Package ‘vectools’

June 7, 2021

Title Advanced Vector Toolkit
Version 0.3.0
Date 2021-06-07
License GPL (>= 2)
Maintainer Abby Spurdle <spurdle.a@gmail.com>
Author Abby Spurdle

Description Image objects/arrays, geometric objects/arrays (in 2D/3D), block matrices and matrix arrays, with related binary operators and plain text markup. Mid-level plotting/rendering functions (as building blocks for high-level plotting functions), with dual support for base and grid graphics systems. Also, sequence/set utilities, linear transformations and additional color matrix/array transformations.

Note Suggested packages png/jpeg, grid and colorspace are required for some functions/examples.

Imports methods
Suggests png, jpeg, grid, colorspace
Contact Primary <spurdle.a@gmail.com>, Secondary <spurdle.a@aol.com>

R topics documented:

00_binary_operators ........................................ 2
01_subsetting_operators .................................... 5
02_language-like_functions ................................. 6
03_constants .................................................. 7
04_new_generic_functions ................................... 8
10_raster_images ............................................. 9
15_simple_geometric_objects ............................... 11
16_other_main_geometric_objects ......................... 12
17_regular_polygons ........................................ 13
Matrix Array Multiplication

Description

Multiplication on GeomArray and MatrixArray objects.

These functions call the Transformation Functions.
Usage

#simple matrix-array multiplication
## S4 method for signature 'MatrixArray,MatrixArray'
x %*% y
## S4 method for signature 'MatrixArray,matrix'
x %*% y
## S4 method for signature 'matrix,MatrixArray'
x %*% y

#pre-multiplication
#(with right-side data objects)
a %*[% b
a %*{% b
a %*|% b

#post-multiplication
#(with left-side data objects)
a [%]*% b
a [%]*% b
a [%]*% b

Arguments

a, b  In simple matrix-array multiplication, matrices or MatrixArray(s).

In pre-multiplication, a is the transformation object and b is the data object.
In post-multiplication, a is the data object and b is the transformation object.

The transformation object should be a matrix or MatrixArray.

The data object should a matrix, GeomObject, GeomArray or MatrixArray.
Except for row-wise and col-wise functions (with the vertical bars), where the data object should be a matrix.

x, y  Same as a and b.

Details

The operators are designed for matrix array multiplication.
But can also be used to transform GeomObject(s) and GeomArray(s).

If there are two matrix arrays, each with ten matrices of the same size:
And they are multiplied using the %*% operator, then each pair of matrices is multiplied using matrix multiplication, and a new matrix array is returned, of length ten.

It’s possible to use a single matrix, in which case, its handled the same as a matrix array of length one.
Each pair of matrices needs to be conformable.
And if the lengths of the matrix arrays are different, then one matrix array is recycled.
(But currently, the length longer one needs to a multiple of the length shorter one).

Often these operators can be used to apply transformation matrices to data.
In which case, the transformation matrices will often have one more row or column than the data.

With the right-pad operator `%*%`, an extra row of ones is rbinded to the right hand matrices before multiplication, and strips it off the result. Likewise, with the left-pad operator `%]*%`, an extra column of ones is cbinded to the left hand matrices before multiplication, and then strips it off.
The padded operators can also used on GeomObject(s), including GeomArray(s).

There are a number of variations on this approach.
In cartesian-based matrix array multiplication, the resulting object is the product over the cartesian set of both matrix arrays.
If the matrix arrays have length 10 and 20, then the resulting matrix array will have length 200, not 20.

The `%*|%` and `%|*%` operators perform cartesian matrix array multiplication, using a right or left data object, respectively.
These call the vt3.gmultc.pre and vt3.gmultc.post functions, which provide some more information.
The `%|*%` and `%|*%` operators, perform column-wise and row-wise matrix array multiplication, with right or left padding.
The data object, which should be a matrix, is handled as a MatrixArray of padded column or row vectors.
The double bar operators are the same, except that the don’t pad the operands.
In post-multiplication (but not pre-multiplication) with two or more transformations and left-padding of the left-most matrix array, we need to put all the transformations in parentheses, otherwise the result after the first multiplication will be non-conformable with the subsequent multiplications.

**Value**

Column-wise and row-wise operators return a single matrix.
All other operators should return a GeomArray or MatrixArray.
Note that the recycling order, is subject to change.

**See Also**

MatrixArray
Pre-Multiplication Transformation Matrices
Post-Multiplication Transformation Matrices
vt3.testplot, vt3.testplot3

**Examples**

```r
#refer to:
#Pre-Multiplication or Post-Multiplication
#for better examples
```

```r
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```
vt3.testplot (SQUARE %*% brot2 (pi / 8) )

**Subsetting Operators**

Description
Subsetting operators for vector-like objects.

Usage

```r
## S3 method for class 'ObjectArray'
v[...]
## S3 method for class 'ObjectArray'
v[[...]]
## S3 replacement method for class 'ObjectArray'
v[[...]] <- value

## S3 method for class 'NestMatrix'
v[[i, j, ..., drop=TRUE, zero=TRUE]]
## S3 replacement method for class 'NestMatrix'
v[[i, j]] <- value

## S3 method for class 'PartMatrix'
v[[i, j, ..., drop=TRUE]]
## S3 replacement method for class 'PartMatrix'
v[[i, j]] <- value

## S3 method for class 'GeomArray'
v[...]
## S3 method for class 'ImageArray'
v[...]
## S3 method for class 'MatrixArray'
v[...]
## S3 replacement method for class 'GeomArray'
v[[...]] <- value

# for SectMatrix
getSect (v, ..., drop=TRUE)
```

Arguments

- `v` An ObjectArray (including subclasses) or block matrix object.
- `i, j, ...` The indices.
- `drop` Logical, if true, reduce the number of dimensions, where possible.
- `zero` Logical, if true, replace ZERO with zero, or a zero matrix.
- `value` A suitable value to assign.
Details

ObjectArray (and NestMatrix), are similar to lists, so:
Single bracket subsetting returns another ObjectArray, which is a subset of the original.
Double bracket subsetting gets or sets a single element within the ObjectArray.

For PartMatrix:
Currently, single bracket subsetting is not supported.
Currently, double bracket subsetting indexes a single submatrix.

Note that the number of indices used in an ObjectArray depends on the number of dimensions.
The number of indices in PartMatrix objects is always two.
And the number of indices for SectMatrix subsetting depends on the VMap object.

Note that there are restrictions on what can be assigned:
An ObjectArray may be restricted by its CLASS slot, determined when the object is constructed.
Other objects may have restrictions on the sizes of submatrices.

See Also

ObjectArray
NestMatrix
PartMatrix
SectMatrix

Examples

x <- matrix (1:16, 4, 4)

pm <- as.PartMatrix (x, 2, 2)
nm <- as.NestMatrix (pm)

#all same
x [1:2, 1:2]
nm [[1, 1]]
pm [[1, 1]]

#all same
x [1, 2]
nm [[1, 1]][1, 2]
pm [[1, 1]][1, 2]
Usage

CLASS (v)
ndim (v)

Arguments

v An ObjectArray.

Details

CLASS returns a character giving the CLASS property of the ObjectArray. This is the class or classes of the objects within the array. <OBJECT> means that the ObjectArray can contain any objects. The ndim function returns an integer, giving the number of dimensions.

See Also

ObjectArray

Examples

v <- ObjectArray (c (2, 5), CLASS="MyClass")
CLASS (v)
class (v)
ndim (v)

03_constants

Constant Values

Description

The ZERO constant is for NestMatrix. The other constants are mainly for test plots. Refer to vt3.testplot and vt3.testplot3 functions.

Usage

ZERO
SQUARE
CUBE
#tilted
TCUBE
Details

ZERO is an instance of the class Zero.
This is a generalization of zero matrices.
Which can be used in NestMatrix objects, without being constrained by dimensions.

See Also

NestMatrix
vt3.testplot, vt3.testplot3, spin3d

Examples

vt3.testplot3 (CUBE %*% btrl3 (0, 2, 2.5) )
Details

In general, it’s not necessary to call the plotv function. Calls to plot should call plotv, which in turn, should call the correct method.

The rep2 function is similar to rep_len. (It recycles arguments to a particular length).

The objtag function is used inside formatted object arrays.

The ghead and headt functions are for grouped head and combined head and tail.

See Also

rep.MatrixArray, rep2.MatrixArray
objtag methods
ghead.data.frame
Combined Head and Tail Functions

Examples

plot (TCUBE)
plotv (TCUBE)
ghead (iris)

10_raster_images  Raster Objects

Description

Objects representing raster image data.

The class RImage has two subclasses SImage (for single channel images) and MImage (for multiple channel images).

Currently, pixel values (im) need to be between zero and one.

NOTE: INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.

DO NOT USE SLOTS, DIRECTLY.

NOTE: USING NON-sRGB COLOR SPACES REQUIRES THE colorspace PACKAGE TO BE INSTALLED AND LOADED.

Also, note that there’s also a VImage object, which uses vector graphics.

Usage

as.SImage (im)
as.MImage (im, ..., input="sRGB", storage=input)

get.channel (im, which, ..., as.matrix=FALSE)
Arguments

im For SImage:
A matrix or raster object.

For MImage:
A matrix, an array, a list, a SImage object or a raster object.
Arrays can have two or three dimensions.
In the 3d case, the last dimension can have 1 to 4 items, each representing a channel.
Lists can have 1 to 4 equally-sized matrices, each representing one channel.

For get.channel:
An MImage object.

input, storage Single characters, giving the name of a valid color space from the colorspace package.
Refer to mapcol.
The input argument is ignored, unless the input is a three or four channel 3D array.

Input is the input color space, and storage is the storage color space.

Note that the colorspace package needs to be installed to use non-sRGB colorspaces.
Also, note that the same name is used for three channel images, and four channel images (with an alpha channel).
i.e. "sRGB" can be used for sRGB and sRGB-A.

which Integer, which channel.

as.matrix Logical, if false (default), return an SImage object, If true, return a matrix.
... Ignored.

Value

An SImage, MImage or matrix object.

Examples

library (png)

#adapted from png package
im0 <- readPNG (system.file ("img", "Rlogo.png", package="png") )

#multichannel image
im <- as.MImage (im0)

plot (im)

headt (as.raster (im) )
Description

Points, lines, polygons and text, as S4 objects.
These can be plotting using either the base or grid graphics systems.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Usage

Points (x, y, z=NULL, ..., glist = list () )
Line (x, y, z=NULL, ..., glist = list () )
Polygon (x, y, z=NULL, ..., glist = list () )
Text (text, x, y, z=NULL, ..., glist = list () )

as.Points (v, ...)
as.Line (v, ...)
as.Polygon (v, ...)

Arguments

x, y, z
Equal length numeric vectors, giving coordinates.
If z missing, a 2d object, otherwise, a 3d object.
v
Currently, a two or three column matrix.
text
A character vector of labels.
x, y and z, need to be the same length.
glist
Argument list for plotting functions.
...
Ignored.

Value

Each function returns an S4 object with the corresponding class name.
e.g. Line returns a Line object.

See Also

Grid, VImage
regPolygon

Examples

plot (Points (1:4, 1:4) )
### Description

Geometric objects, resembling a grid, containing quadrilaterals. These can be plotting using either the base or grid graphics systems. These functions take three matrices. For regularly-spaced or semi-regularly-spaced grids, it may be easier to use the `rectGrid` and `rectVImage` functions, which take two vectors and a matrix.

**NOTE:**

INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

### Usage

