Package ‘SymbolizeR’

January 7, 2026

Title Symbolic Probability Calculations

Version 0.1.0

Description A lightweight symbolic probability engine that helps users derive Expectations, Variances, and Covariances using standard R syntax. Uses a capital letter heuristic to distinguish random variables (uppercase) from constants (lowercase).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Aidan J Wagner [aut, cre]

Maintainer Aidan J Wagner `<aidanjwagner03@gmail.com>`

Contents

`are.independent` *Check Independence*

Description

Checks if two random variables have been declared independent.

Usage

```
are.independent(x, y)
```

Arguments

`x`	Character name of first variable (or symbol)
`y`	Character name of second variable (or symbol)

Value

Logical TRUE if x and y are declared independent, FALSE otherwise

`assume.independent` *Declare Independent Random Variables*

Description

Registers random variables as mutually independent. When independence is declared, E(X * Y) simplifies to E(X) * E(Y).

Usage

```
assume.independent(...)
```

Arguments

`...`	Random variable symbols to declare as mutually independent

Value

Invisibly returns the list of independence pairs added

Examples

```
assume.independent(X, Y)
E(X * Y)  # Returns: E(X) * E(Y)

assume.independent(X, Y, Z)  # All three are mutually independent
E(X * Y * Z)  # Returns: E(X) * E(Y) * E(Z)
```

canonical.key *Canonical Form Key*

Description

Creates a canonical string key for a polynomial term. Sorts variables alphabetically and represents as "var1^pow1*var2^pow2".

Usage

```
canonical.key(base)
```

Arguments

base	A base expression

Value

A character string key

classify.type *Classify Expression Type*

Description

Determines if a symbol is a number, random variable, or constant based on naming conventions. Uppercase = RV, lowercase = constant.

Usage

```
classify.type(sym)
```

Arguments

sym	A symbol or expression to classify

Value

Character: "number", "rv", or "const"

clear.definitions *Clear Variable Definitions*

Description

Removes all registered variable definitions from the package environment.

Usage

```
clear.definitions()
```

Examples

```
define(X ~ Normal(mu, sigma))
clear.definitions()
E(X)   # Returns: E(X) (no longer has distribution info)
```

clear.independence *Clear Independence Assumptions*

Description

Removes all independence assumptions.

Usage

```
clear.independence()
```

Examples

```
assume.independent(X, Y)
clear.independence()
E(X * Y)  # Returns: E(X * Y) (no longer factors)
```

collect.terms *Collect Like Terms*

Description

Collects like terms in an expression by flattening to terms, grouping by base, summing coefficients, and reconstructing.

Usage

```
collect.terms(expr)
```

Arguments

`expr`	An expression

Value

A simplified expression with like terms collected

`combine.powers` *Combine Powers*

Description

Combines power expressions: x*x^n = x^(n+1), x^m*x^n = x^(m+n)

Usage

```
combine.powers(left, right)
```

Arguments

`left`	Left expression
`right`	Right expression

Value

Combined power expression, or NULL if not combinable

`contains.var` *Check if Expression Contains Variable*

Description

Recursively checks if an expression tree contains a specific variable symbol.

Usage

```
contains.var(expr, var)
```

Arguments

`expr`	An unevaluated R expression
`var`	A symbol to search for

Value

Logical: TRUE if var appears in expr, FALSE otherwise

Cov *Symbolic Covariance*

Description

Computes the symbolic covariance using the identity Cov(X, Y) = E(XY) - E(X)E(Y)

Usage

```
Cov(x, y)
```

Arguments

`x`	First random variable expression
`y`	Second random variable expression

Value

An unevaluated expression representing the covariance

Examples

```
Cov(X, Y)      # Returns: E(X * Y) - E(X) * E(Y)
```

define *Define Random Variable Distribution*

Description

Registers a random variable with its distribution, enabling automatic moment substitution in E(), Var(), etc.

Usage

```
define(formula)
```

Arguments

`formula`	A formula of the form X ~ Distribution(param1, param2, ...)

