Package ‘mpoly’

June 21, 2017

Type Package
Title Symbolic Computation and More with Multivariate Polynomials
Version 1.0.5
URL https://github.com/dkahle/mpoly
BugReports https://github.com/dkahle/mpoly/issues
Description Symbolic computing with multivariate polynomials in R.
Depends stringr (>= 1.0.0)
Imports partitions, plyr, stats, ggplot2, polynom, orthopolynom, tidyr
Suggests testthat, magrittr, dplyr
License GPL-2
RoxygenNote 6.0.1
NeedsCompilation no
Author David Kahle [aut, cre]
Maintainer David Kahle <david.kahle@gmail.com>
Repository CRAN
Date/Publication 2017-06-21 17:18:13 UTC

R topics documented:

as.function.mpoly .................................................. 2
as.function.mpolyList ........................................... 3
as.mpoly ............................................................... 5
bernstein ............................................................. 7
bernsteinApprox ..................................................... 8
bezier ................................................................. 10
bezierFunction ....................................................... 13
burst ................................................................. 15
chebyshev ........................................................... 16
components ......................................................... 17
deriv.mpoly ......................................................... 19
as.function.mpoly

Change a multivariate polynomial into a function.

Description

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

Usage

```r
## S3 method for class 'mpoly'
as.function(x, varorder = vars(x), vector
    = TRUE, ...)
```
as.function.mpolyList

Change a vector of multivariate polynomials into a function.

Description

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

Arguments

- **x**: an object of class mpoly
- **varorder**: the order in which the arguments of the function will be provided
- **vector**: whether the function should take a vector argument (TRUE) or a series of arguments (FALSE)
- ... any additional arguments

See Also

plug

Examples

```r
p <- mp("x^3 + 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
```
Usage

```r
## S3 method for class 'mpolyList'
as.function(x, varorder = vars(x),
          vector = TRUE, ...)
```

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

Arguments

- `x`: an object of class `mpolyList`
- `varorder`: the order in which the arguments of the function will be provided (default `vars(mpoly)`)
- `vector`: whether the function should take a vector argument (TRUE) or a series of arguments (FALSE)
- `...`: any additional arguments

Examples

```r
# 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
as.mpoly

Convert an object to an mpoly

```r
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 |-> \( \binom{\text{size}}{x} \ p^x (1-p)^{(\text{size}-x)} \) \( x = 0, \ldots, \text{size} \)
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
```
Description

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

Usage

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

Arguments

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

Value

the object formatted as a `mpoly` object.

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

`mp`

Examples

```r
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))
```
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)
Description
Bernstein polynomial approximation

Usage
bernsteinapprox(f, n, lower = 0, upper = 1, indeterminate = "x")

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

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

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

f <- function(x) sin(2*pi*x)

p <- bernsteinapprox(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 <- bernsteinApprox(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 <- bernsteinApprox(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)

<table>
<thead>
<tr>
<th>bezier</th>
<th>Bezier polynomials</th>
</tr>
</thead>
</table>

**Description**

Compute the Bezier polynomials of a given collection of points. Note that using `as.function` 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 `bezierFunction` 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**

`bezierFunction`

**Examples**

```r
p1 <- c(0, 0)
p2 <- c(1, 1)
p3 <- c(2, -1)
p4 <- c(3, 0)
bez <- function(p1, p2, p3, p4)
  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))
 BezPoly <- bezier(points))

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

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

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

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

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

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

```r
ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red", size = 8) +
  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))
 BezPoly <- bezier(points))

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

```r
ggplot(aes(x = x, y = y), data = df) +
  geom_point(data = points, color = "red", size = 8) +
  geom_path(data = points, color = "red") +
  geom_path()
```
## Example 4

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

## Example 5

```r
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")
```

---

### bezierFunction

**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

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

Arguments

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

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


See Also

`bezier`

Examples

```r
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 <- bezierFunction(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()
```
burst

```r
f <- bezierFunction(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()
```

```r
f <- bezierFunction(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 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)

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

Value

a mpoly object or mpolyList object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

- chebyshev.t.polynomials
- chebyshev.u.polynomials
- chebyshev.c.polynomials
- chebyshev.s.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(tidyverse)

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

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

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

<table>
<thead>
<tr>
<th>components</th>
<th>Polynomial components</th>
</tr>
</thead>
</table>

Description

Compute quantities/expressions related to a multivariate polynomial.
Usage

```r
## 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
- `...`: additional arguments

Value

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

Examples

```r
(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)

derv.mpoly

---

**deriv.mpoly**

*Compute partial derivatives of a multivariate polynomial.*

---

**Description**

This is a deriv method for mpoly objects. It does not call the deriv function (from package stats).

