Package 'mvp'

September 5, 2019

Type Package

Title Fast Symbolic Multivariate Polynomials

Version 1.0-8

Depends methods,magrittr

Suggests knitr,rmarkdown,spray,microbenchmark,testthat

VignetteBuilder knitr

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Description Fast manipulation of symbolic multivariate polynomials using the 'Map' class of the Standard Template Library. The package uses print and coercion methods from the 'mpoly' package (Kahle 2013, ``Multivariate polynomials in R''. The R Journal, 5(1):162), but offers speed improvements. It is comparable in speed to the 'spray' package for sparse arrays, but retains the symbolic benefits of 'mpoly'.

License GPL (>= 2)

Imports Rcpp (>= 0.12.3), partitions, mpoly (>= 1.1.0), magic

LinkingTo Rcpp

SystemRequirements C++11

URL https://github.com/RobinHankin/mvp.git

BugReports https://github.com/RobinHankin/mvp/issues

NeedsCompilation yes

Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Repository CRAN

Date/Publication 2019-09-05 04:40:09 UTC

R topics documented:

`mvp-package` *Fast Symbolic Multivariate Polynomials*

Description

Fast manipulation of symbolic multivariate polynomials using the 'Map' class of the Standard Template Library. The package uses print and coercion methods from the 'mpoly' package (Kahle 2013, "Multivariate polynomials in R". The R Journal, 5(1):162), but offers speed improvements. It is comparable in speed to the 'spray' package for sparse arrays, but retains the symbolic benefits of 'mpoly'.

Details

The DESCRIPTION file:

Package:	mvp
Type:	Package
Title:	Fast Symbolic Multivariate Polynomials
Version:	1.0-8
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gma
Depends:	methods,magrittr
Suggests:	knitr,rmarkdown,spray,microbenchmark,testthat
VignetteBuilder:	knitr
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>

Description:	Fast manipulation of symbolic multivariate polynomials using the 'Map' class of the Standard Templa
License:	GPL (>= 2)
Imports:	Rcpp (>= 0.12.3), partitions, mpoly (>= 1.1.0), magic
LinkingTo:	Rcpp
SystemRequirements:	C++11
URL:	https://github.com/RobinHankin/mvp.git
BugReports:	https://github.com/RobinHankin/mvp/issues
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

```
Ops.mvp                 Arithmetic Ops Group Methods for 'mvp' objects
accessors               Accessor methods for mvp objects
allvars                 All variables in a multivariate polynomial
as.function.mvp         Functional form for multivariate polynomials
constant                The constant term
deriv                   Differentiation of 'mvp' objects
horner                  Horner's method
invert                  Replace symbols with their reciprocals
kahle                   A sparse multivariate polynomial
knight                  Chess knight
lose                    Drop empty variables
lowlevel                Low level functions
mpoly                   Conversion to and from mpoly form
mvp                     Multivariate polynomials, mvp objects
mvp-package             Fast Symbolic Multivariate Polynomials
ooom                    One over one minus a multivariate polynomial
print.mvp               Print methods for 'mvp' objects
rmvp                    Random multivariate polynomials
series                  Decomposition of multivariate polynomials by
                        powers
special                 Various functions to create simple multivariate
                        polynomials
spray                   Spray functionality
subs                    Substitution
summary                 Summary methods for mvp objects
zero                    The zero polynomial
```

Author(s)

NA

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

Examples

```
p <- as.mvp("1+x+x*y+x^5")
```

```
p + as.mvp("a+b^6")

p^3

subs(p^4,x="a+b^2")
aderiv(p^2,x=4)
horner(p,1:3)
```

---

accessor *Accessor methods for mvp objects*

Description

Accessor methods for mvp objects

Usage

```
vars(x)
powers(x)
coeffs(x)
coeffs(x) <- value
```

Arguments

x	Object of class mvp
value	Numeric vector of length 1

Details

Access the different parts of an mvp object. The constant term is technically a coefficient but is documented under constant.Rd.

Note

Accessing elements of an mvp object is problematic because the order of the terms of an mvp object is not well-defined. This is because the map class of the STL does not specify an order for the key-value pairs (and indeed the actual order in which they are stored may be implementation dependent). The situation is similar to the hyper2 package which uses the STL in a similar way.

