Package: ashapesampler (via r-universe)

August 28, 2024

Title Generating Alpha Shapes

Version 1.0.0

Description Understanding morphological variation is an important task in many applications. Recent studies in computational biology have focused on developing computational tools for the task of sub-image selection which aims at identifying structural features that best describe the variation between classes of shapes. A major part in assessing the utility of these approaches is to demonstrate their performance on both simulated and real datasets. However, when creating a model for shape statistics, real data can be difficult to access and the sample sizes for these data are often small due to them being expensive to collect. Meanwhile, the landscape of current shape simulation methods has been mostly limited to approaches that use black-box inference---making it difficult to systematically assess the power and calibration of sub-image models. In this R package, we introduce the alpha-shape sampler: a probabilistic framework for simulating realistic 2D and 3D shapes based on probability distributions which can be learned from real data or explicitly stated by the user. The 'ashapesampler' package supports two mechanisms for sampling shapes in two and three dimensions. The first, empirically sampling based on an existing data set, was highlighted in the original main text of the paper. The second, probabilistic sampling from a known distribution, is the computational implementation of the theory derived in that paper. Work based on Winn-Nunez et al. (2024) <doi:10.1101/2024.01.09.574919>.

License GPL (>= 3)

Imports pracma, alphahull, alphashape3d, truncnorm, stats, Rvcg, TDA, doParallel, foreach, parallel, dplyr

Suggests knitr, testthat, rgl, ggplot2, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 3.1.0)

NeedsCompilation no

Author Emily Winn-Nunez [aut, cre]
(<https://orcid.org/0000-0001-6759-5406>), Lorin Crawford [aut]
(<https://orcid.org/0000-0003-0178-8242>)

Maintainer Emily Winn-Nunez <emily_winn-nunez@brown.edu>

Repository CRAN

Date/Publication 2024-01-30 12:00:02 UTC

Contents

`calc_overlap_2D` *Calculate Overlap 2D*

Description

This function calculates the minimum coverage percentage of an alpha ball over the bounded area being considered. 0 is no coverage, 1 means complete coverage. For the square, r is the length of the side. For circle, r is the radius. For the annulus, r and min_r are the two radii.

Usage

```
calc_overlap_2D(alpha, r = 1, rmin = 0.01, bound = "square")
```

Arguments

`alpha`	radius of alpha ball
`r`	length of square, radius of circle, or outer radius of annulus
`rmin`	inner radius of annulus
`bound`	manifold shape, options are "square", "circle", or "annulus"

Value

area of overlap

`calc_overlap_3D` *calculate overlap in three dimensions (calc_overlap_3D)*

Description

Calculates the volume of intersection divided by the volume of the manifold. For the cube, r is the length of the side. For sphere, r is the radius. For the annulus, r and min_r are the two radii.

Usage

```
calc_overlap_3D(alpha, r = 1, rmin = 0.01, bound = "cube")
```

Arguments

alpha	radius of one sphere
r	radius of second sphere or outer radius of shell or length of cube side
rmin	inner radius of shell, only needed if bound=shell
bound	manifold type, options are "cube", "shell", and "sphere"

Value

volume of overlap

---

cap_intersect_vol *Intersection of spheres*

Description

Called for sphere overlaps with alpha > r*sqrt(2). Integral precalculated and numbers plugged in.

Usage

```
cap_intersect_vol(alpha, r)
```

Arguments

alpha	radius 1
r	radius 2

Value

volume of intersection of spheres.

---

circle_overlap_cc *Circle Overlap Centered on Circumference*

Description

Circle overlap cc is subfunction for repeated code in calc_overlap_2D Returns the area of two overlapping circles where one is centered on the other's Circumference. (cc = centered on circumference)

Usage

```
circle_overlap_cc(alpha, r = 1)
```

Arguments

alpha	radius 1
r	radius 2

Value

area of overlap

---

circle_overlap_ia — *Circle Overlap Inner Annulus*

Description

Circle overlap ia (inner annulus) calculates area needed to subtract when calculating area of overlap of annulus and circle.

