

# Package: caracas (via r-universe)

October 26, 2024

**Version** 2.1.1

**Title** Computer Algebra

**Maintainer** Mikkel Meyer Andersen <mik1@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), Matrix, doBy (>= 4.6.15), magrittr

**Suggests** Ryacas, testthat (>= 2.1.0), knitr, rmarkdown, tinytex, magick, pdftools, qpdf

**License** GPL

**SystemRequirements** Python (>= 3.6.0)

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

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

**Roxygen** list(markdown = TRUE, roclets = c("`rd", "`collate", "`namespace"))

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**Repository** <https://r-cas.r-universe.dev>

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

**RemoteRef** HEAD

**RemoteSha** c230dee07c11a5d38eff69bdec84c074889eb583

## Contents

add_prefix . . . . .	4
all_vars . . . . .	4

apart . . . . .	5
as.character.caracas_symbol . . . . .	5
ask . . . . .	6
as_character . . . . .	6
as_character_matrix . . . . .	7
as_diag . . . . .	7
as_expr . . . . .	8
as_func . . . . .	8
as_sym . . . . .	9
as_vec . . . . .	10
cancel . . . . .	11
collect . . . . .	11
colspan . . . . .	12
cumsum.caracas_symbol . . . . .	12
def_sym . . . . .	13
der . . . . .	14
der2 . . . . .	15
diag . . . . .	16
diag-set . . . . .	16
diag.caracas_symbol . . . . .	17
diag<-.caracas_symbol . . . . .	17
diag_ . . . . .	18
diff_mat . . . . .	18
dim.caracas_symbol . . . . .	19
dim<-.caracas_symbol . . . . .	19
doit . . . . .	20
do_la . . . . .	21
drop_remainder . . . . .	22
eval_to_symbol . . . . .	23
expand . . . . .	23
expand_func . . . . .	24
expand_log . . . . .	24
expand_trig . . . . .	25
factor_ . . . . .	25
fraction_parts . . . . .	26
free_symbols . . . . .	26
generic-matrices . . . . .	27
get_basis . . . . .	28
get_py . . . . .	28
get_sympy . . . . .	29
has_sympy . . . . .	29
install_sympy . . . . .	30
int . . . . .	30
is_sym . . . . .	31
jacobian . . . . .	31
kroncker,caracas_symbol,caracas_symbol-method . . . . .	32
lim . . . . .	33
linalg . . . . .	33

listify . . . . .	35
ls_sym . . . . .	36
Math.caracas_symbol . . . . .	36
matrify . . . . .	37
matrix-products . . . . .	37
matrix_ . . . . .	38
matrix_cross_product . . . . .	38
mat_pow . . . . .	39
N . . . . .	39
Ops.caracas_symbol . . . . .	40
print.caracas_scaled_matrix . . . . .	40
print.caracas_solve_sys_sol . . . . .	41
print.caracas_symbol . . . . .	41
prod_ . . . . .	42
rankMatrix_ . . . . .	43
reciprocal_matrix . . . . .	43
rowSums_colSums . . . . .	44
scale_matrix . . . . .	44
score_hessian . . . . .	45
simplify . . . . .	46
solve.caracas_symbol . . . . .	46
solve_lin . . . . .	47
solve_sys . . . . .	48
solve_triangular . . . . .	49
special_matrices . . . . .	49
subs . . . . .	50
sum.caracas_symbol . . . . .	51
sum_ . . . . .	51
symbol . . . . .	52
symbol_class . . . . .	53
symbol_is_matrix . . . . .	53
sympy_func . . . . .	54
sympy_version . . . . .	55
sym_class . . . . .	55
sym_inherits . . . . .	56
t.caracas_symbol . . . . .	56
taylor . . . . .	57
tex . . . . .	57
tex.caracas_scaled_matrix . . . . .	58
texshow . . . . .	59
to_something . . . . .	59
tuplify . . . . .	60
unbracket . . . . .	60
unique.caracas_symbol . . . . .	61
unscale_matrix . . . . .	61
vectorfy . . . . .	62
[.caracas_symbol . . . . .	62
[<-.caracas_symbol . . . . .	63

[%>%](#) ..... 64

**Index** ..... 65

add\_prefix                      *Add prefix to each element of matrix*

### Description

Add prefix to each element of matrix

### Usage

```
add_prefix(x, prefix = "")
```

### Arguments

x                      Numeric or symbolic matrix  
 prefix                A character vector

### Examples

```
if (has_sympy()) {
  X <- matrix_sym(2, 3)
  X
  add_prefix(X, "e")

  X <- matrix(1:6, 3, 2)
  X
  add_prefix(X, "e")
}
```

all\_vars                      *All variables*

### Description

Return all variables in caracas symbol

### Usage

```
all_vars(x)
```

### Arguments

x                      caracas symbol

**Examples**

```
if (has_sympy()){
  x <- vector_sym(5)
  all_vars(x)
}
```

apart

*Partial fraction decomposition on a rational function***Description**

apart() performs a partial fraction decomposition on a rational function

**Usage**

```
apart(x)
```

**Arguments**

x                    A caracas\_symbol

**Examples**

```
if (has_sympy()){
  def_sym(x)
  expr = (4*x**3 + 21*x**2 + 10*x + 12)/(x**4 + 5*x**3 + 5*x**2 + 4*x)
  apart(expr)
}
```

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	<i>Ask for a symbol's property</i>
-----	------------------------------------

---

**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	<i>Coerce symbol to character</i>
--------------	-----------------------------------

---

**Description**

Coerce symbol to character

**Usage**

```
as_character(x)
```

**Arguments**

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

---

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	<i>Convert caracas object to R</i>
---------	------------------------------------

---

**Description**

Potentially calls `doit()`.

