Package ‘mvmesh’

February 11, 2020

Type Package

Title Multivariate Meshes and Histograms in Arbitrary Dimensions

Version 1.6

Date 2020-02-09

Author John P. Nolan

Maintainer John P. Nolan <jpnolan@american.edu>

Depends R (>= 3.0), rcdd, rgl, geometry, abind, SimplicialCubature

Description Define, manipulate and plot meshes on simplices, spheres, balls, rectangles and tubes. Directional and other multivariate histograms are provided.

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-11 21:40:06 UTC

R topics documented:

mvmesh-package *Multivariate meshes and histograms in arbitrary dimensions*

Description

Define, manipulate and plot multivariate meshes/grids in n-dimensional Euclidean space. Multivariate histograms based on these meshes are provided.

Details

A range of multivariate problems require working with simplices, spheres, balls, rectangular and tubular meshes in dimension n > 1. The multivariate histogram functions in this package provide routines to tabulate and display multivariate data. For example, directional histograms tablulate the number of points in a sequence of directions, see function `histDirectional`. Multivariate stable distributions and multivariate extreme value distributions are defined by a measure on a sphere or simplex. Also, simulation of generalized spherical laws involves a triangulation of some surface. Numerical quadrature problems on a region or surface in n space require the ability to specify and work with meshes, e.g. packages `SphericalCubature` and `SimplicialCubature`. Finally, these meshes can be used on their own to create and plot multivariate shapes not in the `rgl` package.

A key goal for this package is that the dimension n is not limited to 2 or 3, but in principle can be arbitrary. Of course, as n increases compute times and required memory will increase quickly. This package uses existing methods from computational geometry that work in arbitrary dimension. Several of these functions were written as prototypes, so getting something to work was the immediate goal, speed was not.

In this documentation we will use the term grid to mean a collection of points, usually approximately evenly spread on a solid or surface. We will use the term mesh to mean both the grid, and the grouping information that tells which points make up the simplices that triangulate/tesselate the region.

Please let me know if you find any mistakes. I will try to fix bugs promptly. Constructive comments for improvements are welcome; actually implementing any suggestions will be dependent on time constraints.

This research was supported by an agreement with Cornell University, Operations Research & Information Engineering, under contract W911NF-12-1-0385 from the Army Research Development and Engineering Command.

Version history:

- 1.0 original package
- 1.1 added functions `histDirectionalQuantileThreshold`, `histDirectionalAbsoluteeThreshold`, `HollowTube`, `SolidTube`, `Lift2UnitSimplex`, `IntersectMultipleSimplicesV`, `IntersectMultipleSimplicesH`, `Intersect2SimplicesH`
- 1.2 added new functions `mvmeshFromSVI`, `mvmeshfromSimplices`, `mvmeshFromVertices`, `rtesselation`, new argument normalize.by.area in `histDirectional`, new argument label.orthants in `histDirectional`. Speed up 3d plots.

- 1.3 replace `rtessellation` with more general `rmvmesh`; add argument 'm' to mvmeshFromSVI; rename `RectangularMesh` to `SolidRectangle`; add `HollowRectangle`; fix `Icosahedron` to correctly set 'm'.
- 1.4 fix a bug in `IntersectMultipleSimplicesH`.
- 1.5 add new return value 'which.simplex' to `TallyHrep`; for each data point x[,i], it identifies which simplex contains that point; minor change in `Intersect2SimplicesH`; change 'octant' to 'orthant' to correctly describe what happens in all dimensions; improve `IntersectMultipleSimplicesH` and `Intersect2SimplicesH`
- 1.6 correct an unused argument in documentation.

Programming details and notes

The remainder of this section describes some of the internal details of the package. It is not needed for the average user.

Points in n-dimensional space are stored in row vectors as is customary in R. All simplices considered in this package are convex. A single convex simplex can be described/stored in two ways:

- A vps x n matrix of (doubles) S; the rows S[1,], S[2,], etc. are the vertices in R^n. The simplex is the convex hull of the vertices. Note: vps stands for 'vertices per simplex'.
- An nV x n matrix of (doubles) vertices V with rows giving the points in R^n, and an integer vector of length vps called SVI (Simplex Vertex Indices) that specifies which vertices make up a simplex.

Both of these descriptions have their uses, so the core functions in this package calculate both. To store all the relevant information needed, the basic functions in this package return an object of class mvmesh. An object of class mvmesh has the following fields, extending the definitions above from a single simplex to a list of simplices:

- type - a string describing the mesh, e.g. "UnitSimplex" (see table below)
- mvmesh.type - an integer specifying the type of mesh (see table below)
- n - dimension of the space
- m - dimension of the mesh, e.g. the unit sphere in n=3 dimensions is an m=2 dimensional surface (see table below)
- vps - vertices per simplex, the number of vertices that define a simplex, which must be the same for all simplices in this mesh (see table below)
- S - an (vps x n x nS) array, with S[, ,k] specifying the vertices of k-th simplex
- V - an (nV x n) matrix giving the distinct vertices in the list of simplices (repeated vertices in S that are on common edges are removed)
- SVI - an integer (vps x nS) matrix which specifies the indices of the vertices that make up the simplices in S. SVI = Simplex Vertex Indices. SVI[,k] gives the subscripts in the vertex array V that determine the k-th simplex in S
- other fields are specific to the type of mesh. Generally, they describe the parameters that were used to generate the mesh

type	mvmesh.type	m	vps
UnitSimplex	1	n-1	n
SolidSimplex	2	n	n+1
UnitSphere, edgewise	3	n-1	n
UnitSphere, dyadic	4	n-1	n
UnitBall, edgewise	5	n	n+1
UnitBall, dyadic	6	n	n+1
SolidRectangle	7	n	2^n
Icosahedron	8	2	3
PolarSphere	9	n-1	2^(n-1)
PolarBall	10	n	2^(n-1) + 1
HollowTube	11	n-1	2*(n-1)
SolidTube	12	n	2*n
HollowRectangle	13	n	2^(n-1)

There are two generic S3 methods for objects of class mvmesh: print and plot. They are basic. The plot command only works for dimensions n=2 and n=3, is slow, and has some limitations. The main goal of this package is to provide grids/meshes in arbitrary dimensions, where plots are not possible.

This package represents points in n dimensional space as double precision numbers. This is convenient, but has potential problems. For example, determining whether points lie on a line or in a plane or on a sphere may not be possible with floating point arithmetic because coordinates can't be represented exactly. The computational geometry package rcdd on CRAN gives a way around this by using exact rational arithmetic. Using rational arithemetic works fine when points can be expressed as rational numbers, but not for points shifted by an irrational number or on more general surfaces, e.g. (sqrt(2)/2,sqrt(2)/2) is on the unit circle, but cannot be represented exactly as a rational number. Since we want to work in more situations, we use floating point numbers everywhere, accepting the fact that points may not be represented exactly. When the required package `rcdd` is loaded, it prints out a warning message about double precision numbers and encourages the use of rational arithmetic. I do not know how to suppress this message. That package warns that using doubles can lead to crashes in certain circumstances. I don't know what circumstances cause crashes; I have not seen any in the kinds of computations done in this package.

Examples

```
UnitSimplex( n=2, k=3 )
UnitBall( n=3, k= 2 )

## Not run:

plot( SolidSimplex( n=2, k=3 ), col="red" )
title("2d solid simplex")

plot( SolidSimplex( n=3, k=4 ) )
plot(  UnitSimplex( n=3,k=4), new.plot=FALSE, col="red", lwd=5 )
title3d("solid and unit simplex in 3d")
rgl.viewpoint( -45, 15)
```

```
# two plots on one window
plot( UnitSphere( n=3, k=2 ), col="blue")
mesh2 <- AffineTransform( UnitBall( n=3,k=2 ), A=diag(c(1,1,1)), shift=c(3,0,0) )
plot( mesh2, new.plot=FALSE, col="magenta" )
title3d("unit sphere and ball in 3d")


demo(mvmesh) # shows a range of meshes
demo(mvhist) # shows a range of multivariate histograms

## End(Not run)
```

`HollowRectangle` *Subdivide a hyperrectangle with a standard grid*

Description

`EdgeSubdivision` implements the

Usage

```
HollowRectangle( a, b, breaks=5, silent=FALSE )
SolidRectangle( a, b, breaks=5, silent=FALSE )
mvmeshRectBreaks( a, b, breaks, silent )
NextMultiIndex( i, n )
```

Arguments

`a`	vector specifying the "lower left" vertex of the rectangle
`b`	vector specifying the "upper right" vertex of the rectangle
`breaks`	a specification of the subdivision scheme. See details below.
`silent`	indicates whether or not to warn the caller if the subdivision determined by 'breaks' covers the whole hyperrectangle [a,b].
`i`	integer vector
`n`	integer vector

Details

`RectangularMesh` computes an rectangular mesh on the hyperrectangle [a,b] = [a[1],b[1]] x [a[2],b[2]] x ... x [a[n],b[n]]. It is similar to the function `mesh` in CRAN package plot3D, but works for dimension d=2,3,4,...

'breaks' determines how each component is divided, it is motivated by the argument breaks in `hist`. If 'breaks' is a vector of length n, then breaks[i] gives the number of evenly sized bins in coordinate i, spread out over the range [a[i],b[i]]. If 'breaks' is a single number m, then each component is subdivided into that many bins, i.e. this is equivalent to breaks=rep(m,n). Thus the default breaks=6 subdivides each coordinate into 6 bins. Finally, if a more complicated subdivision is desired, 'breaks' can a list with n fields. breaks[[i]] should be a vector of dividing points for coordinate i. See the example below. In this last case, where the bin boundaries are explictly defined, 'a' and 'b' are not used (other than a possible warning if the specified bins do not cover the rectangle given by 'a' and 'b').

Value

An object of class "mvmesh" as described in mvmesh.