Usage

```
circle_overlap_ia(alpha, R, r)
```

Arguments

alpha	radius of circle
R	outer radius of annulus
r	inner radius of annulus

Value

area of overlap

---

circumcenter_face — *circumcenter Face*

Description

This function finds the circumcenters of the faces of a simplicial complex given the list of vertex coordinates and the set of faces.

Usage

```
circumcenter_face(v_list, f_list)
```

Arguments

v_list	matrix of vertex coordinates
f_list	matrix with 3 columns with face information.

Value

circ_mat, matrix of coordinates of circumcenters of faces.

`circumcenter_tet` *circumcenter Tetrahedra*

Description

This function finds the circumcenters of the tetrahedra/3-simplices of a simplicial complex given the list of vertex coordinates and the set of tetrahedra.

Usage

```
circumcenter_tet(v_list, t_list)
```

Arguments

`v_list`	matrix of vertex coordinates
`t_list`	matrix of 4 columns with tetrahedra

Value

circ_mat, matrix of coordinates of circumcenters of teterahedra

---

`circ_face_2D` *Circumcenter face - three points in 2D Given 3 sets of coordinates, calculates the circumcenter*

Description

Circumcenter face - three points in 2D Given 3 sets of coordinates, calculates the circumcenter

Usage

```
circ_face_2D(points)
```

Arguments

`points`	3x2 matrix

Value

1x2 vector, coordinates of circumcenter

`circ_face_3D` *Circumcenter face - three points in 3D Given 3 sets of coordinates, calculates the circumcenter*

Description

Circumcenter face - three points in 3D Given 3 sets of coordinates, calculates the circumcenter

Usage

```
circ_face_3D(points)
```

Arguments

`points`	3x3 matrix

Value

1x3 vector, coordinates of circumcenter

`circ_tet_3D` *Circumcenter tetrahedron - 4 points in 3D Given 3D coordinates of 4 points, calculates circumcenter*

Description

Circumcenter tetrahedron - 4 points in 3D Given 3D coordinates of 4 points, calculates circumcenter

Usage

```
circ_tet_3D(points)
```

Arguments

`points`	4x3 matrix

Value

1x3 vector, coordinates of circumcenter

`count_neighbors` *Neighbors function - finds number of neighbors for each point in point cloud.*

Description

Neighbors function - finds number of neighbors for each point in point cloud.

Usage

```
count_neighbors(v_list, complex)
```

Arguments

`v_list`	2 or 3 column matrix
`complex`	simplicial complex object

Value

n_list vector where each entry is number of neighbors for a point

`euclid_dists_point_cloud_2D` *Euclidean Distance Point Cloud 2D*

Description

Calculates the distance matrix of a point from the point cloud.

Usage

```
euclid_dists_point_cloud_2D(point, point_cloud)
```

Arguments

`point`	cartesian coordinates of 2D point
`point_cloud`	3 column matrix with cartesian coordinates of 2D point cloud

Value

vector of distances from the point to each point in the point cloud

euclid_dists_point_cloud_3D *Euclidean Distance Point Cloud 3D*

Description

Calculates the distance matrix of a point from the point cloud.

Usage

```
euclid_dists_point_cloud_3D(point, point_cloud)
```

Arguments

`point`	cartesian coordinates of 3D point
`point_cloud`	3 column matrix with cartesian coordinates of 3D point cloud

Value

vector of distances from the point to each point in the point cloud

extract_complex_edges *Returns the edges of complex.*

Description

Returns the edges of complex.

Usage

```
extract_complex_edges(complex, n_vert = 0)
```

Arguments

`complex`	complex object from TDA packages
`n_vert`	number of vertices in complex; default is 0, specifying this parameter speeds up the function

Value

edge_list data frame or if empty NULL

extract_complex_faces *Returns faces of complex.*

Description

Returns faces of complex.

