

# Package: DatabionicSwarm (via r-universe)

August 9, 2024

**Type** Package

**License** GPL-3

**Title** Swarm Intelligence for Self-Organized Clustering

**Version** 1.3.0

**Date** 2023-10-30

**Maintainer** Michael Thrun <m.thrun@gmx.net>

**Description** Algorithms implementing populations of agents that interact with one another and sense their environment may exhibit emergent behavior such as self-organization and swarm intelligence. Here, a swarm system called Databionic swarm (DBS) is introduced which was published in Thrun, M.C., Ultsch A.: ``Swarm Intelligence for Self-Organized Clustering" (2020), Artificial Intelligence, <DOI:10.1016/j.artint.2020.103237>. DBS is able to adapt itself to structures of high-dimensional data such as natural clusters characterized by distance and/or density based structures in the data space. The first module is the parameter-free projection method called Pswarm (Pswarm()), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is the parameter-free high-dimensional data visualization technique, which generates projected points on the topographic map with hypsometric tints defined by the generalized U-matrix (GeneratePswarmVisualization()). The third module is the clustering method itself with non-critical parameters (DBSclustering()). Clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole. It enables even a non-professional in the field of data mining to apply its algorithms for visualization and/or clustering to data sets with completely different structures drawn from diverse research fields. The comparison to common projection methods can be found in the book of Thrun, M.C.: ``Projection Based Clustering through Self-Organization and Swarm Intelligence" (2018) <DOI:10.1007/978-3-658-20540-9>.

**Imports** Rcpp (>= 1.0.8), RcppParallel (>= 5.1.4), deldir, GeneralizedUmatrix

**Suggests** DataVisualizations, knitr (>= 1.12), rmarkdown (>= 0.9),  
 plotrix, geometry, sp, spdep, parallel, rgl, png,  
 ProjectionBasedClustering, parallelDist, pracma, dendextend

**LinkingTo** Rcpp, RcppArmadillo, RcppParallel

**Depends** R (>= 3.0)

**NeedsCompilation** yes

**SystemRequirements** GNU make, pandoc (>=1.12.3, needed for vignettes)

**LazyLoad** yes

**LazyData** TRUE

**URL** <https://www.deepbionics.org/>

**Encoding** UTF-8

**VignetteBuilder** knitr

**BugReports** <https://github.com/Mthrun/DatabionicSwarm/issues>

**Repository** <https://mthrun.r-universe.dev>

**RemoteUrl** <https://github.com/mthrun/databionicswarm>

**RemoteRef** HEAD

**RemoteSha** 442e97529e88b51ced34360722a23ede36a21e3e

## Contents

DatabionicSwarm-package . . . . .	3
DBSclustering . . . . .	6
DefaultColorSequence . . . . .	8
Delaunay4Points . . . . .	9
DelaunayClassificationError . . . . .	10
Delta3DWeightsC . . . . .	12
DijkstraSSSP . . . . .	12
findPossiblePositionsCsingle . . . . .	13
GeneratePswarmVisualization . . . . .	14
getCartesianCoordinates . . . . .	17
getUmatrix4Projection . . . . .	18
Hepta . . . . .	19
Lsun3D . . . . .	20
plotSwarm . . . . .	20
ProjectedPoints2Grid . . . . .	21
Pswarm . . . . .	22
PswarmCurrentRadiusC2botsPositive . . . . .	24
rDistanceToroidCsingle . . . . .	25
RelativeDifference . . . . .	26
sESOM4BMUs . . . . .	27
setdiffMatrix . . . . .	29
setGridSize . . . . .	29
setPolarGrid . . . . .	30

setRmin . . . . .	31
ShortestGraphPathsC . . . . .	32
trainstepC . . . . .	33
UniquePoints . . . . .	34
<b>Index</b>	<b>37</b>

---

DatabionicSwarm-package

*Swarm Intelligence for Self-Organized Clustering*


---

## Description

Algorithms implementing populations of agents that interact with one another and sense their environment may exhibit emergent behavior such as self-organization and swarm intelligence. Here, a swarm system called Databionic swarm (DBS) is introduced which was published in Thrun, M.C., Ultsch A.: "Swarm Intelligence for Self-Organized Clustering" (2020), Artificial Intelligence, <DOI:10.1016/j.artint.2020.103237>. DBS is able to adapt itself to structures of high-dimensional data such as natural clusters characterized by distance and/or density based structures in the data space. The first module is the parameter-free projection method called Pswarm (Pswarm()), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is the parameter-free high-dimensional data visualization technique, which generates projected points on the topographic map with hypsometric tints defined by the generalized U-matrix (GeneratePswarmVisualization()). The third module is the clustering method itself with non-critical parameters (DBSclustering()). Clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole. It enables even a non-professional in the field of data mining to apply its algorithms for visualization and/or clustering to data sets with completely different structures drawn from diverse research fields. The comparison to common projection methods can be found in the book of Thrun, M.C.: "Projection Based Clustering through Self-Organization and Swarm Intelligence" (2018) <DOI:10.1007/978-3-658-20540-9>.

## Details

For a brief introduction to **DatabionicSwarm** please see the vignette **Short Intro to the Databionic Swarm (DBS)**. The license is CC BY-NC-SA 4.0.

Index: This package was not yet installed at build time.

## Note

For interactive Island Generation of a generalized Umatrix see interactiveGeneralizedUmatrixIsland function in the package **ProjectionBasedClustering**.

If you want to verify your clustering result externally, you can use Heatmap or SilhouettePlot of the CRAN package **DataVisualizations**.