Value

Invisibly returns the distribution info stored

Examples

```
define(X ~ Normal(mu, sigma))
E(X)   # Returns: mu (instead of E(X))
E(X^2) # Returns: sigma^2 + mu^2
```

deriv.sym *Symbolic Differentiation*

Description

Computes the symbolic partial derivative of an expression with respect to a specified variable. Wraps `stats::D` with pre-processing to handle E() calls and post-processing to simplify the result.

Usage

```
## S3 method for class 'sym'
deriv(expr, var)
```

Arguments

`expr`	An R expression (uses non-standard evaluation, no quoting needed)
`var`	The variable symbol to differentiate by (no quoting needed)

Value

A simplified R expression representing the derivative

Examples

```
deriv.sym(x^3, x)               # 3 * x^2
deriv.sym(exp(a * x), x)        # a * exp(a * x)
deriv.sym(x * E(Y), x)          # E(Y) (treats E(Y) as constant)
```

derive.Cov *Derive Covariance Step-by-Step*

Description

Shows step-by-step derivation of the covariance calculation.

Usage

```
derive.Cov(x, y)
```

Arguments

`x`	First random variable expression
`y`	Second random variable expression

Value

A list with steps showing the derivation

Examples

```
derive.Cov(X, Y)
```

`derive.E` *Derive Expectation Step-by-Step*

Description

Shows step-by-step derivation of the expectation calculation, including all rules applied and intermediate results.

Usage

```
derive.E(expr)
```

Arguments

`expr`	An R expression involving random variables and constants

Value

A list with steps showing the derivation

Examples

```
derive.E(a * X + b)
derive.E(X + Y)
```

`derive.Kurtosis` *Derive Kurtosis Step-by-Step*

Description

Shows step-by-step derivation of the kurtosis calculation.

Usage

```
derive.Kurtosis(expr, excess = TRUE)
```

Arguments

`expr`	An R expression involving random variables and constants
`excess`	Logical; if TRUE (default), derives excess kurtosis

Value

A list with steps showing the derivation

Examples

```
derive.Kurtosis(X)
```

`derive.Skewness` *Derive Skewness Step-by-Step*

Description

Shows step-by-step derivation of the skewness calculation.

Usage

```
derive.Skewness(expr)
```

Arguments

`expr`	An R expression involving random variables and constants

Value

A list with steps showing the derivation

Examples

```
derive.Skewness(X)
```

`derive.Var` *Derive Variance Step-by-Step*

Description

Shows step-by-step derivation of the variance calculation.

Usage

```
derive.Var(expr)
```

Arguments

`expr`	An R expression involving random variables and constants

Value

A list with steps showing the derivation

Examples

```
derive.Var(X)
derive.Var(a * X + b)
```

E *Symbolic Expectation*

Description

Computes the symbolic expectation of an expression using the linearity of expectation. Uses non-standard evaluation.

Usage

```
E(expr)
```

Arguments

`expr`	An R expression involving random variables and constants. Random variables are identified by uppercase first letter.

Value

An unevaluated expression representing the expectation

Examples

```
E(X)            # Returns: E(X)
E(a * X)        # Returns: a * E(X)
E(X + Y)        # Returns: E(X) + E(Y)
E(2 * X + 3)    # Returns: 2 * E(X) + 3
```

ensure.expression *Ensure Expression*

Description

Normalizes input to an unevaluated expression, handling both raw calls and variables holding calls.

Usage

```
ensure.expression(expr)
```

Arguments

`expr`	An expression or variable containing an expression

Value

An unevaluated call object

`expand.poly` *Expand Polynomial*

Description

Expands polynomial expressions with integer powers, particularly (A + B)^n patterns. This enables Var() and Cov() to handle linear combinations properly.

Usage

```
expand.poly(expr)
```

Arguments

`expr`	An unevaluated R expression

Details

Currently handles:

- `(A + B)^2` -> `A^2 + 2*A*B + B^2`
- `(A - B)^2` -> `A^2 - 2*A*B + B^2`
- `(A * B)^n` -> `A^n * B^n`

Value

An expanded expression with powers distributed

`expect.recursive` *Recursive Expectation Engine*

Description

Recursively transforms an expression tree by applying the linearity of expectation. Constants pass through unchanged, random variables are wrapped in E(), and operators are handled appropriately.