```r
Grid(x, y, gv=NULL, ..., glist = list(), vlist = NULL)
VImage(x, y, gv=NULL, ..., tf=FALSE, colm, glist = list())
as.Grid(v, ..., glist = list(), vlist = NULL)
```

### Arguments

- **x, y**
  - Numeric matrices of coordinates.
- **gv**
  - A numeric matrix, giving the "z" value at each "x" and "y" pair.
  - If null, a 2D grid is created.
  - Can also be a scalar value, but this is mainly for testing and demo purposes.
- **v**
  - Currently, an FGrid object, such as one returned by the `cgrid` function.
- **glist**
  - Argument list for plotting functions.
- **vlist**
  - Same as glist, except that each element is a matrix, corresponding to each quadrilateral.
  - The number of rows and columns for each element, should be one less than gv.
  - Note that glist and vlist should contain distinct names.
- **tf**
  - Logical, transpose and flip colm.
  - By default, x and y have their standard meanings.
  - Setting tf to true, allows raster-style data to be rendered properly.
- **colm**
  - A character matrix of colors.
  - The number of rows and columns for each element, should be one less than gv.
- **...**
  - Ignored.

### Details

Note that the plot methods for VImage will plots the colors (like a heatmap) if colm is provided. Otherwise, it will plot a grid of lines.
17_regular_polygons

Value

A Grid or VImage object.

See Also

Points, Line, Polygon
RImage-class
rectGrid, rectVImage
Refer to these functions, for examples.
(These use simple vectors for x and y, rather than matrices).

Description

Regular polygons.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Usage

regPolygon (n=4, ..., about.axis, d=1, stagger = (n %% 2 == 0) )

Rect (...), about.axis, center=FALSE, side.length=1,
glist = list ()
Cuboid (...), center=FALSE, side.length=1, glist = list ()

Arguments

n
  Integer, the number of points.
d
  Numeric, the distance.
about.axis
  A single character equal to "x", "y", "z".
  If missing (the default), the object is in 2d space.
  If true, a 2d object is created in 3d space.
stagger
  Logical, if false, the first point is at the top.
  Otherwise, it's rotated half the angle.
  (That is, the angle equal to 2pi divided by n).
center
  Logical, if true, the object is centered about the origin, otherwise, it has the lowermost corner at the origin.
side.length
  Numeric vector of length one or two, the x and y lengths.
glist
  Argument lists for plotting functions.
  Note that, in Cuboid, it's applied to each face.
...
  Ignored.
Value

A regPolygon, Rect or Cuboid object.

Examples

vt3.testplot (Rect (side.length = c (2, 1) ) )

Description

Regularly-spaced or semi-regularly-spaced grids. These functions take two vectors and a matrix. For more general "grids", you can use the Grid and VImage functions, which take three matrices.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Usage

rectGrid (x, y, gv=NULL, ..., glist = list (), vlist=NULL)
rectVImage (x, y, gv=NULL, ..., tf=FALSE, colm, glist = list () )

Arguments

x, y       Equal length (ascending or descending) numeric vectors, giving the grid points, in x and y directions.
gv         A numeric matrix, giving the "z" value at each "x" and "y" pair. If null, a 2D grid is created.
            Can also be a scalar value, but this is mainly for testing and demo purposes.
glist, vlist Argument lists of graphical parameters using in the plotting functions.
            glist applies to all polygons, in vlist the elements should be matrices, where each value is applied to separate polygons.
tf         Logical, transpose and flip gv, for rendering raster-style images, using the standard coordinate system.
colm       A character matrix, of colors.
...         Ignored.

Details

Note that vlist and colm arguments should have dimensions one less than the nrow/ncol of gv.
Value

A Grid or VImage object.

See Also

Grid, VImage

as.Grid

Which can be used on FGrid-class objects.

Examples

library (png)

#adapted from png package
im0 <- readPNG (system.file ("img", "Rlogo.png", package="png") )

#standard raster object
colm <- as.raster (im0)
headt (colm)

#smaller version
I <- seq (1, nrow (colm), 5)
J <- seq (1, ncol (colm), 5)
colm <- colm [I, J]
headt (colm)

#VImage
x <- seq (-1, 1, length (colm))
y <- seq (-1, 1, length (colm))
v <- rectVImage (x, y, 0, tf=TRUE, colm=colm)

if (interactive () )
  spin3d (v *%*% brot3x (pi / 4), t=10)
Arguments

CLASS         A character vector of classes.
Every object in the object array, needs to be of these classes.
Note that the order is not important.

The default, "<OBJECT>", means that objects can be of any class.

n            Positive integer vector, the dimensions.

names        List of character vectors, of names in each dimension, which should match n,
above.
May be NULL or contain NULL values.

default.value Default value, for each element.

v            In as.ObjectArray, a list.
In as.NestMatrix, a matrix or a PartMatrix.

...           Ignored.

Details

ObjectArray(s) are similar to list matrices and list arrays.
(But these are S4 classes, with a number of differences).

Setting individual elements to NULL, doesn’t remove them.
And they support formatting, via objtag methods.

By default, an ObjectArray can contain any objects, however, this can be restricted by the CLASS
argument.

Value

An ObjectArray object.

See Also

Subclasses:
ImageArray
GeomArray
NestMatrix
MatrixArray
PartMatrix, SectMatrix
Similar to NestMatrix.

Subsetting Operators
Standard Methods
Binary Operators
Operations on MatrixArray objects.
Examples

v <- ObjectArray (c (2, 2), CLASS="list")
v [[1, 1]] <- list (1)
v [[2, 1]] <- list (1, 2)
v [[1, 2]] <- list (1, 2, 3)
v [[2, 2]] <- list (1, 2, 3, 4)
v
v [[1, 2]]

Image Arrays

Description

ImageArray objects, where ImageArray is a subclass of ObjectArray, containing RImage (raster image) objects.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Usage

ImageArray (n)
as.ImageArray (im, ..., n)

Arguments

n     Integer vector, the dimensions.
im   A list of RImage objects.
...  Ignored.

Value

An ImageArray object.

See Also

ObjectArray
RImage
VIImage
Examples

```r
library (png)
library (colorspace)

#adapted from png package
im0 <- readPNG ( system.file ("img", "Rlogo.png", package="png") )

#multichannel images
im <- ImageArray (2)
im [[1]] <- as.MImage (im0)
im [[2]] <- as.MImage (im0, storage="HCL")

decomp.plot (im)
```

22_GeomArray

GeomArray

Description

A GeomArray is a subclass of ObjectArray for storing GeomObject(s).

Note that GeomArray is also a GeomObject, and contain other GeomArray(s).
i.e. They can be recursive.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Usage