**Usage**

```r
## 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**

```r
m <- mp('x y + y z + z^2')
derv(m, 'x')
derv(m, 'y')
derv(m, 'z')
derv(m, c('x', 'y', 'z'))
derv(m, 'a')
is.mpoly(derv(m, 'x'))
is.mpolyList(derv(m, c('x', 'y', 'z')))
```
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
rosenbrock <- mp('((1 - x)^2 + 100 (y - x)^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)) +
  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)) +
  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)
runegraph(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")

---

**grobner**

REMOVED – Compute a grobner basis of a list of multivariate polynomials.

---

**Description**

This function has been removed to eliminate mpoly’s dependence on packages that only it uses. To compute a Grobner basis of a collection of multivariate polynomials, checkout the new m2r package, which you can download with the code in the first example.

**Usage**

grobner(mpolyList, varorder = vars(mpolyList), order = "lex")
Arguments

- `mpolyList` an object of class `mpolyList`
- `varorder` order of variables
- `order` total order to be considered for monomials (e.g. lexicographic)

Details

grobner computes a Grobner basis for a collection of multivariate polynomials represented as an object of class `mpolyList`. Note that the polynomials printed after calculation are unlikely to be properly ordered: this is because the order of the monomials displayed is governed by the print method, not the `mpoly`'s themselves.

Value

An object of class `mpolyList`.

Examples

```r
## Not run:

# code to download m2r:
# note that to do this you should have Macaulay2 installed,
# see https://github.com/musicman3320/m2r and
# http://www.math.uiuc.edu/Macaulay2/Downloads/
if(!require(devtools)) install.packages("devtools")
devtools::install_github("musicman3320/m2r")

## End(Not run)
```

hermite

Hermite polynomials

Description

Hermite polynomials as computed by `orthopolynom`.

Usage

```r
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

hermite.h.polynomials, 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(tidyrr)

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))
homogenize

Homogenize a polynomial

Description

Homogenize a polynomial.

Usage

homogenize(x, var = "t")

dehomogenize(x, var = "t")

is.homogeneous(x)

homogeneous_components(x)

Arguments

x an mpoly object

var 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, "t")

xh <- homogenize(x)
dehomogenize(xh) # assumes var = "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**

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

**Arguments**

- `elem` element to insert
- `slot` location of insert
- `v` vector to insert into

**Value**

vector with element inserted

**Examples**

```r
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 mpolylst object

Author(s)

David Kahle calling code from the orthopolynom package

See Also

jacobiNgNpolynomials, jacobiNpNpolynomials 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))
**Description**

Generalized Laguerre polynomials as computed by orthopolynom.

**Usage**

\[ \text{laguerre}(\text{degree}, \alpha = 0, \text{indeterminate} = "x", \text{normalized} = \text{FALSE}) \]

**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**

- glaguerre.polynomials

**Examples**

\[
\begin{align*}
\text{laguerre}(0) \\
\text{laguerre}(1) \\
\text{laguerre}(2) \\
\text{laguerre}(3) \\
\text{laguerre}(4) \\
\text{laguerre}(5) \\
\text{laguerre}(6) \\
\text{laguerre}(2) \\
\text{laguerre}(2, \text{normalized} = \text{TRUE}) \\
\text{laguerre}(0:5) \\
\text{laguerre}(0:5, \text{normalized} = \text{TRUE}) \\
\text{laguerre}(0:5, \text{indeterminate} = "t")
\end{align*}
\]
# visualize the laguerre polynomials

library(ggplot2); theme_set(theme_classic())
library(tidyverse)

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))

---

**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**

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


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

```r
library(ggplot2); theme_set(theme_classic())
library(tidyverse)

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

---

**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**

```r
mp(string, varorder)
```

**Arguments**

- **string** a character string containing a polynomial, see examples
- **varorder** (optional) order of variables in string

**Value**

An object of class mpoly.

**Author(s)**

David Kahle <david.kahle@gmail.com>

**See Also**

- **mpoly**
Examples

```r
(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"))
# mp("x + 2 y + x^2 y", varorder = c("q", "p")) # -> error

(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)
```

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

```r
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
mpolyArithmetic

Value

Object of class mpoly.

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

mp

Examples

```r
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)
```

Description

Arithmetic with multivariate polynomials

Usage

```r
## 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

**Arguments**

- `e1`: an object of class `mpoly`
- `eR`: an object of class `mpoly`

**Value**

object of class `mpoly`

**Examples**

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

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

---

### mpolyEqual

_Determine whether two multivariate polynomials are equal._

**Description**

Determine whether two multivariate polynomials are equal.

**Usage**

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

**Arguments**

- `e1`: an object of class `mpoly`
- `e2`: an object of class `mpoly`
mpolyList

Value

A logical value.

Examples

```r
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")
```

---

**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.
mpolyListArithmetic

Element-wise arithmetic with vectors of multivariate polynomials.

Description

Element-wise arithmetic with vectors of multivariate polynomials.