So the output of coeffs(x) is defined only up to an unknown rearrangement. If all the coefficients are the same, this does not matter. The same considerations apply to the output of vars(), which returns a list of character vectors in an undefined order, and the output of powers(), which returns a numeric list whose elements are in an undefined order. However, even though the order of these three objects is undefined individually, their ordering is jointly consistent in the sense that the first element of coeffs(x) corresponds to the first element of vars(x) and the first element of powers(x). The identity of this element is not defined—but whatever it is, the first element of all three accessor methods refers to it.

Note also that a single term (something like `4a^3*b*c^6`) has the same issue: the variables are not stored in a well-defined order. This does not matter because the algebraic value of the term does not depend on the order in which the variables appear and this term would be equivalent to `4b*c^6*a^3`.

The vignette provides an extensive discussion of this.

Author(s)

Robin K. S. Hankin

See Also

`constant`

Examples

```
a <- rmvp(5)
vars(a)
powers(a)
coeffs(a)

coeffs(a) <- 1  # A simpler object
coeffs(a) <- 0  # The zero polynomial
```

`allvars` *All variables in a multivariate polynomial*

Description

Returns a character vector containing all the variables present in a `mvp` object

Usage

```
allvars(x)
```

Arguments

`x`	object of class `mvp`

Note

The character vector returned is not in any particular order

Author(s)

Robin K. S. Hankin

Examples

```
p <- rmvp(5)
allvars(p)
```

---

as.function.mvp — *Functional form for multivariate polynomials*

Description

Coerces a multivariate polynomial into a function

Usage

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

Arguments

`x`	Multivariate polynomial
`...`	Further arguments (currently ignored)

Author(s)

Robin K. S. Hankin

Examples

```
p <- as.mvp("1+a^2 + a*b^2 + c")
p
f <- as.function(p)

f(a=1)
f(a=1,b=2)
f(a=1,b=2,c=3)             # coerces to a scalar
f(a=1,b=2,c=3,lose=FALSE)  # formal mvp object
```

constant *The constant term*

Description

Get and set the constant term of an mvp object

Usage

```
## S3 method for class 'mvp'
constant(x)
## S3 replacement method for class 'mvp'
constant(x) <- value
## S3 method for class 'numeric'
constant(x)
is.constant(x)
```

Arguments

x	Object of class mvp
value	Scalar value for the constant

Details

The constant term in a polynomial is the coefficient of the empty term. In an mvp object, the map {} -> c, implies that c is the constant.

If x is an mvp object, constant(x) returns the value of the constant in the multivariate polynomial; if x is numeric, it returns a constant multivariate polynomial with value x.

Function is.constant() returns TRUE if its argument has no variables and FALSE otherwise.

Author(s)

Robin K. S. Hankin

Examples

```
a <- rmvp(5)+4
constant(a)
constant(a) <- 33
a

constant(0)  # the zero mvp
```

deriv *Differentiation of mvp objects*

Description

Differentiation of mvp objects

Usage

```
## S3 method for class 'mvp'
deriv(expr, v, ...)
## S3 method for class 'mvp'
aderiv(expr, ...)
```

Arguments

expr	Object of class mvp
v	Character vector. Elements denote variables to differentiate with respect to
...	Further arguments, ignored in deriv() but specifies the differentials in aderiv()

Details

Function deriv(S,v) returns $\frac{\partial^r S}{\partial v_1 \partial v_2 \ldots \partial v_r}$.

Function aderiv() uses the ellipsis construction with the names of the argument being the variable to be differentiated with respect to. Thus aderiv(S,x=1,y=2) returns $\frac{\partial^3 S}{\partial x \partial y^2}$.

