Package: dfdr (via r-universe)

November 23, 2026

Type Package

Title Automatic Differentiation of Simple Functions

Version 0.2.0

Description Implementation of automatically computing derivatives of functions (see Mailund Thomas (2017) <doi:10.1007/978-1-4842-2881-4>). Moreover, calculating gradients and Hessian matrices is possible.

License GPL-3

Encoding UTF-8

Imports methods, purrr, rlang, R6

Suggests tinytest

Repository https://konrad1991.r-universe.dev

Date/Publication 2026-11-23 15:11:17 UTC

RemoteUrl https://github.com/Konrad1991/dfdr

RemoteRef HEAD

RemoteSha 3866460708c7c57b1ca534e59d7b0c44ed4cdc0a

Contents

d 2
fcts 3
fcts_add_fct 4
gradient 5
hessian 5
simplify 6

Index **7**

d *Differentiate a function for a single variable.*

Description

Differentiate a function for a single variable.

Usage

```
d(f, x, derivs = NULL)
```

Arguments

f	The function to differentiate.
x	The variable that f should be differentiated with respect to.
derivs	An S4 class of type *fcts* that defines additional derivatives. See fcts for details.

Details

The following functions are already supported:
sin, sinh, asin, cos, cosh, acos, tan, tanh, atan, exp, log, sqrt, c, vector, numeric, rep and matrix.
Notably, for the functions: c, vector, numeric, rep and matrix the function is ignored during differentiation.

Value

For example function f and symbol x:
df/dx

Examples

```
library(dfdr)
d(sin, x)

f <- function(x) -sin(x)
d(f, x)

# Initialize list
lst <- dfdr::fcts()
# The function which should be added
f <- function(x) x^2
# The dervative function of f
f_deriv <- function(x) 2*x
# add new entry to list
lst <- fcts_add_fct(lst, f, f_deriv)
g <- function(z) f(z)
d(g, z, lst)
```

`fcts` *S4 class* fcts

Description

A S4 class containing additional functions which can be used for calculating derivatives with `d()`.
To create a class the function *fcts()* should be used.
Adding functions is only possible *via* the function *add_fct*.

Details

The following functions are already supported:
sin, sinh, asin, cos, cosh, acos, tan, tanh, atan, exp, log, sqrt, c, vector, numeric, rep and matrix.
Notably, for the functions: c, vector, numeric, rep and matrix the function is ignored during differentiation.

Slots

`funs` A list containing the specified functions. This slot should not be accessed and is used only internally.

See Also

`d()`

Examples

```
library(dfdr)
# Initialize list
lst <- dfdr::fcts()

# The function which should be added
f <- function(x) x^2
# The dervative function of f
f_deriv <- function(x) 2*x

# add new entry to list
lst <- fcts_add_fct(lst, f, f_deriv)

g <- function(z) f(z)
df <- d(g, z, lst)
df
```

`fcts_add_fct` *appending a S4 class of type* fcts

Description

A function which appends a S4 class of type *fcts* with a new function-derivative pair.

Usage

```
fcts_add_fct(lst, f, f_deriv, keep = FALSE)
```

Arguments

`lst`	is the S4 class of type *fcts*. Newly created by `fcts()`
`f`	is the function which should be differentiated. The argument has to be of type function.
`f_deriv`	is a function defining the derivative of *f*. The argument has to be of type function.
`keep`	is a logical value. If set to TRUE the function *f* is ignored of `d()`. The default value is FALSE.

Details

The following functions are already supported:
sin, sinh, asin, cos, cosh, acos, tan, tanh, atan, exp, log, sqrt, c, vector, numeric, rep and matrix.
Notably, for the functions: c, vector, numeric, rep and matrix the function is ignored during differentiation.

Value

a S4 class of type *fcts* extended by the new function-derivative pair.

Note

The body of *f* and *f_deriv* have to be defined without curly brackets.

Examples

```
library(dfdr)
# Initialize list
lst <- dfdr::fcts()

# The function which should be added
f <- function(x) x^2
# The dervative function of f
f_deriv <- function(x) 2*x

# add new entry to list
lst <- fcts_add_fct(lst, f, f_deriv)
```

```
g <- function(z) f(z)
df <- d(g, z, lst)
df
```

`gradient` *Compute the gradient-function of a function.*

Description

Creates a function that computes the derivative of a function with respect to each parameter and return a vector of these.

Usage

```
gradient(f, use_names, ...)
```

Arguments

`f`	A function
`use_names`	Should the gradient add variable names to the output of the function?
`...`	The variable names for which gradients should be calculated

Value

A function that computes the gradient of f at any point.

Examples

```
f <- function(x, y) x^2 + y^2
df <- gradient(f, FALSE, x, y)
df(1, 1)
```

`hessian` *Compute the Hessian-function of a function.*

Description

Creates a function that computes the second-order derivatives of a function with respect to each pair of parameters and return a vector of these.

Usage

```
hessian(f, use_names = FALSE, ...)
```

Arguments

f	A function
use_names	Should the gradient add variable names to the output of the function?
...	The variable names for which gradients should be calculated

Value

A function that computes the gradient of f at any point.

Examples

```
f <- function(x, y) x**2 + y**2
h <- hessian(f, FALSE, x, y)
h(0, 0)
```

---

simplify *Simplify an expression by computing the values for constant expressions*

Description

Simplify an expression by computing the values for constant expressions

Usage

```
simplify(expr)
```

Arguments

expr	An expression

Value

a simplified expression

Examples

```
ex <- quote(a*0 + b^2 + 0)
simplify(ex)
```

Index

d, 2, *3*, *4*

fcts, *2*, 3, *4*
fcts_add_fct, 4

gradient, 5

hessian, 5

simplify, 6