Usage

```
extract_complex_faces(complex, n_vert = 0)
```

Arguments

`complex`	complex object from TDA package
`n_vert`	number of vertices in the complex; default is 0, specifying this parameter speeds up function

Value

face_list data frame of points forming faces in complex

extract_complex_tet *Returns tetrahedra of complex (3 dimensions)*

Description

Returns tetrahedra of complex (3 dimensions)

Usage

```
extract_complex_tet(complex, n_vert = 0)
```

Arguments

`complex`	complex object from TDA package
`n_vert`	number of vertices in the complex; default is 0, specifying this parameter speeds up function

Value

tet_list data frame of points forming tetrahedra in complex

`extreme_pts` *Extreme points Finds the boundary points of a simplicial complex*

Description

Extreme points Finds the boundary points of a simplicial complex

Usage

```
extreme_pts(complex, n_vert, dimension)
```

Arguments

`complex`	complex list object
`n_vert`	number of vertices in the complex
`dimension`	number, 2 or 3

Value

vector of all vertices on the boundary

`generate_ashape2d` *Generate 2D alpha shape*

Description

Generate 2D alpha shape

Usage

```
generate_ashape2d(
  point_cloud,
  J,
  tau,
  delta = 0.05,
  afixed = TRUE,
  mu = NULL,
  sig = NULL,
  sample_rad = NULL,
  acc_rad = NULL,
  k_min = 2,
  eps = 1e-04,
  cores = 1
)
```

Arguments

point_cloud	2 column matrix of all points from all shapes in initial data set
J	number of shapes in initial (sub) data set
tau	tau bound vector for shapes input
delta	probability of not preserving homology; default is 0.05
afixed	boolean, whether to sample alpha or leave fixed based on tau. Default FALSE
mu	mean of truncated distribution from which alpha sampled; default tau/3
sig	standard deviation of truncated distribution from which alpha sampled; default tau/12
sample_rad	radius of ball around each point in point cloud from which to sample; default tau/8
acc_rad	radius of ball to check around potential sampled points for whether to accept or reject new point; default tau/4
k_min	number of points needed in radius tau of point cloud to accept a sample
eps	amount to subtract from tau/2 to give alpha. Defaul 1e-4.
cores	number of computer cores for parallelizing. Default 1.

Value

new_ashape two dimensional alpha shape object from alphahull library

---

generate_ashape3d *Generate 3D alpha shape*

Description

Generate 3D alpha shape

Usage

```
generate_ashape3d(
  point_cloud,
  J,
  tau,
  delta = 0.05,
  afixed = TRUE,
  mu = NULL,
  sig = NULL,
  sample_rad = NULL,
  acc_rad = NULL,
  k_min = 3,
  eps = 1e-04,
  cores = 1
)
```

Arguments

`point_cloud`	3 column matrix of all points from all shapes in initial data set
`J`	number of shapes in initial data set
`tau`	tau bound for the shapes
`delta`	probability of not preserving homology; default is 0.05
`afixed`	boolean, whether to sample alpha or leave fixed based on tau. Default FALSE
`mu`	mean of truncated distribution from which alpha sampled; default tau/3
`sig`	standard deviation of truncated distribution from which alpha sampled; default tau/12
`sample_rad`	radius of ball around each point in point cloud from which to sample; default tau/8
`acc_rad`	radius of ball to check around potential sampled points for whether to accept or reject new point; default tau/4
`k_min`	number of points needed in radius 2 alpha of point cloud to accept a sample
`eps`	amount to subtract from tau/2 to give alpha. Defaul 1e-4.
`cores`	number of cores for parallelizing. Default 1.

Value

new_ashape three dimensional alpha shape object from alphashape3d library

---

`get_alpha_complex` *Get alpha complex*

Description

Generates alpha complex for a set of points and parameter alpha

Usage

```
get_alpha_complex(points, alpha)
```

Arguments

`points`	point cloud for alpha complex, in form of 2 column of 3 column matrix with nonzero number of rows
`alpha`	alpha parameter for building the alpha complex

Value

complex list of vertices, edges, faces, and tetrahedra.

`get_area` *Get area*

Description

Quickly calculate which area needed for a homology bound; here to clean up code above

Usage

```
get_area(r, rmin, bound)
```

Arguments

`r`	side length (square) or radius (circle, annulus)
`rmin`	radius of inner circle for annulus
`bound`	square, circle, or annulus

Value

area, number

`get_volume` *Get volume*

Description

Quickly calculate which volume needed for a homology bound; here to clean up code above

Usage

```
get_volume(r, rmin, bound)
```

Arguments

`r`	side length (cube) or radius (sphere, shell)
`rmin`	radius of inner sphere for shell
`bound`	cube, sphere, shell

Value

volume, number

n_bound_connect_2D — *n Bound Connect 2D*

Description

This is the bound for connectivity based on samples.

