Package 'symengine'

February 25, 2024

Title Interface to the 'SymEngine' Library

Version 0.2.6

Description Provides an R interface to 'SymEngine' <https://github.com/symengine/>, a standalone 'C++' library for fast symbolic manipulation. The package has functionalities for symbolic computation like calculating exact mathematical expressions, solving systems of linear equations and code generation.

Depends R (>= 4.2)

Imports methods, Rcpp

Suggests crayon, pracma, odeintr, testthat (>= 2.1.0), knitr, rmarkdown

LinkingTo Rcpp

SystemRequirements GNU make, cmake, gmp, mpfr

SystemRequirementsNote gmp (deb package: libgmp-dev, rpm package: gmp-devel), mpfr (deb package: libmpfr-dev, rpm package: mpfr-devel)

Encoding UTF-8

URL https://github.com/symengine/symengine.R

BugReports https://github.com/symengine/symengine.R/issues

Additional_repositories https://Marlin-Na.github.io/drat

License GPL (>= 2)

Copyright The R package bundles the 'SymEngine' library source and its subcomponents under 'src/upstream' directory. See file COPYRIGHTS for retained copyright notices as a relicensing and redistribution requirement.

RoxygenNote 7.2.3

Collate 'RcppExports.R' 'basic-getinfo.R' 'classes.R' 'basic.R' 'codegen.R' 'double_visitor.R' 'dxdt.R' 'function_symbol.R' 'knitr.R' 'lambdify.R' 'language_conversion.R' 'matrix.R' 'misc.R' 'ops.R' 'solve.R' 'summary.R' 'symbolic_array.R' 'symengine.R' 'symengine_info.R' 'utils-subset.R' 'vector.R' 'zzz.R'

VignetteBuilder knitr

NeedsCompilation yes

Author Jialin Ma [cre, aut],
Isuru Fernando [aut],
Xin Chen [aut]

Maintainer Jialin Ma <marlin@inventati.org>

Repository CRAN

Date/Publication 2024-02-25 13:50:02 UTC

R topics documented:

==,Basic,Basic-method *Bindings for Operators and Math Functions*

Description

These are S4 methods defined for `Basic`, `VecBasic` and `DenseMatrix`.

Usage

```
## S4 method for signature 'Basic,Basic'
e1 == e2

## S4 method for signature 'Basic,Basic'
e1 != e2

## S4 method for signature 'SymEngineDataType,SymEngineDataType'
Arith(e1, e2)

## S4 method for signature 'SymEngineDataType,vector'
Arith(e1, e2)

## S4 method for signature 'vector,SymEngineDataType'
Arith(e1, e2)

## S4 method for signature 'SymEngineDataType,missing'
e1 - e2

## S4 method for signature 'SymEngineDataType,missing'
e1 + e2

## S4 method for signature 'DenseMatrix,DenseMatrix'
x %*% y

## S4 method for signature 'VecBasic,VecBasic'
x %*% y

## S4 method for signature 'DenseMatrix,VecBasic'
x %*% y

## S4 method for signature 'DenseMatrix,vector'
x %*% y

## S4 method for signature 'VecBasic,DenseMatrix'
x %*% y

## S4 method for signature 'vector,DenseMatrix'
x %*% y
```

```
## S4 method for signature 'SymEngineDataType'
Math(x)

## S4 method for signature 'SymEngineDataType'
sinpi(x)

## S4 method for signature 'SymEngineDataType'
cospi(x)

## S4 method for signature 'SymEngineDataType'
tanpi(x)

## S4 method for signature 'SymEngineDataType'
log(x, base)

## S4 method for signature 'SymEngineDataType'
log2(x)

## S4 method for signature 'SymEngineDataType'
log10(x)

## S4 method for signature 'SymEngineDataType'
log1p(x)

## S4 method for signature 'SymEngineDataType'
expm1(x)

## S4 method for signature 'SymEngineDataType'
sum(x, ..., na.rm = FALSE)

## S4 method for signature 'SymEngineDataType'
prod(x, ..., na.rm = FALSE)
```

Arguments

`e1`, `e2`, `x`, `y`, `base`, `...`	Objects.
`na.rm`	Ignored

Value

== and != will return a logical vector. Other functions will return a `Basic`, `VecBasic` or `DenseMatrix`.

as.character,Basic-method *Some Conversion Methods*

Description

Miscellaneous S4 methods defined for converting a `Basic` or `VecBasic` object to R number/string/language object.

