Package ‘mpoly’

February 20, 2020

Type Package

Title Symbolic Computation and More with Multivariate Polynomials

Version 1.1.1

URL https://github.com/dkahle/mpoly

BugReports https://github.com/dkahle/mpoly/issues

Description Symbolic computing with multivariate polynomials in R.

Imports stringr (>= 1.0.0), stringi, partitions, plyr, stats, ggplot2, polynom, orthopolynom, tidyr

Suggests testthat (>= 2.1.0), magrittr, dplyr, covr

License GPL-2

RoxygenNote 7.0.2

Encoding UTF-8

NeedsCompilation no

Author David Kahle [aut, cre] (<https://orcid.org/0000-0002-9999-1558>)

Maintainer David Kahle <david@kahle.io>

Repository CRAN

Date/Publication 2020-02-20 07:00:08 UTC

R topics documented:

as.function.mpoly 2
as.function.mpolyList 4
as.mpoly 6
bernstein 8
bernstein-approx 9
bezier 11
bezier_function 14
burst 16
chebyshev 17
components 19

deriv.mpoly . . . 20
eq_mp . . . 21
gradient . . . 22
hermite . . . 23
homogenize . . . 25
insert . . . 26
is.wholenumber . . . 27
jacobi . . . 28
laguerre . . . 29
LCM . . . 31
legendre . . . 32
lissajous . . . 34
mp . . . 35
mpoly . . . 36
mpoly-defunct . . . 37
mpoly-equal . . . 37
mpolyArithmetic . . . 38
mpolyList . . . 39
mpolyListArithmetic . . . 40
partitions . . . 41
permutations . . . 42
plug . . . 43
predicates . . . 44
print.mpoly . . . 45
print.mpolyList . . . 46
reorder.mpoly . . . 47
round.mpoly . . . 48
solve_unipoly . . . 49
swap . . . 50
terms.mpoly . . . 51
tuples . . . 52
vars . . . 53

Index **54**

`as.function.mpoly` *Change a multivariate polynomial into a function.*

Description

Transforms an mpoly object into a function which can be evaluated.

Usage

```
## S3 method for class 'mpoly'
as.function(x, varorder = vars(x), vector = TRUE,
  silent = FALSE, ..., plus_pad = 1L, times_pad = 1L, squeeze = TRUE)
```

Arguments

x	an object of class mpoly
varorder	the order of the variables
vector	whether the function should take a vector argument (TRUE) or a series of arguments (FALSE)
silent	logical; if TRUE, suppresses output
...	any additional arguments
plus_pad	number of spaces to the left and right of plus sign
times_pad	number of spaces to the left and right of times sign
squeeze	minify code in the created function

See Also

plug(), as.function.mpolyList()

Examples

```
p <- mp("(x - 1)^2")
(f <- as.function(p))
f(1)
f(seq(0, 2, .1))

p <- mp("x + 3 x y + z^2 x")
(f <- as.function(p))
f(1:3) # -> 16
f(c(1,1,1)) # -> 5

f <- as.function(p, vector = FALSE)
f(1, 2, 3) # -> 16
f(1, 1, 1) # -> 5

f <- as.function(p, varorder = c("z","y","x"), vector = FALSE)
f(3, 2, 1) # -> 16
f(1, 1, 1) # -> 5

# for univariate mpolys, as.function() returns a vectorized function
# that can even apply to arrays
p <- mp("x^2")
f <- as.function(p)
f(1:10)
(mat <- matrix(1:4, 2))
f(mat)


p <- mp("1 2 3 4")
f <- as.function(p)
f(10) # -> 24
```

```
bernstein(1, 2)
s <- seq(0, 1, .01)
as.function(bernstein(1, 2))(s)
plot(
  s,
  as.function(bernstein(1, 2))(s)
)


as.function(mp("x + xx"))
as.function(mp("x + xx"), squeeze = FALSE)
```

`as.function.mpolyList` *Change a vector of multivariate polynomials into a function.*

Description

Transforms an mpolyList object into a function which can be evaluated.

Usage

```
## S3 method for class 'mpolyList'
as.function(x, varorder = vars(x), vector = TRUE,
  silent = FALSE, ..., plus_pad = 1L, times_pad = 1L, squeeze = TRUE)

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

Arguments

`x`	an object of class mpoly
`varorder`	the order of the variables
`vector`	whether the function should take a vector argument (TRUE) or a series of arguments (FALSE)
`silent`	logical; if TRUE, suppresses output
`...`	any additional arguments
`plus_pad`	number of spaces to the left and right of plus sign
`times_pad`	number of spaces to the left and right of times sign
`squeeze`	minify code in the created function

See Also

`plug()`, `as.function.mpolyList()`

Examples

```
# basic examples
(mpolyList <- mp(c("2 x + 1", "x - z^2")))
(f <- as.function(mpolyList))
f(c(1,2)) # -> (2*1 + 1, 1-2^2) = 3 -3

f <- as.function(mpolyList, varorder = c("x","y","z"))
f(c(1,0,2)) # -> 3 -3
f(c(1,4,2)) # -> 3 -3

f <- as.function(mpolyList, varorder = c("x","y","z"), vector = FALSE)
f(1, 0, 2) # -> 3 -3
f(1, 4, 2) # -> 3 -3



# making a gradient function (useful for optim)
mpoly <- mp("x + y^2 + y z")
mpolyList <- gradient(mpoly)
f <- as.function(mpolyList, varorder = vars(mpoly))
f(c(0,2,3)) # -> 1 7 2



# a univariate mpolyList creates a vectorized function
ps <- mp(c("x", "x^2", "x^3"))
f <- as.function(ps)
f
s <- seq(-1, 1, length.out = 11)
f(s)

# another example
ps <- chebyshev(1:3)
f <- as.function(ps)
f(s)

# the binomial pmf as an algebraic (polynomial) map
# from [0,1] to [0,1]^size
# p |-> {choose(size, x) p^x (1-p)^(size-x)}_{x = 0, ..., size}
abinom <- function(size, indet = "p"){
  chars4mp <- vapply(as.list(0:size), function(x){
    sprintf("%d %s^%d (1-%s)^%d", choose(size, x), indet, x, indet, size-x)
  }, character(1))
  mp(chars4mp)
}
(ps <- abinom(2, "p")) # = mp(c("(1-p)^2", "2 p (1-p)", "p^2"))
f <- as.function(ps)

f(.5) # P[X = 0], P[X = 1], and P[X = 2] for X ~ Bin(2, .5)
dbinom(0:2, 2, .5)
```

```
f(.75) # P[X = 0], P[X = 1], and P[X = 2] for X ~ Bin(2, .75)
dbinom(0:2, 2, .75)

# as the degree gets larger, you'll need to be careful when evaluating
# the polynomial.  as.function() is not currently optimized for
# stable numerical evaluation of polynomials; it evaluates them in
# the naive way
all.equal(
  as.function(abinom(10))(.5),
  dbinom(0:10, 10, .5)
)

all.equal(
  as.function(abinom(30))(.5),
  dbinom(0:30, 20, .5)
)