Usage

```
n_bound_connect_2D(alpha, delta = 0.05, r = 1, rmin = 0.01, bound = "square")
```

Arguments

alpha	alpha parameter for alpha shape
delta	probability of isolated point
r	length of square, radius of circle, or outer radius of annulus
rmin	inner radius of annulus
bound	manifold shape, options are "square", "circle", or "annulus"

Value

minimum number of points to meet probability threshold.

n_bound_connect_3D — *N Bound Connect 3D*

Description

Function returns the minimum number of points to preserve the homology with an open cover of radius alpha.

Usage

```
n_bound_connect_3D(alpha, delta = 0.05, r = 1, rmin = 0.01, bound = "cube")
```

Arguments

alpha	radius of open balls around points
delta	probability of isolated point
r	radius of sphere, outer radius of shell, or length of cube side
rmin	inner radius of shell
bound	manifold from which points sampled. Options are sphere, shell, cube

Value

integer of minimum number of points needed

Examples

```
# For a cube with probability 0.05 of isolated points
n_bound_connect_3D(0.2, 0.05,0.9)
# For a sphere with probability 0.01 of isolated points
n_bound_connect_3D(0.2, 0.01, 1, bound="sphere")
# For a shell with probability 0.1 isolated points.
n_bound_connect_3D(0.2, 0.1, 1, 0.25, bound="shell")
```

---

n_bound_homology_2D *n Bound Homology 2D*

Description

#' Function returns the minimum number of points to preserve the homology with an open cover of radius alpha.

Usage

```
n_bound_homology_2D(area, epsilon, tau = 1, delta = 0.05)
```

Arguments

area	area of manifold from which points being sampled
epsilon	size of balls of cover
tau	number bound
delta	probability of not recovering homology

Value

n, number of points needed

n_bound_homology_3D *n Bound Homology 3D*

Description

Calculates number of points needed to be samped from manifold for open ball cover to have same homology as original manifold. See Niyogi et al 2008

Usage

```
n_bound_homology_3D(volume, epsilon, tau = 1, delta = 0.05)
```

Arguments

volume	volume of manifold from which points being sampled
epsilon	size of balls of cover
tau	number bound
delta	probability of not recovering homology

Value

n, number of points needed

readOFF *Read OFF File*

Description

This is a function to read OFF files for triangular meshes into the form that is required to use other functions in the package.

Usage

```
readOFF(file_name)
```

Arguments

file_name	path and name of file to be read

Value

complex_info list object containing two components, "Vertices" which holds the vertex coordinates and "cmplx" which holds the complex list object.

read_alpha_txt — *Read alpha text file*

Description

Read alpha text file

Usage

```
read_alpha_txt(file_name)
```

Arguments

`file_name`	name and path of file to be read. File is of format output by write_alpha_txt function

Value

alpha shape object

runif_annulus — *Uniform Sampling from Annulus*

Description

Returns points uniformly sampled from annulus in plane

Usage

```
runif_annulus(n, rmax = 1, rmin = 0.5)
```

Arguments

`n`	number of points to sample
`rmax`	radius of outer circle of annulus
`rmin`	radius of inner circle of annulus

Value

n by 2 matrix of points sampled

Examples

```
# Sample 100 points from annulus with rmax=1 and rmin=0.5
runif_annulus(100)
# Sample 100 points from annulus with rmax=0.75 and rmin=0.25
runif_annulus(100, 0.75, 0.25)
```

`runif_ball_3D` *Uniform Ball 3D*

Description

Returns points uniformly centered from closed ball of radius r in 3D space

Usage

```
runif_ball_3D(n, r = 1)
```

Arguments

`n`	number of points
`r`	radius of ball, default r=1

Value

n by 3 matrix of points

Examples

```
# Sample 100 points from unit ball
runif_ball_3D(100)
# Sample 100 points from ball of radius 0.5
runif_ball_3D(100, r=0.5)
```

`runif_cube` *r Uniform Cube*

Description

Returns points uniformly sampled from cube or rectangular prism in space.

