Computational geometry for multivariate statistics

John P. Nolan*

Abstract
An open source R package `mvmesh` for working with multivariate meshes is described. This package allows one to work with complex geometric objects in dimensions $n \geq 2$.

Key Words: Multivariate meshes, computational geometry, R package

1. Introduction

The purpose of this paper is to give a quick introduction to some ideas in computational geometry that allow one to work with shapes in n-dimensions, particularly $n > 2$. When the data of interest is supported on a non-rectangular shape, non-standard methods are needed. For example, directional data concerns points the unit sphere. Compositional data is concerned with the proportion of a sample that is of different types, e.g. a sample of ore may contain 20% iron, 30% copper, etc. In such problems, the proportions of different types sum to one, so observations are points on the unit simplex. Multivariate extreme value distributions have a complicated dependence structure that is specified by a measure on the unit simplex. Likewise, multivariate stable distributions are specified by a measure on the unit sphere. To model these two classes in practice, we need to be able to partition the simplex or sphere and to manipulate these objects.

In two dimensions, it is straightforward to approximate most shapes, e.g. a simplex or circle or path. In higher dimensions, it is not so simple. The R package `mvmesh`, Nolan (2015a), is an open source package which is available on the CRAN network that gives a collection of functions to define, subdivide and perform other operations with geometric objects. A key goal in this package is to go beyond 2 and 3 dimensions, providing methods that work in arbitrary dimensions. The `mvmesh` package builds on two existing R packages: `rcdd` by Geyer & Meeden (2015) and `geometry` by Barber et al. (2015) to make is simpler to define and manipulate whole objects. We also add features that combine computational geometry with statistics, e.g. simulating points from a shape and producing multivariate histograms over non-rectangular shapes.

2. Simplices and meshes

The basic building block for representing shapes in a computationally tractable way is a simplex. There are two common ways to represent simplices: the vertex (V) representation and the half-space (H) representation. The V-representation of a simplex is simply a list of the vertices $\mathbf{v}_1, \ldots, \mathbf{v}_k$; the simplex is the closed convex hull of these vertices. The H-representation is given by the intersection of half-spaces. For example, in 3-dimensions the standard solid simplex in Figure 1 can represented by the vertices $\mathbf{v}_1 = (1, 0, 0)$, $\mathbf{v}_2 = (0, 1, 0)$, $\mathbf{v}_3 = (0, 0, 1)$ and $\mathbf{v}_4 = (0, 0, 0)$ or by the intersection of the 4 half-spaces $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, and $x_1 + x_2 + x_3 \leq 1$. These inequalities can always be expressed in

*American University, Department of Mathematics and Statistics, 4400 Massachusetts Avenue, Washington, DC 20016-8050. Supported by contract W911NF-12-1-0385 from the Army Research Office. This paper is a modified version of a talk presented at the CASD 2015, Conference on Applied Statistics in Defense 2015, George Mason University, 21 October 2015.

matrix form as $A\mathbf{x} \leq \mathbf{b}$. By allowing some of the inequalities to be equalities, we can get a lower dimensional simplex. For example, the unit simplex (see Figure 1) can be specified as the closed convex hull of $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$; or by replacing the last inequality above with equality $x_1 + x_2 + x_3 = 1$.

There is a detail to be aware of when dealing with simplices in a computer. To specify points exactly, one could use rational numbers, stored as pairs of integers, for vertices (in the V-representation) or as coefficients in the equalities/inequalities (in the H-representation). Some R packages like `rcdd`, Geyer & Meeden (2015), use the GNU Multiple Precision Library to do this. While this has many advantages in exactly specifying simplices, we do not use it. One reason is that this approach does not work when a vertex is not a rational number; e.g. most points on the unit sphere. Another reason is that we frequently want to do further processing with a simplex, e.g. translate, rotate, or integrate a function over a simplex, and vertices are more conveniently represented as floating point numbers for these purposes. Because of these reasons, we represent vertices as n-tuples of floating point numbers. One consequence of this is that it is generally not possible to determine if a point is on a line, plane, or hyperplane, because of the limited precision of floating point numbers.

