

Package ‘caracas’

October 12, 2022

Version 1.1.2

Title Computer Algebra

Maintainer Mikkel Meyer Andersen <mikl@math.aau.dk>

Encoding UTF-8

Description Computer algebra via the 'SymPy' library (<<https://www.sympy.org/>>).
This makes it possible to solve equations symbolically,
find symbolic integrals, symbolic sums and other important quantities.

Depends R (>= 3.0), methods

Imports reticulate (>= 1.14), magrittr

Suggests Matrix, testthat (>= 2.1.0), knitr, rmarkdown

License GPL

SystemRequirements Python (>= 3.6.0)

URL <https://github.com/r-cas/caracas>

BugReports <https://github.com/r-cas/caracas/issues>

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation no

Author Mikkel Meyer Andersen [aut, cre, cph],
Søren Højsgaard [aut, cph]

Repository CRAN

Date/Publication 2022-02-11 07:50:02 UTC

R topics documented:

as.character.caracas_symbol	3
ask	3
as_character_matrix	4
as_diag	4
as_expr	5

as_sym	5
def_sym	6
der	7
der2	8
diag	9
diag-set	9
diag.caracas_symbol	10
diag<-.caracas_symbol	10
diag_	11
dim.caracas_symbol	11
doit	12
do_la	12
drop_remainder	13
eval_to_symbol	14
expand	14
expand_func	15
expand_log	15
expand_trig	16
fraction_parts	16
get_py	17
get_sympy	17
has_sympy	18
install_sympy	18
int	19
lim	19
linalg	20
listify	22
Math.caracas_symbol	22
matrify	23
matrix-products	23
matrix_	24
mat_pow	24
N	25
Ops.caracas_symbol	26
print.caracas_solve_sys_sol	26
print.caracas_symbol	27
prod_	27
reciprocal_matrix	28
simplify	28
solve_lin	29
solve_sys	29
subs	30
subs_lst	31
subs_vec	31
sum.caracas_symbol	32
sum_	33
symbol	33
sympy_func	34

sympy_version	35
t.caracas_symbol	36
taylor	36
tex	37
tuplify	37
unbracket	38
vec	38
[.caracas_symbol	39
[<-.caracas_symbol	40
%>%	40

Index **41**

as.character.caracas_symbol
Convert symbol to character

Description

Convert symbol to character

Usage

```
## S3 method for class 'caracas_symbol'
as.character(x, replace_I = TRUE, ...)
```

Arguments

x	A caracas_symbol
replace_I	Replace constant I (can both be identity and imaginary unit)
...	not used

ask *Ask for a symbol's property*

Description

Ask for a symbol's property

Usage

```
ask(x, property)
```

Arguments

x	symbol
property	property, e.g. 'positive'

Examples

```
if (has_sympy()) {  
  x <- symbol("x", positive = TRUE)  
  ask(x, "positive")  
}
```

as_character_matrix *Get matrix as character matrix*

Description

Get matrix as character matrix

Usage

```
as_character_matrix(x)
```

Arguments

x caracas symbol

Examples

```
if (has_sympy()) {  
  s <- as_sym("[[r1, r2, r3], [u1, u2, u3]]")  
  s2 <- apply(as_character_matrix(s), 2, function(x) (paste("1/(", x, ")")))  
  as_sym(s2)  
}
```

as_diag *Construct diagonal matrix from vector*

Description

Construct diagonal matrix from vector

Usage

```
as_diag(x)
```

Arguments

x Matrix with 1 row or 1 column that is the diagonal in a new diagonal matrix

Examples

```

if (has_sympy()) {
  d <- as_sym(c("a", "b", "c"))
  D <- as_diag(d)
  D
}

```

`as_expr`*Convert caracas object to R*

Description

Potentially calls `doit()`.

Usage

```
as_expr(x, first_doit = TRUE)
```

Arguments

<code>x</code>	caracas_symbol
<code>first_doit</code>	Try <code>doit()</code> first

`as_sym`*Convert object to symbol*

Description

Variables are detected as a character followed by a number of either: character, number or underscore.

Usage

```
as_sym(x, declare_symbols = TRUE)
```

Arguments

<code>x</code>	R object to convert to a symbol
<code>declare_symbols</code>	declare detected symbols automatically

Details

Default is to declare used variables. Alternatively, the user must declare them first, e.g. by `symbol()`.

Note that matrices can be defined by specifying a Python matrix, see below in examples.

