Package 'mosaicCalc'

May 7, 2020

Type Package

Title Function-Based Numerical and Symbolic Differentiation and Antidifferentiation

Description Part of the Project MOSAIC (<http://mosaic-web.org/>) suite that provides utility functions for doing calculus (differentiation and integration) in R. The main differentiation and antidifferentiation operators are described using formulas and return functions rather than numerical values. Numerical values can be obtained by evaluating these functions.

Version 0.5.1

Depends R (>= 3.0.0), mosaicCore

Imports methods, stats, MASS, mosaic, ggformula, magrittr, rlang

Suggests testthat, knitr, rmarkdown, mosaicData

Author Daniel T. Kaplan <kaplan@macalester.edu>, Randall Pruim <rpruim@calvin.edu>, Nicholas J. Horton <nhorton@amherst.edu>

Maintainer Daniel Kaplan <kaplan@macalester.edu>

VignetteBuilder knitr

License GPL (>= 2)

LazyLoad yes

LazyData yes

URL https://github.com/ProjectMOSAIC/mosaicCalc

BugReports https://github.com/ProjectMOSAIC/mosaicCalc/issues

RoxygenNote 7.0.2

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2020-05-07 13:00:13 UTC

R topics documented:

connector . . . 2
D . . . 2
findZeros . . . 4
fitSpline . . . 5
integrateODE . . . 5
numD . . . 6
plotFun . . . 7
rfun . . . 7
smoother . . . 7
spliner . . . 7

Index **8**

`connector` *Create an interpolating function going through a set of points*

Description

This is defined in the mosaic package: See `connector`.

`D` *Derivative and Anti-derivative operators*

Description

Operators for computing derivatives and anti-derivatives as functions.

Usage

```
D(formula, ..., .hstep = NULL, add.h.control = FALSE)

antiD(formula, ..., lower.bound = 0, force.numeric = FALSE)

makeAntiDfun(.function, .wrt, from, .tol = .Machine$double.eps^0.25)

numerical_integration(f, wrt, av, args, vi.from, ciName = "C", .tol)
```

Arguments

formula	A formula. The right side of a formula specifies the variable(s) with which to carry out the integration or differentiation. On the left side should be an expression or a function that returns a numerical vector of the same length as its argument. The expression can contain unbound variables. Functions will be differentiated as if the formula f(x) ~ x were specified but with x replaced by the first argument of f.
...	Default values to be given to unbound variables in the expression expr. See examples.#' Note that in creating anti-derivative functions, default values of "from" and "to" can be assigned. They are to be written with the name of the variable as a prefix, e.g. y.from.
.hstep	horizontal distance between points used for secant slope calculation in numerical derivatives.
add.h.control	logical indicating whether the returned derivative function should have an additional parameter for setting .hstep. Meaningful only for numerical derivatives.
lower.bound	for numerical integration only, the lower bound used
force.numeric	If TRUE, a numerical integral is performed even when a symbolic integral is available.
.function	function to be integrated
.wrt	character string naming the variable of integration
from	default value for the lower bound of the integral region
.tol	Numerical tolerance. See integrate().
f	A function.
wrt	Character string naming a variable: the var. of integration.
av	A list of the arguments passed to the function calling this.
args	Default values (if any) for parameters.
vi.from	The the lower bound of the interval of integration.
ciName	Character string giving the name of the symbol for the constant of integration.

Details

D attempts to find a symbolic derivative for simple expressions, but will provide a function that is a numerical derivative if the attempt at symbolic differentiation is unsuccessful. The symbolic derivative can be of any order (although the expression may become unmanageably complex). The numerical derivative is limited to first or second-order partial derivatives (including mixed partials). antiD will attempt simple symbolic integration but if it fails it will return a numerically-based anti-derivative.

antiD returns a function with the same arguments as the expression passed to it. The returned function is the anti-derivative of the expression, e.g., antiD(f(x)~x) -> F(x). To calculate the integral of f(x), use F(to) - F(from).

Value

For derivatives, the return value is a function of the variable(s) of differentiation, as well as any other symbols used in the expression. Thus, `D(A*x^2 + B*y ~ x + y)` will compute the mixed partial with respect to x then y (that is, $\frac{d^2 f}{dy\ dx}$). The returned value will be a function of x and y, as well as A and B. In evaluating the returned function, it's best to use the named form of arguments, to ensure the order is correct.

a function of the same arguments as the original expression with a constant of integration set to zero by default, named "C", "D", ... depending on the first such letter not otherwise in the argument list.

Note

`numerical_integration` is not intended for direct use. It packages up the numerical anti-differentiation process so that the contents of functions produced by `antiD` look nicer to human readers.