```r
GeomArray (n, ..., D)
GeomArray2 (n)
GeomArray3 (n)

as.GeomArray (v, ..., n, D)
```

Arguments

- **n**: Single integer, the dimensions.
  Currently, restricted to one dimensional arrays.
- **D**: Integer, two or three, for the number of spatial dimensions.
  Any elements in the GeomArray will need to match this.
  For the as function, D is optional but all elements need to match.
- **v**: A list of GeomObject(s).
- **...**: Ignored.
Description

Block matrices.
(NestMatrix and PartMatrix objects).

PartMatrix objects are generalized by SectMatrix objects, which allow overlapping sections.

Note that PartMatrix objects maybe removed or deprecated, in the future.

NOTE:
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE.
DO NOT USE SLOTS, DIRECTLY.

Examples

# explicit construction
v <- GeomArray (2, D=2)
v [[1]] <- regPolygon (6, d=1)
v [[2]] <- regPolygon (6, d=2)
plot (v)

# construction via matrix array multiplication
v <- regPolygon (6, d=1)
plot (v)
Usage

VMap (n)

NestMatrix (nr, nc, ..., rnames, cnames, conform=TRUE, recursive=FALSE, default.value=ZERO)
as.NestMatrix (v, ..., rnames, cnames, conform=TRUE, recursive=FALSE)

PartMatrix (nr, nc, rsep, csep, ..., rnames, cnames, default.value=0)
SectMatrix (nr, nc, ..., vmap, rnames, cnames, default.value=0)

as.PartMatrix (v, rsep, csep, ..., rnames, cnames)
as.SectMatrix (v, ..., vmap, rnames, cnames)

n22 (...)

Arguments

n Integer, the number of submatrices.

nr, nc Integers, the number of rows and columns

rsep, csep Optional integer vectors, the inter-row and inter-column indices.
Ignored, if v is a NestMatrix.

vmap A VMap object.

rnames, cnames Character vectors of nr row names and nc column names.
May be NULL.
Currently ignored, when a NestMatrix is constructed from a PartMatrix.

default.value The default value of entries in the matrix.

v A matrix or data.frame.
as.PartMatrix also allows a NestMatrix, if it has conformable submatrices and is non-recursive.

conform Logical, if true, submatrices are required to have conformable dimensions.
Note that a NestMatrix may contain ZERO(s) regardless of the whether there’s a conformable requirement or not.

recursive Logical, if true, subelements can be nested matrices.
Otherwise, they must be ZERO or standard matrices.

... Ignored.

Details

A NestMatrix is a subclass of an ObjectArray that contains ZERO or standard matrices.
If recursive is true, then it may also contain other NestMatrix objects.

Note that NestMatrix elements can have a scalar value assigned, but it will be converted to a 1x1 matrix.

A PartMatrix is a subclass of a SectMatrix.
(These are similar to NestMatrix).
A PartMatrix contains a matrix, along with a set of separator lines. (These lines are included, when the object is printed/formatted).

A SectMatrix is similar, but allows arbitrary rectangular sections, defined by the `vmap` argument.

A VMap object is a subclass of ObjectArray, that requires one or more two by two integer matrices. Each two by two matrices defines the indices of the section. The top row is the start and end indices for the rows, and the bottom row is the start and end indices for the columns.

A VMap can have more than one dimensions. And a VMap is created by the PartMatrix constructor, which is two dimensional.

The `n22` function is a convenience function for constructing a one dimensional VMap. It takes a integer vector, where the length is a multiple of four. Each consecutive set of four defines the indices for one section. Within each set, each value describes the row start, row end, column start and column end, in that order.

**Value**

VMap, NestMatrix, PartMatrix and SectMatrix are constructors for object of their classes.

`n22` returns a VMAP object.

**See Also**

ObjectArray

Subsetting Operators

Standard Methods

**Examples**

```r
x <- matrix(1:100, 10, 10)

#NestMatrix
msub <- NestMatrix(4, 4)

nm1 <- NestMatrix(4, 4)

nm2 <- NestMatrix(4, 4, recursive=TRUE, default.value=msub)

nm3 <- as.NestMatrix(PartMatrix(5, 5, 2:4, 2:4))

nm1

nm2

nm3

#PartMatrix
pm <- as.PartMatrix(x, 5, c(2, 4, 6, 8))

pm
```
pm [[1, 4]]

#SectMatrix

vmap <- n22 ( 1, 10, #1 1, 10, 3, 10, #2 3, 10, 5, 10, #3 5, 10, 7, 10, #4 7, 10, 9, 10, #5 9, 10)

sm <- as.SectMatrix (x, vmap=vmap)

getSect (sm, 4)

---

**26_MatrixArray**  
**Matrix Arrays**

**Description**

A MatrixArray is a subclass of ObjectArray. These represent a (currently one dimensional) array of matrices. The package provides binary operators to apply matrix multiplication over the entire array.

**NOTE:**  
INTERNAL STRUCTURE OF OBJECTS IS SUBJECT TO CHANGE. DO NOT USE SLOTS, DIRECTLY.

**Usage**

MatrixArray (n, ..., NR, NC, conform=TRUE, default.value=ZERO)  
as.MatrixArray (v, ..., n, NR, NC, conform=TRUE)

**Arguments**

- **n**  
  Integer, the length  
  Currently, the top-level object is constrained to one dimension.

- **v**  
  A suitable object.  
  Such as a list of matrices.

- **NR, NC**  
  Currently, these need to be set to NA.

- **conform**  
  Logical, if true, submatrices are required to have conformable dimensions.

- **default.value**  
  The default value.

- **...**  
  Ignored.
Value
A MatrixArray object.

See Also
ObjectArray
NestMatrix, GeomArray
Binary Operators

Examples
v <- MatrixArray (2, NR=NA, NC=NA)
v [[1]] <- matrix (1:4, 2, 2)
v [[2]] <- matrix (5:8, 2, 2)
v

t <- diag (c (2, 2) )

u <- t

u
u [[1]]
u [[2]]

Description
Functions to test if an object is of a particular class.

Note that ImageArray, GeomArray, NestMatrix and MatrixArray extend ObjectArray, and PartMatrix extends SectMatrix.

Usage
is.ZERO (object)

is.RImage (object)
is.GeomObject (object)

is.Points (object)
is.Line (object)
is.Polygon (object)
is.Text (object)
is.Grid (object)

is.SImage (object)
is.MImage (object)
is.ImageArray (object)
is.ObjectArray (object)
is.GeomArray (object)
is.NestMatrix (object)
is.PartMatrix (object)
is.SectMatrix (object)
is.MatrixArray (object)

Arguments

object An object to test.

Value

TRUE or FALSE.

Examples

is.ZERO (ZERO)

31_replication_functions

Replication Functions

Description

rep methods, and rep2 methods, functions similar to rep_len.

Usage

## S3 method for class 'ObjectArray'
rep(x, n=1, ..., each=1, times=n)
## S3 method for class 'ImageArray'
rep(x, n=1, ..., each=1, times=n)
## S3 method for class 'GeomArray'
rep(x, n=1, ..., each=1, times=n)
## S3 method for class 'MatrixArray'
rep(x, n=1, ..., each=1, times=n)

## S3 method for class 'ObjectArray'
rep2(v, n, ..., n.out=n)
## S3 method for class 'ImageArray'
rep2(v, n, ..., n.out=n)
## S3 method for class 'GeomArray'
rep2(v, n, ..., n.out=n)
## S3 method for class 'MatrixArray'
rep2(v, n, ..., n.out=n)
32_combine_methods

Arguments

- **x, v**: A one-dimensional ObjectArray (incl subclasses).
- **n**: Default value for n.out and times, see below.
- **each**: Integer, the number of times, each element is repeated, consecutively.
- **times**: Integer, the number of times, the sequence is repeated, periodically.
- **n.out**: Integer, the length of the output object.
- **...**: Ignored.

Value

A new one-dimensional ImageArray, ObjectArray, GeomArray or MatrixArray object.

Examples

```r
rep (eq.brot2 (4), 10)
```

---

**Description**

Combine one-dimensional ObjectArray(s).

**Usage**

```r
## S3 method for class 'ObjectArray'
c(...)
```

**Arguments**

- **...**: An argument list of one-dimensional ObjectArray(s), including subclasses.

**Value**

A new ObjectArray object.

**Examples**

```r
v1 <- rep (eq.brot2 (4), 2)
v2 <- rep (eq.brot2 (6), 2)
c (v1, v2)
```
**33_standard_methods**  

**Standard Methods**

**Description**

Length, dim, print, format, head and tail methods for vector-like objects, from this package.

**Usage**

```r
## S3 method for class 'GeomObject'
print(x, ...)
## S3 method for class 'MImage'
print(x, ...)
## S3 method for class 'SImage'
print(x, ...)

## S3 method for class 'ObjectArray'
length(x)
## S3 method for class 'SectMatrix'
length(x)

## S3 method for class 'ObjectArray'
dim(x)
## S3 method for class 'SectMatrix'
dim(x)

## S3 method for class 'ObjectArray'
dimnames(x)
## S3 method for class 'SectMatrix'
dimnames(x)

## S3 replacement method for class 'ObjectArray'
dimnames(x) <- value
## S3 replacement method for class 'SectMatrix'
dimnames(x) <- value

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

## S3 method for class 'ObjectArray'
format(x, ...)
## S3 method for class 'SectMatrix'
format(x, ..., na.string="")
## S3 method for class 'ZERO'
format(x, ...)