Examples

```

if (has_sympy()) {
  x <- symbol("x")
  A <- matrix(c("x", 0, 0, "2*x"), 2, 2)
  A
  B <- as_sym(A)
  B
  2*B
  dim(B)
  sqrt(B)
  D <- as_sym("[[1, 4, 5], [-5, 8, 9]]")
  D
}

```

def_sym

Define caracas symbols in global environment

Description

Define caracas symbols in global environment

Usage

```
def_sym(..., charvec = NULL, warn = FALSE, env = parent.frame())
```

Arguments

...	Names for new symbols, also supports non-standard evaluation
charvec	Take each element in this character vector and define as caracas symbols
warn	Warn if existing variable names are overwritten
env	Environment to assign variable in

Value

Names of declared variables (invisibly)

See Also

[symbol\(\)](#), [as_sym\(\)](#)

Examples

```

if (has_sympy()) {
  ls()
  def_sym(n1, n2, n3)
  ls()
  def_sym("x1", "x2", "x3")
  ls()
  def_sym("x1", "x2", "x3", warn = TRUE)
  ls()
  def_sym(i, j, charvec = c("x", "y"))
  ls()
}

```

der

*Symbolic differentiation of an expression***Description**

Symbolic differentiation of an expression

Usage

```
der(expr, vars, simplify = TRUE)
```

Arguments

expr	A caracas_symbol
vars	variables to take derivate with respect to
simplify	Simplify result

Examples

```

if (has_sympy()) {
  x <- symbol("x")
  y <- symbol("y")
  f <- 3*x^2 + x*y^2
  der(f, x)
  g <- der(f, list(x, y))
  g
  dim(g)
  G <- matlify(g)
  G
  dim(G)

  h <- der(g, list(x, y))
  h
  dim(h)
}

```

```
as.character(h)
H <- matlify(h)
H
dim(H)

g %>%
  der(list(x, y)) %>%
  der(list(x, y)) %>%
  der(list(x, y))
}
```

der2

Symbolic differentiation of second order of an expression

Description

Symbolic differentiation of second order of an expression

Usage

```
der2(expr, vars, simplify = TRUE)
```

Arguments

expr	A caracas_symbol
vars	variables to take derivate with respect to
simplify	Simplify result

Examples

```
if (has_sympy()) {
  x <- symbol("x")
  y <- symbol("y")
  f <- 3*x^2 + x*y^2
  der2(f, x)
  h <- der2(f, list(x, y))
  h
  dim(h)
  H <- matlify(h)
  H
  dim(H)
}
```

diag	<i>Matrix diagonal</i>
------	------------------------

Description

Matrix diagonal

Usage

```
diag(x, ...)
```

Arguments

x	Object x
...	Passed on

diag-set	<i>Replace matrix diagonal</i>
----------	--------------------------------

Description

Replace matrix diagonal

Usage

```
diag(x) <- value
```

Arguments

x	Object x
value	Replacement value

diag.caracas_symbol *Matrix diagonal*

Description

Matrix diagonal

Usage

```
## S3 method for class 'caracas_symbol'
diag(x, ...)
```

Arguments

x	Object x
...	Not used

diag<-.caracas_symbol *Replace diagonal*

Description

Replace diagonal

Usage

```
## S3 replacement method for class 'caracas_symbol'
diag(x) <- value
```

Arguments

x	A caracas_symbol.
value	Replacement value

Examples

```
if (has_sympy()) {
  A <- matrix(c("a", 0, 0, 0, "a", "a", "a", 0, 0), 3, 3)
  B <- as_sym(A)
  B
  diag(B)
  diag(B) <- "b"
  B
  diag(B)
}
```

diag_	<i>Symbolic diagonal matrix</i>
-------	---------------------------------

Description

Symbolic diagonal matrix

Usage

```
diag_(x, n = 1L, declare_symbols = TRUE, ...)
```

Arguments

x	Character vector with diagonal
n	Number of times x should be repeated
declare_symbols	Passed on to <code>as_sym()</code> when constructing symbolic matrix
...	Passed on to <code>rep(x, n, ...)</code>

Examples

```
if (has_sympy()) {
  diag_(c("a", "b", "c"))
  diag_("a", 2)
}
```

dim.caracas_symbol	<i>Dimensions of a caracas symbol</i>
--------------------	---------------------------------------

Description

Dimensions of a caracas symbol

Usage

```
## S3 method for class 'caracas_symbol'
dim(x)
```

Arguments

x	caracas symbol
---	----------------

doit *Perform calculations setup previously*

Description

Perform calculations setup previously

Usage

```
doit(x)
```

Arguments

x A caracas_symbol

Examples

```
if (has_sympy()) {  
  x <- symbol('x')  
  res <- lim(sin(x)/x, "x", 0, doit = FALSE)  
  res  
  doit(res)  
}
```

do_la *Do linear algebra operation*

Description

Do linear algebra operation

Usage

```
do_la(x, slot, ...)
```

Arguments

x A matrix for which a property is requested
slot The property requested
... Auxillary arguments

Value

Returns the requested property of a matrix.