Examples

```
SolidRectangle( a=c(1,3), b=c(2,7), breaks=2 )
SolidRectangle( a=c(1,3), b=c(2,7), breaks=c(4,10) )
SolidRectangle( a=c(1,3), b=c(2,7),
    breaks=list( seq(1,3,by=0.25), seq(2,7,by=1) ) )
HollowRectangle( a=c(1,3), b=c(2,7), breaks=2 )
HollowRectangle( a=c(1,3), b=c(2,7), breaks=c(4,10) )
HollowRectangle( a=c(1,3), b=c(2,7),
    breaks=list( seq(1,3,by=0.25), seq(2,7,by=1) ) )


## Not run:
plot( SolidRectangle( a=c(1,3), b=c(2,7), breaks=3 ), show.labels=TRUE )
plot( SolidRectangle( a=c(1,3), b=c(2,7), breaks=c(4,10) ), show.labels=TRUE )
plot( SolidRectangle( a=c(1,3), b=c(2,7),
    breaks=list( seq(1,3,by=0.25), seq(2,7,by=1) ) ), show.labels=TRUE )
plot( SolidRectangle( a=c(1,3), b=c(2,7), breaks=3 ), show.labels=TRUE,
    label.values=letters[1:9], col='green' )

plot( HollowRectangle( a=c(1,3,0), b=c(6,7,6), breaks=3 ), show.labels=TRUE, col='blue')
plot( HollowRectangle( a=c(1,3), b=c(2,7), breaks=3 ), show.labels=TRUE )
plot( HollowRectangle( a=c(1,3), b=c(2,7), breaks=c(4,10) ), show.labels=TRUE )
plot( HollowRectangle( a=c(1,3), b=c(2,7),
    breaks=list( seq(1,3,by=0.25), seq(2,7,by=1) ) ), show.labels=TRUE )
plot( HollowRectangle( a=c(1,3), b=c(2,7), breaks=3 ), show.labels=TRUE,
    label.values=letters[1:9], col='green' )

## End(Not run)
```

HollowTube *Define tubes in n-dimensions*

Description

Define a 'horizontal' tube, either hollow or solid, in n dimensions

Usage

```
HollowTube( n, k.x=1, k.circumference=2, method="dyadic", p=2 )
SolidTube( n, k.x=1, k.circumference=2, method="dyadic", p=2 )
```

Arguments

`n`	Dimension of the space
`k.x`	Number of subdivisions along the x[1] direction (first component)
`k.circumference`	Number of subdivisions around the circumference; note the meaning of this depends on the value of `method`.
`method`	"dyadic" or "edgewise": the former recursively subdivides the sphere to get a more uniform grid; the latter uses a faster method using one edgewise subdivision.
`p`	Power used in the l^p norm; p=2 is the Euclidean norm

Details

`HollowTube` computes an approximation to a tube, an (n-1) dimensional surface in n space. The tube is 'horizontal', e.g. the center line of the tube is the x-axis with 0 <= x[1] <= 1 and radius 1; use `AffineTransform` to rotate, stretch or translate. The mesh is basically constructed by taking the cross product of an x[1] subdivision with an (n-1) dimensional sphere; the optional arguments `k.circumference,method,p` are used in a call to `UnitSphere` to specify the sphere. The default value of p=2 gives a tube with round/Euclidean cross section; using a different p will make the cross sections of the tube a ball in the Lp norm.

`SolidTube` computes an approximation to a solid tube, an n dimensional solid in n space.

Value

an object of class "mvmesh" as described in `mvmesh`.

Examples

```
HollowTube( n=3 )
SolidTube( n=3 )

## Not run:
plot( HollowTube(  n=3, k.x=3, k.circumference=2 ), show.faces=TRUE, col='red', alpha=0.5 )
plot( SolidTube( n=3, k.x=5, k.circumference=2 ), col='blue' )

# use non-Euclidean sphere to define wall of tube
plot( HollowTube(  n=3, k.x=10, k.circumference=2, p=0.6 ), col='green')

## End(Not run)
```

mvhist *Multivariate histograms*

Description

Tabulate and plot histograms for multivariate data, including directional histograms

Usage

```
histDirectional( x, k, p=2, plot.type="default", freq=TRUE, positive.only=FALSE,
     report="summary", label.orthants=TRUE, normalize.by.area=FALSE, ... )
histDirectionalQuantileThreshold( x, probs=1, p=2, k=3, positive.only=FALSE, ... )
histDirectionalAbsoluteThreshold( x, thresholds=0, p=2, k=3, positive.only=FALSE,...)
histRectangular( x, breaks=10, plot.type="default", freq=TRUE, report="summary", ... )
histSimplex( x, S, plot.type="default", freq=TRUE, report="summary", ... )

TallyHrep( x, H, report="summary" )
DrawPillars( S, height, shift=rep(0.0,3), ... )
```