Most interesting geometric objects are more complicated than simplices. To be computationally accessible, they are approximated by a list of simplices. We will use the term mesh to mean a list of vertices along with the grouping necessary to specify what points are in what simplices. The `mvmesh` package (the name is a contraction of MultiVariate MESH) has functions to define several standard shapes, some of which are shown in Figure 1. The canonical solid simplex is $\{\mathbf{x} = (x_1, \ldots, x_n) : x_i \geq 0, \sum_{i=1}^n x_i \leq 1\}$. The basic building block for a surface is the unit simplex $\{\mathbf{x} : x_i \geq 0, \sum_{i=1}^n x_i = 1\}$. In general, any m-dimensional simplex in $\mathbb{R}^n$ is a linear transformation of these basic (possibly lower dimensional) simplices. These shapes can be subdivided into regions with equal volume/area by specifying a parameter. For example, `SolidSimplex(n=3,k=4)` generates the solid simplex in the upper left of the first figure. The argument $n = 3$ specifies a simplex in $\mathbb{R}^3$, the $k = 4$ value subdivides each side of the standard simplex into 4 pieces and computes the resulting subdivision using the k-edge subdivision algorithm of Edelsbrunner & Grayson (1999).

Shapes with curvature are approximated by a tessellation, e.g. lists of simplices. The ℓ_p solid unit ball is $\{\mathbf{x} : \|\mathbf{x}\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p} \leq 1\}$, and the unit sphere is $\{\mathbf{x} : \|\mathbf{x}\|_p = 1\}$. These meshes are simplicial approximations to the true shape, with different subdivision schemes possible. It is impossible to subdivide equally in general dimension, but an approximately equal area subdivision is possible using a $k = 2$ edge subdivision on each octant recursively. It is also possible use a polar coordinates approach to subdivide the unit ball and sphere, however this has two drawbacks. First, the regions are not all of the same type: at a pair of antipodal points the simplices that make up the surface of the sphere have fewer vertices than all the other simplices. For example, the sphere in $\mathbb{R}^3$ with the standard latitude and longitude grid has all longitude lines meeting at the north and south poles. The second reason is that the approximating simplices will have very different areas near these poles than in other regions. A rectangular mesh is a simple rectilinear partition of a hyperrectangle. Solid tubes in $\mathbb{R}^n$ are defined by crossing an $(n-1)$-dimensional ball with an interval; a hollow tube is the cross of a unit sphere with an interval.

It is also possible to build ad hoc geometric shapes by specifying the vertices and a grouping directly. This takes some effort, especially in higher dimensions, but allows one to work with custom shapes. Figure 2 shows a surface that could be used to model the shape of an explosion. It also shows an object constructed to show the name of the CASD 2015 Conference by specifying vertices and line segments that spell out the letters and numbers,

as well as a trefoil note, both in $\mathbb{R}^3$. One advantage of constructing such objects as an mvmesh object is that they can be plotted using the functions described below to plot and simulate from the mesh.