Usage

```
expect.recursive(expr)
```

Arguments

`expr`	An unevaluated R expression

Details

- Numerics and constants return unchanged
- Random variables (uppercase) are wrapped in `E()`
- Addition/subtraction: E(X + Y) = E(X) + E(Y)
- Const * RV: E(a*X) = a*E(X)
- RV * RV: Returns E(X*Y) (cannot simplify without independence)

Value

A transformed expression with E() applied symbolically

`expr.to.latex` *Expression to LaTeX (Recursive)*

Description

Recursively converts an R expression tree to LaTeX.

Usage

```
expr.to.latex(expr)
```

Arguments

`expr`	An R expression

Value

LaTeX string

`extract.coef.base` *Simplify Expression*

Description

Recursively simplifies an expression by removing identity elements and evaluating trivial operations.

Extracts the coefficient and base from an expression. For a*X returns list(coef=a, base=X). For X returns list(coef=1, base=X).

Usage

```
extract.coef.base(expr)
```

Arguments

`expr`	An expression

Details

Simplification rules applied:

- `x + 0` -> `x`
- `x * 1` -> `x`
- `x * 0` -> `0`
- `0 / x` -> `0`
- `E(c)` where c is constant -> `c`
- `a*X + b*X` -> `(a+b)*X`

Value

A simplified expression

List with ’coef’ and ’base’, or NULL if not extractable

`extract.factors` *Extract Monomial Factors*

Description

Extracts all factors from a product expression and their powers. Returns a named list of powers keyed by variable name.

Usage

```
extract.factors(expr)
```

Arguments

`expr`	An expression (product of powers)

Value

A named list: list(vars = c("mu" = 2, "sigma" = 2), other = NULL)

`extract.linear.exp` *Extract Linear Exponent*

Description

For exp(-B*x) or exp(B*x), extracts B (with sign).

Usage

```
extract.linear.exp(expr, var)
```

Arguments

`expr`	An exp() call
`var`	The variable symbol

Value

The coefficient B where exponent = B*x, or NULL

`extract.power` *Extract Power Base and Exponent*

Description

For expressions like x^a, extracts base and exponent.

Usage

```
extract.power(expr, var)
```

Arguments

`expr`	The expression
`var`	The variable symbol

Value

List with 'base' and 'exp', or NULL if not a power of var

`extract.quadratic` *Extract Quadratic Coefficients*

Description

Extracts coefficients from a quadratic expression in var: c2*x^2 + c1*x + c0. Returns NULL if not quadratic.

Usage

```
extract.quadratic(expr, var)
```

Arguments

`expr`	The expression to analyze
`var`	The variable symbol

Value

A list with c2, c1, c0 coefficients, or NULL

`factors.to.expr` *Reconstruct Expression from Factors*

Description

Reconstructs an R expression from a canonical factor representation.

Usage

```
factors.to.expr(vars)
```

Arguments

`vars`	Named list of variable powers

Value

An R expression

`flatten.to.terms` *Flatten Expression to Terms*

Description

Flattens an expression into a list of (coefficient, base) pairs. Handles nested additions and subtractions.

Usage

```
flatten.to.terms(expr, sign = 1)
```

Arguments

`expr`	An expression
`sign`	The sign multiplier (1 or -1)

Value

A list of lists with ’coef’ and ’base’

`get.first.moment` *Get First Moment for Known Distribution*

Description

Returns the symbolic first moment E(X) if the variable has a registered distribution.

Usage

```
get.first.moment(var_name)
```

Arguments

`var_name`	Character name of the variable

Value

The moment expression, or NULL if not defined

`get.mgf` *Get Moment Generating Function for Known Distribution*

Description

Returns the symbolic MGF M_X(t) = E(e^(tX)) if the variable has a registered distribution.