# the function produced is vectorized:
number_of_probs <- 11
probs <- seq(0, 1, length.out = number_of_probs)
(mat <- f(probs))
colnames(mat) <- sprintf("P[X = %d]", 0:2)
rownames(mat) <- sprintf("p = %.2f", s)
mat
```

`as.mpoly` *Convert an object to an mpoly*

Description

mpoly is the most basic function used to create objects of class mpoly.

Usage

```
as.mpoly(x, ...)
```

Arguments

`x`	an object
`...`	additional arguments to pass to methods

Value

the object formated as a mpoly object.

Author(s)

David Kahle `<david@kahle.io>`

See Also

mp()

Examples

```
library(ggplot2); theme_set(theme_classic())
library(dplyr)

n <- 101
s <- seq(-5, 5, length.out = n)



# one dimensional case

df <- data.frame(x = seq(-5, 5, length.out = n)) %>%
  mutate(y = -x^2 + 2*x - 3 + rnorm(n, 0, 2))

(mod <- lm(y ~ x + I(x^2), data = df))
(p <- as.mpoly(mod))
qplot(x, y, data = df) +
  stat_function(fun = as.function(p), colour = "red", size = 1)

(mod <- lm(y ~ poly(x, 2, raw = TRUE), data = df))
(p <- as.mpoly(mod))
qplot(x, y, data = df) +
  stat_function(fun = as.function(p), colour = "red", size = 1)

(mod <- lm(y ~ poly(x, 1, raw = TRUE), data = df))
(p <- as.mpoly(mod))
qplot(x, y, data = df) +
  stat_function(fun = as.function(p), colour = "red", size = 1)



# two dimensional case with ggplot2

df <- expand.grid(x = s, y = s) %>%
  mutate(z = x^2 - y^2 + 3*x*y + rnorm(n^2, 0, 3))
qplot(x, y, data = df, geom = "raster", fill = z)

(mod <- lm(z ~ x + y + I(x^2) + I(y^2) + I(x*y), data = df))
(mod <- lm(z ~ poly(x, y, degree = 2, raw = TRUE), data = df))
(p <- as.mpoly(mod))
df$fit <- apply(df[,c("x","y")], 1, as.function(p))

qplot(x, y, data = df, geom = "raster", fill = fit)

qplot(x, y, data = df, geom = "raster", fill = z - fit) # residuals
```

`bernstein` *Bernstein polynomials*

Description

Bernstein polynomials

Usage

```
bernstein(k, n, indeterminate = "x")
```

Arguments

`k`	Bernstein polynomial k
`n`	Bernstein polynomial degree
`indeterminate`	indeterminate

Value

a mpoly object

Author(s)

David Kahle

Examples

```
bernstein(0, 0)

bernstein(0, 1)
bernstein(1, 1)

bernstein(0, 1, "t")

bernstein(0:2, 2)
bernstein(0:3, 3)
bernstein(0:3, 3, "t")


bernstein(0:4, 4)
bernstein(0:10, 10)
bernstein(0:10, 10, "t")
```

```
bernstein(0:20, 20, "t")

## Not run:   # visualize the bernstein polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(0, 1, length.out = 101)
N <- 10 # number of bernstein polynomials to plot
(bernPolys <- bernstein(0:N, N))

df <- data.frame(s, as.function(bernPolys)(s))
names(df) <- c("x", paste0("B_", 0:N))
head(df)

mdf <- gather(df, degree, value, -x)
head(mdf)

qplot(x, value, data = mdf, geom = "line", color = degree)


## End(Not run)
```

`bernstein-approx` *Bernstein polynomial approximation*

Description

Bernstein polynomial approximation

Usage

```
bernstein_approx(f, n, lower = 0, upper = 1, indeterminate = "x")

bernsteinApprox(...)
```

Arguments

`f`	the function to approximate
`n`	Bernstein polynomial degree
`lower`	lower bound for approximation
`upper`	upper bound for approximation
`indeterminate`	indeterminate
`...`	...

Value

a mpoly object

Author(s)

David Kahle

Examples

```
## Not run:   # visualize the bernstein polynomials

library(ggplot2); theme_set(theme_bw())
library(reshape2)




f <- function(x) sin(2*pi*x)
p <- bernstein_approx(f, 20)
round(p, 3)

x <- seq(0, 1, length.out = 101)
df <- data.frame(
  x = rep(x, 2),
  y = c(f(x), as.function(p)(x)),
  which = rep(c("actual", "approx"), each = 101)
)
qplot(x, y, data = df, geom = "line", color = which)





p <- bernstein_approx(sin, 20, pi/2, 1.5*pi)
round(p, 4)

x <- seq(0, 2*pi, length.out = 101)
df <- data.frame(
  x = rep(x, 2),
  y = c(sin(x), as.function(p)(x)),
  which = rep(c("actual", "approx"), each = 101)
)
qplot(x, y, data = df, geom = "line", color = which)
```

```
p <- bernstein_approx(dnorm, 15, -1.25, 1.25)
round(p, 4)

x <- seq(-3, 3, length.out = 101)
df <- data.frame(
  x = rep(x, 2),
  y = c(dnorm(x), as.function(p)(x)),
  which = rep(c("actual", "approx"), each = 101)
)
qplot(x, y, data = df, geom = "line", color = which)

## End(Not run)
```

bezier	*Bezier polynomials*

Description

Compute the Bezier polynomials of a given collection of points. Note that using `mpoly::as.function.mpoly()` on the resulting Bezier polynomials is made numerically stable by taking advantage of de Casteljau's algorithm; it does not use the polynomial that is printed to the screen. See `bezier_function()` for details.

Usage

```
bezier(..., indeterminate = "t")
```

Arguments

`...`	either a sequence of points or a matrix/data frame of points, see examples
`indeterminate`	the indeterminate of the resulting polynomial

Value

a mpoly object

Author(s)

David Kahle

See Also

bezier_function()

Examples

```
p1 <- c(0,  0)
p2 <- c(1,  1)
p3 <- c(2, -1)
p4 <- c(3,  0)
bezier(p1, p2, p3, p4)


points <- data.frame(x = 0:3, y = c(0,1,-1,0))
bezier(points)


points <- data.frame(x = 0:2, y = c(0,1,0))
bezier(points)








# visualize the bernstein polynomials

library(ggplot2); theme_set(theme_bw())

s <- seq(0, 1, length.out = 101)



## example 1
points <- data.frame(x = 0:3, y = c(0,1,-1,0))
(bezPolys <- bezier(points))

f <- as.function(bezPolys)
df <- as.data.frame(f(s))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red") +
  geom_path(data = points, color = "red") +
  geom_path()
```

```
## example 1 with weights
f <- as.function(bezPolys, weights = c(1,5,5,1))
df <- as.data.frame(f(s))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red") +
  geom_path(data = points, color = "red") +
  geom_path()




## example 2
points <- data.frame(x = 0:2, y = c(0,1,0))
(bezPolys <- bezier(points))
f <- as.function(bezPolys)
df <- as.data.frame(f(s))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red") +
  geom_path(data = points, color = "red") +
  geom_path()