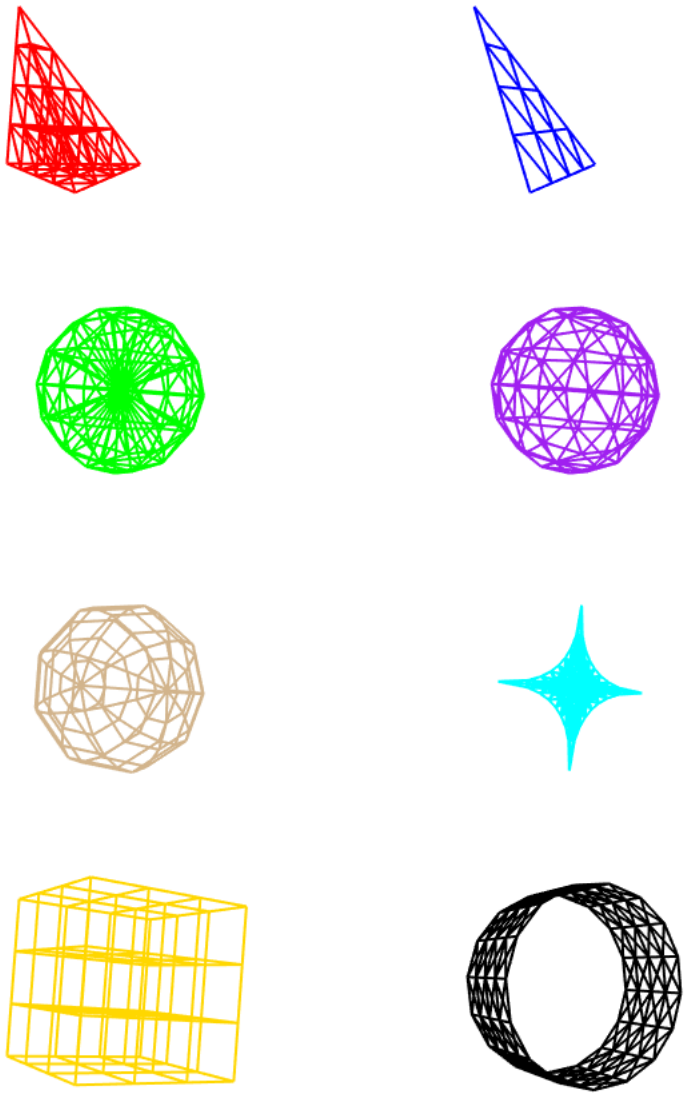

Figure 1: Basic shapes in $\mathbb{R}^3$: starting from top left, by row the objects are solid simplex, unit simplex, unit ball, unit sphere, polar sphere, $\ell_{1/2}$ unit sphere, rectangular mesh, hollow tube.

Multivariate meshes are implemented in the package as S3 objects of class "mvmesh", e.g. lists with multiple fields. The key fields are shown in Table 1. Other fields are specific to the type of mesh and generally describe the parameters that were used to generate the mesh.

3. Showing and manipulating shapes

Once a mesh is defined, there are standard operations that can be performed on any mesh. There are print and plot methods for an mvmesh object. The plot uses standard R graphics

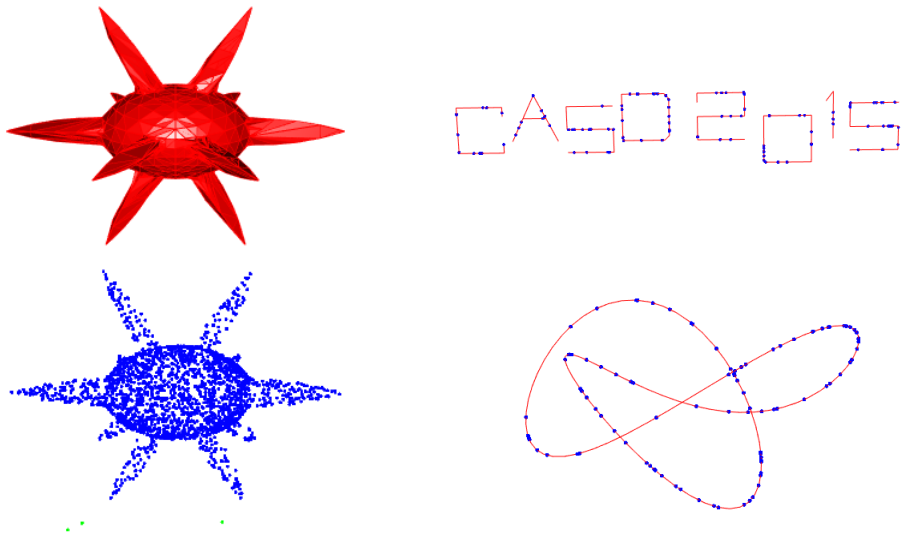

Figure 2: Custom shapes: an explosion, CASD 2015 logo, and a trefoil knot. The blue points are points randomly generated from the underlying red object.

type	a string describing the mesh, e.g. "UnitSimplex"
n	dimension of the space
m	dimension of the mesh
vps	vertices per simplex, the number of vertices that define a simplex, which must be the same for all simplices in this mesh
S	an (vps $\times$ n $\times$ nS) array, with S[i , ,k] specifying the i-th vertex of the k-th simplex
V	an (nV $\times$ n) matrix giving the coordinates of the distinct vertices in the list of simplices (repeated vertices in S that are on common edges are removed)
SVI	an integer (vps $\times$ nS) matrix which specifies the indices of the vertices that make up the simplices in S. SVI[,k] gives the subscripts in the vertex array V that determine the k-th simplex in S

Table 1: Basic fields in an mvmesh object.

in two dimensions and the `rgl` package Adler et al. (2016) in three dimensions.