Author(s)

Robin K. S. Hankin

See Also

taylor

Examples

```
p <- rmvp(10,9,9,letters[1:4])
deriv(p,letters[1:3])
deriv(p,rev(letters[1:3]))  # should be the same

aderiv(p,a=1,b=2,c=1)

## verify the chain rule:
x <- rmvp(7,symbols=6)
v <- allvars(x)[1]
s <- as.mvp("1  +  y  -  y^2 zz  +  y^3 z^2")
LHS <- subsmvp(deriv(x,v)*deriv(s,"y"),v,s)   # dx/ds*ds/dy
RHS <- deriv(subsmvp(x,v,s),"y")              # dx/dy
```

```
LHS - RHS # should be zero
```

---

horner *Horner's method*

Description

Horner's method for multivariate polynomials

Usage

```
horner(P,v)
```

Arguments

P	Multivariate polynomial
v	Numeric vector of coefficients

Details

Given a polynomial

$$p(x) = a_0 + a_1 + a_2 x^2 + \cdots + a_n x^n$$

it is possible to express $p(x)$ in the algebraically equivalent form

$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right)$$

which is much more efficient for evaluation, as it requires only n multiplications and n additions, and this is optimal. But this is not implemented here because it's efficient. It is implemented because it works if x is itself a (multivariate) polynomial, and that is the second coolest thing ever. The coolest thing ever is the `Reduce()` function.

Author(s)

Robin K. S. Hankin

See Also

`ooom`

Examples

```
horner("x",1:5)
horner("x+y",1:3)

w <- as.mvp("x+y^2")
stopifnot(1 + 2*w + 3*w^2 == horner(w,1:3))  # note off-by-one issue

"x+y+x*y" %>% horner(1:3) %>% horner(1:2)
```

`invert` *Replace symbols with their reciprocals*

Description

Given an `mvp` object, replace one or more symbols with their reciprocals

Usage

```
invert(p, v)
```

Arguments

`p`	Object (coerced to) `mvp` form
`v`	Character vector of symbols to be replaced with their reciprocal; missing interpreted as replace all symbols

Author(s)

Robin K. S. Hankin

See Also

`subs`

Examples

```
invert("x")

invert(rmvp(10,7,7,letters[1:3]),"a")
```

kahle — *A sparse multivariate polynomial*

Description

A sparse multivariate polynomial inspired by Kahle (2013)

Usage

```
kahle(n = 26, r = 1, p = 1, coeffs = 1, symbols = letters)
```

Arguments

n	Number of different symbols to use
r	Number of symbols in a single term
p	Power of each symbol in each terms
coeffs	Coefficients of the terms
symbols	Alphabet of symbols

Author(s)

Robin K. S. Hankin

References

David Kahle 2013. "**mpoly**: multivariate polynomials in R". *R Journal*, volume 5/1.

See Also

special

Examples

```
kahle()  # a+b+...+z
kahle(r=2,p=1:2)  # Kahle's original example

## example where mvp runs faster than spray (mvp does not need a 200x200 matrix):
k <- kahle(200,r=3,p=1:3,symbols=paste("x",sprintf("%02d",1:200),sep=""))
system.time(ignore <- k^2)
#system.time(ignore <- mvp_to_spray(k)^2)   # needs spray package loaded
```

knight *Chess knight*

Description

Generating function for a chess knight on an infinite d-dimensional chessboard

Usage

```
knight(d, can_stay_still = FALSE)
```

Arguments

d	Dimension of the board
can_stay_still	Boolean, with default `FALSE` meaning that the knight is obliged to move and `FALSE` meaning that it has the option of remaining on its square

Note

The function is a slight modification of `spray::knight()`.

Author(s)

Robin K. S. Hankin

Examples

```
knight(2)      # regular chess knight on a regular chess board
knight(2,TRUE) # regular chess knight that can stay still

# Q: how many ways are there for a 4D knight to return to its starting
# square after four moves?

# A:
constant(knight(4)^4)

# Q ...and how many ways in four moves or fewer?

# A1:
constant(knight(4,TRUE)^4)

# A2:
constant((1+knight(4))^4)
```

lose — *Drop empty variables*

Description

Convert an mvp object which is a pure constant into a scalar whose value is the coefficient of the empty term.

A few functions in the package (currently subs(), subsy()) take a lose argument that behaves much like the drop argument in base extraction.