Usage

```
## S4 method for signature 'Basic'
as.character(x)

## S4 method for signature 'Basic'
as.numeric(x)

## S4 method for signature 'Basic'
as.integer(x)

## S4 method for signature 'VecBasic'
as.character(x)

## S4 method for signature 'VecBasic'
as.numeric(x)

## S4 method for signature 'VecBasic'
as.integer(x)

as.language(x)

## S4 method for signature 'Basic'
as.language(x)
```

Arguments

x	The object to be converted.

Value

Same as default methods of these generics. `as.language()` may return `symbol`, `integer`, `double` or `call`.

`as.matrix.DenseMatrix` *Methods Related to DenseMatrix*

Description

These are miscellaneous S3/S4 methods defined for `DenseMatrix` class.

Usage

```
## S3 method for class 'DenseMatrix'
as.matrix(x, ...)

## S4 method for signature 'DenseMatrix'
dim(x)

## S4 replacement method for signature 'DenseMatrix'
dim(x) <- value

## S4 replacement method for signature 'VecBasic'
dim(x) <- value

## S4 replacement method for signature 'Basic'
dim(x) <- value

## S4 replacement method for signature 'DenseMatrix'
dimnames(x) <- value

## S4 method for signature 'DenseMatrix'
dimnames(x)

## S4 method for signature 'DenseMatrix'
length(x)

## S4 method for signature 'DenseMatrix,ANY'
x[[i, j, ...]]

## S4 replacement method for signature 'DenseMatrix'
x[[i, j, ...]] <- value

## S4 method for signature 'DenseMatrix'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'DenseMatrix'
x[i, j, ...] <- value
```

Arguments

`x`	A DenseMatrix object.

`i, j, value, ..., drop`
: Arguments for subsetting, assignment or replacing.

Value

Same or similar with the generics of these methods.

`cbind.SymEngineDataType` *Joining DenseMatrix*

Description

S3 methods of `cbind` and `rbind` defined for `DenseMatrix` and `VecBasic`.

Usage

```
## S3 method for class 'SymEngineDataType'
cbind(..., deparse.level)

## S3 method for class 'SymEngineDataType'
rbind(..., deparse.level)
```

Arguments

`...`	DenseMatrix, VecBasic or R objects.
`deparse.level`	Not used.

Value

`DenseMatrix` S4 object.

`codegen` *Code Generation*

Description

Generate C/MathML/LaTeX/JavaScript code string from a `Basic` or `VecBasic` object.

Usage

```
codegen(x, type = c("ccode", "mathml", "latex", "jscode"))
```

Arguments

`x`	A Basic or a VecBasic object.
`type`	One of "ccode", "mathml", "latex" and "jscode".

Value

A character vector.

`D,SymEngineDataType-method` *Derivatives of a Symbolic Expression*

Description

S4 method of `D` defined for `Basic`. It returns the derivative of `expr` with regards to `name`. `name` may be missing if there is only one symbol in `expr`.

Usage

```
## S4 method for signature 'SymEngineDataType'
D(expr, name)
```

Arguments

`expr`	A Basic object.
`name`	A character vector or a Basic object of type Symbol.

Value

Same type as `expr` argument.

Examples

```
expr <- S(~ exp(x))
D(expr) == expr
expr <- S(~ x^2 + 2*x + 1)
D(expr)
```

`det` *Calculate the Determinant of DenseMatrix*

Description

S4 method of `det` defined for `DenseMatrix`.