## example 3
points <- data.frame(x = c(-1,-2,2,1), y = c(0,1,1,0))
(bezPolys <- bezier(points))
f <- as.function(bezPolys)
df <- as.data.frame(f(s))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red") +
  geom_path(data = points, color = "red") +
  geom_path()




## example 4
points <- data.frame(x = c(-1,2,-2,1), y = c(0,1,1,0))
(bezPolys <- bezier(points))
f <- as.function(bezPolys)
df <- as.data.frame(f(s))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red") +
  geom_path(data = points, color = "red") +
  geom_path()
```

```
## example 5
qplot(speed, dist, data = cars)

s <- seq(0, 1, length.out = 201)
p <- bezier(cars)
f <- as.function(p)
df <- as.data.frame(f(s))
qplot(speed, dist, data = cars) +
  geom_path(data = df, color = "red")

# the curve is not invariant to permutations of the points
# but it always goes through the first and last points
permute_rows <- function(df) df[sample(nrow(df)),]
p <- bezier(permute_rows(cars))
f <- as.function(p)
df <- as.data.frame(f(s))
qplot(speed, dist, data = cars) +
  geom_path(data = df, color = "red")
```

`bezier_function` *Bezier function*

Description

Compute the Bezier function of a collection of polynomials. By Bezier function we mean the Bezier curve function, a parametric map running from t = 0, the first point, to t = 1, the last point, where the coordinate mappings are linear combinations of Bernstein polynomials.

Usage

```
bezier_function(points, weights = rep(1L, nrow(points)))

bezierFunction(...)
```

Arguments

`points`	a matrix or data frame of numerics. the rows represent points.
`weights`	the weights in a weighted Bezier curve
`...`	...; used internally

Details

The function returned is vectorized and evaluates the Bezier curve in a numerically stable way with de Castlejau's algorithm (implemented in R).

Value

function of a single parameter

Author(s)

David Kahle

References

http://en.wikipedia.org/wiki/Bezier_curve, http://en.wikipedia.org/wiki/De_Casteljau's_algorithm

See Also

bezier()

Examples

```
library(ggplot2); theme_set(theme_bw())


t <- seq(0, 1, length.out = 201)
points <- data.frame(x = 0:3, y = c(0,1,-1,0))


f <- bezier_function(points)
df <- as.data.frame(f(t))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red", size = 8) +
  geom_path(data = points, color = "red") +
  geom_path()




f <- bezier_function(points, weights = c(1,5,5,1))
df <- as.data.frame(f(t))

ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red", size = 8) +
  geom_path(data = points, color = "red") +
  geom_path()




f <- bezier_function(points, weights = c(1,10,10,1))
df <- as.data.frame(f(t))

ggplot(aes(x = x, y = y), data = df) +
```

```
  geom_point(data = points, color = "red", size = 8) +
  geom_path(data = points, color = "red") +
  geom_path()
```

burst *Enumerate integer r-vectors summing to n*

Description

Determine all r-vectors with nonnegative integer entries summing to n. Note that this is not intended to be optimized.

Usage

```
burst(n, r = n)
```

Arguments

n	integer to sum to
r	number of components

Value

a matrix whose rows are the n-tuples

Examples

```
burst(4)

burst(4, 4)
burst(4, 3)
burst(4, 2)

rowSums(burst(4))
rowSums(burst(4, 3))
rowSums(burst(4, 2))


burst(10, 4) # all possible 2x2 contingency tables with n=10
burst(10, 4) / 10 # all possible empirical relative frequencies
```

chebyshev *Chebyshev polynomials*

Description

Chebyshev polynomials as computed by orthopolynom.

Usage

```
chebyshev(degree, kind = "t", indeterminate = "x", normalized = FALSE)

chebyshev_roots(k, n)
```

Arguments

`degree`	degree of polynomial
`kind`	`"t"` or `"u"` (Chebyshev polynomials of the first and second kinds), or `"c"` or `"s"`
`indeterminate`	indeterminate
`normalized`	provide normalized coefficients
`k, n`	the k'th root of the n'th chebyshev polynomial

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

`orthopolynom::chebyshev.t.polynomials()`, `orthopolynom::chebyshev.u.polynomials()`, `orthopolynom::chebyshev.c.polynomials()`, `orthopolynom::chebyshev.s.polynomials()`, `http://en.wikipedia.org/wiki/Chebyshev_polynomials`

Examples

```
chebyshev(0)
chebyshev(1)
chebyshev(2)
chebyshev(3)
chebyshev(4)
chebyshev(5)
chebyshev(6)
chebyshev(10)
```

```
chebyshev(0:5)
chebyshev(0:5, normalized = TRUE)
chebyshev(0:5, kind = "u")
chebyshev(0:5, kind = "c")
chebyshev(0:5, kind = "s")
chebyshev(0:5, indeterminate = "t")



# visualize the chebyshev polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(-1, 1, length.out = 201)
N <- 5 # number of chebyshev polynomials to plot
(cheb_polys <- chebyshev(0:N))

# see ?bernstein for a better understanding of
# how the code below works

df <- data.frame(s, as.function(cheb_polys)(s))
names(df) <- c("x", paste0("T_", 0:N))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)



# roots of chebyshev polynomials
N <- 5
cheb_roots <- chebyshev_roots(1:N, N)
cheb_fun <- as.function(chebyshev(N))
cheb_fun(cheb_roots)



# chebyshev polynomials are orthogonal in two ways:
T2 <- as.function(chebyshev(2))
T3 <- as.function(chebyshev(3))
T4 <- as.function(chebyshev(4))

w <- function(x) 1 / sqrt(1 - x^2)
integrate(function(x) T2(x) * T3(x) * w(x), lower = -1, upper = 1)
integrate(function(x) T2(x) * T4(x) * w(x), lower = -1, upper = 1)
integrate(function(x) T3(x) * T4(x) * w(x), lower = -1, upper = 1)

(cheb_roots <- chebyshev_roots(1:4, 4))
sum(T2(cheb_roots) * T3(cheb_roots) * w(cheb_roots))
sum(T2(cheb_roots) * T4(cheb_roots) * w(cheb_roots))
sum(T3(cheb_roots) * T4(cheb_roots) * w(cheb_roots))

sum(T2(cheb_roots) * T3(cheb_roots))
sum(T2(cheb_roots) * T4(cheb_roots))
```

```
sum(T3(cheb_roots) * T4(cheb_roots))
```

---

components *Polynomial components*

Description

Compute quantities/expressions related to a multivariate polynomial.

Usage

```
## S3 method for class 'mpoly'
x[ndx]

LT(x, varorder = vars(x), order = "lex")

LC(x, varorder = vars(x), order = "lex")

LM(x, varorder = vars(x), order = "lex")

multideg(x, varorder = vars(x), order = "lex")

totaldeg(x)

monomials(x)

exponents(x, reduced = FALSE)
```

Arguments

x	an object of class mpoly
ndx	a subsetting index
varorder	the order of the variables
order	a total order used to order the terms
reduced	if TRUE, don't include zero degrees

Value

An object of class mpoly or mpolyList, depending on the context

Examples

```
(p <- mp("x y^2 + x (x+1) (x+2) x z + 3 x^10"))
p[2]
p[-2]
p[2:3]