## S3 method for class 'VecLike'
```

head(x, n=6, ...)
## S3 method for class 'raster'
head(x, ...)
## S3 method for class 'VecLike'
tail(x, n=6, ...)
## S3 method for class 'raster'
tail(x, ...)

Arguments

x A suitable object.
na.string String, what to format NAs as.
n Integer, the number of items.
value A list of character vectors, matching the dimensions of the object.
... Ignored.

Details

Note that the dimensions of object arrays and nested matrices are the dimensions of the top level object. However, the dimensions of sectioned matrices (including partitioned matrices) are the dimensions of the combined matrix.

The format method for ObjectArray, calls the objtag function, for each of its elements.

The format method for sectioned and partitioned matrices, adds plain-text markup, for separators and section boxes.

Value

Most functions perform the same action as standard R functions.

The format methods return formatted character matrices.

Currently, some of the head, tail functions return formatted character matrices, however, it’s possible this may change in the future.

Examples

x <- matrix (1:100, 10, 10)
pm <- as.PartMatrix (x, 5, c (2, 4, 6, 8) )
nm <- as.NestMatrix (pm)

dim (nm)
dim (pm)

head (nm, 2)
head (pm, 2)
nm
pm
34_as_raster_methods  As Raster Methods

Description

Create standard raster objects from RImage (SImage and MImage) objects, from this package.

Usage

as.raster.SImage (x, ..., optfit=FALSE,
colf = vt3.linearshader (reverse=invert),
invert=TRUE)

as.raster.MImage (x, ..., which, invert.gs=TRUE)

Arguments

x An RImage object.
which Integer, which channel.
  If missing, all channels are used.
invert, invert.gs Logical, if true, reverse the colors.
  In MImage, ignored, except for GS and GS-A (one and two channel images), or
  single-channel images via the which arg.
optfit Logical, if true, fit colors to a reduced interval (min value to max value), increas-
  ing the visible color variation.
colf A function that maps a numeric matrix to a character matrix of R colors.
  This includes a shader function from this package.
... Ignored.

See Also

RImage

Refer to above link for example.

36_other_conversion_functions

Other Conversion Functions

Description

Functions to convert ObjectArray(s) to lists, and NestMatrix and SectMatrix objects to matrices.
Usage

## S3 method for class 'ObjectArray'
as.list(x, ...)

## S3 method for class 'MImage'
as.list(x, ...)

## S3 method for class 'SImage'
as.matrix(x, ...)

## S3 method for class 'NestMatrix'
as.matrix(x, ...)

## S3 method for class 'SectMatrix'
as.matrix(x, ...)

Arguments

x                An ObjectArray/NestMatrix or PartMatrix/SectMatrix object.

Note that nested matrices need to be conformable and non-recursive.

...   Ignored.

Examples

x <- NestMatrix (2, 2)
x [[1, 1]] <- matrix("AA", 2, 2)
x [[1, 2]] <- matrix("AB", 2, 2)
x [[2, 1]] <- matrix("BA", 2, 2)
x [[2, 2]] <- matrix("BB", 2, 2)

x

as.matrix (x)

Description

Simple plotting functions for testing both 2d and 3d transformations.

Note that currently, the spin3d function only doesn’t work well for objects, where the x/y/z ranges are considerably different.

e.g. An x and y range between -10 and 10, and a z range between 0 and 1000.

Usage

vt3.testplot (v=SQUARE, ..., 
main="", xlab="x", ylab="y", 
xlim = c (-4.5, 4.5), ylim=xlim, 
grid.lines=TRUE, cross.hairs=grid.lines,
proj="xy", sort=TRUE)

vt3.testplot3 (v=CUBE, ..., main = c("proj=xy", "proj=xz", "proj=yz"), xlab="x", ylab="y", zlab="z", xlim = c(-4.5, 4.5), ylim=xlim, zlim=xlim, grid.lines=TRUE, cross.hairs=grid.lines, sort=TRUE)

spin3d (v=CUBE, ..., t=5)

Arguments

v       A GeomObject, which includes a GeomArray.
main, xlab, ylab, zlab
       Strings, the main, x axis and y axis titles.
       Note that in 3D plots, these map from 3D to 2D.
xlim, ylim, zlim
       Numeric vectors of length two, giving the x, y and z ranges.
       Note that in 3D plots, these map from the 3D coordinates to 2D coordinates.
grid.lines Logical, add grid lines.
cross.hairs Logical, add lines running through the origin.
proj    String, either "xy", "xz" or "yz", projection for 3D objects.
sort    Logical, if true (the default), depth-sort 3D objects.
t       Numeric, approximate number of seconds.
...     Ignored.

See Also

MatrixArray
Binary Operators
This describes how to perform operations on MatrixArray objects.
Pre-Multiplication Transformation Matrices
Post-Multiplication Transformation Matrices
SQUARE, CUBE

Examples

vt3.testplot (SQUARE %]*% brot2 (pi / 8) )

vt3.testplot3 (TCUBE)

if (interactive () )
spin3d (v=CUBE)
Description

Plotting methods for GeomObject(s).
These use a dual approach that supports both base graphics and grid.
There are plotting methods for both 2D and 3D objects.

Usage

## S3 method for class 'RImage'
plotv(im, ...)

## S3 method for class 'Points'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'Line'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'Polygon'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'Text'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'Grid'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'VImage'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy")

## S3 method for class 'ImageArray'
plotv(im, ..., rby=TRUE, add=FALSE)

## S3 method for class 'GeomArray'
plotv(v, ..., gsys="graphics", add=FALSE, proj="xy", sort=TRUE)

plot_RImage (im, ..., gsys="graphics",
    xlim = c (0, 1), ylim = c (0, 1),
    orient = "r",
    add=FALSE, interpolate=FALSE)

Arguments

v A GeomObject.
xlim, ylim Length-two numeric vectors, giving the boundaries of the image.
gsys String, graphics system, either "graphics" or "grid".
add Logical, if true, add to existing plot, if false, a new plot is created.
proj String, the projection, for 3D objects.
    Either "xy", "xz", "yz".
sort Logical, if true, sort the objects, first.
Note that 3D objects need to be sorted to render correctly.
However, they only need to be sorted once.
interpolate Logical, if true, interpolate.
orient String, the orientation.
Either "v" for standard vector graphics (such as plotting heatmaps), "r", for standard raster graphics, but with the vertical axis running down, or "r.flip", for the same but with the vertical axis running up.
Note that "r" and "r.flip" should produce the same result, if add=FALSE.
im An RImage (raster image) or ImageArray object.
rby Logical, if true, plot by rows.
... Ignored.
Except for plot_RImage, where they're passed to the as.raster method.
And also for the other plotting functions for RImage objects, where they're passed to plot_RImage, and then to the as.raster method.

Examples
plot (TCUBE)

Description
Plot multiple images, or multiple channels.
Note that init.image.board initializes the plot (in base graphics only) and returns the previous par settings.
Subsequent calls are required to plot each image.
Alternatively, you can use the plot method for ImageArray(s).

Usage
init.image.board (nr, nc, ..., rby=TRUE)
decomp.plot (im, ..., rby=TRUE, colfs)

Arguments
im An RImage or ImageArray
nr, nc Integers, number of rows and columns.
rby Logical, if true, fill by row first.
colfs A list of color functions. Refer to shader.
... Ignored.
See Also

RImage and ImageArray.
Refer to the above links for examples.

Description

Compute the spatial ranges of GeomObject(s).
In principle, this corresponds to the xlim/ylim arguments in the base graphics system.
(i.e. If you needed to initialize a plotting window, then the range methods give the xlim/ylim values).

Usage

## S3 method for class 'GeomArray'
range(v, ...)
## S3 method for class 'Grid'
range(v, ...)
## S3 method for class 'Line'
range(v, ...)
## S3 method for class 'Points'
range(v, ...)
## S3 method for class 'Polygon'
range(v, ...)
## S3 method for class 'Text'
range(v, ...)
## S3 method for class 'VImage'
range(v, ...)

Arguments

v A GeomObject.

... Ignored.

Value

A two-column matrix.

Examples

range (TCUBE)
**Description**

Element-level formatting functions for object arrays.

**Usage**

```r
## S3 method for class 'ObjectArray'
objtag(v, ...)
## S3 method for class 'NestMatrix'
objtag(v, ...)
## S3 method for class 'PartMatrix'
objtag(v, ...)
## S3 method for class 'SectMatrix'
objtag(v, ...)
## S3 method for class 'MatrixArray'
objtag(v, ...)
## S3 method for class 'Zero'
objtag(v, ...)
## S3 method for class 'function'
objtag(v, ...)
## S3 method for class 'list'
objtag(v, ...)
## S3 method for class 'data.frame'
objtag(v, ...)
## S3 method for class 'matrix'
objtag(v, ...)
## Default S3 method:
objtag(v, ...)
## S3 method for class 'GeomObject'
objtag(v, ...)
## S3 method for class 'GeomArray'
objtag(v, ...)
## S3 method for class 'RImage'
objtag(im, ...)
## S3 method for class 'ImageArray'
objtag(im, ...)
```

**Arguments**

- `im, v` An object.
... Ignored.

Details

These functions map an object to a single compact string, regardless of the length of the object.
The format method for ObjectArray calls the objtag function for each of its elements.
To format an object of a different class, you need to write an (S3) objtag method for that class.

Value

Each method returns a single compact string.
(i.e. A length-one character vector, that’s relatively short).
If you write a new objtag method, it should do the same.

See Also

format.ObjectArray

Examples

# simple classes
alphabet.1 <- function ()
structure (LETTERS, class="alphabet.1")
alphabet.2 <- function ()
structure (sample (LETTERS), class="alphabet.2")

# simple 2x2 ObjectArray
x <- ObjectArray (c (2, 2) )
x [[1, 1]] <- alphabet.1 ()
x [[2, 1]] <- alphabet.1 ()
x [[1, 2]] <- alphabet.2 ()
x [[2, 2]] <- alphabet.2 ()

# printed with default formatting
x

# objtag methods
objtag.alphabet.1 <- function (x)
paste ("<A1 ", x [1], ":", x [26], ">", sep="")
objtag.alphabet.2 <- function (x)
paste ("<A2 ", x [1], ":", x [26], ">", sep="")

# reprinted with custom formatting
x
**46_grouped_head**  

*Grouped Head*

**Description**

Grouped head function for data.frame(s).

**Usage**

```r
## S3 method for class 'data.frame'
ghead(v, nh=3, ..., gname)
```

**Arguments**

- `v` A data.frame.
- `nh` Integer, the number of rows per group.
- `gname` A string, giving the name of the grouping variable. If missing, defaults to the first string or factor, with non-unique values.
- `...` Ignored.

**Value**

Currently, returns a formatted character matrix, however, it’s possible this may change in the future.

**See Also**

- `headt.data.frame`

**Examples**

```r
ghead(iris)
```

---

**47_combined_head_and_tail**  

*Combined Head and Tail Functions*

**Description**

Combined head and tail functions.
Usage

```r
## S3 method for class 'ObjectArray'
headt(v, nh=3, nt=nh, ...)
## S3 method for class 'MatrixLike'
headt(v, nh=3, nt=nh, ...)
## S3 method for class 'data.frame'
headt(v, nh=3, nt=nh, ...)
## S3 method for class 'matrix'
headt(v, nh=3, nt=nh, ...)