**Usage**

```
as_expr(x, first_doit = TRUE)

## S3 method for class 'caracas_symbol'
as.expression(x, ...)

## S3 method for class 'caracas_solve_sys_sol'
as.expression(x, ...)
```

**Arguments**

x	caracas_symbol
first_doit	Try <code>doit()</code> first
...	not used

**Examples**

```
if (has_sympy()) {
  v <- vector_sym(2)
  x <- as_expr(v)
  x
  y <- as.expression(v)
  y
}
```

---

as_func	<i>Convert expression into function object.</i>
---------	---

---

**Description**

Convert expression into function object.

**Usage**

```
as_func(x, order = NULL, vec_arg = FALSE)

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



**Arguments**

x	caracas expression.
order	desired order of function argument. Defaults to alphabetical ordering.
vec_arg	should the function take vector valued argument.
...	not used

**Examples**

```

if (has_sympy()) {
  def_sym(b0, b1, b2, k, x)
  e <- b1 + (b0 - b1)*exp(-k*x) + b2*x

  f1 <- as_func(e)
  f1
  f1(1, 2, 3, 4, 5)
  f1 <- as_func(e, order = sort(all_vars(e)))
  f1(1, 2, 3, 4, 5)
  f2 <- as_func(e, vec_arg = TRUE)
  f2
  f2(c(1, 2, 3, 4, 5))
  f2 <- as_func(e, order = sort(all_vars(e)), vec_arg = TRUE)
  f2
  f2(c(1,2,3,4,5))

  f1a <- as.function(e)
  f1a
  f1a(1, 2, 3, 4, 5)
  f1(1, 2, 3, 4, 5)
}

```

---

as\_sym

---

*Convert R object to caracas 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**

x	R object to convert to a symbol
declare_symbols	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
}
```

---

as\_vec

*Stacks matrix to vector*

---

## Description

Stacks matrix to vector

## Usage

```
as_vec(x)
```

## Arguments

x                    Matrix

## Examples

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

---

cancel	<i>Put rational function into standard form</i>
--------	---

---

**Description**

cancel() will take any rational function and put it into the standard canonical form

**Usage**

```
cancel(x)
```

**Arguments**

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

**Examples**

```
if (has_sympy()){
  def_sym(x, y, z)
  expr = cancel((x**2 + 2*x + 1)/(x**2 + x))
  cancel(expr)
  expr = (x*y**2 - 2*x*y*z + x*z**2 + y**2 - 2*y*z + z**2)/(x**2 - 1)
  cancel(expr)
  factor_(expr)
}
```

---

collect	<i>Collects common powers of a term in an expression</i>
---------	--

---

**Description**

Collects common powers of a term in an expression

**Usage**

```
collect(x, a)
```

**Arguments**

x, a	A caracas_symbol
------	------------------

**Examples**

```
if (has_sympy()){
  def_sym(x, y, z)
  expr = x*y + x - 3 + 2*x**2 - z*x**2 + x**3
  collect(expr, x)
}
```

---

colspan	<i>Column space (range) of a symbolic matrix</i>
---------	--

---

**Description**

Column space (range) of a symbolic matrix

**Usage**

```
colspan(x)
```

**Arguments**

x	Symbolic matrix
---	-----------------

**Examples**

```
if (has_sympy()) {
  X1 <- matrix_(paste0("x_",c(1,1,1,1, 2,2,2,2, 3,4,3,4)), nrow = 4)
  X1
  colspan(X1)
  do_la(X1, "columnspace")
  rankMatrix_(X1)

  X2 <- matrix_(paste0("x_",c(1,1,1,1, 0,0,2,2, 3,4,3,4)), nrow = 4)
  X2
  colspan(X2)
  do_la(X2, "columnspace")
  rankMatrix_(X2)
}
```

---

cumsum.caracas_symbol	<i>Cumulative Sums</i>
-----------------------	------------------------

---

**Description**

Cumulative Sums

**Usage**

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

**Arguments**

x	Elements to sum
---	-----------------

**Examples**

```
if (has_sympy()) {  
  A <- matrix(1:9, 3)  
  cumsum(A)  
  B <- matrix_sym(3, 3)  
  cumsum(B)  
  C <- vector_sym(3)  
  cumsum(C)  
}
```

---

def\_sym

*Define (invisibly) caracas symbols in global environment*

---

**Description**

Define (invisibly) caracas symbols in global environment

Define symbol for components in vector

**Usage**

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

```
def_sym_vec(x, 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	The environment in which the assignment is made.
x	Character vector.

**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) # Do not run as will cause a warning
  ls()
  def_sym(i, j, charvec = c("x", "y"))
  ls()
}

if (has_sympy()) {
  def_sym(z1, z2, z3)
  u <- paste0("u", seq_len(3))
  ## Creates symbols u1, u2, u3 and binds to names u1, u2, u3 in R.
  def_sym_vec(u)
  ## Same as (but easier than)
  def_sym(u1, u2, u3)
  ## Notice: this creates matrix [u1, u2, u3]
  as_sym(u)
}