LT(p)
LC(p)
LM(p)

multideg(p)
totaldeg(p)
monomials(p)

exponents(p)
exponents(p, reduce = TRUE)
lapply(exponents(p), is.integer)

homogeneous_components(p)
```

`deriv.mpoly` *Compute partial derivatives of a multivariate polynomial.*

Description

This is a deriv method for mpoly objects. It does not call the `stats::deriv()`.

Usage

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

Arguments

`expr`	an object of class mpoly
`var`	character - the partial derivative desired
`...`	any additional arguments

Value

An object of class mpoly or mpolyList.

Examples

```
m <- mp("x y + y z + z^2")
deriv(m, "x")
deriv(m, "y")
deriv(m, "z")
deriv(m, c("x","y","z"))
deriv(m, "a")
is.mpoly(deriv(m, "x"))
is.mpolyList( deriv(m, c("x","y","z")) )
```

`eq_mp` *Convert an equation to a polynomial*

Description

Convert characters of the form `"p1 = p2"` (or similar) to an mpoly object representing `p1 -p2`.

Usage

```
eq_mp(string, ...)
```

Arguments

`string`	a character string containing a polynomial, see examples
`...`	arguments to pass to `mpoly()`

Value

An object of class mpoly or mpolyList.

Author(s)

David Kahle `<david@kahle.io>`

See Also

`mpoly()`

Examples

```
eq_mp(c("y = x", "y ==  (x + 2)"))
```

gradient *Compute gradient of a multivariate polynomial.*

Description

This is a wrapper for deriv.mpoly.

Usage

```
gradient(mpoly)
```

Arguments

mpoly	an object of class mpoly

Value

An object of class mpoly or mpolyList.

See Also

deriv.mpoly()

Examples

```
m <- mp("x y + y z + z^2")
gradient(m)


# gradient descent illustration using the symbolically
# computed gradient of the rosenbrock function (shifted)
rosenbrock <- mp("(1 - x)^2 + 100 (y - x^2)^2")
fn <- as.function(rosenbrock)
(rosenbrock_gradient <- gradient(rosenbrock))
gr <- as.function(rosenbrock_gradient)

# visualize the function
library(ggplot2)
s <- seq(-5, 5, .05)
df <- expand.grid(x = s, y = s)
df$z <- apply(df, 1, fn)
ggplot(df, aes(x = x, y = y)) +
  geom_raster(aes(fill = z + 1e-10)) +
  scale_fill_continuous(trans = "log10")

# run the gradient descent algorithm using line-search
# step sizes computed with optimize()
current <- steps <- c(-3,-4)
change <- 1
tol <- 1e-5
```

```
while(change > tol){
  last  <- current
  delta <- optimize(
    function(delta) fn(current - delta*gr(current)),
    interval = c(1e-10, .1)
  )$minimum
  current <- current - delta*gr(current)
  steps   <- unname(rbind(steps, current))
  change  <- abs(fn(current) - fn(last))
}
steps <- as.data.frame(steps)
names(steps) <- c("x", "y")

# visualize steps, note the optim at c(1,1)
# the routine took 5748 steps
ggplot(df, aes(x = x, y = y)) +
  geom_raster(aes(fill = z + 1e-10)) +
  geom_path(data = steps, color = "red") +
  geom_point(data = steps, color = "red", size = .5) +
  scale_fill_continuous(trans = "log10")

# it gets to the general region of space quickly
# but once it gets on the right arc, it's terrible
# here's what the end game looks like
last_steps <- tail(steps, 100)
rngx <- range(last_steps$x); sx <- seq(rngx[1], rngx[2], length.out = 201)
rngy <- range(last_steps$y); sy <- seq(rngy[1], rngy[2], length.out = 201)
df <- expand.grid(x = sx, y = sy)
df$z <- apply(df, 1, fn)
ggplot(df, aes(x = x, y = y)) +
  geom_raster(aes(fill = z)) +
  geom_path(data = last_steps, color = "red", size = .25) +
  geom_point(data = last_steps, color = "red", size = 1) +
  scale_fill_continuous(trans = "log10")
```

hermite *Hermite polynomials*

Description

Hermite polynomials as computed by orthopolynom.

Usage

```
hermite(degree, kind = "he", indeterminate = "x", normalized = FALSE)
```

Arguments

degree	degree of polynomial
kind	"he" (default, probabilists', see Wikipedia article) or "h" (physicists)
indeterminate	indeterminate
normalized	provide normalized coefficients

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

orthopolynom::hermite.h.polynomials(), orthopolynom::hermite.he.polynomials(), http://en.wikipedia.org/wiki/Hermite_polynomials

Examples

```
hermite(0)
hermite(1)
hermite(2)
hermite(3)
hermite(4)
hermite(5)
hermite(6)
hermite(10)

hermite(0:5)
hermite(0:5, normalized = TRUE)
hermite(0:5, indeterminate = "t")



# visualize the hermite polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(-3, 3, length.out = 201)
N <- 5 # number of hermite polynomials to plot
(hermPolys <- hermite(0:N))

# see ?bernstein for a better understanding of
# how the code below works

df <- data.frame(s, as.function(hermPolys)(s))
```

```
names(df) <- c("x", paste0("T_", 0:N))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)


# hermite polynomials are orthogonal with respect to the gaussian kernel:
He2 <- as.function(hermite(2))
He3 <- as.function(hermite(3))
He4 <- as.function(hermite(4))

w <- dnorm
integrate(function(x) He2(x) * He3(x) * w(x), lower = -Inf, upper = Inf)
integrate(function(x) He2(x) * He4(x) * w(x), lower = -Inf, upper = Inf)
integrate(function(x) He3(x) * He4(x) * w(x), lower = -Inf, upper = Inf)
```

`homogenize` *Homogenize a polynomial*

Description

Homogenize a polynomial.

Usage

```
homogenize(x, indeterminate = "t")

dehomogenize(x, indeterminate = "t")

is.homogeneous(x)

homogeneous_components(x)
```

Arguments

`x`	an mpoly object, see `mpoly()`
`indeterminate`	name of homogenization

Value

a (de/homogenized) mpoly or an mpolyList

Examples

```
x <- mp("x^4 + y + 2 x y^2 - 3 z")
is.homogeneous(x)
(xh <- homogenize(x))
is.homogeneous(xh)

homogeneous_components(x)

homogenize(x, "o")

xh <- homogenize(x)
dehomogenize(xh) # assumes indeterminate = "t"
plug(xh, "t", 1) # same effect, but dehomogenize is faster



# the functions are vectorized
(ps <- mp(c("x + y^2", "x + y^3")))
(psh <- homogenize(ps))
dehomogenize(psh)


# demonstrating a leading property of homogeneous polynomials
library(magrittr)
p  <- mp("x^2 + 2 x + 3")
(ph <- homogenize(p, "y"))
lambda <- 3
(d <- totaldeg(p))
ph %>%
  plug("x", lambda*mp("x")) %>%
  plug("y", lambda*mp("y"))
lambda^d * ph
```

`insert` *Insert an element into a vector.*

Description

Insert an element into a vector.

Usage

```
insert(elem, slot, v)
```

Arguments

`elem`	element to insert

`slot`	location of insert
`v`	vector to insert into

Value

vector with element inserted

Examples

```
insert(2, 1, 1)
insert(2, 2, 1)
insert('x', 5, letters)
```

`is.wholenumber` *Test whether an object is a whole number*

Description

Test whether an object is a whole number.

Usage

```
is.wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

`x`	object to be tested
`tol`	tolerance within which a number is said to be whole

Value

Vector of logicals.