Examples

```
if (has_sympy()) {
  A <- matrix(c("a", "0", "0", "1"), 2, 2) %>% as_sym()

  do_la(A, "QR")
  QRdecomposition(A)

  do_la(A, "eigenval")
  eigenval(A)

  do_la(A, "eigenvec")
  eigenvec(A)

  do_la(A, "inv")
  inv(A)

  do_la(A, "echelon_form")
  do_la(A, "rank")

  do_la(A, "det") # Determinant
  det(A)
}
```

drop_remainder	<i>Remove remainder term</i>
----------------	------------------------------

Description

Remove remainder term

Usage

```
drop_remainder(x)
```

Arguments

x Expression to remove remainder term from

See Also

[taylor\(\)](#)

Examples

```
if (has_sympy()) {
  def_sym(x)
  f <- cos(x)
  ft_with_0 <- taylor(f, x0 = 0, n = 4+1)
```

```

ft_with_0
ft_with_0 %>% drop_remainder() %>% as_expr()
}

```

eval_to_symbol *Create a symbol from a string*

Description

Create a symbol from a string

Usage

```
eval_to_symbol(x)
```

Arguments

x String to evaluate

Value

A caracas_symbol

Examples

```

if (has_sympy()) {
  x <- symbol('x')
  (1+1)*x^2
  lim(sin(x)/x, "x", 0)
}

```

expand *Expand expression*

Description

Expand expression

Usage

```
expand(x)
```

Arguments

x A caracas_symbol

expand_func	<i>Expand a function expression</i>
-------------	-------------------------------------

Description

Expand a function expression

Usage

```
expand_func(x)
```

Arguments

x	A caracas_symbol
---	------------------

expand_log	<i>Expand a logarithmic expression</i>
------------	--

Description

Note that force as described at <https://docs.sympy.org/latest/tutorial/simplification.html#expand-log> is used meaning that some assumptions are taken.

Usage

```
expand_log(x)
```

Arguments

x	A caracas_symbol
---	------------------

Examples

```
if (has_sympy()) {  
  x <- symbol('x')  
  y <- symbol('y')  
  z <- log(x*y)  
  z  
  expand_log(z)  
}
```

expand_trig	<i>Expand a trigonometric expression</i>
-------------	--

Description

Expand a trigonometric expression

Usage

```
expand_trig(x)
```

Arguments

x	A caracas_symbol
---	------------------

fraction_parts	<i>Get numerator and denominator of a fraction</i>
----------------	--

Description

Get numerator and denominator of a fraction

Usage

```
fraction_parts(x)
```

Arguments

x	Fraction
---	----------

Examples

```
if (has_sympy()) {  
  x <- as_sym("a/b")  
  frac <- fraction_parts(x)  
  frac  
  frac$numerator  
  frac$denominator  
}
```

get_py	<i>Access 'py' object</i>
--------	---------------------------

Description

Get the 'py' object. Note that it gives you extra responsibilities when you choose to access the 'py' object directly.

Usage

```
get_py()
```

Value

The 'py' object with direct access to the library.

Examples

```
if (has_sympy()) {  
  py <- get_py()  
}
```

get_sympy	<i>Access 'SymPy' directly</i>
-----------	--------------------------------

Description

Get the 'SymPy' object. Note that it gives you extra responsibilities when you choose to access the 'SymPy' object directly.

Usage

```
get_sympy()
```

Value

The 'SymPy' object with direct access to the library.

Examples

```
if (has_sympy()) {  
  sympy <- get_sympy()  
  sympy$solve("x**2-1", "x")  
}
```

has_sympy	<i>Check if 'SymPy' is available</i>
-----------	--------------------------------------

Description

Check if 'SymPy' is available

Usage

```
has_sympy()
```

Value

TRUE if 'SymPy' is available, else FALSE

Examples

```
has_sympy()
```

install_sympy	<i>Install 'SymPy'</i>
---------------	------------------------

Description

Install the 'SymPy' Python package into a virtual environment or Conda environment.