Usage

```r
## 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

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

mpolyList(mpolyList(p1, p2, p3))
p <- mpolyList("x + 1")
mpolyList(p)

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

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

**Examples**

```r
( ms1 <- mp( c('x + 1', 'x + 2') ) )
( ms2 <- mp( c('x + 1', 'y + 2') ) )
ms1 + ms2
ms1 - ms2
ms1 * ms2
```

---

**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**

```r
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**

```r
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

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

indeterminate

value

Value
an mpoly object
Examples

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

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)
```

<table>
<thead>
<tr>
<th>predicates</th>
<th>mpoly predicate functions</th>
</tr>
</thead>
</table>

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

```r
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)
```

---

**Description**

This is the major function used to view multivariate polynomials.

**Usage**

```r
## S3 method for class 'mpoly'
print(x, varorder, order, stars = FALSE, silent = FALSE, ...)
```
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 `cat`

Value

Invisible string of the printed object.

Examples

```r
p <- mp("2 x^5 - 3 y^2 + x y^3")
p
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")) # -> 2 x^5 - 3 y^2 + y^3 x
print(p, stars = TRUE)
```

Description

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

Usage

```r
# S3 method for class 'mpolyList'
print(x, varorder = vars(x), order,
    silent = FALSE, ...)
```
Arguments

- **x**: an object of class `mpolyList`
- **varorder**: the order of the variables
- **order**: a total order used to order the monomials in the printing
- **silent**: logical; if TRUE, suppresses output
- **...**: arguments to pass to `print.mpoly`

Value

Invisible character vector of the printed objects.

Examples

```r
mL <- mp(c('x + 1', 'y - 1', 'x y^2 z + x^2 z^2 + z^2 + x^3'))
mL

ps <- print(mL, silent = TRUE)
ps

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'))
s <- print(mL, varorder = c('z','y','x'))
str(s)
```

---

**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

```r
## 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
- **...**: additional arguments

Value

An object of class `mpoly`.

Examples

```r
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

```r
## 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@gmail.com>

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**

```r
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)
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, 5)
tuples(1:2, 5, 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))) # x, y, xy
mp("x + y")^n

r <- 3 # number of variables, e.g. x, y, z
\begin{verbatim}
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
\end{verbatim}

\section*{vars}

\textit{Determine the variables in a mpoly object.}

\section*{Description}

Determine the variables in a mpoly object.

\section*{Usage}

\texttt{vars(mpoly)}

\section*{Arguments}

\begin{itemize}
  \item \texttt{mpoly} an object of class mpoly
\end{itemize}

\section*{Value}

A character vector of the variable names.

\section*{Examples}

\begin{verbatim}
list <- list(
  c(x = 5, coef = 2),
  c(y = 2, coef = -3),
  c(x = 1, y = 3, coef = 1)
)
p <- mpoly(list)
vars(p)
\end{verbatim}
Index

*.mpoly (mpolyArithmetic), 33
*.mpolyList (mpolyListArithmetic), 36
+.mpoly (mpolyArithmetic), 33
+.mpolyList (mpolyListArithmetic), 36
-.mpoly (mpolyArithmetic), 33
-.mpolyList (mpolyListArithmetic), 36
== (mpolyEqual), 34
[.mpoly (components), 17
^.mpoly (mpolyArithmetic), 33

as.function, 10
as.function.bezier
    as.function.mpolyList, 3
as.function.mpoly, 2
as.function.mpolyList, 3
as.mpoly, 5

bernstein, 7
bernsteinApprox, 8
bezier, 10, 14
bezierFunction, 11, 13
burst, 15

cat, 41
chebyshev, 16
chebyshev.c.polynomials, 16
chebyshev.s.polynomials, 16
chebyshev.t.polynomials, 16
chebyshev.u.polynomials, 16
components, 17

dehomogenize (homogenize), 24
deriv.mpoly, 19, 20

exponents (components), 17
glaguerre.polynomials, 28
gradient, 20
groebner, 21

hermite, 22

hermite.h.polynomials, 23
hermite.he.polynomials, 23
homogeneous_components (homogenize), 24
homogenize, 24

insert, 25
is.bernstein (predicates), 39
is.bezier (predicates), 39
is.chebyshev (predicates), 39
is.constant (predicates), 39
is.homogeneous (homogenize), 24
is.linear (predicates), 39
is.mpoly (predicates), 39
is.mpolyList (predicates), 39
is.unipoly (predicates), 39
is.wholenumber, 26

jacobi, 26
jacobi.g.polynomials, 27
jacobi.p.polynomials, 27

laguerre, 28
LC (components), 17
LCM, 29
legendre, 30
legendre.polynomials, 30
LM (components), 17
LT (components), 17

monomials (components), 17
mp, 6, 31, 33, 44
mpoly, 24, 31, 32
mpoly-package (mpoly), 32
mpolyArithmetic, 33
mpolyEqual, 34
mpolyList, 35
mpolyListArithmetic, 36
multideg (components), 17

package-mpoly (mpoly), 32
partitions, 37
permutations, 38
plug, 3, 38
predicates, 39
print.mpoly, 40, 42
print.mpolyList, 41
reorder.mpoly, 42
round.mpoly, 43
solve_unipoly, 45
swap, 45
terms.mpoly, 46
totaldeg (components), 17
tuples, 47
vars, 48