## S3 method for class 'raster'
headt(v, nh=3, nt=nh, ...)

## Default S3 method:
headt(v, nh=3, nt=nh, ...)
```

Arguments

- `v` A suitable object.
- `nh, nt` Integers, the number of head/tail elements/rows.
  In matrix-based objects, can also be a vector of length two.
  (In which case, the second value applies to the number of columns).
  Note that currently, in PartMatrix and SectMatrix objects, nh and nt includes formatted lines.
  However, this may change, in the future.
- `...` Ignored.

Value

Currently, some of the functions return formatted character matrices, however, it’s possible this may change in the future.

See Also

`ghead.data.frame`

Examples

```r
x <- matrix (1:2700, 30, 90)
rsep <- seq (3, 27, 3)
csep <- seq (3, 87, 3)

pm <- as.PartMatrix (x, rsep, csep)

headt (pm, 10, c (5, 1) )
```
Description

Convenience functions to test uniqueness, or to return the number of unique/duplication values, or a unique sorted vector.

The which.unique function returns a vector of indices. This will produce a vector of unique values, if the original vector is subsetted using the indices.

Usage

\[
\begin{align*}
\text{is.eeach.equal} & \quad (v) \\
\text{is.eeach.unique} & \quad (v) \\
\text{n.unique} & \quad (v) \\
\text{n.duplicated} & \quad (v) \\
\text{which.unique} & \quad (v) \\
\text{usv} & \quad (v)
\end{align*}
\]

Arguments

- \( v \) An integer/numeric vector.

See Also

- Sequence Properties
- Optimization-Related Functions
- Other Sequence Functions

Examples

\[
\begin{align*}
x & \leftarrow c(2, 2, 1, 10, 12) \\
n.unique & \quad (x) \\
usv & \quad (x)
\end{align*}
\]
**Description**

Functions to test basic sequence properties, excluding unique-related properties.

**Usage**

- `is.sttincreasing(v)`
- `is.sttdecreasing(v)`
- `is.nondecreasing(v)`
- `is.nonincreasing(v)`

**Arguments**

- `v` An integer/numeric vector.

**Value**

All functions return true or false.

**See Also**

- Unique-Related Functions
- Optimization-Related Functions
- Other Sequence Functions

**Examples**

```r
x <- c(0, 0, 1, 2)

is.sttincreasing(x)
is.nondecreasing(x)
```

---

**Description**

Sequence based functions, involving indices (or intervals) of minima or maxima.
Usage

which.mins (v, ..., what="I", ends=FALSE)
which.maxs (v, ..., what="I", ends=FALSE)
which.opts (v, ..., what="I", ends=FALSE)

which.level (v, ..., what="intervals")

which.decreasing (v, ..., what="intervals")
which.increasing (v, ..., what="intervals")
which.nondecreasing (v, ..., what="intervals")
which.nonincreasing (v, ..., what="intervals")
which.nonlevel (v, ..., what="intervals")

mins (v, ..., allow.intervals=FALSE, ends=FALSE)
maxs (v, ..., allow.intervals=FALSE, ends=FALSE)
opts (v, ..., allow.intervals=FALSE, ends=FALSE)

Arguments

v
   An integer/numeric vector, with length two or more.
allow.intervals
   Logical, if true, optimal sections (which two or more equal values) are allowed, if false and there are optimal sections, an error is generated.
what
   String, either:
   "I" (integer vector of indices)
   "intervals" (two-column integer matrix, representing intervals)
   "NMP" (integer vector, same as "NMP-"
   "NMP." (integer vector of indices, near mid points, rounds down given even length subintervals)
   "NMP+" (integer vector of indices, near mid points, rounds up given even length subintervals)
   "XMP" (numeric vector of exact midpoints)
   "first" (integer vector of first indices)
   "last" (integer vector of last indices)
ends
   Logical vector, of length one or two, to include the first/last values as possible optima.
...
   Ignored.

Details

These functions are designed to find indices or subintervals matching a pattern.
When what="I", the solutions need to be single indices.
Otherwise, an error is generated.
(Note that "I" is not allowed, for the which.level function).
When what="intervals" the functions return the starting and ending indices of intervals.
For other return types, a single index is returned for each interval.
Value

All "which" functions return an integer vector, which the exception of what="intervals" which returns a two-column integer matrix and what="XMP" which returns a numeric vector.

The mins and maxs functions return values from the original vector.
They may contain duplicated values.

See Also

Unique-Related Functions
Sequence Properties
(Other than unique-related properties).
Other Sequence Functions

Examples

x <- c (2, 1, 0, 1, 2, 1, 0, 1, 2)
which.mins (x)
mins (x)

Description

Compute midpoints and endpoints from a vector.

Usage

midpoints (v)
endpoints (v)

Arguments

v An integer/numeric vector.

Value

The midpoints function returns a vector, with a length one less than the original vector.
The endpoints function returns a length two vector.

See Also

Unique-Related Functions
Sequence Properties
(Other than unique-related properties).
Optimization-Related Functions
Examples

```r
x <- 1:4

midpoints(x)
endpoints(x)
```

---

**Description**

Computes the cartesian product of one or more sets.

By default, a matrix is returned, if possible.

Each row represents one combination.

In some cases, `as.list` (to return lists) needs to be set to `true`.
(If there's a lack of supporting methods).

**Usage**

```r
cprod(..., sets = list(...), gby.last=FALSE, as.list=FALSE)
```

**Arguments**

- `sets` List containing one or more vectors.
  Note that standard vectors can be replaced with other objects.
  However, each object requires a `rep` method with an `each` and `times` argument, and you need to set `as.list` to `true`.

- `gby.last` Logical, if `false` (default), group by the first set, then the second, and so on.
  If `true`, group by the last set, then the second to last, and so on.

- `as.list` Logical, if `true`, return a list, rather than a matrix.
  Each element corresponds to one column from the corresponding matrix.

- `...` Argument list containing one or more standard vectors.
  Often, easier way to specify sets, above.

**Examples**