Usage

```
get.mgf(var_name, t_expr)
```

Arguments

`var_name`	Character name of the variable
`t_expr`	The expression for t in e^(tX)

Value

The MGF expression, or NULL if not defined

get.nth.moment *Get nth Moment for Known Distribution*

Description

Returns the symbolic nth raw moment E(X^n) if the variable has a registered distribution and the moment is implemented.

Usage

```
get.nth.moment(var_name, n)
```

Arguments

var_name	Character name of the variable
n	The moment order (positive integer)

Value

The moment expression, or NULL if not defined

get.second.moment *Get Second Moment for Known Distribution*

Description

Returns the symbolic second moment E(X^2) if the variable has a registered distribution.

Usage

```
get.second.moment(var_name)
```

Arguments

var_name	Character name of the variable

Value

The moment expression, or NULL if not defined

`integrate.sym` *Symbolic Definite Integration*

Description

Computes a definite integral using kernel recognition. Recognizes Gamma, Gaussian, and Beta distribution kernels and returns their known normalizing constants. Falls back to an unevaluated Integrate() call if no pattern matches.

Usage

```
integrate.sym(expr, var, lower = -Inf, upper = Inf)
```

Arguments

`expr`	The integrand (uses non-standard evaluation, no quoting needed)
`var`	The variable of integration (no quoting needed)
`lower`	Lower bound of integration (default -Inf)
`upper`	Upper bound of integration (default Inf)

Value

Simplified expression of the result OR unevaluated Integrate() call

Examples

```
# Gamma kernel: integral of x^2 * exp(-x) from 0 to Inf = Gamma(3) = 2
integrate.sym(x^2 * exp(-x), x, 0, Inf)

# Gaussian kernel: integral of exp(-x^2) from -Inf to Inf = sqrt(pi)
integrate.sym(exp(-x^2), x, -Inf, Inf)

# Beta kernel: integral of x * (1-x) from 0 to 1 = Beta(2, 2) = 1/6
integrate.sym(x * (1-x), x, 0, 1)
```

`Kurtosis` *Symbolic Kurtosis*

Description

Computes the symbolic excess kurtosis using the fourth standardized moment. Excess Kurtosis = E((X - mu)^4) / sigma^4 - 3

Usage

```
Kurtosis(expr, excess = TRUE)
```

Arguments

`expr`	An R expression involving a random variable
`excess`	Logical; if TRUE (default), returns excess kurtosis (subtract 3)

Value

An unevaluated expression representing the kurtosis formula

Examples

```
Kurtosis(X)  # Returns symbolic excess kurtosis formula

# With defined distribution
define(X ~ Normal(mu, sigma))
Kurtosis(X)  # Returns 0 (normal has excess kurtosis 0)
```

`match.beta.kernel` *Match Beta Kernel*

Description

Attempts to match x^(A-1) * (1-x)^(B-1) pattern. Returns Beta(A, B) = Gamma(A)*Gamma(B)/Gamma(A+B)

Usage

```
match.beta.kernel(var_part, var)
```

Arguments

`var_part`	The variable-dependent part of integrand
`var`	The variable symbol

Value

The integral result, or NULL if no match

`match.gamma.kernel` *Match Gamma Kernel*

Description

Attempts to match x^(A-1) * exp(-B*x) pattern. Returns Gamma(A) / B^A

Usage

```
match.gamma.kernel(var_part, var)
```

Arguments

`var_part`	The variable-dependent part of integrand
`var`	The variable symbol

Value

The integral result, or NULL if no match

`match.gaussian.kernel` *Match Gaussian Kernel*

Description

Attempts to match exp(-A*x^2 + B*x + C) pattern. Returns sqrt(pi/A) * exp(B^2/(4*A) + C)

Usage

```
match.gaussian.kernel(var_part, var)
```

Arguments

`var_part`	The variable-dependent part of integrand
`var`	The variable symbol

Value

The integral result, or NULL if no match

`moment` *Compute nth Raw Moment*

Description

Computes the nth raw moment E(X^n) of a random variable. For defined distributions, returns symbolic expressions. For undefined distributions, returns E(X^n) as-is.

