.dual(x, deriv = 0)

## S4 method for signature 'dual'
psigamma(x, deriv = 0)

## S4 method for signature 'dual,dual'
beta(a, b)

## S4 method for signature 'dual,numericOrArray'
beta(a, b)

## S4 method for signature 'numericOrArray,dual'
beta(a, b)

## S4 method for signature 'dual,dual'
lbeta(a, b)

## S4 method for signature 'dual,numericOrArray'
lbeta(a, b)

## S4 method for signature 'numericOrArray,dual'
lbeta(a, b)
```

```
factorial.dual(x)

lfactorial.dual(x)

## S4 method for signature 'dual,numeric'
choose(n, k)

## S4 method for signature 'dual,numeric'
lchoose(n, k)
```

Arguments

x	function argument (dual or numeric object)
base	base to which log is computed
y	first argument of atan2 function (dual or numeric)
...	extra arguments to trunc (unused)
deriv	integer argument to psigamma
a, b	arguments of beta and lbeta (dual or nueumeric)
n	first argument of choose and lchoose (dual)
k	second argument of choose and lchoose (numeric)

Details

The derivative of ‘abs‘ is set to be the function ‘sign‘, so its derivative in 0 is considered as null. You may want to redefine ‘abs‘ using ‘dualFun1‘ to get an undefined derivative.

Value

All functions return dual objects.

Examples

```
x <- dual(1)
y <- log(x)
y
d(y)
```

---

matmult *Matrix Arithmetic*

Description

Methods and functions for dual matrix arithmetic

Usage

```
matrixprod_dn(x, y)

matrixprod_nd(x, y)

matrixprod_dd(x, y)

## S4 method for signature 'dual,numericOrArray'
x %*% y

## S4 method for signature 'numericOrArray,dual'
x %*% y

## S4 method for signature 'dual,dual'
x %*% y

## S4 method for signature 'dual,dual'
crossprod(x, y)

## S4 method for signature 'dual,numericOrArray'
crossprod(x, y)

## S4 method for signature 'numericOrArray,dual'
crossprod(x, y)

## S4 method for signature 'dual,missing'
crossprod(x, y)

## S4 method for signature 'dual,dual'
tcrossprod(x, y)

## S4 method for signature 'dual,numericOrArray'
tcrossprod(x, y)

## S4 method for signature 'numericOrArray,dual'
tcrossprod(x, y)

## S4 method for signature 'dual,missing'
tcrossprod(x, y)
```

Arguments

x, y	Dual or numeric matrices or vectors

Details

All methods are the analog of the corresponding methods for matrices. The functions ‘matrixprod_dd‘, ‘matrixprod_nd‘ and ‘matrixprod_dn‘ are for multiplication of two dual objects, of a

numeric and a dual object, or of a dual and a numeric object, respectively. You may use these functions to save the method dispatching time.

Value

A dual object.

Examples

```
x <- dual( matrix(c(0,1,3,1), 2, 2) )
y <- x %*% c(2,-2)
d(y, "x1.1")
```

`matrix` *Methods for ‘matrix‘, ‘array‘, ‘as.matrix‘ and ‘as.vector‘*

Description

Methods for ‘matrix‘, ‘array‘, ‘as.matrix‘ and ‘as.vector‘

Usage

```
## S4 method for signature 'dual'
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)

## S4 method for signature 'dual'
array(data = NA, dim = length(data), dimnames = NULL)

## S3 method for class 'dual'
as.matrix(x, ...)

## S4 method for signature 'dual'
as.matrix(x, ...)

## S3 method for class 'dual'
as.vector(x, mode = "any")

## S4 method for signature 'dual'
as.vector(x, mode = "any")
```

Arguments

`data, x`	A dual object
`nrow`	the desired number of rows
`ncol`	the desired number of cols
`byrow`	if ‘TRUE‘ the matrix is filled by rows
`dimnames`	A ‘dimnames‘ attributes for a matrix or an array

dim	A ‘dim‘ attributes for an array
...	additional arguments (ignored)
mode	The mode of the vector to create

Details

The default behaviour for ‘as.matrix‘ dans ‘as.vector‘ is to drop the derivatives. This can be modified using ‘salad‘ (to use with care). The prefered method to change the shape is to use ‘dim<-‘.

Value

A dual object for ‘matrix‘ and ‘array‘, a base object for ‘as.matrix‘ and ‘as.vector‘.

See Also

shape, salad, dual-class

Examples