Usage

```
det(x, ...)

## S4 method for signature 'DenseMatrix'
det(x, ...)
```

Arguments

x	A DenseMatrix object.
...	Unused.

Value

A Basic object.

Examples

```
mat <- Matrix(LETTERS[1:9], 3)
det(mat)
```

DoubleVisitor *Double Visitor*

Description

Construct DoubleVisitor object from Basic or VecBasic and use it to numerically evaluate symbolic expressions.

Usage

```
DoubleVisitor(
  exprs,
  args,
  perform_cse = TRUE,
  llvm_opt_level = if (symengine_have_component("llvm")) 3L else -1L
)

visitor_call(visitor, input, do_transpose = FALSE)
```

Arguments

exprs	A Basic object or a VecBasic object to be evaluated.
args	A VecBasic object indicating order of input arguments. Can be missing.
perform_cse	Boolean.
llvm_opt_level	Integer. If negative, it will return a LambdaDoubleVisitor, otherwise it will return a LLVMDoubleVisitor with the specified optimization level.
visitor	A DoubleVisitor object.
input	A numeric matrix. Each row is input value for one argument.
do_transpose	Boolean. Matters when exprs is a VecBasic. If true, output will have each column for one symbolic expression, otherwise each row for one symbolic expression.

Details

DoubleVisitor constructs the visitor and visitor itself is callable. visitor_call is the low level function to call the visitor with input.

Value

DoubleVisitor returns a callable LambdaDoubleVisitor or LLVMDoubleVisitor. visitor_call returns a numeric vector or matrix.

See Also

lambdify.

Examples

```
a <- S("a")
b <- S("b")
c <- S("c")
vec <- c(log(a), log(a)/log(b) + c)
func <- DoubleVisitor(vec, args = c(a, b, c))
args(func)

## Use closure
func(a = 1:10, b = 10:1, c = 1.43)

## Use visitor_call
input <- rbind(a = 1:10, b = 10:1, c = 1.43)
visitor_call(func, input, do_transpose = TRUE)
```

---

dxdt *Solve System of Ordinary Differential Equations*

Description

This is a wrapper of the odeintr R package using symengine objects to specify the ODE system and C code generation functionality from symengine to generate the C++ source. The dxdt function and defined == S4 method allow one to intuitively specify the ODE system with symengine objects. The ODESystem will generate C++ source and compile on the fly with Rcpp. Then predict can be used to get results.

Usage

```
dxdt(x)

## S4 method for signature 'DxdtOdeConstructor,ANY'
e1 == e2

ODESystem(
```

```
  odesys,
  ...,
  method = "rk5_i",
  atol = 1e-06,
  rtol = 1e-06,
  compile = TRUE
)

## S4 method for signature 'ODESystem'
predict(object, init, duration, step_size = 1, start = 0)
```

Arguments

`x`	A SymEngine Basic object of type Symbol or a R object that will be converted to `Symbol(x)`.
`e1`	A DxdtOdeConstructor S4 object which can be returned by ‘dxdt‘.
`e2`	A Basic object or an R object that will be converted to ‘S(e2)‘.
`odesys, ...`	DxdtOde S4 objects that can be returned with ‘dxdt(x) == rhs‘. Or ‘odesys‘ can be a list of DxdtOde S4 objects when there is no dot arguments.
`method, atol, rtol`	Passed to ‘odeintr::compile_sys‘.
`compile`	Logical, whether to compile the C++ source. Useful if you only want to obtain the code.
`object`	A ODESystem S4 object.
`init`	A numeric vector specifying the initial conditions. It can be named with the variable names or it can be unnamed but in the same of order of equations.
`duration, step_size, start`	Passed to the function generated by ‘odeintr::compile_sys‘.

Value

`dxdt` returns a DxdtOdeConstructor S4 object.

S4 method of ‘==‘ for "DxdtOdeConstructor" returns a DxdtOde S4 object.

‘ODESystem‘ returns a ODESystem S4 object.

‘predict‘ returns a dataframe.