Usage

```
## S3 method for class 'mvp'
lose(x)
```

Arguments

x	Object of class mvp

Author(s)

Robin K. S. Hankin

See Also

subs

Examples

```
m1 <- as.mvp("1+bish +bash^2 + bosh^3")
m2 <- as.mvp("bish +bash^2 + bosh^3")

m1-m2         # an mvp object
lose(m1-m2)   # numeric
```

lowlevel *Low level functions*

Description

Various low-level functions that call the C routines

Usage

```
mvp_substitute(allnames,allpowers,coefficients,v,values)
mvp_substitute_mvp(allnames1, allpowers1, coefficients1, allnames2, allpowers2,
    coefficients2, v)
mvp_vectorised_substitute(allnames, allpowers, coefficients, M, nrows, ncols, v)
mvp_prod(allnames1,allpowers1,coefficients1,allnames2,allpowers2,coefficients2)
mvp_add(allnames1, allpowers1, coefficients1, allnames2, allpowers2,coefficients2)
simplify(allnames,allpowers,coefficients)
mvp_deriv(allnames, allpowers, coefficients, v)
mvp_power(allnames, allpowers, coefficients, n)
```

Arguments

`allnames,allpowers,coefficients,allnames1,allpowers1,coefficients1,allnames2,allpowers2,coefficient`
Variables sent to the C routines

Details

These functions call the functions defined in `RcppExports.R`

Note

These functions are not intended for the end-user. Use the syntatic sugar (as in `a+b` or `a*b` or `a^n`), or functions like `mvp_plus_mvp()`, which are more user-friendly

Author(s)

Robin K. S. Hankin

mpoly *Conversion to and from mpoly form*

Description

The **mpoly** package by David Kahle provides similar functionality to this package, and the functions documented here convert between mpoly and `mvp` objects. The mvp package uses `mpoly::mp()` to convert character strings to `mvp` objects.

Usage

```
mpoly_to_mvp(m)
## S3 method for class 'mvp'
as.mpoly(x,...)
```

Arguments

`m`	object of class mvp
`x`	object of class mpoly
`...`	further arguments, currently ignored

Author(s)

Robin K. S. Hankin

See Also

`spray`

Examples

```
x <- rmvp(5)

x == mpoly_to_mvp(mpoly::as.mpoly(x))        # should be TRUE
```

---

mvp — *Multivariate polynomials, mvp objects*

Description

Create, test for, an coerce to, `mvp` objects

Usage

```
mvp(vars, powers, coeffs)
is_ok_mvp(vars,powers,coeffs)
is.mvp(x)
as.mvp(x,...)
```

Arguments

`vars`	List of variables comprising each term of an `mvp` object
`powers`	List of powers corresponding to the variables of the `vars` argument
`coeffs`	Numeric vector corresponding to the coefficients to each element of the `var` and `powers` lists
`x`	Object possibly of class `mvp`
`...`	Further arguments, passed to the methods

Details

Function `mvp()` is the formal creation mechanism for `mvp` objects. However, it is not very user-friendly; it is better to use `as.mvp()` in day-to-day use.

Function `is_ok_mvp()` checks for consistency of its arguments.

Author(s)

Robin K. S. Hankin

Examples

```
mvp(list("x" , c("x","y"), "a",c("y","x")),list(1,1:2,3,c(-1,4)),1:4)

## Note how the terms appear in an arbitrary order, as do
## the symbols within a term.


kahle  <- mvp(
    vars   = split(cbind(letters,letters[c(26,1:25)]),rep(seq_len(26),each=2)),
    powers = rep(list(1:2),26),
    coeffs = 1:26
)

## again note arbitrary order of terms and symbols within a term
```

`ooom` *One over one minus a multivariate polynomial*

Description

Uses Taylor's theorem to give one over one minus a multipol

Usage

```
ooom(P,n)
```

Arguments

`n`	Order of expansion
`P`	Multivariate polynomial

Author(s)

Robin K. S. Hankin

See Also

horner

Examples

```
ooom("x",5)
ooom("x",5) * as.mvp("1-x")  # zero through fifth order


ooom("x+y",4)

"x+y" %>% ooom(5) %>% `-`(1) %>% ooom(3)
```

Ops.mvp — *Arithmetic Ops Group Methods for* mvp *objects*

Description

Allows arithmetic operators to be used for multivariate polynomials such as addition, multiplication, integer powers, etc.