Examples

```
is.wholenumber(seq(-3,3,.5))
```

`jacobi` *Jacobi polynomials*

Description

Jacobi polynomials as computed by orthopolynom.

Usage

```
jacobi(
  degree,
  alpha = 1,
  beta = 1,
  kind = "p",
  indeterminate = "x",
  normalized = FALSE
)
```

Arguments

`degree`	degree of polynomial
`alpha`	the first parameter, also called p
`beta`	the second parameter, also called q
`kind`	"g" or "p"
`indeterminate`	indeterminate
`normalized`	provide normalized coefficients

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

`orthopolynom::jacobi.g.polynomials()`, `orthopolynom::jacobi.p.polynomials()`, `http://en.wikipedia.org/wiki/Jacobi_polynomials`

Examples

```
jacobi(0)
jacobi(1)
jacobi(2)
jacobi(3)
jacobi(4)
jacobi(5)
jacobi(6)
jacobi(10, 2, 2, normalized = TRUE)

jacobi(0:5)
jacobi(0:5, normalized = TRUE)
jacobi(0:5, kind = "g")
jacobi(0:5, indeterminate = "t")



# visualize the jacobi polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(-1, 1, length.out = 201)
N <- 5 # number of jacobi polynomials to plot
(jacPolys <- jacobi(0:N, 2, 2))

df <- data.frame(s, as.function(jacPolys)(s))
names(df) <- c("x", paste0("P_", 0:N))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)

qplot(x, value, data = mdf, geom = "line", color = degree) +
  coord_cartesian(ylim = c(-30, 30))
```

`laguerre` *Generalized Laguerre polynomials*

Description

Generalized Laguerre polynomials as computed by orthopolynom.

Usage

```
laguerre(degree, alpha = 0, indeterminate = "x", normalized = FALSE)
```

Arguments

degree	degree of polynomial
alpha	generalization constant
indeterminate	indeterminate
normalized	provide normalized coefficients

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

orthopolynom::glaguerre.polynomials(), http://en.wikipedia.org/wiki/Laguerre_polynomials

Examples

```
laguerre(0)
laguerre(1)
laguerre(2)
laguerre(3)
laguerre(4)
laguerre(5)
laguerre(6)

laguerre(2)
laguerre(2, normalized = TRUE)

laguerre(0:5)
laguerre(0:5, normalized = TRUE)
laguerre(0:5, indeterminate = "t")



# visualize the laguerre polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(-5, 20, length.out = 201)
N <- 5 # number of laguerre polynomials to plot
(lagPolys <- laguerre(0:N))

# see ?bernstein for a better understanding of
# how the code below works
```

```
df <- data.frame(s, as.function(lagPolys)(s))
names(df) <- c("x", paste0("L_", 0:N))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)

qplot(x, value, data = mdf, geom = "line", color = degree) +
  coord_cartesian(ylim = c(-25, 25))


# laguerre polynomials are orthogonal with respect to the exponential kernel:
L2 <- as.function(laguerre(2))
L3 <- as.function(laguerre(3))
L4 <- as.function(laguerre(4))

w <- dexp
integrate(function(x) L2(x) * L3(x) * w(x), lower = 0, upper = Inf)
integrate(function(x) L2(x) * L4(x) * w(x), lower = 0, upper = Inf)
integrate(function(x) L3(x) * L4(x) * w(x), lower = 0, upper = Inf)
```

LCM *Compute the least common multiple of two numbers.*

Description

A simple algorithm to compute the least common multiple of two numbers

Usage

```
LCM(x, y)
```

Arguments

x	an object of class numeric
y	an object of class numeric

Value

The least common multiple of x and y.

Examples

```
LCM(5,7)
LCM(5,8)
LCM(5,9)
LCM(5,10)
Reduce(LCM, 1:10) # -> 2520
```

`legendre` *Legendre polynomials*

Description

Legendre polynomials as computed by orthopolynom.

Usage

```
legendre(degree, indeterminate = "x", normalized = FALSE)
```

Arguments

`degree`	degree of polynomial
`indeterminate`	indeterminate
`normalized`	provide normalized coefficients

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

`orthopolynom::legendre.polynomials()`, `http://en.wikipedia.org/wiki/Legendre_polynomials`

Examples

```
legendre(0)
legendre(1)
legendre(2)
legendre(3)
legendre(4)
legendre(5)
legendre(6)

legendre(2)
legendre(2, normalized = TRUE)

legendre(0:5)
legendre(0:5, normalized = TRUE)
legendre(0:5, indeterminate = "t")
```

```
# visualize the legendre polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyr)

s <- seq(-1, 1, length.out = 201)
N <- 5 # number of legendre polynomials to plot
(legPolys <- legendre(0:N))

# see ?bernstein for a better understanding of
# how the code below works

df <- data.frame(s, as.function(legPolys)(s))
names(df) <- c("x", paste0("P_", 0:N))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)


# legendre polynomials and the QR decomposition
n <- 201
x <- seq(-1, 1, length.out = n)
mat <- cbind(1, x, x^2, x^3, x^4, x^5)
Q <- qr.Q(qr(mat))
df <- as.data.frame(cbind(x, Q))
names(df) <- c("x", 0:5)
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)

Q <- apply(Q, 2, function(x) x / x[n])
df <- as.data.frame(cbind(x, Q))
names(df) <- c("x", paste0("P_", 0:5))
mdf <- gather(df, degree, value, -x)
qplot(x, value, data = mdf, geom = "line", color = degree)


# chebyshev polynomials are orthogonal in two ways:
P2 <- as.function(legendre(2))
P3 <- as.function(legendre(3))
P4 <- as.function(legendre(4))

integrate(function(x) P2(x) * P3(x), lower = -1, upper = 1)
integrate(function(x) P2(x) * P4(x), lower = -1, upper = 1)
integrate(function(x) P3(x) * P4(x), lower = -1, upper = 1)

n <- 10000L
u <- runif(n, -1, 1)
2 * mean(P2(u) * P3(u))
2 * mean(P2(u) * P4(u))
2 * mean(P3(u) * P4(u))

(2/n) * sum(P2(u) * P3(u))
(2/n) * sum(P2(u) * P4(u))
(2/n) * sum(P3(u) * P4(u))
```

`lissajous` *Lissajous polynomials*

Description

The Lissajous polynomials are the implicit (variety) descriptions of the image of the parametric map x = cos(m t + p), y = sin(n t + q).

Usage

```
lissajous(m, n, p, q, digits = 3)
```

Arguments

`m, n, p, q`	Trigonometric coefficients, see examples for description
`digits`	The number of digits to round coefficients to, see `round.mpoly()`. This is useful for cleaning terms that are numerically nonzero, but should be.

