Package 'mosaicCalc'

July 27, 2017

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 0.5.0

Date 2017-07-26

Depends R (>= 3.0.0), mosaicCore

Imports methods, stats, MASS, mosaic

Suggests testthat, knitr, rmarkdown

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

Maintainer Randall Pruim <rpruim@calvin.edu>

License GPL (>= 2)

LazyLoad yes

LazyData yes

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

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

RoxygenNote 6.0.1.9000

NeedsCompilation no

Repository CRAN

Date/Publication 2017-07-27 11:22:02 UTC

R topics documented:

`.polyExp` *Takes a call and returns its polynomial coefficients*

Description

Takes a call and returns its polynomial coefficients

Takes a call and returns its polynomial coefficients as numerics.

Method for putting a polynomial together given the coefficients and power from .polyExp()

Usage

```
.polyExp(tree, .x., params, iterate = 1)

.polyExp.num(tree, .x.)

.makePoly(form, poly)
```

Arguments

`tree`	A call that will be parsed and simplified recursively
`.x.`	the variable name with respect to which the polynomial should be most simplified
`params`	All names of free variables. If there are no free variables, the value should be ""
`iterate`	The number of times the call is nested. Default and proper value when called from the outside is 1
`form`	original formula - provides information on which variable the polynomial was reduced with respect to.
`poly`	output of .polyExp()

Details

Will work on any call as long as it can be reduced to a polynomial with respect the the variable and each of the parameters. Operates recursively, reducing each of the coefficients with respect to the extra parameters in turn. Calls .polyExp.num when all remaining coefficients are numeric to reduce the expression more fully.

works with the same structure as .polyExp() but will return only if all coefficients reduce to numeric values.

Value

A list containing a list, `coeffs`, of coefficients ordered high to low (i.e. the list (2,3,4) would correspond to the polynomial 2*x^2+3*x+4) and value, `pow`, indicating the order of the polynomial. If the expression is not a polynomial, this method returns an empty list or an error.

A list containing a list, `coeffs`, of coefficients ordered high to low (i.e. the list (2,3,4) would correspond to the polynomial 2*x^2+3*x+4) and value, `pow`, indicating the order of the polynomial. If the expression is not a polynomial, this method returns an empty list or an error.

A formula whose left hand side is a polynomial that fits the description given with the input poly.

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)

## Default S3 method:
D(formula, ..., .hstep = NULL, add.h.control = FALSE)

## S3 method for class 'formula'
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
```

---

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)

setInterval(C, wrt, h)

setCorners(C, var1, var2, h)

dfdx(.function, .wrt, .hstep)
```

```
d2fdxdy(.function, .var1, .var2, .hstep)

d2fdx2(.function, .wrt, .hstep)

numerical.first.partial(f, wrt, h, av)

numerical.second.partial(f, wrt, h, av)

numerical.mixed.partial(f, var1, var2, h, av)
```

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
C	list of arguments for evaluating the function at the "center" point
wrt	character string naming the variable with respect to which differentiation is to be done
h	the finite-difference step size
var1	character string naming the first variable with respect to which differentiation is to be done
var2	character string naming the second variable with respect to which differentiation is to be done
.function	function to be differentiated
.wrt	character string naming the variable with respect to which differentiation is to be done
.var1	character string naming the first variable with respect to which differentiation is to be done
.var2	character string naming the second variable with respect to which differentiation is to be done
f	function to differentiate
av	arguments to the function calling this
.step	the finite-difference step size

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

Numerical partials

These functions are not intended for direct use. They just package up the numerical differentiation process to make functions returned by numD and D easier to read.

Note

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

Helper function for numD for unmixed partials

Helper function for numD for mixed partials

Helper function for numD for first-order derivs.

Helper function for numD for second-order mixed partials

Helper function for numD for second-order derivs

Not for direct use. This just packages up the numerical differentiation process to make functions returned by numD and D easier to read.

Not for direct use. This just packages up the numerical differentiation process to make functions returned by numD and D easier to read.

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

See Also

D, symbolicD, makeFun, antiD, plotFun

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))
```

rkintegrate *A simple Runge-Kutta integrator*

Description

Integrates ordinary differential equations using a Runge-Kutta method

Usage

```
rkintegrate(fun, x0, tstart = 0, tend = 1, dt = NULL)
```

Arguments

fun	the dynamical function with arguments state (a vector) and t.
x0	the initial condition, a vector with one element for each state variable
tstart	starting time
tend	ending time for integration
dt	step size for integration

Details

This is mainly for internal use by integrateODE.

Value

a list containing x, a matrix of the state with one row for each time step and a vector t containing the times of those steps.

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

symbolicD *Symbolic Derivatives*

Description

Constructs symbolic derivatives of some mathematical expressions

Usage

```
symbolicD(formula, ..., .order = NULL)
```

Arguments

formula	a mathematical expression (see examples and plotFun)
...	additional parameters, typically default values for mathematical parameters
.order	a number specifying the order of a derivative with respect to a single variable

Details

Uses the built-in symbolic differentiation function to construct a formula for the derivative and packages this up as a function. The .order argument is just for convenience when programming high-order derivatives, e.g. the 5th derivative w.r.t. one variable.

Value

a function implementing the derivative

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

See Also

D, numD, makeFun, antiD, plotFun

Examples

```
symbolicD( a*x^2 ~ x)
symbolicD( a*x^2 ~ x&x)
symbolicD( a*sin(x)~x, .order=4)
symbolicD( a*x^2*y+b*y ~ x, a=10, b=100 )
```

---

symbolicInt *Find the symbolic integral of a formula*

Description

Find the symbolic integral of a formula

Use recursion to find a symbolic antiderivative

Attempts symbolic integration of some mathematical/arithmetical forms

Attempts symbolic integration of some mathematical forms

Attempts symbolic integration of some mathematical forms using trigonometric substitution

Takes a call and returns its affine coefficients.

Usage

```
symbolicInt(form, ...)

symbolicAntiD(form, ...)

.intArith(form, ...)

.intMath(form, ...)

.intTrig(form, num, den, .x.)

.affine.exp(tree, .x.)
```

Arguments

form	an object of type formula to be integrated. Rhs of formula indicates which variable to integrate with respect to. Must only have one variable.
num	numerator
den	denominator
.x.	the variable name
tree	the expression to be analyzed
...	extra parameters

Details

This symbolic integrator recognizes simple polynomials and functions such as sin, cos, tan, sinh, cosh, tanh, sqrt, and exp.

It will not perform more complicated substitutions or integration by parts.

Value

symbolicInt returns a function whose body is the symbolic antiderivative of the formula. If this method does not recognize the formula, it will return an error.

a formula implementing giving symbolic anti-derivative. If the formula isn't found by the algorithm, an error is thrown.

An expression with the integral, or throws an error if unsuccessful.

An expression with the integral, or throws an error if unsuccessful.

An expression with the integral, or throws an error if unsuccessful.

A list with values of a and b satisfying a*.x.+b = tree. If the expression is not affine, returns an empty list.

Index