```r
cprod(1:2, 1:4)
cprod(1:2, 1:4, gby.last=TRUE)
```
Description

The seqt function is just a wrapper for the base::seq function, which multiples the result by 2 pi.
Note that the eq.arot2 and eq.brot2 functions can be used to generate a circular transformation matrix.

Usage

seqt (...)  

Arguments

...  Arguments for the base::seq function

Examples

seqt (0, 1, 10)

Description

Constructors for S4 objects, containing a matrix (gv) and coordinates (x and y), or (xb and yb).
Essentially, the gv slot is equivalent to the matrix returned by base::outer, over the x and y values.
DGrid is for integer inputs and CGrid is for numeric inputs.
The StepGrid version is for step functions, where xb and yb represent breakpoints.
The gv values are evaluated using the midpoints.
The xgrid.hdim function and related functions, can be used to produce arrays in higher dimensions.

Usage

DGrid (x = 1:nrow (gv), y = 1:ncol (gv), gv)
CGrid (x = 1:nrow (gv), y = 1:ncol (gv), gv)
StepGrid (xb = 0:nrow (gv) + 0.5, yb = 0:ncol (gv) + 0.5, gv)
dgrid (sf='*', xlim, ylim=xlim, ..., cf, iterate=FALSE, x, y)
cgrid (sf='*', xlim, ylim=xlim, ..., n=8, cf, iterate=FALSE, x, y)
stepgrid (sf='*', xlim, ylim=xlim, ..., n=8, cf, iterate=FALSE, xb, yb)
Arguments

sf
A suitable function.
It’s first argument should be a matrix/list/etc, if cf provided, otherwise it should have leading arguments, for x and y.
xlim, ylim
Length-two integer (for DGrid) or numeric (for CGrid/SGrid) vectors.
These give the limits for the sequences.
n
Integer vector of length one or two, the number or rows and columns.
For CGrid, this is also the length of the x and y sequences.
For StepGrid, xb and yb will be one longer.
iterate
Logical, if true, evaluate the function with one bin/point, at a time. Note that this is not done in an efficient way, you’ll get a faster result if you vectorize your function.
cf
A function of two vectors, used to combine x and y into a single object.
The sf function is called using the result of the cf function.
x, y
Integer or numeric vectors.
xb, yb
Same as x and y, but one longer, for breakpoints/endpoints.
gv
A matrix of grid values, equivalent to the result of sf.
...
Currently, ignored.

Value

An DGrid, CGrid or SGrid object.

See Also
dgrid.hdim, cgrid.hdim, xgrid.hdim

Examples

f <- function (v)
  v[,1] + v[,2]

#simple addition problem
dgrid (`+`, c(0, 4), c(1, 5))

#same, but f operates on matrix
dgrid (f, c(0, 4), c(1, 5), cf=cbind)

Description

Outer product generalizations, allowing an arbitrary function and an arbitrary number of variables. Note that the DGrid and CGrid functions provide additional functionality in the 2D case.
Usage

dgrid.hdim (sf=`*`, ..., cf, lims = rbind (...), drop=TRUE, iterate=FALSE)
cgrid.hdim (sf=`*`, ..., cf, lims = rbind (...), drop=TRUE, iterate=FALSE, n=10)
xgrid.hdim (sf=`*`, ..., cf, seqs = list (...), drop=TRUE, iterate=FALSE)

Arguments

sf
A suitable function.
By default, the number of arguments needs to match the main argument list.
(But a single argument is required if cf is not missing).
And the length of the return value needs to match the length of the arguments.

lims
A two-column matrix of limits.
If a row is constant, then a constant (rather than sequence) is used for evaluation.
(And if drop is true, then there will be one less dimensions in the result).

seqs
A list of vectors.
...
Alternative ways of specifying lims and seqs, see above.
In dgrid.hdim and cgrid.hdim these should be pairs of integers and numerics,
respectively.
(Or a single value, for a constant).

drop
Logical, if true, array dimensions with one value are dropped.
iterate
Logical, if true, evaluate the function with one bin/point, at a time.
Note that this is not done in an efficient way, you’ll get a faster result if you
vectorize your function.

cf
A function used to combine the main arguments.
n
Integer vector, of length one, or equal to the length of the main arguments, giving
the number of evaluation points, in each dimension.
Ignored, where the main arguments are constants.
(Rather than pairs defining ranges, or sequences).

Value

An array.

By default (with drop=TRUE), the number of dimensions will equal the number of main arguments,
except where any dimension has a single evaluation bin/point.

See Also

FGrid-class

Examples

f3a <- function (x, y, z)
  x + 10 * y + 100 * z
f3b <- function (x)
  x [,1] + 10 * x [,2] + 100 * x [,3]
x <- seq(0, 10, length.out=4)
xgrid.hd (f3a, x, x, x)

#same as above, but using xlim/ylim style input
cgrid.hd (f3a, c(0, 10), c(0, 10), c(0, 10), n=4)
cgrid.hd (f3b, c(0, 10), c(0, 10), c(0, 10), cf=cbind, n=4)

#drop argument
#(here, the default drops the first dimension)
cgrid.hd (f3a, 0, c(0, 10), c(0, 10), drop=FALSE, n=4)
cgrid.hd (f3a, 0, c(0, 10), c(0, 10), n=4)

#dropping two dimensions

cgrid.hd (f3a, c(0, 10), 0, 0, n=4)

#different n values
cgrid.hd (f3a, c(0, 10), c(0, 10), c(0, 10), n=c(2, 3, 4))

---

70_transformation_functions

*Transformation Functions.*

**Description**

Transformation functions for GeomArray and MatrixArray objects.
The generalize the binary operators, used for Matrix Array Multiplication.

**Usage**

vt3.mmult (a, b)
vt3.mmultc (a, b, ..., gby.rhs=FALSE)

vt3.gmult.pre (a, b)
vt3.gmultc.pre (a, b, ..., gby.trans=TRUE, rolling=TRUE)

vt3.gmult.post (a, b)
vt3.gmultc.post (a, b, ..., gby.trans=TRUE, rolling=TRUE)

vt3.cmult.pre (a, b, ..., pack=TRUE)
vt3.rmult.post (a, b, pack=TRUE)

**Arguments**

a, b

In simple matrix-array multiplication, matrices or MatrixArray(s).

In pre-multiplication, a is the transformation object and b is the data object.
In post-multiplication, a is the data object and b is the transformation object.
The transformation object should be a matrix or MatrixArray.

The data object should be a matrix, GeomObject, GeomArray or MatrixArray. Except for row-wise and col-wise functions (vt3.cmult.pre/vt3.rmult.post), where the data object should be a matrix.

- **gby.rhs**: Logical, if true, group by the sub-objects within the RHS object.
- **gby.trans**: Logical, if true (default), group by (or nest by) sub-objects within the transformation object.
- **rolling**: Logical, if true (default), extend the arrays (repeating where necessary), otherwise, nest the expansions.
- **pack**: Logical, if true (the default) add/strip and extra row/column to the data object. This allows a 3 by 3 transformation matrix to be multiply a two-column data matrix.

- Ignored, for GeomObject(s).
- Ignored.

**Details**

Reiterating, these functions generalize the binary operators, used for Matrix Array Multiplication. Please refer to them, for an overview.

**Value**

- A matrix for the vt3.cmult.pre/vt3.rmult.post functions. (These are used for row-wise and column-wise matrix multiplication).
- A matrix, GeomObject, GeomArray or MatrixArray for other functions.

By default, cartesian-based multiplication uses the rolling option, where the length of the output will match the length of the cartesian product.

- If the nested/nonrolling option is used, then the length will match the length of the grouping operand. (This is determined by the gby.rhs and gby.trans args).

**See Also**

Matrix Array Multiplication

**Examples**

```r
x <- matrix(1:4, 2, 2)
vt3.gmult.pre(ascl2(1:4), x)
```
Description

Functions for Mapping 3D GeomObject(s) to 2D GeomObject(s).

Usage

## S3 method for class 'GeomArray'
vt3.proj(v, proj="xy", sort=TRUE, ...)

## S3 method for class 'Grid'
vt3.proj(v, proj="xy", ...)
## S3 method for class 'Line'
vt3.proj(v, proj="xy", ...)
## S3 method for class 'Point'
vt3.proj(v, proj="xy", ...)
## S3 method for class 'Polygon'
vt3.proj(v, proj="xy", ...)
## S3 method for class 'Text'
vt3.proj(v, proj="xy", ...)

Arguments

- **v** A 3D GeomObject.
- **proj** String, including "xy", "xz" or "yz"
- **sort** Logical, if true, sort the GeomArray, which is necessary to plot the corresponding 2D-based arrays correctly.
- ... Ignored.

Value

A 2D GeomObject.

Note that plotting a 3D object, automatically projects it.

See Also

range.GeomArray

Examples

```R
range (TCUBE)
range (vt3.proj (TCUBE) )
range (vt3.proj (TCUBE, "xz") )
```
Description

Constructors for constructing one or more transformation matrices.

Usage

ascl2 (x, y=x, ..., about)
atrl2 (x=0, y=0)
arot2 (theta, ..., about)

ascl3 (x, y=x, z=x)
atrl3 (x=0, y=0, z=0)
arot3x (theta)
arot3y (theta)
arot3z (theta)

Arguments

x, y, z Numeric vectors, x, y and z transformation parameters.
theta Numeric vector, the angle in radians.
about Length-two numeric vector, giving center point for transformation.
... Ignored.

Details

These matrices are designed for pre-multiplication. They go before the data matrix.
Pre and post scaling matrices are the same.
In 2d rotation matrices, positive theta values rotate counter-clockwise.
3d rotation matrices, are the same as the 2d matrices, except for having an extra row and column.
(Whether they’re clockwise or counter-clockwise, depends on one’s coordinate system).

Value

A matrix or a 1d MatrixArray.
2d transformations are 3x3 matrices and 3d transformations are 4x4 matrices.
Post-Multiplication Transformation Matrices

Description

Constructors for constructing one or more transformation matrices.

Usage

bscl2 (x, y=x, ..., about)
btrl2 (x=0, y=0)
brot2 (theta, ..., about)

bscl3 (x, y=x, z=x)
btrl3 (x=0, y=0, z=0)

brot3x (theta)
brot3y (theta)
brot3z (theta)

Arguments

x, y, z  Numeric vectors, x, y and z transformation parameters.
theta  Numeric vector, the angle in radians.
about  Length-two numeric vector, giving center point for transformation.
...  Ignored.

Examples

# 2d static examples
# centered hexagon
v <- regPolygon (6)
vt3.testplot (v)

# four hexagons, different sizes
v2 <- ascl2 (seq (1.3, 0.7,, 4) ) %*% v
vt3.testplot (v2)
Details

These matrices are designed for post-multiplication.
They go after the data matrix.
Pre and post scaling matrices are the same.
In 2d rotation matrices, positive theta values rotate counter-clockwise.
3d rotation matrices, are the same as the 2d matrices, except for having an extra row and column.
(Whether they’re clockwise or counter-clockwise, depends on one’s coordinate system).

Value

A matrix or a 1d MatrixArray.
2d transformations are 3x3 matrices and 3d transformations are 4x4 matrices.

See Also

MatrixArray
Binary Operators
This describes how to perform operations on MatrixArray objects.
Pre-Multiplication Transformation Matrices
vt3.testplot, vt3.testplot

Examples

#2d static examples
v <- regPolygon (6)
vt3.testplot (v)

#four hexagons, different sizes
v2 <- v %*% bscl2 (seq (1.3, 0.7,, 4) )
vt3.testplot (v2)
Arguments

- n: Integer, the number of distinct points.
- start: Numeric, giving the starting angle in radians.
- ...: Ignored.

Details

These functions are wrappers for `arot2` and `brot2`. They compute a vector of equally-spaced theta values, around a circle. Then construct the transformation matrices using the theta values. This can be used to create polygons.

Note that swapping functions, reverses the direction of rotation. (e.g. Changing `eq.arot2` to `eq.brot2`).

Value

A `MatrixArray` of 3x3 rotation matrices.

See Also

- `Binary Operators`
- `MatrixArray`
- `Pre-Multiplication Transformation Matrices`
- `Post-Multiplication Transformation Matrices`
- `vt3.testplot`

Examples