We next describe some of the more useful general operations that can be performed on mvmesh objects. The function `V2Hrep` converts a list of simplices specified by the vertex representation to a list of half-space representations; the function `H2Vrep` does the inverse. Objects can be scaled, rotated, and shifted using function `AffineTransformation`. Several meshes can be combined together to make a more complicated mesh using `mvmeshCombine`, and there are functions to intersect meshes in V or H representations: `IntersectMultipleSimplicesV` and `IntersectMultipleSimplicesH`.

Another useful operation is simulation from a mesh. The function `rtessellation` simulates from a tessellation; it uses the S field from a mesh, i.e. the list of simplices, and a vector of weights specifying the weight of each simplex. A point is simulated by selecting a simplex S_j according to those weights, then simulating a point uniformly distributed on the simplex. This last step is performed by simulating a uniform distribution on the unit simplex of dimension m using the standard method of simulating from a Dirichlet distribution with shape parameter $(1, 1, \ldots, 1)$ and then linearly mapping that to the simplex S.

Figure 2 shows some examples. At the bottom left of the figure, points have been randomly sampled from the explosion shape in the upper left. The figure on the right shows points sampled from both the CASD 2015 logo and a trefoil knot superimposed on the underlying mvmesh objects.

While not a part of the `mvmesh` package, one can numerically evaluate the integral of a function over a mesh by using package `SimplicialCubature` Nolan (2015b).

4. Multivariate histograms

We end with an application of the `mvmesh` package to calculating multivariate histograms. First we show some examples of histograms in dimensions 2 and 3. Figure 3 shows a histogram of two dimensional data on the left and three dimensional data on the right, both using a rectangular mesh. In Figure 4, the left plot shows the two dimensional spread of points contained in a circle by counting how many points are in pie-shaped sectors; the right plot shows counts of points distributed throughout the three dimensional solid simplex. The construction of multivariate histograms is explained through several cases.

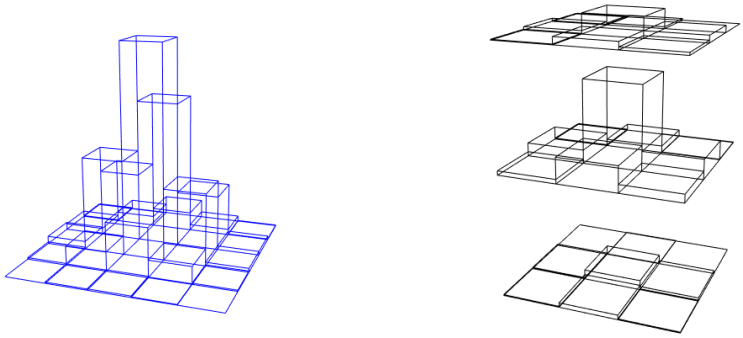

Figure 3: Histograms based on rectangular meshes in 2 and 3 dimensions. The right hand plot slices the data into 3 horizontal bands and computes histograms on each slice.

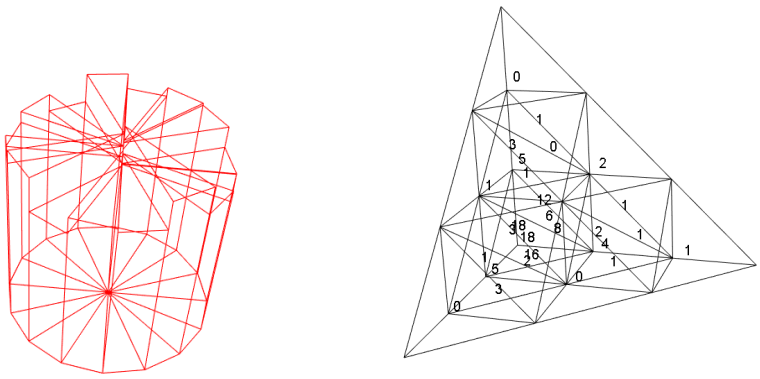

Figure 4: Histogram by circular sector in 2 dimensions on the left; height of the plot shows frequency. On the right, counts of number of data points in a subdivision of a solid simplex in 3 dimensions.