```
x <- dual(c(1,2,0,4))
y <- matrix(x, 2, 2)
y
as.matrix(y)
dim(x) <- c(2,2)
x
```

---

optiWrap *Wrapper for optimisation with automatically computed gradient*

Description

Wrapper for calling `stats::optim` with a gradient computed by automatic differentiation

Usage

```
optiWrap(
  par,
  fn,
  ...,
  method = c("BFGS", "L-BFGS-B", "CG"),
  lower = -Inf,
  upper = Inf,
  control = list(),
  hessian = FALSE,
  trace = FALSE
)
```

Arguments

par	Initial value
fn	Function to be minimized
...	Further argument to be passed to 'fn'
method	Optimization method
lower, upper	Bounds on the variables for 'L-BFGS-B'
control	A list of control parameters passed to 'optim'
hessian	If 'TRUE' a *numerically* differentiated matrix is returned.
trace	If 'TRUE', keep trace of the visited points

Details

The gradient of fn is computed using unlist(d(fn(x))). It is computed at the same time as fn(x)' and stored for when optim calls the gradient. In most cases this should be more efficient than defining gr = \(x) unlist(d(f(dual(x)))).

Parameters 'method' 'lower' 'upper' 'control' and 'hessian' are passed directly to optim.

See Also

optim

Examples

```
f <- function(x) (x[1] - x[2])**4 + (x[1] + 2*x[2])**2 + x[1] + x[2]

X <- seq(-1, 0.5, by = 0.01)
Y <- seq(-1, 0.5, by = 0.01)
Z <- matrix(NA_real_, nrow = length(X), ncol = length(Y))
for(i in seq_along(X)) for(j in seq_along(Y)) Z[i,j] <- f(c(X[i],Y[j]))

contour(X,Y,Z, levels = c(-0.2, 0, 0.3, 2**(0:6)), main = "BFGS")
opt <- optiWrap(c(0,0), f, method = "BFGS", trace = TRUE)
lines(t(opt$trace), type = "o", col = "red")
```

---

outer *Outer product for dual objects*

Description

Method extending to dual object the usual method method

Usage

```
outer.dual(X, Y, FUN = "*", ...)

## S4 method for signature 'dual,dual'
outer(X, Y, FUN = "*", ...)

## S4 method for signature 'numericOrArray,dual'
outer(X, Y, FUN = "*", ...)

## S4 method for signature 'dual,numericOrArray'
outer(X, Y, FUN = "*", ...)

## S4 method for signature 'dual,dual'
X %o% Y

## S4 method for signature 'numericOrArray,dual'
X %o% Y

## S4 method for signature 'dual,numericOrArray'
X %o% Y
```

Arguments

X, Y	arguments of 'FUN'
FUN	function to use in the outer product
...	extra arguments passed to 'FUN'

Details

Methods extending 'outer' and '

Value

A dual matrix.

Examples

```
x <- dual(1:3)
outer(x, x)
d(outer(x,x), "x2")
```

rep *Replicate elements of a dual vector*

Description

A method extending ‘rep‘ to dual objects

Usage

```
## S3 method for class 'dual'
rep(x, ...)
```

Arguments

x	a dual vector
...	extra parameters (typically, ’times’, ’length.out’ or ’each’)

Value

A dual object.

Examples

```
x <- rep( dual(1:2), each = 4 )
x
d(x)
```

salad *Salad options*

Description

Set or get options values for package ’salad’

Usage

```
salad(...)
```

Arguments

...	options to be defined, using ’name = value’, or name(s) of option(s) to get.

Details

Currently, only one option can be defined, `drop.derivatives`, which modifies the bevahiour of S3 methods `as.vector` and `as.matrix` and corresponding S4 methods. The default value is set to 'TRUE', which means that `as.vector` and `as.matrix` will return a 'base' objects, without derivatives. Setting `drop.derivatives = FALSE` will make these functions return an object of class dual. This might be useful to re-use exiting code, but may cause some functions to break, and should be use with care.

Use `salad()` to get the current value of all options, or `salad(name)` to get the current value of a given option.

Value

A list with the defined options, or a single element when `salad(name)` is used.

Examples