```

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 <- matrifify(g)
G
dim(G)

h <- der(g, list(x, y))
h
dim(h)
as.character(h)
H <- matrifify(h)
H
dim(H)

g %>%
  der(list(x, y), simplify = FALSE) %>%
  der(list(x, y), simplify = FALSE) %>%
  der(list(x, y), simplify = FALSE)
}

```

---

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\_                      *Symbolic diagonal matrix*

---

**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 as_sym() when constructing symbolic matrix
...	Passed on to rep(x, n, ...)

**Examples**

```
if (has_sympy()) {
  diag_(c(1,3,5))
  diag_(c("a", "b", "c"))
  diag_("a", 2)
  diag_(vector_sym(4))
}
```

---

diff\_mat                      *Difference matrix*

---

**Description**

Difference matrix

**Usage**

```
diff_mat(N, l = "-1", d = 1)
```

**Arguments**

N	Number of rows (and columns)
l	Value / symbol below main diagonal
d	Value / symbol on main diagonal

**Examples**

```
if (has_sympy()){
  Dm <- diff_mat(4)
  Dm
  y <- vector_sym(4, "y")
  Dm %**% y
}
```

---

dim.caracas\_symbol      *Dimensions of a caracas symbol*

---

**Description**

Dimensions of a caracas symbol

**Usage**

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

**Arguments**

x                      caracas symbol

---

dim<- .caracas\_symbol      *Dimensions of a caracas symbol*

---

**Description**

Dimensions of a caracas symbol

**Usage**

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

**Arguments**

x                      caracas symbol  
value                  new dimension

**Examples**

```
if (has_sympy()) {  
  v <- vector_sym(4)  
  v  
  dim(v)  
  dim(v) <- c(2, 2)  
  v  
  m <- matrix_sym(2, 2)  
  dim(m)  
  dim(m) <- c(4, 1)  
  m  
}
```

---

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, "LU")
  LUdecomposition(A)

  do_la(A, "cholesky", hermitian = FALSE)
  chol(A, hermitian = FALSE)

  do_la(A, "singular_value_decomposition")
  do_la(A, "svd")
  svd_res <- svd_(A)
  svd_res
  U_expr <- svd_res$U |> as_expr()
  U_expr
  eval(U_expr, list(a = 3+2i))

  b <- symbol("b", real = TRUE)
  B <- matrix(c("b", "0", "0", "1"), 2, 2) %>% as_sym(declare_symbols = FALSE)
  svd_(B)

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

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

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

do_la(A, "trace")
trace_(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	<i>Create a symbol from a string</i>
----------------	--------------------------------------

---

**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	<i>Expand expression</i>
--------	--------------------------

---

**Description**

Expand expression

**Usage**

```
expand(x, ...)
```

**Arguments**

x	A caracas_symbol
...	Pass on to SymPy's expand, e.g. force = TRUE

**Examples**

```

if (has_sympy()) {
  def_sym(x)
  y <- log(exp(x))
  simplify(y)
  expand(simplify(y))
  expand(simplify(y), force = TRUE)
  expand_log(simplify(y))
}

```

---

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

*Expand a trigonometric expression*

---

**Description**

Expand a trigonometric expression

**Usage**

```
expand_trig(x)
```

**Arguments**

x                    A caracas\_symbol

---

factor\_

*Expand expression*

---

**Description**

Expand expression

**Usage**

```
factor_(x)
```

**Arguments**

x                    A caracas\_symbol

**Examples**

```
if (has_sympy()){  
  def_sym(x, y, z)  
  factor_(x**3 - x**2 + x - 1)  
  factor_(x**2*z + 4*x*y*z + 4*y**2*z)  
}
```

`fraction_parts`      *Get numerator and denominator of a fraction*

---

**Description**

Get numerator and denominator of a fraction

**Usage**

```
fraction_parts(x)
```

```
numerator(x)
```

```
denominator(x)
```

**Arguments**

`x`                      Fraction

**Examples**

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

---

`free_symbols`      *Get free symbol in expression*

---

**Description**

Get free symbol in expression

**Usage**

```
free_symbols(x)
```

**Arguments**

`x`                      Expression in which to get the free symbols in

**Examples**

```
if (has_sympy()) {  
  def_sym(a, b)  
  x <- (a - b)^4  
  free_symbols(x)  
}
```

---

generic-matrices

*Generate generic vectors and matrices*

---

**Description**

Generate generic vectors and matrices.

**Usage**

```
vector_sym(n, entry = "v")  
  
matrix_sym(nrow, ncol, entry = "v")  
  
matrix_sym_diag(nrow, entry = "v")  
  
matrix_sym_symmetric(nrow, entry = "v")
```

**Arguments**

n	Length of vector
entry	The symbolic name of each entry.
nrow, ncol	Number of rows and columns

**Examples**

```
if (has_sympy()) {  
  vector_sym(4, "b")  
  matrix_sym(3, 2, "a")  
  matrix_sym_diag(4, "s")  
  matrix_sym_symmetric(4, "s")  
}
```

get\_basis

*Get basis*

---

**Description**

Get basis

**Usage**