Usage

```
runif_cube(n, xmin = 0, xmax = 1, ymin = 0, ymax = 1, zmin = 0, zmax = 1)
```

Arguments

`n`	number of points to be sampled
`xmin`	miniumum x coordinate
`xmax`	maximum x coordinate
`ymin`	minimum y coordinate
`ymax`	maximum y coordinate
`zmin`	minimum z coordinate
`zmax`	maximum z coordinate

Value

n by 3 matrix of points

Examples

```
# Sample 100 points from unit cube
runif_cube(100)
# Sample 100 points from unit cube centered on origin
runif_cube(100, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
```

`runif_disk` *Uniform sampling from disk*

Description

Returns points uniformly sampled from disk of radius r in plane

Usage

```
runif_disk(n, r = 1)
```

Arguments

n	number of points to sample
r	radius of disk

Value

points n by 2 matrix of points sampled

Examples

```
# Sample 100 points from unit disk
runif_disk(100)
# Sample 100 points from disk of radius 0.7
runif_disk(100, 0.7)
```

runif_shell_3D *Uniform Shell 3D*

Description

Returns points uniformly sampled from spherical shell in 3D

Usage

```
runif_shell_3D(n, rmax = 1, rmin = 0.5)
```

Arguments

n	number of points
rmax	radius of outer sphere
rmin	radius of inner sphere

Value

n by 3 matrix of points

Examples

```
# Sample 100 points with defaults rmax=1, rmin=0.5
runif_shell_3D(100)
# Sample 100 points with rmax=0.75, rmin=0.25
runif_shell_3D(100, 0.75, 0.25)
```

runif_square *Uniform Sampling from Square*

Description

Returns points uniformly sampled from square or rectangle in plane.

Usage

```
runif_square(n, xmin = 0, xmax = 1, ymin = 0, ymax = 1)
```

Arguments

n	number of points
xmin	minimum x coordinate
xmax	maximum x coordinate
ymin	minimum y coordinate
ymax	maximum y coordinate

Value

n by 2 matrix of points

Examples

```
# Sample 100 points from unit square
runif_square(100)
# Sample 100 points from unit square centered at origin
runif_square(100, 0.5, 0.5, 0.5, 0.5)
```

`sampling2Dashape` *Sampling 2D alpha shapes*

Description

This function takes parameter input from user and returns list of two dimensional alpha shape objects from the ahull package.

Usage

```
sampling2Dashape(
  N,
  n.dependent = TRUE,
  nconnect = TRUE,
  nhomology = FALSE,
  n.noise = FALSE,
  afixed = FALSE,
  mu = 0.24,
  sigma = 0.05,
  delta = 0.05,
  n = 20,
  alpha = 0.24,
  lambda = 3,
  r = 1,
  rmin = 0.25,
  bound = "square"
)
```

Arguments

`N`	number of alpha shapes to sample
`n.dependent`	boolean, whether the number of points n are dependent on alpha
`nconnect`	boolean, whether user wants shapes to have one connected component with high probability
`nhomology`	boolean, whether user wants shapes to preserve homology of underlying manifold with high probability

`n.noise`	boolean, whether to add noise variable to number of points n for more variety in shapes
`afixed`	boolean, whether alpha is fixed for all shapes sampled
`mu`	mean value of truncated normal from which alpha is sampled
`sigma`	standard deviation of truncated normal distribution from which alpha is sampled
`delta`	probability of getting disconnected shape or not preserving homology
`n`	minimum number of points to be sampled for each alpha shape
`alpha`	chosen fixed alpha; only used if afixed = TRUE
`lambda`	parameter for adding noise to n; only used if n.noise=TRUE
`r`	length of radius of circle, side length of square, or outer radius of annulus
`rmin`	inner radius of annulus
`bound`	compact manifold to be sampled from; either square, circle, or annulus

Value

list of alpha shapes of length N

`sampling3Dashape` *Sample 3D alpha shapes*

Description

This function takes parameter input from user and returns list of three dimensional alpha shape objects from the ahull package.