4.1 Tallying points in solid simplices

We start with the simplest case: a mesh is given by a list of solid simplices $S_1, \ldots, S_M$. For a data set $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we wish to tally how many points are in each simplex. The approach is straightforward: for each simplex j, find the half-space representation $A_j\mathbf{x} \leq \mathbf{b}_j$. It is now a simple double loop to test each point $\mathbf{x}_i$ against each H-representation for simplex: if $A_j\mathbf{x}_i \leq \mathbf{b}_j$, we say the point is in the j-th simplex.

There are two details with this procedure: points not in any simplex and points on the boundary of two or more simplices. To handle the first issue, we keep a count of the number of rejects, i.e. points that do not satisfy any set of constraints. For the second, we count the point as being in the first (based on order of the list of simplices) simplex j that satisfies $A_j\mathbf{x}_i \leq \mathbf{b}_j$, but also check to see if that point satisfies any other set of constraints, e.g. a tie. There is a argument to the tally function that controls how much reporting gets done: if `report`="summary", then a single message is printed saying how many ties there are; if `report`="all", then every point that satisifes more than one set of constraints is individually reported; if `report`="none", then no messages are printed out. The default is "summary". In all cases, the counts of how many ties and how many rejects are observed and returns these values to the user.

4.2 Cones and directional histograms

A cone is an unbounded region easily specified by the H-representation. For example, in 2-dimensions the 3 inequalities $x_1 \geq 0$, $2x_2 \geq x_1$ and $x_2 \leq 3x_1$ determine a cone in the first quadrant with $x_1/2 \leq x_2 \leq 3x_1$. Given a list of such cones, we can use the tally procedure described above to count how many data points are in each cone. This is particularly useful to calculate directional histograms in n-dimensions. The function `histDirectional` computes the H-representations of cones determined by the origin and a spherical triangles generated by the function `UnitSphere` and then tallies the number of data points in each cone. This sphere can be in any ℓ_p metric, and if the data is concentrated in the positive octant, the plot can be restricted to just that octant. For example, Figure 5 shows a sequence of directional histograms. The top left is a scatterplot of the a synthetic data set, simulated from mixing 5000 light tailed (in the radial direction) values with uniformly distributed angle in the first quadrant with 100 heavy tailed data values that are concentrated along certain rays. The top right plot shows a directional histogram of all the data, using a Euclidean sphere to determine 9 bins, and showing frequency by length of the rays in each direction. A common technique in extreme value theory is to look for directional dependence in extreme data values. This is done by thresholding, i.e. picking some threshold R and then forming a directional histogram using only data points with $\|x_i\|_p \geq R$. The bottom two plots show this procedure with $R = 1$ and $R = 4$. In the last plot, it is clear that the extremes are concentrated along certain directions. (In both multivariate extreme value distributions and multivariate stable distributions, this occurs when the spectral/angular measure is discrete.) Here thresholding was specified by a specific value of R, another function allows thresholding by quantiles of the radii of sample points.

Figure 6 shows a three dimensional example. Here a synthetic nonnegative data set is analyzed to show directional spread based on the ℓ_1 ball = unit simplex. When a 3d plot like this is generated in R, it is interactive, so it can be rotated to more easily see where data is concentrated. Other display modes are described below, including one showing 4 dimensional data.

As noted above, determining whether a point is on a line/plane/hyperplane is not possible in general using floating point numbers. Therefore doing histograms on a surface is not feasible in general. For certain cases, e.g. a sphere or unit simplex, one can do a directional