```
get_basis(x)
```

**Arguments**

x                    Caracas vector / matrix

**Examples**

```
if (has_sympy()) {  
  x <- vector_sym(3)  
  get_basis(x)  
  
  W <- matrix(c("r_1", "r_1", "r_2", "r_2", "0", "0", "u_1", "u_2"), nrow=4)  
  W <- as_sym(W)  
  get_basis(W)  
}
```

---

get\_py

*Access 'py' object*

---

**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	<i>Integrate a function</i>
-----	-----------------------------

---

**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)
}

```

---

is_sym	<i>Is object a caracas symbol</i>
--------	-----------------------------------

---

**Description**

Is object a caracas symbol

**Usage**

```
is_sym(x)
```

**Arguments**

x	object
---	--------

---

jacobian	<i>Compute Jacobian</i>
----------	-------------------------

---

**Description**

Compute Jacobian

**Usage**

```
jacobian(expr, vars)
```

**Arguments**

expr	'caracas expression'.
vars	variables to take derivative with respect to

**See Also**

[score\(\)](#), [hessian\(\)](#) [der\(\)](#)

**Examples**

```

if (has_sympy()) {
  x <- paste0("x", seq_len(3))
  def_sym_vec(x)
  y1 <- x1 + x2
  y2 <- x1^2 + x3
  y <- c(y1, y2)
  jacobian(y, x)
  u <- 2 + 4*x1^2
  jacobian(u, x1)
}

```

---

kronecker, caracas\_symbol, caracas\_symbol-method  
*Kronecker product of two matrices*

---

**Description**

Computes the Kronecker product of two matrices.

**Usage**

```

## S4 method for signature 'caracas_symbol,caracas_symbol'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

```

**Arguments**

`X, Y` matrices as caracas symbols.  
`FUN` a function; it may be a quoted string.  
`make.dimnames` Provide dimnames that are the product of the dimnames of 'X' and 'Y'.  
`...` optional arguments to be passed to 'FUN'.

**Value**

Kronecker product of A and B.

**Examples**

```

if (has_sympy()) {
  A <- matrix_sym(2, 2, "a")
  B <- matrix_sym(2, 2, "b")
  II <- matrix_sym_diag(2)
  EE <- eye_sym(2,2)
  JJ <- ones_sym(2,2)

  kronecker(A, B)
  kronecker(A, B, FUN = "+")
  kronecker(II, B)
}

```



```

    kronecker(EE, B)
    kronecker(JJ, B)
}

```

---

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, matrix = TRUE)

nullspace(x, matrix = TRUE)

rowspace(x, matrix = TRUE)

singular_values(x)

inv(x, method = c("gauss", "lu", "cf", "yac"))

eigenval(x)

eigenvec(x)

GramSchmidt(x)

pinv(x)

rref(x)

QRdecomposition(x)

LUdecomposition(x)

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

svd_(x, ...)

det(x, ...)

trace_(x)
```

**Arguments**

x	A matrix for which a property is requested.
matrix	When relevant should a matrix be returned.
method	The default works by Gaussian elimination. The alternatives are \$LU\$ decomposition (lu), the cofactor method (cf), and Ryacas (yac).
...	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)
  LUdecomposition(A)
  #chol(A) # error
  chol(A, hermitian = FALSE)
  eigenval(A)
  eigenvec(A)
  inv(A)
  det(A)
  rowspace(A)
  columnSpace(A)
  nullspace(A)

  ## Matrix inversion:
  d <- 3
  m <- matrix_sym(d, d)
  print(system.time(inv(m)))           ## Gauss elimination
  print(system.time(inv(m, method="cf"))) ## Cofactor
  print(system.time(inv(m, method="lu"))) ## LU decomposition
  if (requireNamespace("Ryacas")){
    print(system.time(inv(m, method="yac"))) ## Use Ryacas
  }

  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***Convert object to list of elements*

---

**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)
}
```

---

ls_sym	<i>List defined symbols</i>
--------	-----------------------------

---

**Description**

List defined symbols

**Usage**

```
ls_sym()
```

---

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  
  matrify(f)  
  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\_                      *Symbolic matrix*

**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)
}
```

matrix\_cross\_product      *Matrix cross product*

**Description**

Matrix cross product

**Usage**

```
crossprod_(x, y = NULL)
```

```
tcrossprod_(x, y = NULL)
```

**Arguments**

x, y                    caracas matrices

---

mat\_pow                    *Matrix power*

---

**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_scaled_matrix`  
*Print scaled matrix*

---

**Description**

Print scaled matrix

**Usage**

```

## S3 method for class 'caracas_scaled_matrix'
print(x, ...)

```

**Arguments**

`x`                    A `caracas_scaled_matrix`  
`...`                Passed to `print.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 <a href="#">print.caracas_symbol()</a>

**Examples**

```
if (has_sympy()) {
  x <- symbol('x')
  solve_sys(x^2, -1, 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
}
```

---

```
print.caracas_symbol
```

*Print symbol*

---

**Description**

Print symbol

**Usage**