Examples

```
# A differential equation specified with dxdt and ==
x <- Symbol("x")
eq <- dxdt(x) == 1/exp(x)
print(eq)
## Not run:
## Lorenz system
use_vars(x, y, z)
sigma <- 10
rho <- 28
```

```
beta <- 8/3
lorenz_sys <- ODESystem(
    dxdt(x) == sigma * (y - x),
    dxdt(y) == (rho - z) * x - y,
    dxdt(z) == - beta * z + x * y
)
res <- predict(
    lorenz_sys, init = c(x = 1, y = 1, z = 1), duration = 100, step_size = 0.001
)
plot(res[, c(2, 4)], type = 'l', col = "steelblue", main = "Lorenz Attractor")

## End(Not run)
```

`evalf` *Evaluating a SymEngine Object*

Description

This function will evaluate a SymEngine object to its "numerical" form with given precision. User may further use `as.double()` to convert to R value.

Usage

```
evalf(expr, bits = 53L, complex = FALSE)
```

Arguments

`expr`	A SymEngine object.
`bits`	The precision.
`complex`	Whether or not to be evaluated as a complex number.

Value

Same type as `expr` argument.

Examples

```
expr <- Constant("pi")
evalf(expr)
as.double(evalf(expr)) == pi
```

expand *Expand a Symbolic Expression*

Description

This function takes a SymEngine object and return its expanded form.

Usage

```
expand(x)
```

Arguments

x	A Basic/VecBasic/DenseMatrix S4 object.

Value

Same type as input.

Examples

```
expr <- S(~ (x + y) ^ 3)
expand(expr)
```

Function *Create a FunctionSymbol*

Description

`FunctionSymbol` creates a Basic object with type `FunctionSymbol`. `Function` returns a generator.

Usage

```
Function(name)

FunctionSymbol(name, args)
```

Arguments

name	Name of the function symbol
args	Dependent symbols

Value

`FunctionSymbol` returns a `Basic`. `Function` returns a function that will return a `Basic`

See Also

S

Examples

```
f <- Function("f")
a <- Symbol("a")
b <- Symbol("b")
f(a, b)
e <- f(a, f(a + b))
D(e, a)
FunctionSymbol("f", c(a,b))
```

---

get_type *Get Information about Basic Object*

Description

These functions are used to access the underlying properties of a Basic object.

Usage

```
get_type(x)

get_args(x)

get_hash(x)

get_str(x)

free_symbols(x)

function_symbols(x)

get_name(x)

get_prec(x)
```

Arguments

x	A Basic object.

Details

get_type Return the internal type

get_args Return the internal arguments of a Basic object as a VecBasic

get_hash Return the hash as a string

get_str Return the string representation of the Basic object

free_symbols Return free symbols in an expression

function_symbols Return function symbols in an expression

get_name Return name of a Basic object of type FunctionSymbol

get_prec Return precision of a Basic object of type RealMPFR

Value

- `get_type()`, `get_hash()`, `get_str()`, `get_name()` return a string.
- `get_args()`, `free_symbols()`, `function_symbols()` return a `VecBasic` S4 object.
- `get_prec()` returns an integer.

`lambdify` *Convert A Basic/VecBasic Object to R Function*

Description

These functions currently use `DoubleVisitor` to convert a Basic/VecBasic object to a `DoubleVisitor` which essentially is a S4 class extending R function.

Usage

```
lambdify(x, args, backend = c("auto", "lambda", "llvm"), perform_cse = TRUE)

## S3 method for class 'BasicOrVecBasic'
as.function(x, args, backend = "auto", perform_cse = TRUE, ...)
```

Arguments

`x`	A Basic object or a VecBasic object.
`args`	A VecBasic object specifying the arguments of the resulted function. It will be passed to `DoubleVisitor` and can be missing.
`backend`	One of "auto", "lambda" and "llvm". If "auto", `getOption("lambdify.backend")` will be used to determine the value. If that option is not set, it will be determined based on `symengine_have_component("llvm")`.
`perform_cse`	Passed to `DoubleVisitor`.
`...`	Not used

Value

A `DoubleVisitor` S4 object.