Examples

```
D(sin(t) ~ t)
D(A*sin(t) ~ t )
D(A*sin(2*pi*t/P) ~ t, A=2, P=10) # default values for parameters.
f <- D(A*x^3 ~ x + x, A=1) # 2nd order partial -- note, it's a function of x
f(x=2)
f(x=2,A=10) # override default value of parameter A
g <- D(f(x=t, A=1)^2 ~ t)  # note: it's a function of t
g(t=1)
gg <- D(f(x=t, A=B)^2 ~ t, B=10)  # note: it's a function of t and B
gg(t=1)
gg(t=1, B=100)
f <- makeFun(x^2~x)
D(f(cos(z))~z) #will look in user functions also
antiD( a*x^2 ~ x, a = 3)
antiD( A/x~x ) # This gives a warning about no default value for A
F <- antiD( A*exp(-k*t^2 ) ~ t, A=1, k=0.1)
F(t=Inf)
one = makeFun(1 ~ x + y)
by.x = antiD(one(x=x, y=y) ~ x, y=1)
by.xy = antiD(by.x(x = sqrt(1-y^2), y = y) ~ y)
4 * by.xy(y = 1) # area of quarter circle
```

`findZeros` *Find zeros of a function*

Description

This is defined in the mosaic package: See `findZeros` for finding zeros of a function of one variable and `findZerosMult` for finding zeros of functions of two or more variables.

This is defined in the mosaic package: See `findZeros`

fitSpline *Find zeros of a function*

Description

This is defined in the mosaic package: See fitSpline

integrateODE *Integrate ordinary differential equations*

Description

A formula interface to integration of an ODE with respect to "t"

Usage

```
integrateODE(dyn, ..., tdur)
```

Arguments

dyn	a formula specifying the dynamics, e.g. dx ~ -a*x for $dx/dt = -ax$.
tdur	the duration of integration. Or, a list of the form list(from=5,to=10,dt=.001)
...	arguments giving additional formulas for dynamics in other variables, assignments of parameters, and assignments of initial conditions

Details

The equations must be in first-order form. Each dynamical equation uses a formula interface with the variable name given on the left-hand side of the formula, preceded by a d, so use dx~-k*x for exponential decay. All parameters (such as k) must be assigned numerical values in the argument list. All dynamical variables must be assigned initial conditions in the argument list. The returned value will be a list with one component named after each dynamical variable. The component will be a spline-generated function of t.

Value

a list with splined function of time for each dynamical variable

Examples

```
soln = integrateODE(dx~r*x*(1-x/k), k=10, r=.5, tdur=20, x=1)
soln$x(10)
soln$x(30) # outside the time interval for integration
# plotFun(soln$x(t)~t, tlim=range(0,20))
soln2 = integrateODE(dx~y, dy~-x, x=1, y=0, tdur=10)
# plotFun(soln2$y(t)~t, tlim=range(0,10))
# SIR epidemic
epi = integrateODE(dS~ -a*S*I, dI ~ a*S*I - b*I, a=0.0026, b=.5, S=762, I=1, tdur=20)
```

numD *Numerical Derivatives*

Description

Constructs the numerical derivatives of mathematical expressions

Usage

```
numD(formula, ..., .hstep = NULL, add.h.control = FALSE)
```

Arguments

formula	a mathematical expression (see examples and plotFun)
...	additional parameters, typically default values for mathematical parameters
.hstep	numerical finite-difference step (default is 1e-6 or 1e-4 for first and second-order derivatives, respectively)
add.h.control	arranges the returned function to have a .hstep argument that can be used to demonstrate convergence and error

Details

Uses a simple finite-difference scheme to evaluate the derivative. The function created will not contain a formula for the derivative. Instead, the original function is stored at the time the derivative is constructed and that original function is re-evaluated at the finitely-spaced points of an interval. If you redefine the original function, that won't affect any derivatives that were already defined from it. Numerical derivatives, particularly high-order ones, are unstable. The finite-difference parameter .hstep is set, by default, to give reasonable results for first- and second-order derivatives. It's tweaked a bit so that taking a second derivative by differentiating a first derivative will give reasonably accurate results. But, if taking a second derivative, much better to do it in one step to preserve numerical accuracy.

Value

a function implementing the derivative as a finite-difference approximation

Note

WARNING: In the expressions, do not use variable names beginning with a dot, particularly .f or .h

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

Examples

```
g = numD( a*x^2 + x*y ~ x, a=1)
g(x=2,y=10)
gg = numD( a*x^2 + x*y ~ x&x, a=1)
gg(x=2,y=10)
ggg = numD( a*x^2 + x*y ~ x&y, a=1)
ggg(x=2,y=10)
h = numD( g(x=x,y=y,a=a) ~ y, a=1)
h(x=2,y=10)
f = numD( sin(x)~x, add.h.control=TRUE)
# plotFun( f(3,.hstep=h)~h, hlim=range(.00000001,.000001))
# ladd( panel.abline(cos(3),0))
```

`plotFun` *Plot functions of one and two variables using lattice system*

Description

This is defined in the mosaic package: See `plotFun`

`rfun` *Generate a "natural looking" function of one or multiple variables*

Description

This is defined in the mosaic package: See `rfun`.

`smoother` *Create a smoothing function approximating a cloud of points*

Description

This is defined in the mosaic package: See `smoother`.

`spliner` *Construct a spline function going through a set of points*

Description

This is defined in the mosaic package: See `spliner`.

Index

antiD *(D)*, 2

connector, *2*, 2

D, 2

findZeros, *4*, 4
findZerosMult, *4*
fitSpline, *5*, 5

integrate, *3*
integrateODE, 5

makeAntiDfun *(D)*, 2

numD, 6
numerical_integration *(D)*, 2

plotFun, *6*, *7*, 7

rfun, *7*, 7

smoother, *7*, 7
spliner, *7*, 7