```
## S3 method for class 'caracas_symbol'
print(
  x,
  prompt = getOption("caracas.prompt", default = "c: "),
  method = getOption("caracas.print.method", default = "utf8"),
  rowvec = getOption("caracas.print.rowvec", default = TRUE),
  ...
)
```

**Arguments**

x	A caracas_symbol
prompt	Which prompt/prefix to print (default: 'c: ')
method	What way to print (utf8, prettyascii, ascii, compactascii)
rowvec	FALSE to print column vectors as is
...	not used

---

prod_	<i>Product of a function</i>
-------	------------------------------

---

**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)
}
```

---

rankMatrix\_                    *Rank of matrix*

---

**Description**

Rank of matrix

**Usage**

```
rankMatrix_(x)
```

**Arguments**

x                    Numeric or symbolic matrix

**Examples**

```
if (has_sympy()) {
  X <- matrix_(paste0("x_",c(1,1,1,1,2,2,2,2,3,4,3,4)), nrow=4)
  X
  rankMatrix_(X)
  colspan(X)
}
```

---

reciprocal\_matrix            *Elementwise reciprocal matrix*

---

**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)
}
```

---

rowSums\_colSums

*Form Row and Column Sums*


---

**Description**

Form Row and Column Sums

**Usage**

```
rowSums_(x)
```

```
colSums_(x)
```

**Arguments**

x                      Symbolic matrix

**Examples**

```
if (has_sympy()) {
  X <- matrix_(paste0("x_", c(1,1,1,1,2,2,2,2,3,4,3,4)), nrow=4)
  rowSums_(X)
  colSums_(X)
}
```

---

scale\_matrix

*Create list of factors as in a product*


---

**Description**

Create list of factors as in a product

**Usage**

```
scale_matrix(X, k = NULL, divide = TRUE)
```

**Arguments**

X	matrix
k	scalar to be factored out
divide	Should X be divided with k before constructing scaled matrix?

**Examples**

```

if (has_sympy()) {
  V <- matrix_sym(2, 2, "v")
  a <- symbol("a")

  K <- a*V
  scale_matrix(K, a)
  scale_matrix(V, a, divide = FALSE)

  Ks <- scale_matrix(V, a, divide = FALSE)
  Ks
  W <- matrix_sym(2, 2, "w")
  unscale_matrix(Ks) %**% W
  unscale_matrix(Ks) %**% W |> scale_matrix(a)
  Ksi <- unscale_matrix(Ks) |> inv() |> scale_matrix(a/det(unscale_matrix(Ks)))
  (Ksi |> unscale_matrix()) %**% (Ks |> unscale_matrix()) |> simplify()
  tex(Ksi)
}

```

---

score\_hessian

*Score and Hessian matrix*


---

**Description**

Compute column vector of first derivatives and matrix of second derivatives of univariate function.

**Usage**

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

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

**Arguments**

expr	'caracas expression'.
vars	variables to take derivative with respect to.
simplify	Try to simplify result using <code>simplify()</code> ; may be time consuming.

**See Also**

[jacobian\(\)](#), [der\(\)](#)

**Examples**

```

if (has_sympy()) {
  def_sym(b0, b1, x, x0)
  f <- b0 / (1 + exp(b1*(x-x0)))
  S <- score(f, c(b0, b1))
  S
  H <- hessian(f, c(b0, b1))
  H
}

```

---

simplify

*Simplify expression*


---

**Description**

Simplify expression

**Usage**

```
simplify(x)
```

**Arguments**

x                    A caracas\_symbol

---

solve.caracas\_symbol

*Solve a System of Linear Equations*


---

**Description**

Solve a System of Linear Equations

**Usage**

```

## S3 method for class 'caracas_symbol'
solve(a, b, ...)

```

**Arguments**

a                    caracas\_symbol  
b                    If provided, either a caracas\_symbol (if not, as\_sym() is called on the object)  
...                   Not used

**Examples**

```
if (has_sympy()) {
  A <- matrix_sym(2, 2, "a")
  b <- vector_sym(2, "b")
  # Inverse of A:
  solve(A)
  inv(A)
  solve(A) %*% A |> simplify()
  # Find x in Ax = b
  x <- solve(A, b)
  A %*% x |> simplify()
  solve(A, c(2, 1)) |> simplify()
}
```

---

`solve_lin`*Solve a linear system of equations*

---

**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

**Examples**

```
if (has_sympy()) {
  A <- matrix_sym(2, 2, "a")
  b <- vector_sym(2, "b")
  # Inverse of A:
  solve_lin(A) %*% A |> simplify()
  # Find x in Ax = b
  x <- solve_lin(A, b)
  A %*% x |> simplify()
}
```

---

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

<code>lhs</code>	Equation (or equations as row vector/1xn matrix)
<code>rhs</code>	Equation (or equations as row vector/1xn matrix)
<code>vars</code>	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  
}
```



---

solve\_triangular      *Solve lower or upper triangular system*

---

**Description**

Solve lower or upper triangular system

**Usage**

```
solve_lower_triangular(a, b, ...)
```

```
solve_upper_triangular(a, b, ...)
```

**Arguments**

a	caracas_symbol
b	If provided, either a caracas_symbol (if not, as_sym() is called on the object)
...	Not used

**Examples**

```
if (has_sympy()) {
  A <- matrix_sym(3, 3)
  A[upper.tri(A)] <- 0
  solve_lower_triangular(A) |> simplify()
  A <- matrix_sym(3, 3)
  A[lower.tri(A)] <- 0
  solve_upper_triangular(A) |> simplify()
}
```