```r
# equivalent to:
# vt3.testplot (regPolygon (6, d=4) )
ps <- c (0, 4) %|*% eq.brot2 (6, start = pi / 6)
v <- as.Polygon (ps)
vt3.testplot (v)
```

Description

These functions return functions (or a list of functions). The resulting functions, map a matrix of numeric values to a character matrix of colors.

They are designed for rendering single channel images, and also for rendering multiple channels, separately. (In contrast to rendering a multichannel image, as a single raster image).
**Usage**

```r
vt3.linear.shader (col0="black", col1="white", min=0, max=1, reverse=FALSE)
```

```r
vt3.rgb.shaders ()
vt3.hcl.shaders ()
vt3.hsv.shaders ()
```

```r
vt3.hue.shader (hmin=0, hmax=360, ..., c=50, l=50, na.col="white")
```

**Arguments**

- `hmin, hmax` Single numeric values, the min and max hue values.
- `c, l` Single numeric values, the C and L values in HCL color space.
- `na.col` The color returned for NA values.
- `col0, col1` Strings, the lower and upper colors.
- `min, max` The min and max values to interpolate between.
- `reverse` Logical, if true, it’s equivalent to swapping min and max.
- `...` Ignored.

**Value**

- `vt3.linear.shader` returns a function.
- The others return a list of four functions.

**See Also**

- `as.raster.SImage`, `as.raster.MImage`

**Examples**