Usage

```
## S3 method for class 'mvp'
Ops(e1, e2)
mvp_negative(S)
mvp_times_mvp(S1,S2)
mvp_times_scalar(S,x)
mvp_plus_mvp(S1,S2)
mvp_plus_numeric(S,x)
mvp_eq_mvp(S1,S2)
```

Arguments

e1,e2,S,S1,S2	Objects of class mvp
x	Scalar, length one numeric vector

Details

The function Ops.mvp() passes unary and binary arithmetic operators "+", "-", "*" and "^" to the appropriate specialist function.

The most interesting operator is "*", which is passed to mvp_times_mvp(). I guess "+" is quite interesting too.

Value

The high-level functions documented here return an object of mvp, the low-level functions documented at lowlevel.Rd return lists. But don't use the low-level functions.

Author(s)

Robin K. S. Hankin

See Also

lowlevel

Examples

```
p1 <- rmvp(3)
p2 <- rmvp(3)

p1*p2

p1+p2

p1^3


p1*(p1+p2) == p1^2+p1*p2  # should be TRUE
```

---

print — *Print methods for* mvp *objects*

Description

Print methods for mvp objects: to print, an mvp object is coerced to mpoly form and the mpoly print method used.

Usage

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

Arguments

x	Object of class mvp, coerced to mpoly form
...	Further arguments

Value

Returns its argument invisibly

Author(s)

Robin K. S. Hankin

Examples

```
a <- rmvp(4)
a
print(a)
print(a,stars=TRUE)
print(a,varorder=rev(letters))
```

`rmvp` *Random multivariate polynomials*

Description

Random multivariate polynomials, intended as quick “get you going” examples of `mvp` objects

Usage

```
rmvp(n, size = 6, pow = 6, symbols = 6)
```

Arguments

`n`	Number of terms to generate
`size`	Maximum number of symbols in each term
`pow`	Maximum power of each symbol
`symbols`	Symbols to use; if numeric, interpret as the first `symbols` letters of the alphabet

Details

What you see is what you get, basically. A term such as `a^2*b*a^3` will be simplified to `a^5*b`, so powers in the result may be larger than argument `pow`.

Value

Returns a multivariate polynomial, an object of class `mvp`

Author(s)

Robin K. S. Hankin

Examples

```
rmvp(5)
rmvp(5,symbols=state.abb)
```

series — *Decomposition of multivariate polynomials by powers*

Description

Power series of multivariate polynomials, in various forms

Usage

```
trunc(S,n)
trunc1(S,...)
series(S,v,showsymb=TRUE)
## S3 method for class 'series'
print(x,...)
onevarpow(S,...)
taylor(S,vx,va,debug=FALSE)
mvp_taylor_onevar(allnames,allpowers,coefficients, v, n)
mvp_taylor_allvars(allnames,allpowers,coefficients, n)
mvp_taylor_onepower_onevar(allnames, allpowers, coefficients, v, n)
mvp_to_series(allnames, allpowers, coefficients, v)
```

Arguments

`S`	Object of class `mvp`
`n`	Non-negative integer specifying highest order to be retained
`v`	Variable to take Taylor series with respect to. If missing, total power of each term is used (except for `series()` where it is mandatory)
`x,...`	Object of class `series` and further arguments, passed to the print method; in `trunc1()` a list of variables to truncate
`showsymb`	In function `series()`, Boolean, with default `TRUE` meaning to substitute variables like `x_m_foo` with `(x-foo)` for readability reasons
`vx,va,debug`	In function `taylor()`, names of variables to take series with respect to; and a Boolean with default `FALSE` meaning to return the mvp and `TRUE` meaning to return the string that is passed to `eval()`
`allnames,allpowers,coefficients`	Components of `mvp` objects

Details

Function `onevarpow()` returns just the terms in which symbol `v` appears with power `n`.

Function `series` returns a power series expansion of powers of variable `v`. The print method for `series` objects is sensitive to the value of `getOption("mvp_mult_symbol")`; set this to `"*"` to get `mpoly`-compatible output.

Function `taylor()` is a convenience wrapper for `series()`.

Functions `mvp_taylor_onevar()`, `mvp_taylor_allvars()` and `mvp_to_series()` are low-level helper functions that are not intended for the user.