Arguments

`x`	data in an (n x d) matrix; rows are d-dimensional data vectors
`k`	number of subdivisions
`p`	power of p-norm
`freq`	TRUE for a frequency histogram, FALSE for a relative frequency histogram. See note about normalize.by.area
`normalize.by.area`	if TRUE, then the counts are normalized by the surface area of the corresponding simplex on the sphere. This is useful since in general the surface area varies and counts will vary accordingly. In particular, isotropic data will not appear isotropic without setting this to TRUE. If TRUE, the value of freq is ignored: the histogram always shows count/surface area
`breaks`	specifes the subdivision of the region; see 'breaks' in `SolidRectangle`
`plot.type`	type of plot, see details below
`positive.only`	If TRUE, look only in the first orthant
`report`	level of warning messages; one of "summary", "all", "none".
`label.orthants`	If plot.type="index", this controls whether or not the orthants are labeled on the plot.
`probs`	vector of probabilites specifying what fraction of the extremes to keep
`thresholds`	vector of thresholds specifying cutoff for extremes to keep
`...`	Optional arguments to plot
`S`	(vps x d x nS) array of simplices in V representation, see `V2Hrep`
`H`	array of simplices in H representation, see `V2Hrep`
`height`	vector of length nS giving the heights of the pillars
`shift`	shift of the pillars, typically (0,0,0) for 2d data or (0,0,z0) for 3d data

Details

Calculate and plot multivariate histograms. `histDirectional` plots a directional histogram for all the data, `histDirectionalQuantileThreshold` plots `m=length(probs)` directional histograms, with plot i using the top probs[i] fraction of the data, `histDirectionalAbsoluteThreshold` plots `m=length(cut.off)` directional histograms, with plot i using the top probs[i] fraction of the data, `histSimplex` plots histogram based on simplices specified in `S`, `histRectangular` plots histogram based on a rectangular grid,

In all cases, the bins are simplices described in the H-representation and tallied by `TallyHrep`. `TallyCones` does a similar function for cones from the origin and generated by a list of base simplices.

'plot.type' values depend on the type of plot being used. Possible values are:

- "none" - does not show a plot, just return the counts
- "index" - shows a histogram of simplex index number versus count, does not show the geometry, but works in any dimension
- "pillars" - shows a 3D plot with pillars/columns having base the shape of the simplices and height proportional to frequency counts. When the points are 2D, this works for `histRectangular` and `histSimplex`; when the points are 3D, this only works for `histRectangular`. `DrawPillars` is used to plot the pillars.
- "counts" - shows frequency counts as a number in the center of each simplex
- "radial" - `histDirectional` only, shows radial spikes proportional to the counts
- "grayscale" - `histDirectional` only, color codes simplices proportional to the counts
- "orthogonal" - `histDirectional` only, shows radial spikes proportional to the counts
- "default" - type depends on the dimension of the data and type of histogram

Value

A plot is drawn (unless plot.type="none"). A list is returned invisibly, with fields:

- counts - frequency count in each bin
- nrejects - number of x values not in any bin
- nties - number of points in more than one bin (if bins are set up to be non-overlapping, this should only occur on a shared edge between two simplices)
- nx - total number of data points in x
- rel.freq - counts/nx
- rel.rejects - nrejects/nx
- mesh - object of type mvmesh, see `mvmesh`
- plot.type - input value
- report - input value

Warning

This is experimental code, and not throughly tested. If you have problems, please let me know.

Examples

```
# two dimensional, isotropic
x <- matrix( rnorm(8000), ncol=2 )
histDirectional( x, k=1 )

## Not run:

histRectangular( x, breaks=5 )

# some directional 2-dim data
n <- 1000
A <- matrix( c(1,2, 4,1), nrow=2,ncol=2)
x2 <- matrix( 0.0, nrow=n, ncol=2 )
for (i in 1:n) { x2[i,] <- A
dev.new(); par(mfrow=c(2,2))
plot(x2,main="Raw data",col='red')
histDirectionalQuantileThreshold( x2, probs=c(1,0.25,0.1), p=1,
    positive.only=TRUE, col='green',lwd=3)
dev.new(); par(mfrow=c(2,2))
histDirectionalAbsoluteThreshold( x2, thresholds=c(0,50,100,200), p=1,
    positive.only=TRUE, col='blue',lwd=3)


# three dimensional positive data
x3 <- matrix( abs(rnorm(9000)), ncol=3 )
histDirectional( x3, k=3, positive.only=TRUE, col='blue', lwd=3 )
histRectangular( x3, breaks=4 )

demo(mvhist) # shows a range of multivariate histograms
}


## End(Not run)
```

---

mvmesh-geom *Miscellaneous computational geometry and utility functions*

Description

EdgeSubdivision calculates an equal area/volume subdivision of a simplex. AffineTransform defines a new mesh by translating all points x to x' = A Rotate2D and Rotate3D calculate rotation matrices for use by AffineTransform.

Icosahedron returns the vertices of an icosahedron with vertices on the unit sphere

Other functions are internal functions, use at your own risk.

Usage

```
EdgeSubdivision( n, k )
EdgeSubdivisionMulti( V, SVI, k, normalize = FALSE, p = 2)
ConvertBase( m, b, n)
NumVertices( n, k, single = TRUE)
PointCoord( S, color )
SimplexCoord( S, color )
SVIFromColor( S, T )

MatchRow(v, table, first = 1, last = nrow(table))
AffineTransform( mesh, A, shift )
Rotate2D( theta )
Rotate3D( theta )
Icosahedron( )

V2Hrep( S )
H2Vrep( H )
SatisfyHrep( x, Hsingle )
HrepCones( S )
IntersectMultipleSimplicesV( S1, S2 )
IntersectMultipleSimplicesH( H1, H2, skip.redundant=FALSE )
Intersect2SimplicesH( H1, H2, tessellate=FALSE, skip.redundant=FALSE )
Lift2UnitSimplex(S)
```

