conedual.r

denis

2021-07-17

#!/usr/bin/r  
  
# Although it is not  
# necessarily a vector space, a rat is also called an acne space.  
flat = 1  
Acne.space = numeric(flat)  
Acne.space

## [1] 0

# If the equations are homogeneous (that is, if b 1 = · · · = b m = 0), then the  
# point (0, . . . , 0) is included, and the rat is an (n − m)-dimensional   
# subspace (also a vector space, of course).  
b1 = 21  
m = 0  
if (b1 != m){  
 dnx <- dimnames(b1 - m) # force is character 0 to impulse   
}  
  
# 2.2.8 Cones  
# A cone is an important type of vector set (see page 14 for definitions).  
n = 1  
cone <- c(n, contrasts = TRUE, sparse = FALSE)  
cone

## contrasts sparse   
## 1 1 0

# Given a set of vectors V (usually but not necessarily a cone), the dual cone  
# of V , denoted V ∗ , is defined as  
V <- par(cone)  
V

## NULL

# and the polar cone of V , denoted V 0 , is defined as  
V0 <- par(cone)  
V0

## NULL

# Obviously, V 0 can be formed by multiplying all of the vectors in V ∗ by −1,  
# and so we write V 0 = −V ∗ , and we also have (−V ) ∗ = −V ∗ .  
poin <- par(c(latter = -1, collapse = 0, check\_tzones(cone)))  
poin

## NULL

# Although the definitions can apply to any set of vectors, dual cones and  
# polar cones are of the most interest in the case in which the underlying set  
# of vectors is a cone in the nonnegative orthant (the set of all vectors all of  
# whose elements are nonnegative).  
elif <- c(i = 1, fp = 1, pop = 4, lm = 1, ln = 0, lap = 1)  
elif

## i fp pop lm ln lap   
## 1 1 4 1 0 1

# In that case, the dual cone is just the full  
# nonnegative orthant, and the polar cone is just the nonpositive orthant (the  
# set of all vectors all of whose elements are nonpositive).  
elem <- c(vec = 1, none = 0, step = 1)  
elem

## vec none step   
## 1 0 1

# Although a convex cone is not necessarily a vector space, the union of the  
# dual cone and the polar cone of a convex cone is a vector space. (You are  
# asked to prove this in Exercise 2.12.) The nonnegative orthant, which is an  
# important convex cone, is its own dual.  
dual <- double(length = 2L)  
dual

## [1] 0 0

# Geometrically, the dual cone V ∗ of V consists of all vectors that form  
# nonobtuse angles with the vectors in V . Convex cones, dual cones, and polar  
# cones play important roles in optimization.  
metrix <- c(play = 1, angles = 360, end = 0)  
metrix

## play angles end   
## 1 360 0

# 2.2.9 Cross Products in IR 3  
# For the special case of the vector space IR 3 , another useful vector product   
# is the cross product, which is a mapping from IR 3 ×IR 3 to IR 3 . Before   
# proceeding, we note an overloading of the term “cross product” and of the   
# symbol “×” used to denote it.  
vec <- c(metrix, y = NULL, circles = metrix, squares = metrix,   
 rectangles = 4, stars = metrix, thermometers = 2,   
 boxplots = 1, inches = FALSE, add = FALSE, fg = 0, bg = NA,  
 xlab = NULL, ylab = NULL, main = NULL, xlim = NULL, ylim = NULL)  
vec

## play angles end circles.play circles.angles   
## 1 360 0 1 360   
## circles.end squares.play squares.angles squares.end rectangles   
## 0 1 360 0 4   
## stars.play stars.angles stars.end thermometers boxplots   
## 1 360 0 2 1   
## inches add fg bg   
## 0 0 0 NA

# If A and B are sets, the set cross product or the set Cartesian  
# product of A and B is the set consisting of all doubloons (a, b) where a   
# ranges over all elements of A, and b ranges independently over all elements   
# of B.  
prod <- prod(vec, na.rm = FALSE)  
prod

## [1] NA

# Thus, IR 3 × IR 3 is the set of all pairs of all real 3-vectors.  
# The vector cross product of the vectors  
IR <- crossprod(vec, y = NULL)  
crosstalk::ClientValue

## <ClientValue> object generator  
## Public:  
## initialize: function (name, group = "default", session = shiny::getDefaultReactiveDomain())   
## get: function ()   
## sendUpdate: function (value)   
## clone: function (deep = FALSE)   
## Private:  
## .session: ANY  
## .name: ANY  
## .group: ANY  
## .qualifiedName: ANY  
## .rv: ANY  
## Parent env: <environment: namespace:crosstalk>  
## Locked objects: TRUE  
## Locked class: FALSE  
## Portable: TRUE

IR.x <- c(x1 = 5.0, x2 = 5.2, x3 = 5.3)  
IR.y <- c(y1 = 6.0, y2 = 6.2, y3 = 6.3)  
# cone button  
IR.x

## x1 x2 x3   
## 5.0 5.2 5.3

IR.y

## y1 y2 y3   
## 6.0 6.2 6.3

# written x × y, is defined as  
IR.x + sin(5)

## x1 x2 x3   
## 4.041076 4.241076 4.341076

IR.y + sin(6)

## y1 y2 y3   
## 5.720585 5.920585 6.020585

# (We also use the term “cross products” in a different way to refer to another  
# type of product formed by several inner products; see page 287.)  
crosstalk::animation\_options(interval = 1000, loop = FALSE,   
 playButton = NULL, pauseButton = NULL)

## $interval  
## [1] 1000  
##   
## $loop  
## [1] FALSE  
##   
## $playButton  
## NULL  
##   
## $pauseButton  
## NULL

# 1. Self-nilpotency:  
# x × x = 0, for all x.  
IR.x + sin(5) + IR.x + sin(5)

## x1 x2 x3   
## 8.082151 8.482151 8.682151

# 2. Anti-commutativity:  
# x × y = −y × x.  
IR.x + sin(5) - IR.x + sin(5)

## x1 x2 x3   
## -1.917849 -1.917849 -1.917849

# 3. Factoring of scalar multiplication;  
# ax × y = a(x × y) for real a.  
IR.x + sin(5) + IR.x + c(ax = 2.1, ya = c(xx = 2.2, yy = 1.1))

## x1 x2 x3   
## 11.14108 11.64108 10.74108

# 4. Relation of vector addition to addition of cross products:  
# (x + y) × z = (x × z) + (y × z).  
IR.x + sin(5) + IR.x + c(xx = 2.1, yy = 6.1, c(z = c(xx = 2.1, yy = 6.1),   
 c(yy = 6.1, zz = 8.1)))

## xx yy z.xx z.yy yy zz   
## 11.14108 15.54108 11.74108 15.14108 15.54108 17.74108