---

special\_matrices      *Special matrices: zeros\_sym, ones\_sym, eye\_sym*

---

**Description**

Special matrices: zeros\_sym, ones\_sym, eye\_sym

**Usage**

```
zeros_sym(nrow, ncol)
```

```
ones_sym(nrow, ncol)
```

```
eye_sym(nrow, ncol)
```

**Arguments**

nrow, ncol      Number of rows and columns of output

**See Also**

[diag\\_\(\)](#), [matrix\\_sym\(\)](#), [vector\\_sym\(\)](#)

**Examples**

```
if (has_sympy()){
  zeros_sym(3, 4)
  ones_sym(3, 4)
  eye_sym(3, 4)
}
```

---

subs	<i>Substitute symbol for value</i>
------	------------------------------------

---

**Description**

Substitute symbol for value

**Usage**

```
subs(sym, nms, vls)
```

**Arguments**

sym              Expression  
nms              Names of symbols (see Details)  
vls              Values that nms is substituted with (see Details)

**Details**

Two different ways to call this function is supported:

1. Supplying nms as a named list and omitting vls. If two components have the same name, the behaviour is undefined.
2. Supplying both nms and vls See Examples.

**Examples**

```

if (has_sympy()) {
  x <- symbol('x')
  e <- 2*x^2
  e
  subs(e, "x", "2")
  subs(e, x, 2)
  subs(e, list(x = 2))

  A <- matrix_sym(2, 2, "a")
  B <- matrix_sym(2, 2, "b")
  e <- A %*% A
  subs(e, A, B)
}

```

---

sum.caracas_symbol	<i>Summation</i>
--------------------	------------------

---

**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

*Create a symbol*

---

**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")  
}
```

---

symbol_class	<i>Ask type of caracas symbol</i>
--------------	-----------------------------------

---

**Description**

Ask type of caracas symbol

**Usage**

```
symbol_class(x)
```

**Arguments**

x                    An object, a caracas object is expected

---

symbol_is_matrix	<i>Check if object is a caracas matrix</i>
------------------	--

---

**Description**

Check if object is a caracas matrix

**Usage**

```
symbol_is_matrix(x)
```

**Arguments**

x                    An object

**Examples**

```

if (has_sympy() && sympy_version() >= "1.6") {
  x <- vector_sym(4)
  symbol_is_matrix(x) ## TRUE
  x2 <- as.character(x) ## "Matrix([[v1], [v2], [v3], [v4]])"
  symbol_is_matrix(x2) ## TRUE
  x3 <- as_character_matrix(x) ## R matrix
  symbol_is_matrix(x3) ## FALSE
}

```

---

sympy\_func

*Call a SymPy function directly on x*


---

**Description**

Extend caracas by calling SymPy functions directly.

**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)
}

```



---

<code>sym_inherits</code>	<i>Ask if type of caracas symbol is of a requested type</i>
---------------------------	---

---

**Description**

Ask if type of caracas symbol is of a requested type

**Usage**

```
sym_inherits(x, what)
```

**Arguments**

<code>x</code>	An object, a caracas object is expected
<code>what</code>	Requested type (e.g. atomic, vector, list, matrix)

---

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

---

**Description**

Transpose of matrix

**Usage**

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

**Arguments**

<code>x</code>	If <code>caracas_symbol</code> treat as such, else call <code>base::t()</code> .
----------------	--



---

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, zero_as_dot = FALSE, matstr = NULL, ...)
```

**Arguments**

x	A caracas_symbol
zero_as_dot	Print zero as dots
matstr	Replace <code>\begin{matrix}</code> with another environment, e.g. <code>pmatrix</code> . If vector of length two, the second element is an optional argument.
...	Other arguments passed along

**Examples**

```

if (has_sympy()) {
  S <- matrix_sym_symmetric(3, "s")
  S[1, 2] <- "1-x"
  S
  tex(S)
  tex(S, matstr = "pmatrix")
  tex(S, matstr = c("pmatrix", "r"))
}

```

---

```
tex.caracas_scaled_matrix
```

*Export scaled matrix to tex*

---

**Description**

Export scaled matrix to tex

**Usage**

```

## S3 method for class 'caracas_scaled_matrix'
tex(x, ...)

```

**Arguments**

x	scaled matrix
...	Other arguments passed along

---

texshow	<i>Dump latex representation of sympy object.</i>
---------	---

---

**Description**

Dump latex representation of sympy object and compile document into pdf.

**Usage**

```
texshow(x)
```

**Arguments**

x	An object that can be put in latex format with caracas' <code>tex()</code> function or a character string with tex code (in math mode).
---	---

**Value**

Nothing, but a .tex file and a .pdf file is generated.

**Examples**

```
if (has_sympy()) {
  S <- matrix_sym_symmetric(3, "s")
  S
  ## Not run:
  texshow(S)
  texshow(paste0("S = ", tex(S)))

  ## End(Not run)
}
```

---

to_something	<i>Coerce caracas object</i>
--------------	------------------------------

---

**Description**

Coerce caracas object

**Usage**