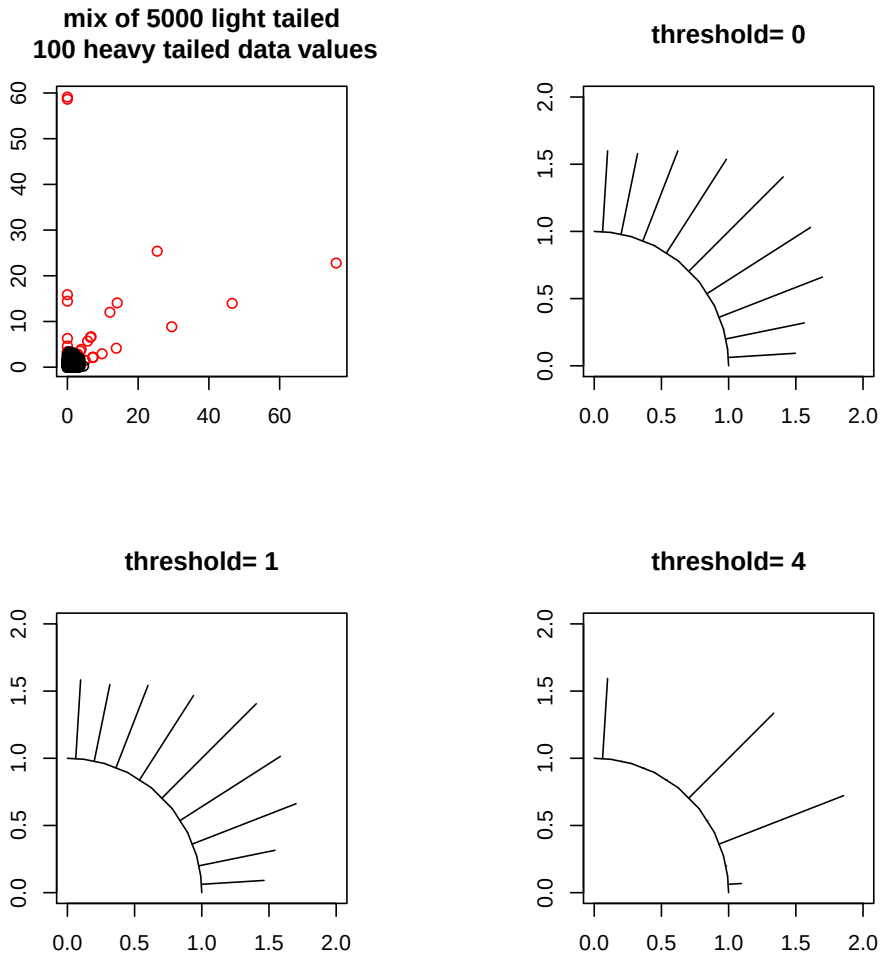

Figure 5: Directional histogram in 2 dimensions for non-negative data. Top left shows a scatterplot of data points, top right shows a histogram of directions of all points. Lower left looks eliminates data with $\|x\|_2 \leq 1$, lower right eliminates $\|x\|_2 \leq 4$, revealing the directional dependence in the extremes.

histogram, but a general solution will require a rule to determine when a data point is "in" a simplex.

4.3 Plotting multvariate histograms

There are multiple types of histogram plots that can be drawn, several of which are shown in Figures 5, 6 and 7. A full list is:

"none" does not show a plot, just return the counts.

"counts" shows frequency counts as a number in the center of each simplex (right plot in Figure 4).

"pillars" shows a 3D plot with pillars/columns having base the shape of the simplices and height proportional to frequency counts (both plots in Figure 3 and left plot

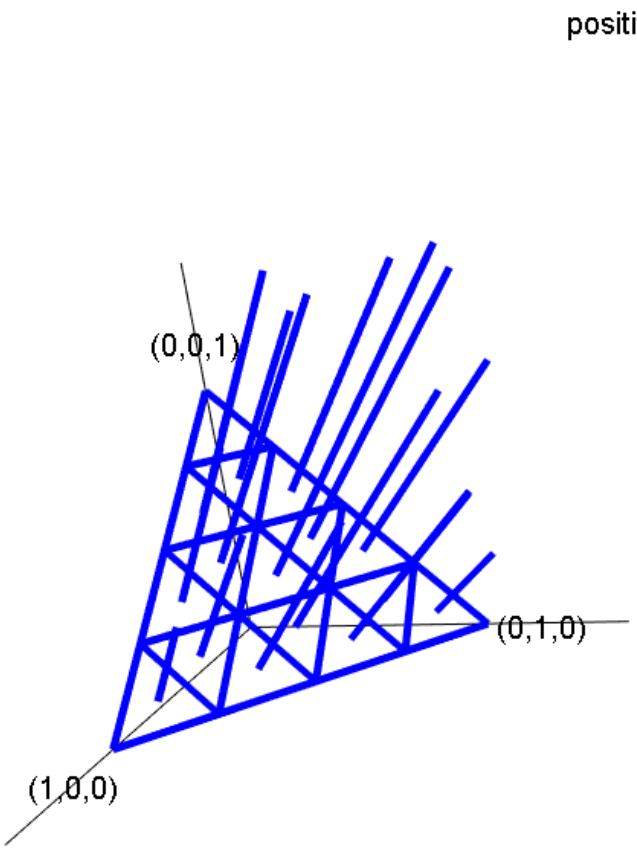

Figure 6: Directional histogram for non-negative three dimensional data using ℓ_1 unit ball.

in Figure 4). When the points are 2D, this works for `histRectangular` and `histSimplex`; when the points are 3D, this only works for `histRectangular`

"radial" `histDirectional` only, shows radial spikes proportional to the counts (Figure 5).