Author(s)

Robin K. S. Hankin

See Also

deriv

Examples

```
trunc(as.mvp("1+x")^6,2)

trunc(as.mvp("1+x+y")^3,2)      # neglects all terms with total power>2
trunc1(as.mvp("1+x+y")^3,x=2) # terms like y^3 are treated as constants

p <- horner("x+y",1:4)

onevarpow(p,x=2)   # coefficient of x^2
onevarpow(p,x=3)   # coefficient of x^3

onevarpow(as.mvp("1+x+x*y^2  + z*y^2*x"),x=1,y=2)

series(rmvp(10),"a")

# Works well with pipes:

f <- function(n){as.mvp(sub('n',n,'1+x^n*y'))}
Reduce(`*`,lapply(1:6,f)) %>% series('y')
Reduce(`*`,lapply(1:6,f)) %>% series('x')


p %>% trunc(2)
p %>% trunc1(x=2)
(p %>% subs(x="x+dx") -p) %>% trunc1(dx=2)


## Third order taylor expansion of f(x)=sin(x+y) for x=1.1, about x=1:
sinxpy <- horner("x+y",c(0,1,0,-1/6,0,+1/120,0,-1/5040,0,1/362880))  # sin(x+y)
dx <- as.mvp("dx")
t3 <- sinxpy  + aderiv(sinxpy,x=1)*dx + aderiv(sinxpy,x=2)*dx^2/2 + aderiv(sinxpy,x=3)*dx^3/6
t3 %<>% subs(x=1,dx=0.1)  # t3 = Taylor expansion of sin(y+1.1)
t3 %>% subs(y=0.3)  - sin(1.4)  # numeric; should be small
```

`special` *Various functions to create simple multivariate polynomials*

Description

Various functions to create simple `mvp` objects such as single-term, homogenous, and constant multivariate polynomials.

Usage

```
product(v,symbols=letters)
homog(d,power=1,symbols=letters)
linear(x,power=1,symbols=letters)
xyz(n,symbols=letters)
numeric_to_mvp(x)
```

Arguments

`d,n`	An integer; generally, the dimension or arity of the resulting `mvp` object
`v,power`	Integer vector of powers
`x`	Numeric vector of coefficients
`symbols`	Character vector for the symbols

Value

All functions documented here return a `mvp` object

Note

The functions here are related to their equivalents in the multipol and spray packages, but are not exactly the same.

Function `constant()` is documented at `constant.Rd`, but is listed below for convenience.

Author(s)

Robin K. S. Hankin

See Also

`constant`, `zero`

Examples

```
product(1:3)        #      a * b^2 * c^3
homog(3)            #      a + b + c
homog(3,2)          #      a^2  + a b + a c + b^2 + b c + c^2
linear(1:3)         #      1*a + 2*b + 3*c
constant(5)         #      5
xyz(5)              #      a*b*c*d*e
```

spray *Spray functionality*

Description

Convert between spray objects and mvp objects

Usage

```
spray_to_mvp(L, symbols = letters)
mvp_to_spray(S)
```

Arguments

L	Object of class mvp
symbols	character vector of symbols
S	Spray object

Author(s)

Robin K. S. Hankin

Examples

```
mvp_to_spray(rmvp(5))
spray_to_mvp(spray::spray(diag(6),1:6))
```

subs *Substitution*

Description

Substitute symbols in an mvp object for numbers or other multivariate polynomials

Usage

```
subs(S, ..., lose = TRUE)
subsy(S, ..., lose = TRUE)
subvec(S, ...)
subsmvp(S,v,X)
varchange(S,...)
varchange_formal(S,old,new)
namechanger(x,old,new)
```

Arguments

`S,X`	Multivariate polynomials
`...`	named arguments corresponding to variables to substitute
`lose`	Boolean with default `TRUE` meaning to return a scalar (the constant) in place of a constant `mvp` object
`v`	A string corresponding to the variable to substitute
`old,new,x`	The old and new variable names respectively; `x` is a character vector

Details

Function `subs()` substitutes variables for `mvp` objects, using a natural R idiom. Observe that this type of substitution is sensitive to order:

```
> p <- as.mvp("a b^2")
> subs(p,a="b",b="x")
mvp object algebraically equal to
x^3
> subs(p,b="x",a="b")  # same arguments, different order
mvp object algebraically equal to
b x^2
```

Functions `subsy()` and `subsmpv()` are lower-level functions, not really intended for the end-user. Function `subsy()` substitutes variables for numeric values (order matters if a variable is substitued more than once). Function `subsmpv()` takes a `mvp` object and substitutes another `mvp` object for a specific symbol.