```
to_list(x)
```

```
to_vector(x)
```

```
to_matrix(x)
```

**Arguments**

x                    a caracas object is expected

---

tuplify                    *Convert object to tuple*

---

**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                    *Remove inner-most dimension*

---

**Description**

Remove inner-most dimension

**Usage**

unbracket(x)

**Arguments**

x                    Array symbol to collapse dimension from

**Examples**

```

if (has_sympy()) {
  x <- as_sym(paste0("x", 1:3))
  y <- as_sym("y")
  l <- list(x, y)
  l
  unbracket(l)
}

```

---

unique.caracas\_symbol *Extract unique elements*

---

**Description**

Extract unique elements

**Usage**

```

## S3 method for class 'caracas_symbol'
unique(x, incomparables = FALSE, ...)

```

**Arguments**

x	A caracas vector or matrix
incomparables	Same meaning as for other unique methods
...	Additional arguments; currently not used.

**Examples**

```

if (has_sympy()){
  v <- vector_sym(4)
  v2 <- rep(v, each=2)
  unique(v2)
}

```

---

unscale\_matrix *Extract matrix from scaled matrix*

---

**Description**

Extract matrix from scaled matrix

**Usage**

```

unscale_matrix(X)

```

**Arguments**

`x` scaled matrix created with `scale_matrix()`

**Examples**

```
if (has_sympy()) {
  V <- matrix_sym(2, 2, "v")
  a <- symbol("a")
  Ks <- scale_matrix(V, a, divide = FALSE)
  Ks
  unscale_matrix(Ks)
  V %**% a
}
```

---

`vectorfy` *Creates symbol vector from list of caracas symbols*

---

**Description**

Creates symbol vector from list of caracas symbols

**Usage**

```
vectorfy(x)
```

**Arguments**

`x` Symbol to be coerced to vector

---

[.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[, 3] <- vector_sym(3)
  B
}
```

---

%>%

*Pipe*

---

**Description**

Pipe operator

**Arguments**

lhs, rhs      specify what lhs and rhs are



# Index

- \* **assumptions**
  - ask, 6
- \* **calculus**
  - der, 14
  - der2, 15
  - drop\_remainder, 22
  - int, 30
  - jacobian, 31
  - lim, 33
  - prod\_, 42
  - score\_hessian, 45
  - sum\_, 51
  - taylor, 57
- \* **caracas\_symbol**
  - all\_vars, 4
  - as\_character, 6
  - as\_expr, 8
  - as\_func, 8
  - as\_sym, 9
  - def\_sym, 13
  - doit, 20
  - fraction\_parts, 26
  - free\_symbols, 26
  - is\_sym, 31
  - listify, 35
  - ls\_sym, 36
  - matrify, 37
  - N, 39
  - subs, 50
  - sym\_inherits, 56
  - symbol, 52
  - symbol\_class, 53
  - sympy\_func, 54
  - to\_something, 59
  - tuplify, 60
  - unbracket, 60
  - vectorfy, 62
- \* **linalg**
  - add\_prefix, 4
  - as\_character\_matrix, 7
  - as\_diag, 7
  - as\_vec, 10
  - colspan, 12
  - diag, 16
  - diag-set, 16
  - diag.caracas\_symbol, 17
  - diag\_, 18
  - diff\_mat, 18
  - dim.caracas\_symbol, 19
  - dim<- .caracas\_symbol, 19
  - do\_la, 21
  - generic-matrices, 27
  - get\_basis, 28
  - kronecker, caracas\_symbol, caracas\_symbol-method, 32
  - linalg, 33
  - mat\_pow, 39
  - matrix-products, 37
  - matrix\_, 38
  - matrix\_cross\_product, 38
  - rankMatrix\_, 43
  - reciprocal\_matrix, 43
  - rowSums\_colSums, 44
  - scale\_matrix, 44
  - special\_matrices, 49
  - symbol\_is\_matrix, 53
  - t.caracas\_symbol, 56
  - unscale\_matrix, 61
- \* **lowlevel**
  - eval\_to\_symbol, 23
- \* **output**
  - as.character.caracas\_symbol, 5
  - print.caracas\_scaled\_matrix, 40
  - print.caracas\_solve\_sys\_sol, 41
  - print.caracas\_symbol, 41
  - tex, 57
  - tex.caracas\_scaled\_matrix, 58
  - texshow, 59