Usage

```
install_sympy(method = "auto", conda = "auto")
```

Arguments

method	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
conda	Path to conda executable (or "auto" to find conda using the PATH and other conventional install locations).

Value

None

int *Integrate a function*

Description

If no limits are provided, the indefinite integral is calculated. Otherwise, if both limits are provided, the definite integral is calculated.

Usage

```
int(f, var, lower, upper, doit = TRUE)
```

Arguments

f	Function to integrate
var	Variable to integrate with respect to (either string or caracas_symbol)
lower	Lower limit
upper	Upper limit
doit	Evaluate the integral immediately (or later with <code>doit()</code>)

Examples

```
if (has_sympy()) {
  x <- symbol("x")

  int(1/x, x, 1, 10)
  int(1/x, x, 1, 10, doit = FALSE)
  int(1/x, x)
  int(1/x, x, doit = FALSE)
  int(exp(-x^2/2), x, -Inf, Inf)
  int(exp(-x^2/2), x, -Inf, Inf, doit = FALSE)
}
```

lim *Limit of a function*

Description

Limit of a function

Usage

```
lim(f, var, val, dir = NULL, doit = TRUE)
```

Arguments

f	Function to take limit of
var	Variable to take limit for (either string or caracas_symbol)
val	Value for var to approach
dir	Direction from where var should approach val: '+' or '-'
doit	Evaluate the limit immediately (or later with <code>doit()</code>)

Examples

```
if (has_sympy()) {  
  x <- symbol("x")  
  lim(sin(x)/x, "x", 0)  
  lim(1/x, "x", 0, dir = '+')  
  lim(1/x, "x", 0, dir = '-')  
}
```

linalg

Do linear algebra operation

Description

Performs various linear algebra operations like finding the inverse, the QR decomposition, the eigenvectors and the eigenvalues.

Usage

columnspace(x)

nullspace(x)

rowspace(x)

singular_values(x)

inv(x)

eigenval(x)

eigenvec(x)

GramSchmidt(x)

pinv(x)

rref(x)

```
QRdecomposition(x)
det(x, ...)
```

Arguments

x	A matrix for which a property is requested
...	Auxillary arguments

Value

Returns the requested property of a matrix.

See Also

[do_la\(\)](#)

Examples

```
if (has_sympy()) {
  A <- matrix(c("a", "0", "0", "1"), 2, 2) %>% as_sym()

  QRdecomposition(A)
  eigenval(A)
  eigenvec(A)
  inv(A)
  det(A)

  A <- matrix(c("a", "b", "c", "d"), 2, 2) %>% as_sym()
  evec <- eigenvec(A)
  evec
  evec1 <- evec[[1]]$eigvec
  evec1
  simplify(evec1)

  lapply(evec, function(l) simplify(l$eigvec))

  A <- as_sym("[[1, 2, 3], [4, 5, 6]]")
  pinv(A)
}
```

listify	<i>Convert object to list of elements</i>
---------	---

Description

Convert object to list of elements

Usage

```
listify(x)
```

Arguments

x	Object
---	--------

Examples

```
if (has_sympy()) {
  x <- as_sym("Matrix([[b1*x1/(b2 + x1)], [b1*x2/(b2 + x2)], [b1*x3/(b2 + x3)]])")
  listify(x)

  xT <- t(x)
  listify(xT)
}
```

Math.caracas_symbol	<i>Math functions</i>
---------------------	-----------------------

Description

If x is a matrix, the function is applied component-wise.

Usage

```
## S3 method for class 'caracas_symbol'
Math(x, ...)
```

Arguments

x	caracas_symbol.
...	further arguments passed to methods

matrify	<i>Creates matrix from array symbol</i>
---------	---

Description

Creates matrix from array symbol

Usage

```
matrify(x)
```

Arguments

x Array symbol to convert to matrix

Examples

```
if (has_sympy()) {  
  x <- symbol("x")  
  y <- symbol("y")  
  f <- 3*x^2 + x*y^2  
  h <- der2(f, list(x, y))  
  h  
  dim(h)  
  H <- matrify(h)  
  H  
  dim(H)  
}
```

matrix-products	<i>Matrix multiplication</i>
-----------------	------------------------------

Description

Matrix multiplication

Matrix multiplication

Usage

```
x %*% y
```

```
## S3 method for class 'caracas_symbol'
```

```
x %*% y
```