See Also

`DoubleVisitor`

LCM *Some Special Math Functions*

Description

These are some special mathematical functions and functions related to number theory.

Usage

```
LCM(a, b)

GCD(a, b)

nextprime(a)

factorial(x)

## S4 method for signature 'SymEngineDataType'
factorial(x)

choose(n, k)

## S4 method for signature 'SymEngineDataType'
choose(n, k)

zeta(a)

lambertw(a)

dirichlet_eta(a)

erf(a)

erfc(a)

## S4 method for signature 'SymEngineDataType,SymEngineDataType'
atan2(y, x)

kronecker_delta(x, y)

lowergamma(x, a)

uppergamma(x, a)

## S4 method for signature 'SymEngineDataType,SymEngineDataType'
beta(a, b)
```

```
## S4 method for signature 'SymEngineDataType'
psigamma(x, deriv = 0L)

## S4 method for signature 'SymEngineDataType'
digamma(x)

## S4 method for signature 'SymEngineDataType'
trigamma(x)
```

Arguments

`a`, `b`, `x`, `y`, `n`, `k`, `deriv`
: SymEngine objects (`Basic`/`VecBasic`/`DenseMatrix`). Some functions require Integer type.

Value

Same type as input.

`length,VecBasic-method`

Methods Related to VecBasic

Description

Miscellaneous S4 methods defined for `VecBasic` class.

Usage

```
## S4 method for signature 'VecBasic'
length(x)

## S3 method for class 'VecBasic'
rep(x, ...)

## S3 method for class 'Basic'
rep(x, ...)

## S3 method for class 'VecBasic'
unique(x, ...)

## S4 method for signature 'BasicOrVecBasic'
c(x, ...)

## S4 method for signature 'VecBasic,numeric'
x[[i, j, ...]]

## S4 method for signature 'VecBasic'
```

```
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'VecBasic'
x[[i]] <- value

## S4 replacement method for signature 'VecBasic'
x[i, j, ...] <- value
```

Arguments

`x`	Basic object or Vecbasic object.
`i, j, ..., drop, value`	Arguments for subsetting or replacing.

Value

Same or similar to the generics.

`Matrix` *DenseMatrix Constructor*

Description

This function constructs a symbolic matrix (`DenseMatrix` S4 object) with a similar interface with R's `matrix` function.

Usage

```
Matrix(data, nrow = 1L, ncol = 1L, byrow = FALSE)
```

Arguments

`data`	A R object.
`nrow, ncol`	Number of rows and columns.
`byrow`	Boolean value. Whether the data should be filled by row or by column.

Value

`DenseMatrix` S4 object.

S *Converting R object to Basic*

Description

'S' and 'Basic' converts a R object to a Basic object. 'Symbol', 'Real' and 'Constant' construct a Basic object with type "Symbol", "RealDouble"/"RealMPFR" and "Constant", respectively.

Usage

```
S(x)

Basic(x)

Symbol(x)

Constant(x)

Real(x, prec = NULL)
```

Arguments

x	A R object.
prec	If supplied, the argument will be parsed as a Basic object of type RealMPFR.

Details

For double vector, 'S' will check whether it is a whole number – if true, it will be converted to a Integer type. If this behavior is not desired, you can use 'Basic' or 'as(x, "Basic")'.

Value

A Basic S4 object.

Examples

```
S("(x + y)^2")
S(~ (x + y)^2)
S(NaN)
S(42)
Basic(42)
as(42, "Basic")
pi <- Constant("pi")
evalf(pi)
if (symengine_have_component("mpfr"))
    evalf(pi, 300)
Real(42)
if (symengine_have_component("mpfr"))
    Real(42, prec = 140)
```

`solve` *Solve Symbolic Equations*

Description

Solve system of symbolic equations or solve a polynomial equation. Depending on types of arguments, it supports different modes. See Details and Examples.