Value

a mpoly object

See Also

`chebyshev()`, Merino, J. C (2003). Lissajous figures and Chebyshev polynomials. The College Mathematics Journal, 34(2), pp. 122-127.

Examples

```
lissajous(3, 2,  -pi/2, 0)
lissajous(4, 3,  -pi/2, 0)
```

mp *Define a multivariate polynomial.*

Description

mp is a smart function which attempts to create a formal mpoly object from a character string containing the usual representation of a multivariate polynomial.

Usage

```
make_indeterminate_list(vars)

mp(string, varorder, stars_only = FALSE)
```

Arguments

`vars`	a character vector of indeterminates
`string`	a character string containing a polynomial, see examples
`varorder`	(optional) order of variables in string
`stars_only`	if you format your multiplications using asterisks, setting this to `TRUE` will reduce preprocessing time

Value

An object of class mpoly.

Author(s)

David Kahle `<david@kahle.io>`

See Also

`mpoly()`

Examples

```
( m <- mp("x + y + x y") )
is.mpoly( m )
unclass(m)


mp("x + 2 y + x^2 y + x y z")
mp("x + 2 y + x^2 y + x y z", varorder = c("y", "z", "x"))

( ms <- mp(c("x + y", "2 x")) )
is.mpolyList(ms)
```

```
gradient( mp("x + 2 y + x^2 y + x y z") )
gradient( mp("(x + y)^10") )

# mp and the print methods are kinds of inverses of each other
( polys <- mp(c("x + y", "x - y")) )
strings <- print(polys, silent = TRUE)
strings
mp(strings)
```

---

mpoly *Multivariate polynomials in R.*

Description

A package for symbolic computation and more with multivariate polynomials

mpoly is the most basic function used to create objects of class mpoly. However, it is not a general purpose function; for that see `mp()`.

Usage

```
mpoly(list, varorder)
```

Arguments

`list`	a list from which to construct an mpoly object
`varorder`	(optional) a character vector setting the intrinsic variable order of the polynomial

Value

Object of class mpoly.

Author(s)

David Kahle `<david@kahle.io>`

See Also

`mp()`

Examples

```
list <- list(
  c(x = 1, coef = 1, y = 0),
  c(x = 0, y = 1, coef = 2),
  c(y = 1, coef = -6),
  c(z = 1, coef = -3, x = 2),
  c(x = 1, coef = 0, x = 3),
  c(t = 1, coef = 4, t = 2, y = 4),
  c(x = 1),
  c(x = 1),
  c(coef = 5),
  c(coef = 5),
  c(coef = -5)
)

mpoly(list) # 3 x  -  4 y  -  3 x^2 z  +  4 y^4 t^3  +  5
mpoly(list, varorder = c("y", "z", "t", "x"))

list <- list(  c(x = 5, x = 2, coef = 5, coef = 6, y = 0) )
mpoly(list)
```

---

mpoly-defunct *Defunct mpoly functions*

Description

This is a list of past functions of the mpoly package.

Details

The following are defunct mpoly functions:

- grobner

---

mpoly-equal *Determine whether two multivariate polynomials are equal.*

Description

Determine whether two multivariate polynomials are equal.

Usage

```
## S3 method for class 'mpoly'
e1 == e2

## S3 method for class 'mpoly'
e1 != e2
```

Arguments

e1	an object of class mpoly
e2	an object of class mpoly

Value

A logical value.

Examples

```
p1 <- mp("x + y + 2 z")
p1 == p1

p2 <- reorder(p1, order = "lex", varorder = c("z","y","x"))
p1 == p2
p2 <- reorder(p1, order = "lex", varorder = c("z","w","y","x"))
p1 == p2
p1 == ( 2 * p2 )

p1 <- mp("x + 1")
p2 <- mp("x + 1")
identical(p1, p2)
p1 == p2

mp("x + 1") == mp("y + 1")
mp("2") == mp("1")
mp("1") == mp("1")
mp("0") == mp("-0")
```

---

mpolyArithmetic *Arithmetic with multivariate polynomials*

Description

Arithmetic with multivariate polynomials

Usage

```
## S3 method for class 'mpoly'
e1 + e2

## S3 method for class 'mpoly'
e1 - e2

## S3 method for class 'mpoly'
e1 * e2

## S3 method for class 'mpoly'
e1 ^ e2

## S3 method for class 'mpolyList'
e1 ^ e2
```

Arguments

e1	an object of class mpoly
e2	an object of class mpoly

Value

object of class mpoly

Examples

```
p <- mp("x + y")
p + p
p - p
p * p
p^2
p^10


mp("(x+1)^10")
p + 1
2*p
```

`mpolyList` *Define a collection of multivariate polynomials.*

Description

Combine a series of mpoly objects into a mpolyList.

Usage

```
mpolyList(...)
```

Arguments

...	a series of mpoly objects.

Value

An object of class mpolyList.

Examples

```
( p1 <- mp("t^4 - x") )
( p2 <- mp("t^3 - y") )
( p3 <- mp("t^2 - z") )
( ms <- mpolyList(p1, p2, p3) )
is.mpolyList( ms )

mpolyList(mp("x + 1"))
p <- mp("x + 1")
mpolyList(p)

ps <- mp(c("x + 1", "y + 2"))
is.mpolyList(ps)


f <- function(){
  a <- mp("1")
  mpolyList(a)
}
f()
```

`mpolyListArithmetic` *Element-wise arithmetic with vectors of multivariate polynomials.*

Description

Element-wise arithmetic with vectors of multivariate polynomials.

Usage

```
## S3 method for class 'mpolyList'
e1 + e2

## S3 method for class 'mpolyList'
e1 - e2

## S3 method for class 'mpolyList'
e1 * e2
```

Arguments

`e1`	an object of class mpolyList
`e2`	an object of class mpolyList

Value

An object of class mpolyList.

Examples

```
( ms1 <- mp( c("x", 'y') ) )
( ms2 <- mp( c("y", '2 x^2') ) )
ms1 + ms2
ms1 - ms2
ms1 * ms2
ms1^3
```

`partitions` *Enumerate the partitions of an integer*

Description

Determine all unrestricted partitions of an integer. This function is equivalent to the function parts in the partitions package.

Usage

```
partitions(n)
```

Arguments

`n`	an integer

Value

a matrix whose rows are the n-tuples

Author(s)

Robin K. S. Hankin, from package partitions

Examples

```
partitions(5)
str(partitions(5))
```

`permutations` *Determine all permutations of a set.*

Description

An implementation of the Steinhaus-Johnson-Trotter permutation algorithm.