Arguments

x Object x
y Object y

See Also

[base::%*%\(\)](#)

[base::%*%\(\)](#)

matrix_	<i>Symbolic matrix</i>
---------	------------------------

Description

Symbolic matrix

Usage

```
matrix_(..., declare_symbols = TRUE)
```

Arguments

... Passed on to [matrix\(\)](#)
declare_symbols Passed on to [as_sym\(\)](#) when constructing symbolic matrix

Examples

```
if (has_sympy()) {
  matrix_(1:9, nrow = 3)
  matrix_("a", 2, 2)
}
```

mat_pow	<i>Matrix power</i>
---------	---------------------

Description

Matrix power

Usage

```
mat_pow(x, pow = "1")
```


Arguments

x	A caracas_symbol, a matrix.
pow	Power to raise matrix x to

Examples

```
if (has_sympy() && sympy_version() >= "1.6") {
  M <- matrix_(c("1", "a", "a", 1), 2, 2)
  M
  mat_pow(M, 1/2)
}
```

N

Numerical evaluation

Description

Numerical evaluation

Usage

```
N(x, digits = 15)
```

Arguments

x	caracas object
digits	Number of digits

Examples

```
if (has_sympy()) {
  n_2 <- as_sym("2")
  n_pi <- as_sym("pi", declare_symbols = FALSE)
  x <- sqrt(n_2) * n_pi
  x
  N(x)
  N(x, 5)
  N(x, 50)
  as.character(N(x, 50))
}
```

Ops.caracas_symbol *Math operators*

Description

Math operators

Usage

```
## S3 method for class 'caracas_symbol'
Ops(e1, e2)
```

Arguments

e1	A caracas_symbol.
e2	A caracas_symbol.

print.caracas_solve_sys_sol
Print solution

Description

Print solution

Usage

```
## S3 method for class 'caracas_solve_sys_sol'
print(
  x,
  simplify = getOption("caracas.print.sol.simplify", default = TRUE),
  ...
)
```

Arguments

x	A caracas_symbol
simplify	Print solution in a simple format
...	Passed to print.caracas_symbol()

print.caracas_symbol *Print symbol*

Description

Print symbol

Usage

```
## S3 method for class 'caracas_symbol'
print(
  x,
  caracas_prefix = TRUE,
  prettyascii = getOption("caracas.print.prettyascii", default = FALSE),
  ascii = getOption("caracas.print.ascii", default = FALSE),
  rowvec = getOption("caracas.print.rowvec", default = TRUE),
  ...
)
```

Arguments

x	A caracas_symbol
caracas_prefix	Print 'caracas' prefix
prettyascii	TRUE to print in pretty ASCII format rather than in utf8
ascii	TRUE to print in ASCII format rather than in utf8
rowvec	FALSE to print column vectors as is
...	not used

prod_ *Product of a function*

Description

Product of a function

Usage

```
prod_(f, var, lower, upper, doit = TRUE)
```

Arguments

f	Function to take product of
var	Variable to take product for (either string or caracas_symbol)
lower	Lower limit
upper	Upper limit
doit	Evaluate the product immediately (or later with <code>doit()</code>)

Examples

```

if (has_sympy()) {
  x <- symbol("x")
  p <- prod_(1/x, "x", 1, 10)
  p
  as_expr(p)
  prod(1/(1:10))
  n <- symbol("n")
  prod_(x, x, 1, n)
}

```

reciprocal_matrix	<i>Elementwise reciprocal matrix</i>
-------------------	--------------------------------------

Description

Elementwise reciprocal matrix

Usage

```
reciprocal_matrix(x, numerator = 1)
```

Arguments

x	Object x
numerator	The numerator in the result.

Examples

```

if (has_sympy()) {
  s <- as_sym("[[r1, r2, r3], [u1, u2, u3]]")
  reciprocal_matrix(s, numerator = 7)
}

```

simplify	<i>Simplify expression</i>
----------	----------------------------

Description

Simplify expression

Usage

```
simplify(x)
```

Arguments

x	A caracas_symbol
---	------------------

solve_lin	<i>Solve a linear system of equations</i>
-----------	---

Description

Find x in $Ax = b$. If b not supplied, the inverse of A is returned.

Usage

```
solve_lin(A, b)
```

Arguments

A	matrix
b	vector

solve_sys	<i>Solves a system of non-linear equations</i>
-----------	--

Description

If called as `solve_sys(lhs, vars)` the roots are found. If called as `solve_sys(lhs, rhs, vars)` the solutions to $lhs = rhs$ for $vars$ are found.