Arguments

`v`	a vector of length n
`table`	matrix of size m3 x n
`first`	row to start search
`last`	row to end search
`mesh`	object of class "mvmesh"
`A`	n x n matrix
`shift`	shift vector of length n
`theta`	rotation angle; in 2D, this is a single angle; in 3D is it a vector of length 3, with theta[i] giving rotation around i-th axis
`k`	number of subdivisions
`n`	dimension of simplex
`V`	matrix of vertices; each row is a point in R^n
`normalize`	TRUE to normalize vertices to lie on the unit sphere in the l^p norm
`p`	power in the l^p norm
`S,S1,S2`	matrix of size (vps x n) specifying the vertices of a single simplex; S[j,] is the j-th vertex of S
`SVI`	Simplex Vertex Indices, see `mvmesh`

m	positive integer to be converted to base 'b'
b	positive integer, the base used to expess 'x'
single	If TRUE, return only one value; if FALSE, return table of values
color	color matrix, internal matrix used by EdgeSubdivision to subdivide a simplex
T	array giving a list of color matrices
H,H1,H2	array of simplices in the H-representation, H[,,k] is the H-representation for the k-th simplex
x	matrix with columns giving the points
Hsingle	matrix giving the H-representation of a single simplex
tessellate	TRUE to tessellate the resulting intersection
skip.redundant	TRUE to skip the call to rcdd::redundant

Details

AffineTransform computes a new mesh from a previous one, with each vertex v being replaced by A Rotate3D computes a 3D rotation matrix.

Icosahedron returns the vertices of the icosahedron with vertices on the unit sphere

H2Vrep converts from the half-space (H) representation to the vertex (V) representation of a simplex. V2Hrep converts from the V-representation to the H-representation. It is assumed that all the resulting value are of the same dimension. If this is not the case, an error will occur. To work with such cases, call the function separately for each simplex and save the result in different size objects. The one place where this can occur with mvmesh objects is with a PolarSphere or PolarBall: at the places where polar coordinates are nonunique, vertices will repeat and the H-representation will have fewer constraints than other simplices.

IntersectMultipleSimplicesV computes the pairwise intersection of two lists of simplices given in the V-representation. IntersectMultipleSimplicesH computes the pairwise intersection of two lists of simplices given in the H-representation. Intersect2SimplicesH computes the intersection of two simplices, both specified in the H-representation.

Lift2UnitSimplex reverses the projection from the unit simplex in n-space to the first (n-1) coordinates. That is, it 'lifts' each (n-1) dimensional simplex in R^(n-1) to the unit simplex in R^n by appending an n-th coordinate, with x[n] <- 1-sum(x[1:(n-1)]).

Value

MatchRow returns an integer vector, showing which rows of table match v. If there are no matches, it returns integer(0).

AffineTransform returns an object of class "mvmesh". Rotate2D returns a 2 x 2 rotation matrix, Rotate3D returns a 3 x 3 rotation matrix.

EdgeSubdivision computes an edgewise subdivision of a simplex using the method of Edelsbrunner and Grayson. The algorithm of Concalves, et. al. was implemented in R. It is a coordinate free method. ConvertBase is an internal routine used by the subdivision algorithm. NumVertices is a utility routine to recursively calculate the number of vertices in an edgewise subdivision.

EdgeSubdivMulti is roughly a vectorized version of EdgeSubdivison. It takes a list of simplices, and performs a k-subdivision of each simplex for function UnitSphere and related functions. Since some simplices may share edges, the same vertex can be occur multiple times, so this function goes through the resulting vertices and eliminates repeats. This function is not meant to be called by an end user; it is not guaranteed to be general.

`ConvertBase` is an internal function that converts a positve integer 'x' to an 'n' digit base 'b' representation. `NumVertices` is an internal function that computes the number of simplices in an edgewise subdivision (without doing the subdivision). `PointCoord` is an internal function that computes a single vertex of a simplex. `SimplexCoord` is an internal function that computes the coordinates of a simplex 'S' given color matrix 'color'. `SVIFromColor` is an internal function that computes the SVI from a starting simplex 'S' and color array 'T'.

Note that rays and lines are not allowed in `V2Hrep`; use rcdd funtion `makeH` directly to use them.

`EdgeSubdivision` returns a color matrix, a coordinate free representaion of the subdivision. One generally uses `UnitSimplex` or `UnitBall` to get a vertex representation of the subdivision.

`EdgeSubdivMulti` returns a list of class 'mvmesh'

References

Edelsbrunner and Grayson, Discrete Comput. Geom., Vol 24, 707-719 (2000).

Goncalves, Palhares, Takahashi, and Mesquita, Algorithm 860: SimpleS – an extension of Freudenthal's simplex subdivision, ACM Trans. Math. Softw., 32, 609-621 (2006).