Function `subvec()` substitutes the symbols of `S` with numerical values. It is vectorised in its ellipsis arguments with recycling rules and names behaviour inherited from `cbind()`. However, if the first element of `...` is a matrix, then this is interpreted by rows, with symbol names given by the matrix column names; further arguments are ignored. Unlike `subs()`, this function is generally only useful if all symbols are given a value; unassigned symbols take a value of zero.

Function `varchange()` makes a *formal* variable substitution. It is useful because it can take non-standard variable names such as "`(a-b)`" or "?", and is used in `taylor()`. Function `varchange_formal()` does the same task, but takes two character vectors, `old` and `new`, which might be more convenient than passing named arguments. Remember that non-standard names might need to be quoted; also you might need to escape some characters, see the examples. Function `namechanger()` is a low-level helper function that uses regular expression idiom to substitute variable names.

Value

Returns a multivariate polynomial, object of class `mvp`, or a numeric vector (`subvec()`).

Author(s)

Robin K. S. Hankin

See Also

`lose`

Examples

```
p <- rmvp(6,2,2,letters[1:3])
p
subs(p,a=1)
subs(p,a=1,b=2)

subs(p,a="1+b x^3",b="1-y")
subs(p,a=1,b=2,c=3,lose=FALSE)

do.call(subs,c(list(as.mvp("z")),rep(c(z="C+z^2"),5)))

subvec(p,a=1,b=2,c=1:5)   # supply a named list of vectors

M <- matrix(sample(1:3,26*3,replace=TRUE),ncol=26)
colnames(M) <- letters
rownames(M) <- c("Huey", "Dewie", "Louie")
subvec(kahle(r=3,p=1:3),M)  # supply a matrix

varchange(as.mvp("1+x+xy + x*y"),x="newx") # variable xy unchanged

kahle(5,3,1:3) %>% subs(a="a + delta")

pnew <- varchange(p,a="]")  # nonstandard variable names OK
p111 <- varchange_formal(p,"\\]","a")
```

`summary` *Summary methods for mvp objects*

Description

Summary methods for mvp objects and extraction of typical terms

Usage

```
## S3 method for class 'mvp'
summary(object, ...)
## S3 method for class 'summary.mvp'
print(x, ...)
rtypical(object,n=3)
```

Arguments

`x,object`	Multivariate polynomial, class `mvp`
`n`	In `rtypical()`, number of terms (in addition to the constant) to select
`...`	Further arguments, currently ignored

Details

The summary method prints out a list of interesting facts about an mvp object such as the longest term or highest power. Function rtypical() extracts the constant if present, and a *random* selection of terms of its argument.

Author(s)

Robin K. S. Hankin

Examples

```
summary(rmvp(40))
rtypical(rmvp(1000))
```

---

zero   *The zero polynomial*

Description

Test for a multivariate polynomial being zero

Usage

```
is.zero(x)
```

Arguments

x	Object of class mvp

Details

Function is.zero() returns TRUE if x is indeed the zero polynomial. It is defined as length(vars(x))==0 for reasons of efficiency, but conceptually it returns x==constant(0).

(Use constant(0) to create the zero polynomial).

Note

I would have expected the zero polynomial to be problematic (cf the **freegroup** and **permutations** packages, where similar issues require extensive special case treatment). But it seems to work fine, which is a testament to the robust coding in the STL.

A general mvp object is something like

{{"x" -> 3,"y" -> 5} -> 6,{"x" -> 1,"z" -> 8} -> -7}}

which would be $6x^3y^5 - 7xz^8$. The zero polynomial is just {}. Neat, eh?

Author(s)

Robin K. S. Hankin

See Also

constant

Examples

```
constant(0)

t1 <- as.mvp("x+y")
t2 <- as.mvp("x-y")

stopifnot(is.zero(t1*t2-as.mvp("x^2-y^2")))
```

Index