```
salad("drop.derivatives")
x <- dual(matrix(c(1,2,3,4), 2, 2))
salad(drop.derivatives = FALSE)
as.vector(x)
salad(drop.derivatives = TRUE)
as.vector(x)
```

---

shape *Dual objects length, dim, names and dimnames*

Description

S3 methods for length, dim, names and dimnames

Usage

```
## S3 method for class 'dual'
length(x)

## S3 method for class 'dual'
dim(x)

## S3 replacement method for class 'dual'
dim(x) <- value

## S3 method for class 'dual'
dimnames(x)

## S3 replacement method for class 'dual'
dimnames(x) <- value
```

```
## S3 method for class 'dual'
names(x)

## S3 replacement method for class 'dual'
names(x) <- value
```

Arguments

x	a dual object
value	for replacement methods, the new value

Details

As the methods ‘dimnames‘ and ‘dimnanes<-.dual‘ have been defined, you can use ‘rownames‘ and ‘colnames‘ as with numeric matrices (see examples).

Value

Return values are similar to the base methods.

Examples

```
x <- dual( matrix(c(1,0,2,3,2,4), 2, 3) )
dim(x)
length(x)
rownames(x) <- c("L1", "L2")
x
d(x, "x1.1")

# modifying dim is the recommended way to change dual object shape
dim(x) <- NULL
x

# back to matrix shape
dim(x) <- c(2, 3)
x
```

Summary *Summary methods for objects of class dual*

Description

Methods extending to dual objects the corresponding methods for numeric objects.

Usage

```
## S3 method for class 'dual'
sum(x, ..., na.rm = FALSE)

## S4 method for signature 'numericOrArray'
sum(x, ..., na.rm = FALSE)

## S3 method for class 'dual'
prod(x, ..., na.rm = FALSE)

## S4 method for signature 'numericOrArray'
prod(x, ..., na.rm = FALSE)

## S3 method for class 'dual'
max(x, ..., na.rm = TRUE)

## S4 method for signature 'numericOrArray'
max(x, ..., na.rm = TRUE)

## S3 method for class 'dual'
min(x, ..., na.rm = TRUE)

## S4 method for signature 'numericOrArray'
min(x, ..., na.rm = TRUE)

## S3 method for class 'dual'
range(x, ..., na.rm = TRUE)

## S4 method for signature 'numericOrArray'
range(x, ..., na.rm = TRUE)

## S4 method for signature 'dual'
which.min(x)

## S4 method for signature 'dual'
which.max(x)
```

Arguments

x	a dual object
...	extra arguments
na.rm	if 'TRUE', NA values are removed

Details

For 'max' and 'min', the derivative is equal to the derivative of maximum element as identified by 'which.max' and 'which.min'. This is unfortunately problematic in presence of ties. If this is an issue, you may redefine this function (at the expense of speed).

Value

'which.min' and 'which.max' return an integer, the other methods return a dual object.

Examples

```
x <- dual( c(1,2,4) )
sum(x)
d(sum(x), "x1")
```

---

t *Transposition of matrices and arrays*

Description

Transposition of matrices and arrays

Usage

```
## S3 method for class 'dual'
t(x)

## S3 method for class 'dual'
aperm(a, perm = NULL, resize = TRUE, ...)
```

Arguments

x, a	a dual matrix or array
perm	subscript permutation vector
resize	if 'TRUE' (default) the array is reshaped
...	extra arguments (ignored)

Value

A dual matrix or array.

Examples

```
x <- dual( matrix(c(1,2,0,3), 2, 2) )
t(x)