- \* **simple\_algebra**
  - Math.caracas\_symbol, 36
  - Ops.caracas\_symbol, 40
- \* **simplify**
  - apart, 5
  - cancel, 11
  - collect, 11
  - expand, 23
  - expand\_func, 24
  - expand\_log, 24
  - expand\_trig, 25
  - factor\_, 25
  - simplify, 46
- \* **solve**
  - solve.caracas\_symbol, 46
  - solve\_lin, 47
  - solve\_sys, 48
  - solve\_triangular, 49
- \* **sympy**
  - get\_py, 28
  - get\_sympy, 29
  - has\_sympy, 29
  - install\_sympy, 30
  - sympy\_version, 55
- \* **vectors**
  - [.caracas\_symbol, 62
  - [<-.caracas\_symbol, 63
  - cumsum.caracas\_symbol, 12
  - diag<-.caracas\_symbol, 17
  - sum.caracas\_symbol, 51
- [.caracas\_symbol, 62
- [<-.caracas\_symbol, 63
- %%(matrix-products), 37
- %>, 64
- add\_prefix, 4
- all\_vars, 4
- apart, 5
- as.character.caracas\_symbol, 5
- as.expression.caracas\_solve\_sys\_sol  
(as\_expr), 8
- as.expression.caracas\_symbol(as\_expr),  
8
- as.function.caracas\_symbol(as\_func), 8
- as\_character, 6
- as\_character\_matrix, 7
- as\_diag, 7
- as\_expr, 8
- as\_func, 8
- as\_sym, 9
- as\_sym(), 13, 52
- as\_vec, 10
- ask, 6
- base::%%(), 38
- base::t(), 56
- cancel, 11
- chol.caracas\_symbol(linalg), 33
- collect, 11
- colspan, 12
- colSums\_(rowSums\_colSums), 44
- columnspace(linalg), 33
- crossprod\_(matrix\_cross\_product), 38
- cumsum.caracas\_symbol, 12
- def\_sym, 13
- def\_sym\_vec(def\_sym), 13
- denominator(fraction\_parts), 26
- der, 14
- der(), 31, 45
- der2, 15
- det(linalg), 33
- diag, 16
- diag-set, 16
- diag.caracas\_symbol, 17
- diag<-.caracas\_symbol, 17
- diag<-(diag-set), 16
- diag\_, 18
- diag\_(), 50
- diff\_mat, 18
- dim.caracas\_symbol, 19
- dim<-.caracas\_symbol, 19
- do\_la, 21
- do\_la(), 35
- doit, 20
- doit(), 8, 30, 33, 42, 52
- drop\_remainder, 22
- drop\_remainder(), 57
- eigenval(linalg), 33
- eigenvec(linalg), 33
- eval\_to\_symbol, 23
- expand, 23
- expand\_func, 24
- expand\_log, 24
- expand\_trig, 25
- eye\_sym(special\_matrices), 49

- factor\_, 25
- fraction\_parts, 26
- free\_symbols, 26
- generic-matrices, 27
- get\_basis, 28
- get\_py, 28
- get\_sympy, 29
- GramSchmidt (linalg), 33
- has\_sympy, 29
- hessian (score\_hessian), 45
- hessian(), 31
- install\_sympy, 30
- int, 30
- inv (linalg), 33
- is\_sym, 31
- jacobian, 31
- jacobian(), 45
- kronecker, caracas\_symbol, caracas\_symbol-method, 32
- lim, 33
- linalg, 33
- listify, 35
- ls\_sym, 36
- LUdecomposition (linalg), 33
- mat\_pow, 39
- Math.caracas\_symbol, 36
- matrify, 37
- matrix(), 38
- matrix-products, 37
- matrix\_, 38
- matrix\_cross\_product, 38
- matrix\_sym (generic-matrices), 27
- matrix\_sym(), 50
- matrix\_sym\_diag (generic-matrices), 27
- matrix\_sym\_symmetric (generic-matrices), 27
- N, 39
- nullspace (linalg), 33
- numerator (fraction\_parts), 26
- ones\_sym (special\_matrices), 49
- Ops.caracas\_symbol, 40
- pinv (linalg), 33
- print.caracas\_scaled\_matrix, 40
- print.caracas\_solve\_sys\_sol, 41
- print.caracas\_symbol, 41
- print.caracas\_symbol(), 40, 41
- prod\_, 42
- QRdecomposition (linalg), 33
- rankMatrix\_, 43
- reciprocal\_matrix, 43
- rowSpace (linalg), 33
- rowSums\_ (rowSums\_colSums), 44
- rowSums\_colSums, 44
- rref (linalg), 33
- scale\_matrix, 44
- scale\_matrix(), 62
- score (score\_hessian), 45
- score(), 31
- score\_hessian, 45
- simplify, 46
- singular\_values (linalg), 33
- solve.caracas\_symbol, 46
- solve\_lin, 47
- solve\_lower\_triangular (solve\_triangular), 49
- solve\_sys, 48
- solve\_triangular, 49
- solve\_upper\_triangular (solve\_triangular), 49
- special\_matrices, 49
- subs, 50
- sum.caracas\_symbol, 51
- sum\_, 51
- svd\_ (linalg), 33
- sym\_class, 55
- sym\_inherits, 56
- symbol, 52
- symbol(), 10, 13
- symbol\_class, 53
- symbol\_is\_matrix, 53
- sympy\_func, 54
- sympy\_version, 55
- t.caracas\_symbol, 56
- taylor, 57
- taylor(), 22
- tcrossprod\_ (matrix\_cross\_product), 38

`tex`, [57](#)  
`tex.caracas_scaled_matrix`, [58](#)  
`texshow`, [59](#)  
`to_list(to_something)`, [59](#)  
`to_matrix(to_something)`, [59](#)  
`to_something`, [59](#)  
`to_vector(to_something)`, [59](#)  
`trace_(linalg)`, [33](#)  
`tuplify`, [60](#)  
  
`unbracket`, [60](#)  
`unique.caracas_symbol`, [61](#)  
`unscale_matrix`, [61](#)  
  
`vector_sym(generic-matrices)`, [27](#)  
`vector_sym()`, [50](#)  
`vectorfy`, [62](#)  
  
`zeros_sym(special_matrices)`, [49](#)