Usage

```
solve(a, b, ...)

## S4 method for signature 'DenseMatrix'
solve(a, b, ...)

## S4 method for signature 'VecBasic'
solve(a, b, ...)

## S4 method for signature 'Basic'
solve(a, b, ...)
```

Arguments

`a, b`	Objects, see details.
`...`	Not used.

Details

`solve` is a generic function dispatched on the class of the first argument.

- If `a` is a (square) DenseMatrix, it solves the equation `a %*% x = b` for `x`. (similar to `solve.default()`)
- If `a` is a DenseMatrix and `b` is missing, `b` is taken to be an identity matrix and `solve` will return the inverse of `a`. (similar to `solve.default()`)
- If `a` is a VecBasic, it solves the system of linear equations represented by `a` with regards to symbols represented in `b`.
- If `a` is a Basic, it solves the polynomial equation represented by a with regards to the symbol represented in `b`.

Value

A `VecBasic` or `DenseMatrix` S4 object.

Examples

```
## Inverse of a symbolic matrix
mat <- Matrix(c("A", "B", "C", "D"), 2)
solve(mat)

## Solve a %*% x == b
a <- Matrix(c("a11", "a21", "a12", "a22"), 2) # a is a 2x2 matrix
b <- Vector("b1", "b2")                       # b is a length 2 vector
solve(a, b)                                   # Solution of x (2x1 matrix)

## Solve the system of linear equations represented by a with regards to
## symbols in b
a <- Vector(~ -2*x + y - 4,  # A system of linear equations
            ~  3*x + y - 9)
b <- Vector(~x, ~y)          # Symbols to solve (x and y)
solve(a, b)                  # Solution of x and y
```

---

subs *Substitute Expressions in SymEngine Objects*

Description

This function will substitute `expr` with pairs of values in the dot arguments. The length of dot arguments must be a even number.

Usage

```
subs(expr, ...)
```

Arguments

`expr`	A `Basic` S4 object.
`...`	Pairs of Basic objects or values can be converted to `Basic`. In the order of "from1, to1, from2, to2, ...".

Value

Same type as `expr`.

symengine *symengine: R interface to SymEngine C++ library for symbolic computation*

Description

symengine is a R package for symbolic computation.

Details

SymEngine library is a standalone fast symbolic manipulation library written in C++. It allows computation over mathematical expressions in a way which is similar to the traditional manual computations of mathematicians and scientists. The R interface of the library tries to provide a user-friendly way to do symbolic computation in R and can be integrated into other packages to help solve related tasks. The design of the package is somehow similar to the SymPy package in Python. Unlike some other computer algebra systems, it does not invent its own language or domain specific language but uses R language to manipulate the symbolic expressions.

symengine uses the S4 dispatch system extensively to differentiate between calculation over normal R objects and symengine objects. For example, the semantics of `sin` in `expr <- Symbol("x"); sin(expr)` is different from the `sin` used over normal R numbers.

Basic class

`Basic` is simply a S4 class holding a pointer representing a symbolic expression in symengine. `Basic` objects have the same S4 class but can have different C-level representations which can be accessed via `get_type()`. For example, `Basic(~ 1/2)` will have "Rational" type and `Basic(1/2)` will have "RealDouble" type.

A `Basic` object will also have a list of associated sub-components which can be accessed via `get_args()`. For example, `(expr <- S("x") * 3L * S("a"))` will have type "Mul", and `as.list(get_args(expr))` will show the three factors of the multiplication.

A `Basic` object can be constructed via `Basic()`, `S()`, `Symbol()`, `Constant()` or `Real()`.

VecBasic and DenseMatrix class

VecBasic and DenseMatrix are S4 classes representing a symbolic vector or matrix. They can be constructed with `Vector()`, `V()`, `Matrix()`, `c()`, `rbind()` or `cbind()`. For example the following code will construct a 2x3 matrix.

```
vec <- Vector("a", "b")
cbind(vec, vec^2L, c(S("c"), S("d")))
```

The following functions are expected to work naturally with VecBasic and DenseMatrix classes.