# creation of an array using dim<-
y <- dual( c(1,-1) ) + 1:12
dim(y) <- c(2,3,2)
z <- aperm(y, c(2,3,1))
z
d(z, "x1")
```

Index

∗ **dual**
matrix, 24
*,dual,dual-method *(*Arithmetic*)*, 3
*,dual,numeric-method *(*Arithmetic*)*, 3
*,numeric,dual-method *(*Arithmetic*)*, 3
+,dual,dual-method *(*Arithmetic*)*, 3
+,dual,missing-method *(*Arithmetic*)*, 3
+,dual,numericOrArray-method *(*Arithmetic*)*, 3
+,numericOrArray,dual-method *(*Arithmetic*)*, 3
-,dual,dual-method *(*Arithmetic*)*, 3
-,dual,missing-method *(*Arithmetic*)*, 3
-,dual,numericOrArray-method *(*Arithmetic*)*, 3
-,numericOrArray,dual-method *(*Arithmetic*)*, 3
/,dual,dual-method *(*Arithmetic*)*, 3
/,dual,numeric-method *(*Arithmetic*)*, 3
/,numeric,dual-method *(*Arithmetic*)*, 3
[,dual,index,index-method *(*Extract*)*, 14
[,dual,index,missing-method *(*Extract*)*, 14
[,dual,missing,index-method *(*Extract*)*, 14
[,dual,missing,missing-method *(*Extract*)*, 14
[<-,dual,index,index,dual-method *(*Extract*)*, 14
[<-,dual,index,index,logicalOrNumericOrArray-method *(*Extract*)*, 14
[<-,dual,index,missing,dual-method *(*Extract*)*, 14
[<-,dual,index,missing,logicalOrNumericOrArray-method *(*Extract*)*, 14
[<-,dual,missing,index,dual-method *(*Extract*)*, 14
[<-,dual,missing,index,logicalOrNumericOrArray-method *(*Extract*)*, 14
[<-,dual,missing,missing,dual-method *(*Extract*)*, 14
[<-,dual,missing,missing,logicalOrNumericOrArray-method *(*Extract*)*, 14
%*%,dual,dual-method *(*matmult*)*, 22
%*%,dual,numericOrArray-method *(*matmult*)*, 22
%*%,numericOrArray,dual-method *(*matmult*)*, 22
%o%,dual,dual-method *(*outer*)*, 26
%o%,dual,numericOrArray-method *(*outer*)*, 26
%o%,numericOrArray,dual-method *(*outer*)*, 26
^,dual,dual-method *(*Arithmetic*)*, 3
^,dual,numeric-method *(*Arithmetic*)*, 3
^,numeric,dual-method *(*Arithmetic*)*, 3

abs.dual *(*MathFun*)*, 19
acos.dual *(*MathFun*)*, 19
acosh.dual *(*MathFun*)*, 19
aperm *(*t*)*, 32
apply, *3*
apply,dual-method, 2
Arithmetic, 3
array,dual-method *(*matrix*)*, 24
as *(*matrix*)*, 24
asin.dual *(*MathFun*)*, 19
asinh.dual *(*MathFun*)*, 19
atan.dual *(*MathFun*)*, 19
atan2,dual,dual-method *(*MathFun*)*, 19
atan2,dual,numericOrArray-method *(*MathFun*)*, 19
atan2,numericOrArray,dual-method *(*MathFun*)*, 19
atanh.dual *(*MathFun*)*, 19

beta,dual,dual-method *(*MathFun*)*, 19
beta,dual,numericOrArray-method *(*MathFun*)*, 19

beta,numericOrArray,dual-method (MathFun), 19
bind, 5

c, 6
c,numericOrArray-method (c), 6
c-dual (c), 6
cbind2,dual,dual-method (bind), 5
cbind2,dual,missing-method (bind), 5
cbind2,dual,numericOrArray-method (bind), 5
cbind2,numericOrArray,dual-method (bind), 5
ceiling.dual (MathFun), 19
choose,dual,numeric-method (MathFun), 19
colMeans (colSums), 7
colMeans,dual-method (colSums), 7
colMeans.dual (colSums), 7
colSums, 7
colSums,dual-method (colSums), 7
colSums.dual (colSums), 7
Compare,dual,ANY-method (Comparison), 8
Comparison, 8
concat0 (c), 6
cos.dual (MathFun), 19
cosh.dual (MathFun), 19
cospi.dual (MathFun), 19
crossprod,dual,dual-method (matmult), 22
crossprod,dual,missing-method (matmult), 22
crossprod,dual,numericOrArray-method (matmult), 22
crossprod,numericOrArray,dual-method (matmult), 22