Usage

```
solve_sys(lhs, rhs, vars)
```

Arguments

lhs	Equation (or equations as row vector/1xn matrix)
rhs	Equation (or equations as row vector/1xn matrix)
vars	vector of variable names or symbols

Value

A list with solutions (with class `caracas_solve_sys_sol` for compact printing), each element containing a named list of the variables' values.

Examples

```
if (has_sympy()) {
  x <- symbol('x')
  exp1 <- 2*x + 2
  exp2 <- x
  solve_sys(cbind(exp1), cbind(exp2), x)

  x <- symbol("x")
  y <- symbol("y")
  lhs <- cbind(3*x*y - y, x)
  rhs <- cbind(-5*x, y+4)
  sol <- solve_sys(lhs, rhs, list(x, y))
  sol
}
```

subs

Substitute symbol for value

Description

Substitute symbol for value

Usage

```
subs(s, x, v)
```

Arguments

s	Expression
x	Name of symbol (character)
v	Value for x

See Also

[subs_vec\(\)](#), [subs_lst\(\)](#)

Examples

```
if (has_sympy()) {
  x <- symbol('x')
  e <- 2*x^2
  e
  subs(e, "x", "2")
  y <- as_sym("2")
  subs(e, "x", y)
}
```

`subs_lst`*Substitute symbol for of value given by a list*

Description

Useful for substituting solutions into expressions.

Usage

```
subs_lst(s, x)
```

Arguments

<code>s</code>	Expression
<code>x</code>	Named list of values

See Also

[subs\(\)](#), [subs_vec\(\)](#)

Examples

```
if (has_sympy()) {  
  p <- as_sym(paste0("p", 1:3))  
  y <- as_sym(paste0("y", 1:3))  
  a <- as_sym("a")  
  l <- sum(y*log(p))  
  L <- -1 + a*(sum(p) - 1)  
  g <- der(L, c(a, p))  
  sols <- solve_sys(g, c(a, p))  
  sol <- sols[[1L]]  
  sol  
  H <- der2(L, c(p, a))  
  H  
  H_sol <- subs_lst(H, sol)  
  H_sol  
}
```

`subs_vec`*Substitute af vector of symbols for a vector of values*

Description

Substitute af vector of symbols for a vector of values

Usage

```
subs_vec(s, x, v)
```

Arguments

s	Expression
x	Names of symbol (vector)
v	Values for x (vector)

See Also

[subs\(\)](#), [subs_lst\(\)](#)

Examples

```
if (has_sympy()) {
  x <- as_sym(paste0('x', 1:3))
  e <- 2*x^2
  e
  subs_vec(e, x, 1:3)
  subs_vec(e, x, x^2)
}
```

sum.caracas_symbol *Summation*

Description

Summation

Usage

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

Arguments

...	Elements to sum
na.rm	Not used

sum_	<i>Sum of a function</i>
------	--------------------------

Description

Sum of a function

Usage

```
sum_(f, var, lower, upper, doit = TRUE)
```

Arguments

f	Function to take sum of
var	Variable to take sum for (either string or caracas_symbol)
lower	Lower limit
upper	Upper limit
doit	Evaluate the sum immediately (or later with <code>doit()</code>)

Examples

```
if (has_sympy()) {  
  x <- symbol("x")  
  s <- sum_(1/x, "x", 1, 10)  
  as_expr(s)  
  sum(1/(1:10))  
  n <- symbol("n")  
  simplify(sum_(x, x, 1, n))  
}
```

symbol	<i>Create a symbol</i>
--------	------------------------

Description

Find available assumptions at <https://docs.sympy.org/latest/modules/core.html#module-sympy-core.assumptions>.

Usage

```
symbol(x, ...)
```

Arguments

x	Name to turn into symbol
...	Assumptions like <code>positive = TRUE</code>

Value

A `caracas_symbol`

See Also

[as_sym\(\)](#)