Examples

```
Icosahedron( )

T <- EdgeSubdivision( n=2, k=2 )
T

ConvertBase( 10, 2, 6 )  # note order of digits

NumVertices( n=4, k=8, single=FALSE )

S <- rbind( diag(rep(1,2)), c(0,0) ) # solid simplex in 2D
PointCoord( S, T[,,1] )

SimplexCoord( S, T[,,1] )

SVIFromColor( S, T )

S1 <- rbind( c(0,0,0), diag( rep(1,3) ) )
S2 <- rbind( c(1,1,1), diag( rep(1,3) ) )
S3 <- rbind( c(1,1,1), c(0,1,0), c(1,0,0), c(1,1,0) )
S <- array( c(S1,S2,S3), dim=c(4,3,3) )

( H1 <- V2Hrep( S ) )
( S4 <- H2Vrep( H1 ) )

( H2 <- HrepCones( UnitSphere(n=2,k=1)$S )[,,2] )  # cone between 0 <= y <= x, x >= 0
x <- matrix( rnorm(100), ncol=2 )
( i <- SatisfyHrep( x, H2 ) )
x[i,]
```

```
(table <- matrix( c(1:12,1:3 ), ncol=3, byrow=TRUE ))
MatchRow( 1:3, table )

## Not run:
plot( Icosahedron( ), col="green" )

mesh <- SolidSimplex( n=3, k=2 )
plot(mesh, col="blue")
mesh2 <- AffineTransform( mesh, A=Rotate3D( rep(pi/2,3) ), shift=c(1,1,1) )
plot(mesh2, new.plot=FALSE, col="red" )



## End(Not run)
```

---

mvmesh-methods *Methods to print and draw mvmesh objects*

Description

Print summary of a mesh and plot 2D and 3D simplices. The 2D plot routines use the standard R plots; 3D plot routines use the rgl package.

Usage

```
## S3 method for class 'mvmesh'
print( x, ... )
## S3 method for class 'mvmesh'
plot( x, new.plot=TRUE, show.points=FALSE, show.edges=TRUE, show.faces=FALSE,
                        show.labels = FALSE, label.values=NULL, ... )
DrawSimplex2d(S,label,show.labels,mvmesh.type,show.edges=TRUE,show.faces=FALSE,...)
DrawSimplex3d(S,label,show.labels,mvmesh.type,show.edges=TRUE,show.faces=FALSE,...)
```

Arguments

`x`	an object of class "mvmesh", usually from one of the functions `UnitSimplex`, `SolidSimplex`, `UnitSphere`, `UnitBall`, `RectangularMesh`, etc.
`new.plot`	If TRUE, start a new plot; otherwise add to an existing plot
`show.points`	If TRUE, show vertices (use cex= to change size)
`show.edges`	If TRUE, show edges
`show.faces`	If TRUE, fill in solid faces (only works in certain cases); otherwise show edges
`show.labels`	If TRUE, an identifying label will be drawn inside each simplex
`label.values`	values to display if show.label=TRUE; defaults to 1,2,3,...
`...`	Optional argument to plot functions to set color, alpha, etc.
`label`	Integer to label current simplex

S	a simplex, an n x m matrix with columns S[,1],...,S[,m] giving the vertices
mvmesh.type	integer code identifying what type of mesh this is, see the definition of class "mvmesh" in mvmesh.

Details

print will print out summary information about a mesh object

plot will plot a mesh, calling DrawSimplex2d or DrawSimplex3d to plot a each simplex as appropriate for the dimension. These routines are meant to give a basic display; not all rgl capabilities are used.

Value

A plot is drawn, usually nothing is returned

Examples

```
print( SolidSimplex( n=3, k=2 ) )

## Not run:

plot( SolidSimplex( n=3, k=2 ), col='red' )


## End(Not run)
```

---

mvmeshmisc *Miscellaneous functions used by/with mvmesh*

Description

Utilities for working with mvmesh objects

Usage

```
mvmeshFromSimplices(  S )
mvmeshFromSVI(  V, SVI, m )
mvmeshFromVertices( V )
mvmeshCombine( mesh1, mesh2 )
uniqueRowsFromDoubleArray( A )
```

Arguments

S	simplices, an (vps x n x nS) array, with S[, ,k] specifying the vertices of k-th simplex
V	(nV x n) matrix giving the distinct vertices in the list of simplices
SVI	integer (vps x nS) matrix which specifies the indices of the vertices that make up the simplices in S

m	integer dimension of the mesh, less than or equal n=dimension of the space
mesh1, mesh2	objects of class "mvmesh"
A	a matrix of doubles

Details

Experimental functions. They allow one to build mvmesh objects manually by specifying just simplices (`mvmeshFromSimplices`), or just vertices (`mvmeshFromVertices`), or vertices and grouping information (`mvmeshFromSVI`). `mvmeshCombines` combines two meshes with the same values of n, m and vps. The resulting objects can usually be plotted by the plot method, but other operations may fail. In particular, vertices common to both meshes will be repeated.

Value

undocumented

Warning

This is experimental code, and not throughly tested. Function names, arguments, and what they do may change in the future.

Examples

```
## Not run:
demo(mvmeshmisc)

## End(Not run)
```

PolarSphere *Define a mesh on the unit sphere/ball in n-dimensions determined by a polar coordinates grid.*

Description

Subdivide the unit ball or sphere into simplices in arbitrary dimensions using a rectangular grid on the polar parameterization of the sphere.