Usage

```
moment(X, n)
```

Arguments

`X`	A random variable (uses non-standard evaluation)
`n`	The moment order (positive integer)

Value

A symbolic expression for the nth moment

Examples

```
define(X ~ Normal(mu, sigma))
moment(X, 1)  # Returns: mu
moment(X, 2)  # Returns: sigma^2 + mu^2

# For undefined variables
moment(Y, 3)  # Returns: E(Y^3)
```

negate.expr *Safely Negate an Expression*

Description

Negates an expression, computing the result directly if the expression evaluates to a pure number, otherwise creating a unary minus call. Handles double negation: -(-x) returns x.

Usage

```
negate.expr(expr)
```

Arguments

expr	An expression to negate

Value

The negated expression (numeric if input is numeric, otherwise call)

normalize.base *Normalize Base Expression*

Description

Creates a canonical string representation of a base expression for grouping like terms.

Usage

```
normalize.base(base)
```

Arguments

base	A base expression

Value

A character string key

`normalize.product` *Normalize Product*

Description

Recursively normalizes a product expression, extracting all numeric coefficients and combining variable powers.

Usage

```
normalize.product(expr)
```

Arguments

`expr`	A product expression

Value

A list with 'coef' (numeric) and 'base' (expression with combined powers)

`partition.integrand` *Partition Integrand*

Description

Separates an integrand into constant and variable-dependent parts. For a product C * f(x), returns list(const = C, var = f(x)).

Usage

```
partition.integrand(expr, var)
```

Arguments

`expr`	The integrand expression
`var`	The variable of integration (as symbol)

Value

A list with 'const' and 'var' components

pkg.env *Package Environment for Distribution Metadata*

Description

Internal environment storing variable distribution definitions. Variables registered via define() are stored here with their distribution type and parameters.

Usage

```
pkg.env
```

Format

An object of class `environment` of length 0.

print.conditional_moment *Print Conditional Moment*

Description

Custom print method for conditional expressions. Formats 'when(expr, cond)' as 'result when condition'.

Usage

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

Arguments

`x`	A conditional moment object
`...`	Additional arguments

print.derivation *Print Derivation*

Description

Prints a derivation object in a readable format.

Usage

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

Arguments

`x`	A derivation object
`...`	Additional arguments (ignored)

Value

Invisibly returns the derivation object

`print.latex_output` *Print LaTeX Output*

Description

Pretty prints LaTeX code with syntax highlighting intent.

Usage

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

Arguments

`x`	A latex_output object
`...`	Additional arguments (ignored)

`protect.E` *Protect E() Calls Before Differentiation*

Description

Replaces E(X) calls with temporary constant symbols so that stats::D treats them as constants. E() calls are replaced with symbols like **E_X** which stats::D will ignore.

Usage

```
protect.E(expr)
```

Arguments

`expr`	An unevaluated R expression

Value

A list with ’expr’ (modified expression) and ’map’ (substitution map)

`restore.E` *Restore E() Calls After Differentiation*

Description

Replaces temporary symbols back to their original E() calls.

Usage

```
restore.E(expr, map)
```

Arguments

`expr`	An expression with E..._ symbols
`map`	The substitution map from protect.E

Value

The expression with E() calls restored

`show.independence` *Show Independence Assumptions*

Description

Displays all current independence assumptions.

Usage

```
show.independence()
```

Value

A character vector describing independence pairs, or a message if none

Examples

```
assume.independent(X, Y)
assume.independent(A, B, C)
show.independence()
```

`simplify.coef` *Simplify Coefficient Expression*

Description

Simplifies numeric coefficient expressions like 1 + 1 -> 2.