Usage

```
permutations(set)
```

Arguments

`set`	a set

Value

a matrix whose rows are the permutations of set

Examples

```
permutations(1:3)
permutations(c('first','second','third'))
permutations(c(1,1,3))
apply(permutations(letters[1:6]), 1, paste, collapse = '')
```

`plug` *Switch indeterminates in a polynomial*

Description

Switch indeterminates in a polynomial

Usage

```
plug(p, indeterminate, value)
```

Arguments

`p`	a polynomial
`indeterminate`	the indeterminate in the polynomial to switch
`value`	the value/indeterminate to substitute

Value

an mpoly object

Examples

```
# on an mpoly
(p <- mp("(x+y)^3"))
plug(p, "x", 5)
plug(p, "x", "t")
plug(p, "x", "y")
plug(p, "x", mp("2 y"))

plug(p, "x", mp("x + y"))
mp("((x+y)+y)^3")

# on an mpolyList
ps <- mp(c("x+y", "x+1"))
plug(ps, "x", 1)
```

predicates *mpoly predicate functions*

Description

Various functions to deal with mpoly and mpolyList objects.

Usage

```
is.constant(x)

is.mpoly(x)

is.unipoly(x)

is.bernstein(x)

is.bezier(x)

is.chebyshev(x)

is.mpolyList(x)

is.linear(x)
```

Arguments

x	object to be tested

Value

Vector of logicals.

Examples

```
p <- mp("5")
is.mpoly(p)
is.constant(p)

is.constant(mp(c("x + 1", "7", "y - 2")))

p <- mp("x + y")
is.mpoly(p)

is.mpolyList(mp(c("x + 1", "y - 1")))
```

```
is.linear(mp("0"))
is.linear(mp("x + 1"))
is.linear(mp("x + y"))
is.linear(mp(c("0", "x + y")))

is.linear(mp("x + x y"))
is.linear(mp(c("x + x y", "x")))


(p <- bernstein(2, 5))
is.mpoly(p)
is.bernstein(p)

(p <- chebyshev(5))
is.mpoly(p)
is.chebyshev(p)
str(p)
```

`print.mpoly` *Pretty printing of multivariate polynomials.*

Description

This is the major function used to view multivariate polynomials.

Usage

```
## S3 method for class 'mpoly'
print(x, varorder, order, stars = FALSE, silent =
  FALSE, ..., plus_pad = 2L, times_pad = 1L)
```

Arguments

`x`	an object of class mpoly
`varorder`	the order of the variables
`order`	a total order used to order the monomials in the printing
`stars`	print the multivariate polynomial in the more computer-friendly asterisk notation (default FALSE)
`silent`	logical; if TRUE, suppresses output
`...`	additional parameters to go to `base::cat()`
`plus_pad`	number of spaces to the left and right of plus sign
`times_pad`	number of spaces to the left and right of times sign

Value

Invisible string of the printed object.

Examples

```
mp("-x^5 - 3 y^2 + x y^3 - 1")


(p <- mp("2 x^5  -  3 y^2  +  x y^3"))
print(p) # same
print(p, silent = TRUE)
s <- print(p, silent = TRUE)
s

print(p, order = "lex") # -> 2 x^5  +  x y^3  -  3 y^2
print(p, order = "lex", varorder = c("y","x")) # -> y^3 x  -  3 y^2  +  2 x^5
print(p, varorder = c("y","x")) # -> y^3 x  -  3 y^2  +  2 x^5

# this is mostly used internally
print(p, stars = TRUE)
print(p, stars = TRUE, times_pad = 0L)
print(p, stars = TRUE, times_pad = 0L, plus_pad = 1L)
print(p, stars = TRUE, times_pad = 0L, plus_pad = 0L)
print(p, plus_pad = 1L)
```

`print.mpolyList` *Pretty printing of a list of multivariate polynomials.*

Description

This function iterates print.mpoly on an object of class mpolyList.

Usage

```
## S3 method for class 'mpolyList'
print(
  x,
  varorder = vars(x),
  stars = FALSE,
  order,
  silent = FALSE,
  ...,
  plus_pad = 2L,
  times_pad = 1L
)
```

Arguments

x	an object of class mpoly
varorder	the order of the variables

stars	print the multivariate polynomial in the more computer-friendly asterisk notation (default FALSE)
order	a total order used to order the monomials in the printing
silent	logical; if TRUE, suppresses output
...	additional parameters to go to base::cat()
plus_pad	number of spaces to the left and right of plus sign
times_pad	number of spaces to the left and right of times sign

Value

Invisible character vector of the printed objects.

Examples

```
mL <- mp(c("x + 1", "y - 1", "x y^2 z  +  x^2 z^2  +  z^2  +  x^3"))
mL
print(mL, order = "lex")
print(mL, order = "glex")
print(mL, order = "grlex")
print(mL, order = "glex", varorder = c("z","y","x"))
print(mL, order = "grlex", varorder = c("z","y","x"))
print(mL, varorder = c("z","y","x"))

(print(mL, varorder = c("z","y","x"), plus_pad = 1L, silent = TRUE))

(print(mL, silent = TRUE, stars = TRUE, plus_pad = 1L, times_pad = 0L))
```

---

reorder.mpoly *Reorder a multivariate polynomial.*

Description

This function is used to set the intrinsic order of a multivariate polynomial. It is used for both the in-term variables and the terms.

Usage

```
## S3 method for class 'mpoly'
reorder(x, varorder = vars(x), order, ...)
```

Arguments

x	an object of class mpoly
varorder	the order of the variables
order	a total order used to order the terms, "lex", "glex", or "grlex"
...	additional arguments

Value

An object of class mpoly.

Examples

```
list <- list(
  c(x = 1, y = 2, z = 1, coef = 1),
  c(x = 2, y = 0, z = 2, coef = 1),
  c(x = 0, y = 0, z = 2, coef = 1),
  c(x = 3, y = 0, z = 0, coef = 1)
)
(p <- mpoly(list)) # -> x y^2 z  +  x^2 z^2  +  z^2  +  x^3
reorder(p) # -> x y^2 z  +  x^2 z^2  +  z^2  +  x^3
reorder(p, varorder = c("x","y","z"), order = "lex")
    # -> x^3  +  x^2 z^2  +  x y^2 z  +  z^2
reorder(p, varorder = c("x","y","z"), order = "glex")
    # -> x^2 z^2  +  x y^2 z  +  x^3  +  z^2
reorder(p, varorder = c("x","y","z"), order = "grlex")
    # -> x y^2 z  +  x^2 z^2  +  x^3  +  z^2

reorder(mp("x + 1"), varorder = c("y","x","z"), order = "lex")
reorder(mp("x + y"), varorder = c("y","x","z"), order = "lex")
reorder(mp("x y + y + 2 x y z^2"), varorder = c("y","x","z"))
reorder(mp("x^2 + y x + y"), order = "lex")
```

---

round.mpoly — *Round the coefficients of a polynomial*

Description

Round the coefficients of an mpoly object.