The general n-dimensional polar coordinates to and from rectangular coordinates transformations are provided.

Usage

```
PolarSphere(n, breaks=c(rep(4,n-2),8), p = 2, positive.only = FALSE)
PolarBall( n, breaks=c(rep(4,n-2),8), p=2, positive.only=FALSE )
Rectangular2Polar( x )
Polar2Rectangular( r, theta )
```

Arguments

n	Dimension of the space; the Polar sphere is an (n-1) dimensional manifold
breaks	specification of the partition of in the angle space theta. See the definition of 'breaks' in SolidRectangle.
p	Power used in the l^p norm; p=2 is the Euclidean norm
positive.only	TRUE means restrict to the positive orthant; FALSE gives the full ball
r	a vector of radii of length m.
theta	a (n-1) x m matrix of angles.
x	(n x m) matrix, with column j being the point in n-dimensional space.

Details

PolarSphere computes an approximation to the unit sphere using a rectangular grid in the polar angle space. PolarBall uses a partition of the polar sphere and joins those simplices to the origin to approximately partition the unit ball. LpNorm computes the l^p norm of each columns of x.

Polar2Rectangular and Rectangular2Polar convert between the polar coordinate representation (r,theta[1],...,theta[n-1]) and the rectangular coordinates (x[1],...,x[n]).

n dimensional polar coordinates are given by the following:
rectangular x=(x[1],...,x[n]) corresponds to polar (r,theta[1],...,theta[n-1]) by
x[1] = r*cos(theta[1])
x[2] = r*sin(theta[1])*cos(theta[2])
x[3] = r*sin(theta[1])*sin(theta[2])*cos(theta[3])
...
x[n-1]= r*sin(theta[1])*sin(theta[2])*...*sin(theta[n-2])*cos(theta[n-1])
x[n] = r*sin(theta[1])*sin(theta[2])*...*sin(theta[n-2])*sin(theta[n-1])

Here theta[1],...,theta[n-2] in [0,pi), and theta[n-1] in [0,2*pi). This is the parameterization described in the Wikipedia webpage for "n-sphere". Note that this is NOT a 1-1 transformation: when theta[1]=0, it follows that x[2]=x[3]=...=x[n]=0. This is analagous to all longitude lines going through the north pole in standard 3d spherical coordinates.

For multivariate integration, the Jacobian of the above tranformation is J(theta) = r^(n-1) * prod(sin(theta[1:(n-2)])^((n-2):1)); note that theta[n-1] does not appear in the Jacobian.

Value

PolarSphere and PolarBall return an object of class "mvmesh" as described in mvmesh. Polar2Rectangular returns an (n x m) matrix of rectangular coordinates. Rectangular2Polar returns a list with fields:

r	a vector of length m containing the radii
theta	an (n x m) matrix of angles

Examples

```
PolarSphere(  n=3, breaks=4)
PolarBall( n=3, breaks=4 )
```

```
(x <- matrix( 1:10, ncol=2 ))
(a <- Rectangular2Polar( x ))
Polar2Rectangular( a$r, a$theta )

(x <- matrix( 1:12, ncol=4 ))
(a <- Rectangular2Polar( x ))
Polar2Rectangular( a$r, a$theta )

## Not run:
plot( PolarSphere( n=2, breaks=8 ) )
plot( PolarBall( n=2, breaks=8 ) )

plot( PolarSphere( n=3, breaks=c(4,8) ) )
plot( PolarBall( n=3, breaks=c(4,8) ) )

## End(Not run)
```

---

rmvmesh *Simulate from a mesh*

Description

Simulate from a mvmesh object

Usage

```
rmvmesh( n, mesh, weights=rep(1,ncol(mesh$SVI) ) )
```

Arguments

mesh	object of class "mvmesh"
n	number of vectors to simulate
weights	weights used for simulation

Details

rmvmesh allows you to sample from an mvmesh object, simplex j is sampled with probability weights[j]. Note that if the simplices are of different sizes, and the weights are uniform, this will result in uniform sampling among the simplices, but different densities on different faces. See the example below with alternating weights. If you want to get a uniform density, set the weights equal to the m dimensional volume of the simplices that make up the meshes.

rmvmesh works for any mesh where the m dimensional simplices are convex combinations of (m+1) vertices i.e. vps=m+1. This works whatever the dimension of the embedding space is, and whether or not things have been rotated, scaled or shifted by AffineTransform. It also works with an unaltered SolidRectangle or HollowRectangle. mvmesh does not currently work with mvmesh objects of type PolarSphere, PolarBall, HollowTube, or SolidTube; nor does it work with rectangles that have been altered by AffineTransform.

Note that `rmvmesh` samples from the mesh, not from the idealized object. In particular, in the example below with a unit sphere, the sampled points are from the tessellation approximation to the sphere, not from the unit sphere itself. So (with probability one), all points will have length less than 1.

Value

A matrix of values x: x[1,],x[2,],...,x[n,] are vectors sampled from the mesh.