"grayscale" `histDirectional` only, color codes simplices proportional to the counts.

"orthogonal" `histDirectional` only, shows radial spikes proportional to the counts (Figure 5).

"default" type depends on the dimension of the data and type of histogram.

"index" shows a histogram of simplex index number versus count (Figure 7).

The last plot type is the only one that makes sense in dimension greater than 3. It does not show the geometry, but gives a way to capture some information about a data set. For example, Figure 7 shows a histogram of a 4-dimensional data set. It clearly shows that all the data lies in quadrants with a + sign in the 3rd or 4th coordinate.

Most of the examples in this paper can be produced by installing the `mvmesh` package and running the two built-in demos: `demo(mvmesh)` and `demo(mvhist)`.

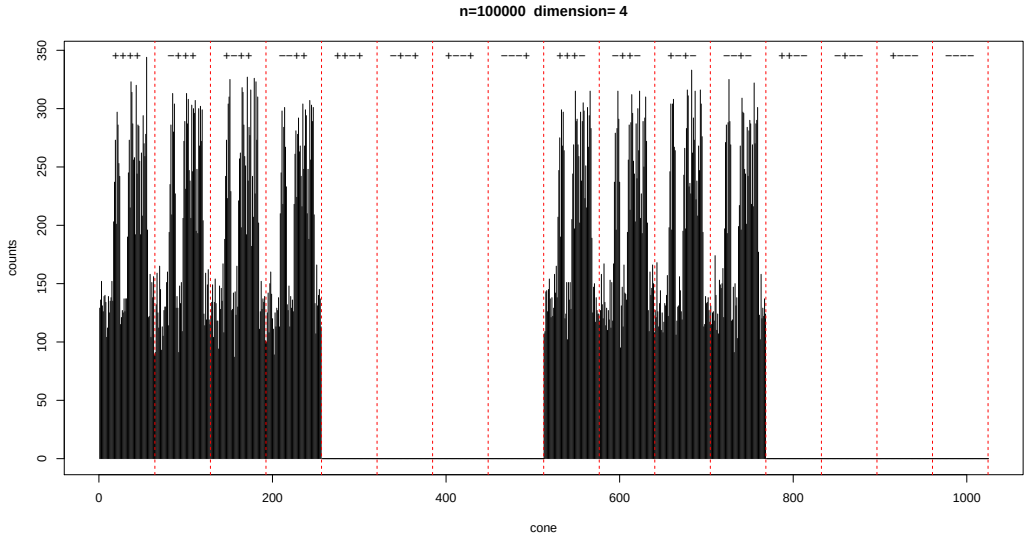

Figure 7: Directional histogram in 4 dimensions, with order based on the order of the cones in $\mathbb{R}^4$. The vertical red dashed lines describe octants; the key above each group is a sequence of + and - signs that shows the sign of each coordinate.

References

Adler, D., Murdoch, D., et al. (2016). *rgl 3D Visualization Using OpenGL*. R package version 0.95.1441, on CRAN.

Barber, C. B., Habel, K., Grasman, R., Gramacy, R. B., Stahel, A., & Sterratt, D. C. (2015). *geometry: Mesh Generation and Surface Tessellation*. R package version 0.3-6, on CRAN.

Edelsbrunner, H., & Grayson, D. R. (1999). Edgewise subdivision of a simplex. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry (Miami Beach, FL, 1999)*, (pp. 24–30 (electronic)). New York: ACM.

Geyer, C. J., & Meeden, G. D. (2015). *rcdd: Computational Geometry*. R package version 1.1-9, on CRAN.

Nolan, J. P. (2015a). *mvmesh: Multivariate Meshes and Histograms in Arbitrary Dimensions*. R package version 1.1, on CRAN.

Nolan, J. P. (2015b). *SimplicialCubature: Integration of Functions Over Simplices*. R package version 1.1, on CRAN.