Examples

```
if (has_sympy()) {  
  x <- symbol("x")  
  2*x  
  
  x <- symbol("x", positive = TRUE)  
  ask(x, "positive")  
}
```

sympy_func

Call a SymPy function directly on x

Description

Call a SymPy function directly on x

Usage

```
sympy_func(x, fun, ...)
```

Arguments

x	Object to call fun on
fun	Function to call
...	Passed on to fun

Examples

```
if (has_sympy()) {
  def_sym(x, a)
  p <- (x-a)^4
  p
  q <- p %>% sympy_func("expand")
  q
  q %>% sympy_func("factor")

  def_sym(x, y, z)
  expr <- x*y + x - 3 + 2*x^2 - z*x^2 + x^3
  expr
  expr %>% sympy_func("collect", x)

  x <- symbol("x")
  y <- gamma(x+3)
  sympy_func(y, "expand_func")
  expand_func(y)
}
```

sympy_version

Get 'SymPy' version

Description

Get 'SymPy' version

Usage

```
sympy_version()
```

Value

The version of the 'SymPy' available

Examples

```
if (has_sympy()) {
  sympy_version()
}
```

t.caracas_symbol	<i>Transpose of matrix</i>
------------------	----------------------------

Description

Transpose of matrix

Usage

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

Arguments

x If caracas_symbol treat as such, else call `base::t()`.

taylor	<i>Taylor expansion</i>
--------	-------------------------

Description

Taylor expansion

Usage

```
taylor(f, x0 = 0, n = 6)
```

Arguments

f	Function to be expanded
x0	Point to expand around
n	Order of remainder term

See Also

[drop_remainder\(\)](#)

Examples

```
if (has_sympy()) {
  def_sym(x)
  f <- cos(x)
  ft_with_0 <- taylor(f, x0 = 0, n = 4+1)
  ft_with_0
  ft_with_0 %>% drop_remainder() %>% as_expr()
}
```

tex	<i>Export object to TeX</i>
-----	-----------------------------

Description

Export object to TeX

Usage

tex(x)

Arguments

x A caracas_symbol

tuplify	<i>Convert object to tuple</i>
---------	--------------------------------

Description

Convert object to tuple

Usage

tuplify(x)

Arguments

x Object

Examples

```
if (has_sympy()) {  
  x <- as_sym("Matrix([[b1*x1/(b2 + x1)], [b1*x2/(b2 + x2)], [b1*x3/(b2 + x3)]])")  
  tuplify(x)  
}
```

unbracket	<i>Remove inner-most dimension</i>
-----------	------------------------------------

Description

Remove inner-most dimension

Usage

```
unbracket(x)
```

Arguments

`x` Array symbol to collapse dimension from

Examples

```
if (has_sympy()) {
  x <- as_sym("[[x1/(b2 + x1)],
              [x2/(b2 + x2)],
              [x3/(b2 + x3)]],
            [[-b1*x1/(b2 + x1)^2],
             [-b1*x2/(b2 + x2)^2],
             [-b1*x3/(b2 + x3)^2]])")
  x
  unbracket(x)

  x <- as_sym("Matrix([[b1*x1/(b2 + x1)], [b1*x2/(b2 + x2)], [b1*x3/(b2 + x3)]])")
}
```

vec	<i>Stacks matrix to vector</i>
-----	--------------------------------

Description

Stacks matrix to vector

Usage

```
vec(x)
```

Arguments

`x` Matrix

Examples

```
if (has_sympy()) {
  A <- as_sym(matrix(1:9, 3))
  vec(A)
}
```

[.caracas_symbol *Extract or replace parts of an object*

Description

Extract or replace parts of an object

Usage

```
## S3 method for class 'caracas_symbol'
x[i, j, ..., drop = TRUE]
```

Arguments

x	A caracas_symbol.
i	row indices specifying elements to extract or replace
j	column indices specifying elements to extract or replace
...	Not used
drop	Simplify dimensions of resulting object

Examples

```
if (has_sympy()) {
  A <- matrix(c("a", 0, 0, 0, "a", "a", "a", 0, 0), 3, 3)
  B <- as_sym(A)
  B[1:2, ]
  B[, 2]
  B[2, , drop = FALSE]
}
```

[<- .caracas_symbol *Extract or replace parts of an object*

Description

Extract or replace parts of an object

Usage

```
## S3 replacement method for class 'caracas_symbol'
x[i, j, ...] <- value
```

Arguments

x	A caracas_symbol.
i	row indices specifying elements to extract or replace
j	column indices specifying elements to extract or replace
...	Not used
value	Replacement value

Examples

```
if (has_sympy()) {
  A <- matrix(c("a", 0, 0, 0, "a", "a", "a", 0, 0), 3, 3)
  B <- as_sym(A)
  B[, 2] <- "x"
  B
}
```