Examples

```
## Not run:
sphere <- UnitSphere( n=3, k=2 )
plot(sphere)
x <- rmvmesh( 1000, sphere )
points3d( x, col='red' )

box <- HollowRectangle( a=c(0,2,-1), b=c(1,5,3), breaks=3 )
plot(box)
x <- rmvmesh( 500, box )
points3d( x, col='blue', size=5 )


plot(box)
nS <- ncol(box$SVI) # number of simplices in box
weights <- rep( c(0,1), nS/2 )  # alternating 0,1 weights
x <- rmvmesh( 10000, box, weights )
points3d( x, col='green', size=5 )


## End(Not run)
```

`UnitSimplex` *Define a mesh on the unit simplex or the canonical simplex*

Description

Defines an equal area/volume subdivision of the unit simplex and the canonical simplex in R^n. The unit simplex is the (n-1) dimensional simplex with vertices (1,0,0,...,0), (0,1,0,...,0), ...,(0,0,0,...,1), i.e. all x >= 0 with sum(x)==1.

The solid simplex is the n dimensional simplex with vertices (1,0,0,...,0), (0,1,0,...,0), ...,(0,0,0,...,1), and (0,0,...,0), i.e. all x >= 0 with sum(x) <= 1.

Usage

```
UnitSimplex(n, k )
SolidSimplex( n, k )
```

Arguments

`n`	dimension of the space
`k`	number of subdivisions

Details

`EdgeSubdivision` is called to do a k-subdivision of each edge, and then that output is converted to a matrix of vertices.

Value

an object of class "mvmesh" as described in `mvmesh`.

Examples

```
UnitSimplex( n=2, k=3 )
SolidSimplex( n=2, k=3 )

UnitSimplex( n=3, k=2 )
SolidSimplex( n=3, k=2 )

UnitSimplex( n=5, k=4 )
SolidSimplex( n=5, k=4 )

## Not run:
plot( UnitSimplex( n=2, k=3 ) )
plot( SolidSimplex( n=2, k=3 ) )

plot( UnitSimplex( n=3, k=2 ) )
plot( SolidSimplex( n=3, k=2 ) )


## End(Not run)
```

`UnitSphere` *Define a mesh on a unit ball in n-dimensions*

Description

Subdivide the unit ball or sphere into approximately equal simplices in arbitrary dimenions.

Usage

```
UnitSphere(n, k, method = "dyadic", p = 2, positive.only = FALSE)
UnitSphereEdgewise(n, k, p, positive.only)
UnitSphereDyadic(n, k, start = "diamond", p, positive.only)
UnitBall( n, k, method="dyadic", p=2, positive.only=FALSE )
LpNorm(x, p)
```

Arguments

`n`	Dimension of the space; the unit sphere is an (n-1) dimensional manifold
`k`	Number of subdivisions
`method`	"dyadic" or "edgewise": the former recursively subdivides the sphere to get a more uniform grid; the latter uses a faster method using one edgewise subdivision.
`p`	Power used in the l^p norm; p=2 is the Euclidean norm
`positive.only`	TRUE means restrict to the positive orthant; FALSE gives the full ball
`start`	starting shape: "diamond" or "icosahedron"
`x`	Matrix of points in n-dimensions; each column is a point in R^n

Details

`UnitSphere` computes a hyperspherical triangle approximation to the unit sphere. It calls either `UnitSphereDyadic` or `UnitSphereEdgewise` based on 'method'. Both work by subdividing the first orthant, and then rotating that subdivision around to other orthants. This is important for some uses of these functions; it guarantees that all vertices of a simplex are in a single orthant. Note that 'k' has a different meaning for the different methods. When method="dyadic", k specifies the number of dyadic subdivisions. When method="edgewise", k specifies the number of subdivisions as in `UnitSimplex`, which is then projected outward to the unit sphere. So when n=2, a dyadic subdivision with k=2 will result in 16 edges, whereas an edgewise subdivions with k=2 results in 8 edges.

`UnitBall` computes an approximate simplicial approximation to the unit ball. Specifically, it generates cones with one vertex at the origin and the other vertices on the surface of the unit sphere; these later vertices are from `UnitSphere`. If k is large, these cones will be very narrow/thin.

Value

an object of class "mvmesh" as described in `mvmesh`.

Examples

```
UnitSphere( n=2, k=2, method="edgewise", positive.only=TRUE )
UnitSphere( n=2, k=2, method="edgewise" )

UnitSphere( n=3, k=2, method="edgewise", positive.only=TRUE )
UnitSphere( n=3, k=2, method="edgewise" )

UnitBall( n=2, k=2, method="edgewise", positive.only=TRUE )
UnitBall( n=2, k=2, method="edgewise" )

UnitSphere( n=3, k=2, method="dyadic", positive.only=TRUE )
UnitSphere( n=3, k=2, method="dyadic" )

UnitBall( n=3, k=2, method="dyadic", positive.only=TRUE )
UnitBall( n=3, k=2, method="dyadic" )

UnitSphere(  n=3, k=2 )
```

```
UnitBall( n=3, k=2 )

x <- c(3,-1,2)
LpNorm( x, p=2 )

## Not run:
plot( UnitSphere(  n=3, k=2 ), show.label=TRUE )
plot( UnitBall( n=3, k=2 ) )


## End(Not run)
```

Index