- `[`, `[[`, `[<-` and `[[<-` for subsetting and assignment.
- `dim()`, `dim<-`, `length()`, `t()`, `det()`, `rbind()`, `cbind()`, `c()`, `rep()`
- `%*%` for matrix multiplication

- `solve(a, b)`: solve `a %*% x = b` where `a` is a square DenseMatrix and `b` is a VecBasic/DenseMatrix.
- `solve(a)`: find the inverse of `a` where `a` is a square DenseMatrix.
- `solve(a, b)`: solve system of linear equations represented by `a` (VecBasic) with regards to symbols in `b` (VecBasic).

Further, the R functions that work on Basic objects (e.g. `sin`) are expected work on VecBasic and DenseMatrix objects as well in a vectorized manner.

Function bindings

The following is a (incomplete) list of functions that are expected to work with symengine objects. Note that these functions can also be used inside a formula or R language objects and passed to S or Basic or Vector to construct symengine objects. For example `S(~ sin(x) + 1)` and `S(quote(sin(x) + 1))`.

- `+`, `-`, `*`, `/`, `^`
- `abs`, `sqrt`, `exp`, `expm1`, `log`, `log10`, `log2`, `log1p`
- `cos`, `cosh`, `sin`, `sinh`, `tan`, `tanh`, `acos`, `acosh`, `asin`, `asinh`, `atan`, `atanh`
- `cospi`, `sinpi`, `tanpi`, `gamma`, `lgamma`, `digamma`, `trigamma`
- `lambertw`, `zeta`, `dirichlet_eta`, `erf`, `erfc`
- `atan2`, `kronecker_delta`, `lowergamma`, `uppergamma`, `psigamma`, `beta`

`symengine_version` *Information about SymEngine Library*

Description

Functions to get symengine logo, version and external libraries built with.

Usage

```
symengine_version()

symengine_ascii_art()

symengine_have_component(
 which = c("mpfr", "flint", "arb", "mpc", "ecm", "primesieve", "piranha", "boost",
    "pthread", "llvm")
)

symengine_compilation_notes()
```

Arguments

`which`	A character vector.

Value

Character vector.

---

t *Transpose (as) a DenseMatrix*

Description

S4 methods of `t` defined for `Basic`, `VecBasic` and `DenseMatrix`.

Usage

```
t(x)

## S4 method for signature 'Basic'
t(x)

## S4 method for signature 'VecBasic'
t(x)

## S4 method for signature 'DenseMatrix'
t(x)
```

Arguments

`x`	A SymEngine object.

Value

A `DenseMatrix` S4 object.

---

use_vars *Initializing Variables*

Description

This is a convenient way to initialize variables and assign them in the given environment.

Usage

```
use_vars(..., .env = parent.frame(), .quiet = FALSE)
```

Arguments

`...`	All the arguments will be quoted and parsed, if a argument is named, the name will be used as the name of variable to assign, otherwise the argument can only be a symbol.
`.env`	Environment to assign.
`.quiet`	Whether to supress the message.

Value

Invisibly returns a list of assigned variables.

Examples

```
use_vars(x, y, expr = "a + b", p = 3.14)
p * x + y
expand(expr^2L)
rm(x, y, expr, p)
```

`Vector` *Symbolic Vector*

Description

A symbolic vector is represented by `VecBasic` S4 class. `Vector` and `V` are constructors of `VecBasic`.

Usage

```
Vector(x, ...)

V(...)
```

Arguments

`x, ...`	R objects.

Details

There are some differences between `Vector` and `V`.

- For double values, `V` will check whether they are whole number, and convert them to integer if so. `Vector` will not.
- `V` does not accept "non-scalar" arguments, like `Vector(c(1,2,3))`.

Value

A `VecBasic`.

Examples

```
a <- S("a")
b <- S("b")
Vector(a, b, a + b, 42L)
Vector(list(a, b, 42L))

Vector(1,2,a)
V(1,2,a)
```

Index