```r
colf <- vt3.linear.shader ()
colf (c (0.33, 0.67) )
```

---

**Description**

Functions for standardizing R color strings.

Note that the `mapcol` function can be used to map between color spaces.
Usage

is.opaque (cols, ..., single.flag=TRUE)
standardize.cols (cols, ..., nchannels, as.list=FALSE, as.array=FALSE)
vt3.prep.cols (cols, nchannels, single.flag=TRUE, ..., as.list=FALSE, as.array=FALSE)

Arguments

cols A character vector/matrix/array, including color names or hex values for sRGB/sRGB-A space.
single.flag Logical, if true, return a single TRUE/FALSE value.
nchannels Integer, either three or four, determining the number of output channels. If missing, there will be three if all input colors are opaque, otherwise four.
as.list, as.array Logical values, if true, return a list or array, see value section.
... Ignored.

Details

The is.opaque function is used to test if the input contains any non-opaque colors.

The standardize.cols function converts all colors into a hex strings. The result will be all 3 channel or 4 channel values, not a mixture of two.

The vt3.prep.cols function is a combination of the two functions above.

Value

By default, is.opaque returns a single logical value. If single.flag is false, it returns a logical vector/matrix/array matching the dimensions of cols.

By default, standardize.cols returns a character vector/matrix/array matching the dimensions of cols. If as.list or as.array are true, it returns a list or array, respectively. In lists, each element corresponds to each channel, and in arrays, the last dimension corresponds to each channel.

vt3.prep.cols returns a length two list, containing the opaque flags and the standardized colors.

Examples

mixed.cols <- c("black", "white", "#000000", "#FFFFFF")
standardize.cols (mixed.cols)
standardize.cols (mixed.cols, as.list=TRUE)
86_color_transformations

Color Conversion

Description

Functions to map vectors/matrices/arrays of colors between color spaces.

THESE FUNCTIONS REQUIRE THE colorspace PACKAGE TO BE INSTALLED AND LOADED.

One exception to this requirement is mapping numeric values to strings, if the "from" color space equals the "to" color space.

Usage

mapcol (colv, ..., as.list = is.list(colv), as.string=FALSE, from="sRGB", to=from, correction=TRUE)
mapcol3 (x, y, z, alpha, ..., as.list=FALSE, as.string=FALSE, from="sRGB", to=from, correction=TRUE)

Arguments

colv A numeric array with two or more dimensions.
(Such as the result of standardize.cols with as.array=TRUE.
The last dimension should have three or four levels, which represent channels.
x, y, z, alpha Numeric vectors.
Alternative way of specifying color.
as.list, as.string Logical, if true, return a list or string, see value section.
from String, the input color space.
to String, the output color space.
Ignored, if as.string is true.
correction Logical, correct the sRGB values.
Ignored, unless converting HCL to sRGB.
... Ignored.

Details

The functions wrap functions from the colorspace package.
The to and from arguments, use names of color spaces from the colorspace package.
This includes:
"XYZ", "RGB", "sRGB", "LAB", "polarLAB", "HSV", "HLS", "LUV" and "polarLUV".
In addition to HCL, with is the same as polarLUV, but the the arguments in a different order.
The same labels are used for three channel and four channel images.
Value

By default, these functions return arrays.
Where the last dimension corresponds to channel.

If as.string is true, a character vector/matrix/array is returned.
If as.list is true, a list is returned, with one element per channel.

See Also

standardize.cols

Examples

library (colorspace)

cols <- standardize.cols (c ("black", "white", "#000000", "#FFFFFF"), as.array=TRUE)

#hues
mapcol (cols, to="HCL")[,1]
Index

[.GeomArray (01_subsetting_operators), 5
[.ImageArray (01_subsetting_operators), 5
[.MatrixArray
(01_subsetting_operators), 5
[.ObjectArray
(01_subsetting_operators), 5
[[.NestMatrix
(01_subsetting_operators), 5
[[.PartMatrix
(01_subsetting_operators), 5
[[<-.GeomArray
(01_subsetting_operators), 5
[[<-.NestMatrix
(01_subsetting_operators), 5
[[<-.ObjectArray
(01_subsetting_operators), 5
[[<-.PartMatrix
(01_subsetting_operators), 5
%*% (00_binary_operators), 2
%**%, MatrixArray, MatrixArray-method
(00_binary_operators), 2
%**%, MatrixArray, matrix-method
(00_binary_operators), 2
%**%, matrix, MatrixArray-method
(00_binary_operators), 2
%*% (00_binary_operators), 2
%*% (00_binary_operators), 2
00_binary_operators, 2
01_subsetting_operators, 5
02_language-like_functions, 6
03_constants, 7
04_new_generic_functions, 8
10_raster_images, 9
15_simple_geometric_objects, 11
16_other_main_geometric_objects, 12
17_regular_polygons, 13
18_rectangular_grids, 14
20_ObjectArray, 15
21_ImageArray, 17
22_GeomArray, 18
25_block_matrices, 19
26_MatrixArray, 22
30_is_functions, 23
31_replication_functions, 24
32_combine_methods, 25
33_standard_methods, 26
34_as_raster_methods, 28
36_other_conversion_functions, 28
40_test_plots, 29
41_plot_methods, 31
42_image_boards, 32
44_range_methods, 33
45_objtag_methods, 34
46_grouped_head, 36
47_combined_head_and_tail, 36
50_unique-related_functions, 38
51_sequence_properties, 39
52_which_functions, 39
55_sequence_properties_2, 41
60_cartesian_products, 42
61_special_sequences, 43
65_outier_generalizations_1, 43
66_outier_generalizations_2, 44
70_transformation_functions, 46
71_projections, 48
75_pre-multiplication_transformation_matrices, 49
76_post-multiplication_transformation_matrices, 50
77_other_transformation_matrices, 51
84_shaders, 52
85_standardized_colors, 53
86_color_transformations, 55
arot2, 52
aro2
(75_pre-multiplication_transformation_matrices, 49)
aro3x
(75_pre-multiplication_transformation_matrices, 49)
aro3y
(75_pre-multiplication_transformation_matrices, 49)
aro3z
(75_pre-multiplication_transformation_matrices, 49)
as.GeomArray
(22_GeomArray, 18)
as.Grid
(15_simple_geometric_objects, 11)
as.ImageArray
(21_ImageArray, 17)
as.Line
(15_simple_geometric_objects, 11)
as.list.MImage
(36_other_conversion_functions, 28)
as.list.ObjectArray
(36_other_conversion_functions, 28)
as.matrix.NestMatrix
(36_other_conversion_functions, 28)
as.matrix.SectMatrix
(36_other_conversion_functions, 28)
as.matrix.SImage
(36_other_conversion_functions, 28)
as.MatrixArray
(26_MatrixArray, 22)
as.MImage
(10_raster_images, 9)
as.NestMatrix
(25_block_matrices, 19)
as.ObjectArray
(20_ObjectArray, 15)
as.PartMatrix
(25_block_matrices, 19)
as.Points
(15_simple_geometric_objects, 11)
as.Polygon
(15_simple_geometric_objects, 11)
as.raster.MImage
(34_as_raster_methods, 28)
as.raster.SImage
as.raster.SImage
as.raster.SImage
ascl2
(75_pre-multiplication_transformation_matrices, 49)
ascl3
(75_pre-multiplication_transformation_matrices, 49)
atrl2
(75_pre-multiplication_transformation_matrices, 49)
atrl3
(75_pre-multiplication_transformation_matrices, 49)
brot2
(76_post-multiplication_transformation_matrices, 50)
brot3x
(76_post-multiplication_transformation_matrices, 50)
brot3y
(76_post-multiplication_transformation_matrices, 50)
brot3z
(76_post-multiplication_transformation_matrices, 50)
bscl2
(76_post-multiplication_transformation_matrices, 50)
bscl3
(76_post-multiplication_transformation_matrices, 50)
btrl2
(76_post-multiplication_transformation_matrices, 50)
btrl3
(76_post-multiplication_transformation_matrices, 50)
c.ObjectArray
(32_combine_methods, 25)
INDEX

CGrid, 44
CGrid(65_outer_generalizations_1), 43
cgrid, 12
cgrid(65_outer_generalizations_1), 43
CGrid-class
(65_outer_generalizations_1), 43
cgrid.hdim, 44
cgrid.hdim
(66_outer_generalizations_2), 44
CLASS(02_language-like_functions), 6
Combined Head and Tail Functions, 9
Combined Head and Tail Functions
(47_combined_head_and_tail), 36
cprod(60_cartesian_products), 42
CUBE, 30
CUBE(03_constants), 7
Cuboid(17_regular_polygons), 13
Cuboid-class(17_regular_polygons), 13
decomp.plot(42_image_boards), 32
DGrid, 44
DGrid(65_outer_generalizations_1), 43
dgrid(65_outer_generalizations_1), 43
DGrid-class
(65_outer_generalizations_1), 43
dgrid.hdim, 44
dgrid.hdim
(66_outer_generalizations_2), 44
dim.ObjectArray(33_standard_methods), 26
dim.SectMatrix(33_standard_methods), 26
dimnames.ObjectArray
(33_standard_methods), 26
dimnames.SectMatrix
(33_standard_methods), 26
dimnames<-.ObjectArray
(33_standard_methods), 26
dimnames<-.SectMatrix
(33_standard_methods), 26
endpoints(55_sequence_properties_2), 41
eq.brot2, 43
eq.brot2
(77_other_transformation_matrices), 51
FGrid-class, 15, 45
FGrid-class
(65_outer_generalizations_1), 43
format.ObjectArray, 35
format.ObjectArray
(33_standard_methods), 26
format.SectMatrix
(33_standard_methods), 26
format.ZERO(33_standard_methods), 26
GeomArray, 2, 16, 23, 46
GeomArray(22_GeomArray), 18
GeomArray-class(22_GeomArray), 18
GeomArray2(22_GeomArray), 18
GeomArray3(22_GeomArray), 18
GeomObject-class
(15_simple_geometric_objects), 11
get.channel(10_raster_images), 9
getSect(01_subsetting_operators), 5
ghead(04_new_generic_functions), 8
ghead.data.frame, 9, 37
ghead.data.frame(46_grouped_head), 36
Grid, 11, 14, 15
Grid(16_other_main_geometric_objects), 12
Grid-class
(16_other_main_geometric_objects), 12
head.raster(33_standard_methods), 26
head.VecLike(33_standard_methods), 26
headt(04_new_generic_functions), 8
headt.data.frame, 36
headt.data.frame
(47_combined_head_and_tail), 36
headt.default
(47_combined_head_and_tail), 36
headt.matrix
(47_combined_head_and_tail), 36
headt.MatrixLike
(47_combined_head_and_tail), 36
objtag.NestMatrix (45_objtag_methods), 34
objtag.ObjectArray (45_objtag_methods), 34
objtag.PartMatrix (45_objtag_methods), 34
objtag.RImage (45_objtag_methods), 34
objtag.SectMatrix (45_objtag_methods), 34
objtag.Zero (45_objtag_methods), 34
Optimization-Related Functions, 38, 39, 41
Optimization-Related Functions (52_which_functions), 39
opts (52_which_functions), 39
Other Sequence Functions, 38, 39, 41
Other Sequence Functions (55_sequence_properties_2), 41
Other Transformation Matrices (77_other_transformationMatrices), 51
PartMatrix, 6, 16, 19
PartMatrix (25_block_matrices), 19
PartMatrix-class (25_block_matrices), 19
plot.VecLike (04_new_generic_functions), 8
plot.RImage (41_plot_methods), 31
plotv (04_new_generic_functions), 8
plotv.GeomArray (41_plot_methods), 31
plotv.Grid (41_plot_methods), 31
plotv.ImageArray (41_plot_methods), 31
plotv.Line (41_plot_methods), 31
plotv.Points (41_plot_methods), 31
plotv.Polygon (41_plot_methods), 31
plotv.RImage (41_plot_methods), 31
plotv.Text (41_plot_methods), 31
plotv.VImage (41_plot_methods), 31
Points, 13
Points (15_simple_geometric_objects), 11
Points-class (15_simple_geometric_objects), 11
Polygon, 13
Polygon (15_simple_geometric_objects), 11
Polygon-class (15_simple_geometric_objects), 11
Post-Multiplication Transformation Matrices, 4, 30, 50, 52
Post-Multiplication Transformation Matrices (76_post-multiplication_transformationMatrices), 50
Pre-Multiplication Transformation Matrices, 4, 30, 51, 52
Pre-Multiplication Transformation Matrices (75_pre-multiplication_transformationMatrices), 49
print.GeomObject (33_standard_methods), 26
print.MImage (33_standard_methods), 26
print.SImage (33_standard_methods), 26
print.VecLike (33_standard_methods), 26
range.GeomArray, 48
range.GeomArray (44_range_methods), 33
range.Grid (44_range_methods), 33
range.Line (44_range_methods), 33
range.Points (44_range_methods), 33
range.Polygon (44_range_methods), 33
range.Text (44_range_methods), 33
range.VImage (44_range_methods), 33
Rect (17_regular_polygons), 13
Rect-class (17_regular_polygons), 13
rectGrid, 12, 13
rectGrid (18_rectangular_grids), 14
rectVImage, 12, 13
rectVImage (18_rectangular_grids), 14
regPolygon, 11
regPolygon (17_regular_polygons), 13
rep.GeomArray (31_replication_functions), 24
rep.ImageArray (31_replication_functions), 24
rep.MatrixArray, 9
rep.MatrixArray (31_replication_functions), 24
rep.ObjectArray (31_replication_functions), 24
rep2 (04_new_generic_functions), 8
rep2.GeomArray (31_replication_functions), 24
rep2.ImageArray (31_replication_functions), 24
rep2.MatrixArray (31_replication_functions), 24
rep2.MatrixArray
  (31_replication_functions), 24
rep2.ObjectArray
  (31_replication_functions), 24
RImage, 17, 28, 33
RImage (10_raster_images), 9
RImage-class, 13
RImage-class (10_raster_images), 9
SectMatrix, 6, 16, 19
SectMatrix (25_block_matrices), 19
SectMatrix-class (25_block_matrices), 19
seqt (61_special_sequences), 43
Sequence Properties, 38, 41
Sequence Properties
  (51_sequence_properties), 39
shader, 28, 32
shader (84_shaders), 52
SImage-class (10_raster_images), 9
spin3d, 8
spin3d (40_test_plots), 29
SQUARE, 30
SQUARE (03_constants), 7
Standard Methods, 16, 19, 21
Standard Methods (33_standard_methods), 26
standardize.cols, 55, 56
standardize.cols
  (85_standardized_colors), 53
StepGrid (65_outer_generalizations_1), 43
stepgrid (65_outer_generalizations_1), 43
StepGrid-class
  (65_outer_generalizations_1), 43
Subsetting Operators, 16, 19, 21
Subsetting Operators
  (01_subsetting_operators), 5
tail.raster (33_standard_methods), 26
tail.VecLike (33_standard_methods), 26
TCUBE (03_constants), 7
Test Plots (40_test_plots), 29
Text (15_simple_geometric_objects), 11
Text-class
  (15_simple_geometric_objects), 11
Transformation Functions, 2
Transformation Functions
  (70_transformation_functions), 46
Unique-Related Functions, 39, 41
Unique-Related Functions
  (50_unique-related_functions), 38
usv (50_unique-related_functions), 38
VecLike-class (20_ObjectArray), 15
VImage, 9, 11, 14, 15, 17
VImage
  (16_other_main_geometric_objects), 12
VImage-class
  (16_other_main_geometric_objects), 12
VMap, 19
VMap (25_block_matrices), 19
VMap-class (25_block_matrices), 19
vt3.cmult.pre
  (70_transformation_functions), 46
vt3.gmult.post
  (70_transformation_functions), 46
vt3.gmult.pre
  (70_transformation_functions), 46
vt3.gmultc.post, 4
vt3.gmultc.post
  (70_transformation_functions), 46
vt3.gmultc.pre, 4
vt3.gmultc.pre
  (70_transformation_functions), 46
vt3.hcl.shaders (84_shaders), 52
vt3.hsv.shaders (84_shaders), 52
vt3.hue.shader (84_shaders), 52
vt3.linear.shader (84_shaders), 52
vt3.mmult
  (70_transformation_functions), 46
vt3.mmultc
  (70_transformation_functions), 46
vt3.prep.cols (85_standardized_colors), 53
vt3.proj (04_new_generic_functions), 8
vt3.proj.GeomArray (71_projections), 48
vt3.proj.Grid (71_projections), 48
vt3.proj.Line (71_projections), 48
vt3.proj.Point (71_projections), 48
vt3.proj.Polygon (71_projections), 48
vt3.proj.Text (71_projections), 48
vt3.rgb.shaders (84_shaders), 52
vt3.rmult.post
   (70_transformation_functions), 46
vt3.testplot. 4, 7, 8, 50–52
vt3.testplot (40_test_plots), 29
vt3.testplot3. 4, 7, 8
vt3.testplot3 (40_test_plots), 29
which.decreasing (52_which_functions), 39
which.increasing (52_which_functions), 39
which.level (52_which_functions), 39
which.maxs (52_which_functions), 39
which.mins (52_which_functions), 39
which.nondecreasing
   (52_which_functions), 39
which.nonincreasing
   (52_which_functions), 39
which.nonlevel (52_which_functions), 39
which.opts (52_which_functions), 39
which.unique
   (50_unique-related_functions), 38
xgrid.hdim, 43, 44
xgrid.hdim
   (66_outer_generalizations_2), 44
ZERO (03_constants), 7
Zero-class (03_constants), 7