%>% *Pipe*

Description

Pipe operator

Arguments

lhs, rhs	specify what lhs and rhs are
----------	------------------------------

Index

- * **assumptions**
 - ask, 3
- * **calculus**
 - der, 7
 - der2, 8
 - drop_remainder, 13
 - int, 19
 - lim, 19
 - prod_, 27
 - sum_, 33
 - taylor, 36
- * **caracas_symbol**
 - as_expr, 5
 - as_sym, 5
 - def_sym, 6
 - doit, 12
 - fraction_parts, 16
 - listify, 22
 - matrify, 23
 - N, 25
 - subs, 30
 - subs_lst, 31
 - subs_vec, 31
 - symbol, 33
 - sympy_func, 34
 - tuplify, 37
 - unbracket, 38
- * **linalg**
 - as_character_matrix, 4
 - as_diag, 4
 - diag, 9
 - diag-set, 9
 - diag.caracas_symbol, 10
 - diag_, 11
 - dim.caracas_symbol, 11
 - do_la, 12
 - linalg, 20
 - mat_pow, 24
 - matrix-products, 23
 - matrix_, 24
 - reciprocal_matrix, 28
 - t.caracas_symbol, 36
 - vec, 38
- * **lowlevel**
 - eval_to_symbol, 14
- * **output**
 - as.character.caracas_symbol, 3
 - print.caracas_solve_sys_sol, 26
 - print.caracas_symbol, 27
 - tex, 37
- * **simple algebra**
 - Math.caracas_symbol, 22
 - Ops.caracas_symbol, 26
- * **simplify**
 - expand, 14
 - expand_func, 15
 - expand_log, 15
 - expand_trig, 16
 - simplify, 28
- * **solve**
 - solve_lin, 29
 - solve_sys, 29
- * **sympy**
 - get_py, 17
 - get_sympy, 17
 - has_sympy, 18
 - install_sympy, 18
 - sympy_version, 35
- * **vectors**
 - [.caracas_symbol, 39
 - [<- .caracas_symbol, 40
 - diag<- .caracas_symbol, 10
 - sum.caracas_symbol, 32
 - [.caracas_symbol, 39
 - [<- .caracas_symbol, 40
 - %% (matrix-products), 23
 - %>, 40
 - as.character.caracas_symbol, 3

as_character_matrix, 4
 as_diag, 4
 as_expr, 5
 as_sym, 5
 as_sym(), 6, 34
 ask, 3

 base::%*%(), 24
 base::t(), 36

 columnSpace (linalg), 20

 def_sym, 6
 der, 7
 der2, 8
 det (linalg), 20
 diag, 9
 diag-set, 9
 diag.caracas_symbol, 10
 diag<- .caracas_symbol, 10
 diag<- (diag-set), 9
 diag_, 11
 dim.caracas_symbol, 11
 do_la, 12
 do_la(), 21
 doit, 12
 doit(), 5, 19, 20, 27, 33
 drop_remainder, 13
 drop_remainder(), 36

 eigenval (linalg), 20
 eigenvec (linalg), 20
 eval_to_symbol, 14
 expand, 14
 expand_func, 15
 expand_log, 15
 expand_trig, 16

 fraction_parts, 16

 get_py, 17
 get_sympy, 17
 GramSchmidt (linalg), 20

 has_sympy, 18

 install_sympy, 18
 int, 19
 inv (linalg), 20

 lim, 19

 linalg, 20
 listify, 22

 mat_pow, 24
 Math.caracas_symbol, 22
 matrifify, 23
 matrix(), 24
 matrix-products, 23
 matrix_, 24

 N, 25
 nullspace (linalg), 20

 Ops.caracas_symbol, 26

 pinv (linalg), 20
 print.caracas_solve_sys_sol, 26
 print.caracas_symbol, 27
 print.caracas_symbol(), 26
 prod_, 27

 QRdecomposition (linalg), 20

 reciprocal_matrix, 28
 rowSpace (linalg), 20
 rref (linalg), 20

 simplify, 28
 singular_values (linalg), 20
 solve_lin, 29
 solve_sys, 29
 subs, 30
 subs(), 31, 32
 subs_lst, 31
 subs_lst(), 30, 32
 subs_vec, 31
 subs_vec(), 30, 31
 sum.caracas_symbol, 32
 sum_, 33
 symbol, 33
 symbol(), 5, 6
 sympy_func, 34
 sympy_version, 35

 t.caracas_symbol, 36
 taylor, 36
 taylor(), 13
 tex, 37
 tuplify, 37

 unbracket, 38

 vec, 38