Usage

```
simplify.coef(coef)
```

Arguments

`coef`	A coefficient expression

Value

Simplified coefficient

`Skewness` *Symbolic Skewness*

Description

Computes the symbolic skewness using the third standardized moment. Skewness = E((X - mu)^3) / sigma^3 = (E(X^3) - 3*mu*sigma^2 - mu^3) / sigma^3

Usage

```
Skewness(expr)
```

Arguments

`expr`	An R expression involving a random variable

Value

An unevaluated expression representing the skewness formula

Examples

```
Skewness(X)  # Returns symbolic skewness formula

# With defined distribution
define(X ~ Normal(mu, sigma))
Skewness(X)  # Returns 0 (normal is symmetric)
```

`substitute.var` *Substitute Variable with Value*

Description

Replaces all occurrences of a variable with a value.

Usage

```
substitute.var(expr, var, val)
```

Arguments

`expr`	The expression
`var`	The variable to replace (symbol)
`val`	The value to substitute

Value

Modified expression

`symbol.to.latex` *Convert Symbol to LaTeX*

Description

Handles Greek letter conversion and subscript notation.

Usage

```
symbol.to.latex(name)
```

Arguments

`name`	Character name of the symbol

Value

LaTeX representation

`tag.conditional` *Tag Conditional Expression*

Description

Checks if an expression is a ’when’ call and tags it with ’conditional_moment’ class.

Usage

```
tag.conditional(expr)
```

Arguments

`expr`	An expression

Value

The expression, possibly with an added class

`to.latex` *Convert Expression to LaTeX*

Description

Converts an R expression tree to publication-ready LaTeX code. Handles standard mathematical operators, Greek letters, and statistical functions like E(), Var(), and Cov().

Usage

```
to.latex(x, delimiters = "none", ...)
```

Arguments

`x`	An R expression, call, or the result of E(), Var(), etc. Can also be a character string representation of an expression.
`delimiters`	Character; type of math delimiters to add. Options: "none" (default), "inline" ($...$), "display" (\[...\])
`...`	Additional arguments (for S3 method dispatch)

Value

A latex_output object containing LaTeX code (prints prettily)

Examples

```
to.latex(E(a * X + b))
to.latex(Var(X), delimiters = "inline")
to.latex(quote(sigma^2 + mu^2), delimiters = "display")
```

to.latex.default *Default LaTeX Conversion*

Description

Converts R expressions to LaTeX using expression tree traversal.

Usage

```
## Default S3 method:
to.latex(x, delimiters = "none", ...)
```

Arguments

`x`	An R expression or call object
`delimiters`	Character; "none", "inline" ($), or "display" (\[\])
`...`	Additional arguments (ignored)

Value

A latex_output object containing LaTeX code

to.latex.derivation *Convert Derivation to LaTeX*

Description

Converts each step of a derivation object to LaTeX format, suitable for including in homework assignments or papers.

Usage

```
## S3 method for class 'derivation'
to.latex(x, delimiters = "none", align = TRUE, ...)
```

Arguments

`x`	A derivation object (from derive.E, derive.Var, etc.)
`delimiters`	Character; type of math delimiters (ignored for derivations)
`align`	Logical; if TRUE, returns an align environment for step-by-step
`...`	Additional arguments (ignored)

Value

A character string containing LaTeX code

`try.combine.like.terms` *Try to Combine Like Terms*

Description

Attempts to combine like terms in addition/subtraction. a*X + b*X -> (a+b)*X, a*X - b*X -> (a-b)*X

Usage

```
try.combine.like.terms(left, right, op)
```

Arguments

`left`	Left simplified expression
`right`	Right simplified expression
`op`	The operator ("+" or "-")

Value

Combined expression, or NULL if terms are not alike

`undefine` *Undefine a Variable*

Description

Removes the distribution definition for a specific variable.

Usage

```
undefine(var_name)
```

Arguments

`var_name`	Character name of the variable to undefine

Examples

```
define(X ~ Normal(mu, sigma))
undefine("X")
```

`Var` *Symbolic Variance*

Description

Computes the symbolic variance using the identity Var(X) = E(X^2) - E(X)^2

Usage

```
Var(expr)
```

Arguments

`expr`	An R expression involving random variables and constants

Value

An unevaluated expression representing the variance

Examples

```
Var(X)          # Returns: E(X^2) - E(X)^2
Var(a * X)      # Returns: a^2 * (E(X^2) - E(X)^2)
```

Index