Usage

```
sampling3Dashape(
  N,
  n.dependent = TRUE,
  nconnect = TRUE,
  nhomology = FALSE,
  n.noise = FALSE,
  afixed = FALSE,
  mu = 0.24,
  sigma = 0.05,
  delta = 0.05,
  n = 20,
  alpha = 0.24,
  lambda = 3,
  r = 1,
  rmin = 0.25,
  bound = "cube"
)
```

Arguments

`N`	number of alpha shapes to sample
`n.dependent`	boolean, whether the number of points n are dependent on alpha
`nconnect`	boolean, whether user wants shapes to have one connected component with high probability
`nhomology`	boolean, whether user wants shapes to preserve homology of underlying manifold with high probability
`n.noise`	boolean, whether to add noise variable to number of points n for more variety in shapes
`afixed`	boolean, whether alpha is fixed for all shapes sampled
`mu`	mean value of truncated normal from which alpha is sampled
`sigma`	standard deviation of truncated normal distribution from which alpha is sampled
`delta`	probability of getting disconnected shape or not preserving homology
`n`	minimum number of points to be sampled for each alpha shape
`alpha`	chosen fixed alpha; only used if afixed = TRUE
`lambda`	parameter for adding noise to n; only used if n.noise=TRUE
`r`	length of radius of circle, side length of square, or outer radius of annulus
`rmin`	inner radius of annulus
`bound`	compact manifold to be sampled from; either cube, sphere, or shell

Value

list of alpha shapes of length N

`sphere_overlap_cs` *sphere overlap when one is centered on circumference of the other*

Description

Sphere overlap cs is subfunction for repeated code in calc_overlap_3D Returns the area of two overlapping spheres where one is centered on the other's surface (cs = centered on surface)

Usage

```
sphere_overlap_cs(alpha, r)
```

Arguments

`alpha`	radius 1
`r`	radius 2

Value

volume of intersection

sphere_overlap_is — *sphere overlap inner shell*

Description

Sphere overlap is (inner shell) calculates area needed to subtract when calculating volume of overlap of shell and sphere.

Usage

```
sphere_overlap_is(alpha, rmax, rmin)
```

Arguments

`alpha`	radius of sphere
`rmax`	outer radius of shell
`rmin`	inner radius of shell

Value

volume of intersection

spherical_cap — *Spherical cap*

Description

Calculates the volume of a sphere cap given radius r and height of cap h

Usage

```
spherical_cap(r, h)
```

Arguments

`r`	radius
`h`	height of cap

Value

v_c volume of spherical cap

tau_bound *tau_bound*

Description

This function finds the bound of tau for one shape, which is the maximum length of the fiber bundle off of a shape for determining the density of points necessary to recover the homology from the open cover. See Niyogi et al 2008. Function checks length of edges and distances to circumcenters from each vertex before checking against the rest of the point cloud and finds the minimum length. We then keep the largest tau to account for the possibility of nonuniformity among points.

Usage

```
tau_bound(v_list, complex, extremes = NULL, cores = 1, sumstat = "mean")
```

Arguments

`v_list`	matrix or data frame of cartesian coordinates of vertices in in point cloud
`complex`	list of each vertex, edge, face, and (in 3D) tetrahedron in a simplicial complex; same form as complex object in TDA package
`extremes`	matrix or data frame of cartesian coordinates of vertices on the boundary of the data frame. If no list given, function will assume all points are extreme and check them all. Inclusion of this parameter speeds up the process both within this function and when calculating alpha because you will get a bigger (but still valid) tau bound.
`cores`	number of cores for parallelizing. Default 1.
`sumstat`	string for summary statistic to be used to get final tau for shape. Default is 'mean'. Options are 'median', 'min', and 'max'.

Value

tau_vec, vector real nonnegative number. Tau values for each point

write_alpha_txt *Write Alpha Text file*

Description

Write Alpha Text file

Usage

```
write_alpha_txt(ashape, file_name)
```

Arguments

`ashape`	alpha shape object, can be 2D or 3D alpha shape
`file_name`	path and name of file to create and write text to

Value

does not return anything; writes file that can be read back to R via read_alpha_txt

Index