Usage

```
## S3 method for class 'mpoly'
round(x, digits = 3)
```

Arguments

x	an mpoly object
digits	number of digits to round to

Value

the rounded mpoly object

Author(s)

David Kahle <david@kahle.io>

See Also

mp()

Examples

```
p <- mp("x + 3.14159265")^4
p
round(p)
round(p, 0)

## Not run:
library(plyr)
library(ggplot2)
library(stringr)

n <- 101
s <- seq(-5, 5, length.out = n)

# one dimensional case
df <- data.frame(x = s)
df <- mutate(df, y = -x^2 + 2*x - 3 + rnorm(n, 0, 2))
qplot(x, y, data = df)
mod <- lm(y ~ x + I(x^2), data = df)
p <- as.mpoly(mod)
qplot(x, y, data = df) +
  stat_function(fun = as.function(p), colour = 'red')

p
round(p, 1)
qplot(x, y, data = df) +
  stat_function(fun = as.function(p), colour = 'red') +
  stat_function(fun = as.function(round(p,1)), colour = 'blue')



## End(Not run)
```

---

solve_unipoly *Solve a univariate mpoly with polyroot*

Description

Solve a univariate mpoly with polyroot

Usage

```
solve_unipoly(mpoly, real_only = FALSE)
```

Arguments

`mpoly`	an mpoly
`real_only`	return only real solutions?

Examples

```
solve_unipoly(mp("x^2 - 1")) # check x = -1, 1
solve_unipoly(mp("x^2 - 1"), real_only = TRUE)
```

`swap` *Swap polynomial indeterminates*

Description

Swap polynomial indeterminates

Usage

```
swap(p, variable, replacement)
```

Arguments

`p`	polynomial
`variable`	the variable in the polynomial to replace
`replacement`	the replacement variable

Value

a mpoly object

Author(s)

David Kahle

Examples

```
(p <- mp("(x + y)^2"))
swap(p, "x", "t")

## the meta data is retained
(p <- bernstein(3, 5))
(p2 <- swap(p, "x", "t"))
is.bernstein(p2)

(p <- chebyshev(3))
(p2 <- swap(p, "x", "t"))
is.chebyshev(p2)
```

---

terms.mpoly *Extract the terms of a multivariate polynomial.*

Description

Compute the terms of an mpoly object as a mpolyList.

Usage

```
## S3 method for class 'mpoly'
terms(x, ...)
```

Arguments

x	an object of class mpoly
...	additional parameters

Value

An object of class mpolyList.

Examples

```
## Not run:  .Deprecated issues a warning

x <- mp("x^2 - y + x y z")
terms(x)
monomials(x)


## End(Not run)
```

`tuples` *Determine all n-tuples using the elements of a set.*

Description

Determine all n-tuples using the elements of a set. This is really just a simple wrapper for expand.grid, so it is not optimized.

Usage

```
tuples(set, n = length(set), repeats = FALSE, list = FALSE)
```

Arguments

`set`	a set
`n`	length of each tuple
`repeats`	if set contains duplicates, should the result?
`list`	tuples as list?

Value

a matrix whose rows are the n-tuples

Examples

```
tuples(1:2, 3)
tuples(1:2, 3, list = TRUE)

apply(tuples(c("x","y","z"), 3), 1, paste, collapse = "")

# multinomial coefficients
r <- 2 # number of variables, e.g. x, y
n <- 2 # power, e.g. (x+y)^2
apply(burst(n,r), 1, function(v) factorial(n)/ prod(factorial(v))) # x, y, xy
mp("x + y")^n

r <- 2 # number of variables, e.g. x, y
n <- 3 # power, e.g. (x+y)^3
apply(burst(n,r), 1, function(v) factorial(n)/ prod(factorial(v)))
mp("x + y")^n

r <- 3 # number of variables, e.g. x, y, z
n <- 2 # power, e.g. (x+y+z)^2
apply(burst(n,r), 1, function(v) factorial(n)/ prod(factorial(v))) # x, y, z, xy, xz, yz
mp("x + y + z")^n
```

vars *Determine the variables in a mpoly object.*

Description

Determine the variables in a mpoly object.

Usage

```
vars(p)
```

Arguments

p	An mpoly or mpolyList object.

Value

A character vector of the variable names.

Examples

```
p <- mp("x + y^2")
vars(p)

p <- mp(c("x + y^2", "y - 2 x"))
vars(p)
```

Index

!=.mpoly (mpoly-equal), 37
*.mpoly (mpolyArithmetic), 38
*.mpolyList (mpolyListArithmetic), 40
+.mpoly (mpolyArithmetic), 38
+.mpolyList (mpolyListArithmetic), 40
-.mpoly (mpolyArithmetic), 38
-.mpolyList (mpolyListArithmetic), 40
== (mpoly-equal), 37
[.mpoly (components), 19
^.mpoly (mpolyArithmetic), 38
^.mpolyList (mpolyArithmetic), 38

as.function.bezier (as.function.mpolyList), 4
as.function.mpoly, 2
as.function.mpolyList, 4
as.function.mpolyList(), *3, 4*
as.mpoly, 6

base::cat(), *45*, *47*
bernstein, 8
bernstein-approx, 9
bernstein_approx (bernstein-approx), 9
bernsteinApprox (bernstein-approx), 9
bezier, 11
bezier(), *15*
bezier_function, 14
bezier_function(), *11, 12*
bezierFunction (bezier_function), 14
burst, 16

chebyshev, 17
chebyshev(), *34*
chebyshev_roots (chebyshev), 17
components, 19

dehomogenize (homogenize), 25
deriv.mpoly, 20
deriv.mpoly(), *22*

eq_mp, 21
exponents (components), 19

gradient, 22

hermite, 23
homogeneous_components (homogenize), 25
homogenize, 25

insert, 26
is.bernstein (predicates), 44
is.bezier (predicates), 44
is.chebyshev (predicates), 44
is.constant (predicates), 44
is.homogeneous (homogenize), 25
is.linear (predicates), 44
is.mpoly (predicates), 44
is.mpolyList (predicates), 44
is.unipoly (predicates), 44
is.wholenumber, 27

jacobi, 28

laguerre, 29
LC (components), 19
LCM, 31
legendre, 32
lissajous, 34
LM (components), 19
LT (components), 19

make_indeterminate_list (mp), 35
monomials (components), 19
mp, 35
mp(), *7*, *36*, *49*
mpoly, 36
mpoly(), *21*, *25*, *35*
mpoly-defunct, 37
mpoly-equal, 37
mpoly::as.function.mpoly(), *11*
mpolyArithmetic, 38
mpolyList, 39

mpolyListArithmetic, 40
multideg *(components)*, 19

orthopolynom::chebyshev.c.polynomials(), *17*
orthopolynom::chebyshev.s.polynomials(), *17*
orthopolynom::chebyshev.t.polynomials(), *17*
orthopolynom::chebyshev.u.polynomials(), *17*
orthopolynom::glaguerre.polynomials(), *30*
orthopolynom::hermite.h.polynomials(), *24*
orthopolynom::hermite.he.polynomials(), *24*
orthopolynom::jacobi.g.polynomials(), *28*
orthopolynom::jacobi.p.polynomials(), *28*
orthopolynom::legendre.polynomials(), *32*

package-mpoly *(mpoly)*, 36
partitions, 41
permutations, 42
plug, 43
plug(), *3, 4*
predicates, 44
print.mpoly, 45
print.mpolyList, 46

reorder.mpoly, 47
round.mpoly, 48
round.mpoly(), *34*

solve_unipoly, 49
stats::deriv(), *20*
swap, 50

terms.mpoly, 51
totaldeg *(components)*, 19
tuples, 52

vars, 53