d, 8, *13*
det (inversion), 18
det,dual-method (inversion), 18
det.dual (inversion), 18
determinant (inversion), 18
diag, 9
diag,dual-method (diag), 9
diag.dual (diag), 9
diag<-,dual,dual-method (diag), 9
diag<-,dual,numericOrArray-method (diag), 9
digamma.dual (MathFun), 19
dim.dual (shape), 29
dim<-.dual (shape), 29
dimnames.dual (shape), 29
dimnames<-.dual (shape), 29
dnorm, 10
dual, 11, *13*
dual-class, 12, *25*
dualFun1, 13

exp.dual (MathFun), 19
expm1.dual (MathFun), 19
Extract, 14

factorial.dual (MathFun), 19
floor.dual (MathFun), 19

gamma.dual (MathFun), 19
gradient.descent, 16

ifelse, 17
ifelse,ANY,dual,numericOrArrayOrDual-method (ifelse), 17
ifelse,ANY,numericOrArray,dual-method (ifelse), 17
inversion, 18

lbeta,dual,dual-method (MathFun), 19
lbeta,dual,numericOrArray-method (MathFun), 19
lbeta,numericOrArray,dual-method (MathFun), 19
lchoose,dual,numeric-method (MathFun), 19
length.dual (shape), 29
lfactorial.dual (MathFun), 19
lgamma.dual (MathFun), 19
log.dual (MathFun), 19
log10.dual (MathFun), 19
log1p.dual (MathFun), 19
log2.dual (MathFun), 19
logNeper (MathFun), 19

MathFun, 19
matmult, 22
matrix, 24
matrix,dual-method (matrix), 24
matrixprod_dd (matmult), 22
matrixprod_dn (matmult), 22
matrixprod_nd (matmult), 22
max (Summary), 30
max,numericOrArray-method (Summary), 30
max.dual (Summary), 30

min (Summary), 30
min,numericOrArray-method (Summary), 30
min.dual (Summary), 30

names.dual (shape), 29
names<-.dual (shape), 29

optim, *26*
optiWrap, 25
outer, 26
outer,dual,dual-method (outer), 26
outer,dual,numericOrArray-method (outer), 26
outer,numericOrArray,dual-method (outer), 26
outer.dual (outer), 26

prod (Summary), 30
prod,numericOrArray-method (Summary), 30
prod.dual (Summary), 30
psigamma,dual-method (MathFun), 19
psigamma.dual (MathFun), 19

range (Summary), 30
range,numericOrArray-method (Summary), 30
range.dual (Summary), 30
rbind2,dual,dual-method (bind), 5
rbind2,dual,missing-method (bind), 5
rbind2,dual,numericOrArray-method (bind), 5
rbind2,numericOrArray,dual-method (bind), 5
rbind2_dd (bind), 5
rep, 28
rowMeans (colSums), 7
rowMeans,dual-method (colSums), 7
rowMeans.dual (colSums), 7
rowSums (colSums), 7
rowSums,dual-method (colSums), 7
rowSums.dual (colSums), 7

salad, *13*, *25*, 28
shape, *25*, 29
sign.dual (MathFun), 19
sin.dual (MathFun), 19
sinh.dual (MathFun), 19
sinpi.dual (MathFun), 19
solve (inversion), 18
solve,dual,dual-method (inversion), 18
solve,dual,missing-method (inversion), 18
solve,dual,numericOrArray-method (inversion), 18
solve,numericOrArray,dual-method (inversion), 18
sqrt.dual (MathFun), 19
sum (Summary), 30
sum,numericOrArray-method (Summary), 30
sum.dual (Summary), 30
Summary, 30

t, 32
tan.dual (MathFun), 19
tanh.dual (MathFun), 19
tanpi.dual (MathFun), 19
tcrossprod,dual,dual-method (matmult), 22
tcrossprod,dual,missing-method (matmult), 22
tcrossprod,dual,numericOrArray-method (matmult), 22
tcrossprod,numericOrArray,dual-method (matmult), 22
trigamma.dual (MathFun), 19
trunc.dual (MathFun), 19

value, *13*
value (d), 8
varnames (d), 8

which.max (Summary), 30
which.max,dual-method (Summary), 30
which.min (Summary), 30
which.min,dual-method (Summary), 30