**Author(s)**

Michal Thrun

Maintainer: Michael Thrun <m.thrun@gmx.net>

**References**

- [Thrun/Ultsch, 2021] Thrun, M. C., and Ultsch, A.: Swarm Intelligence for Self-Organized Clustering, *Artificial Intelligence*, Vol. 290, pp. 103237, doi:10.1016/j.artint.2020.103237, 2021.
- [Thrun/Ultsch, 2021] Thrun, M. C., & Ultsch, A.: Swarm Intelligence for Self-Organized Clustering (Extended Abstract), in Bessiere, C. (Ed.), 29th International Joint Conference on Artificial Intelligence (IJCAI), Vol. IJCAI-20, pp. 5125–5129, doi:10.24963/ijcai.2020/720, Yokohama, Japan, Jan., 2021.
- [Thrun/Ultsch, 2020] Thrun, M. C., & Ultsch, A.: Uncovering High-Dimensional Structures of Projections from Dimensionality Reduction Methods, *MethodsX*, Vol. 7, pp. 101093, DOI doi:10.1016/j.mex.2020.101093, 2020.
- [Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, doi:10.1007/9783658205409, 2018.
- [Ultsch/Thrun, 2017] Ultsch, A., & Thrun, M. C.: Credible Visualizations for Planar Projections, in Cottrell, M. (Ed.), 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM), IEEE Xplore, France, 2017.
- [Thrun et al., 2016] Thrun, M. C., Lerch, F., Loetsch, J., & Ultsch, A.: Visualization and 3D Printing of Multivariate Data of Biomarkers, in Skala, V. (Ed.), International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG), Vol. 24, Plzen, http://wscg.zcu.cz/wscg2016/short/A43-full.pdf, 2016.
- Successfully used in
- [Thrun et al., 2018] Thrun, M. C., Breuer, L., & Ultsch, A. : Knowledge discovery from low-frequency stream nitrate concentrations: hydrology and biology contributions, *Proc. European Conference on Data Analysis (ECDA)*, pp. 46-47, Paderborn, Germany, 2018.
- [Weyer-Menkhoff et al., 2018] Weyer-Menkhoff, I., Thrun, M. C., & Loetsch, J.: Machine-learned analysis of quantitative sensory testing responses to noxious cold stimulation in healthy subjects, *European Journal of Pain*, Vol. 22(5), pp. 862-874, DOI doi:10.1002/ejp.1173, 2018.
- [Kringel et al., 2018] Kringel, D., Geisslinger, G., Resch, E., Oertel, B. G., Thrun, M. C., Heine-mann, S., & Loetsch, J. : Machine-learned analysis of the association of next-generation sequencing based human TRPV1 and TRPA1 genotypes with the sensitivity to heat stimuli and topically applied capsaicin, *Pain*, Vol. 159 (7 ), pp. 1366-1381, DOI doi:10.1097/j.pain.0000000000001222, 2018
- [Thrun, 2019] Thrun, M. C.: : Cluster Analysis of Per Capita Gross Domestic Products, *Entrepreneurial Business and Economics Review (EBER)*, Vol. 7(1), pp. 217-231, DOI: doi:10.15678/EBER.2019.070113, 2019.
- [Lopez-Garcia et al., 2020] Lopez-Garcia, P., Argote, D. L., & Thrun, M. C.: Projection-based Classification of Chemical Groups and Provenance Analysis of Archaeological Materials, *IEEE Access*, Vol. 8, pp. 152439-152451, DOI doi:10.1109/ACCESS.2020.3016244, 2020.

**Examples**

```

data('Lsun3D')
##2d projection, without instant visualization of steps

#Alternative I:
#DistanceMatrix hast to be defined by the user.
InputDistances=as.matrix(dist(Lsun3D$Data))

projection=Pswarm(InputDistances)
#2d projection, with instant visualization

## Not run:
#Alternative II: DataMatrix, Distance is Euclidean per default
projection=Pswarm(Lsun3D$Data,Cls=Lsun3D$Cls,PlotIt=T)

## End(Not run)
#
##Computation of Generalized Umatrix
# If Non Euclidean Distances are used, Please Use \code{MDS}
# from the ProjectionBasedClustering package with the correct OutputDimension
# to generate a new DataMatrix from the distances (see SheppardDiagram
# or KruskalStress)
genUmatrixList=GeneratePswarmVisualization(Data = Lsun3D$Data,

projection$ProjectedPoints,projection$LC)
## Visualizuation of GenerelizedUmatrix,
# Estimation of the Number of Clusters=Number of valleys
library(GeneralizedUmatrix)#install if not installed
GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix,genUmatrixList$Bestmatches)
## Automatic Clustering
# number of Cluster from dendrogram (PlotIt=TRUE) or visualization
Cls=DBScustering(k=3, Lsun3D$Data,

genUmatrixList$Bestmatches, genUmatrixList$LC,PlotIt=FALSE)
# Verification, often its better to mark Outliers manually

GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix,genUmatrixList$Bestmatches,Cls)

## Not run:
# To generate the 3D landscape in the shape of an island
# from the toroidal topograpic map visualization
# you may cut your island interactivly around high mountain ranges
Imx = ProjectionBasedClustering::interactiveGeneralizedUmatrixIsland(genUmatrixList$Umatrix,
genUmatrixList$Bestmatches,Cls)

GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix,
genUmatrixList$Bestmatches, Cls=Cls,Imx = Imx)

## End(Not run)
## Not run:
library(ProjectionBasedClustering)#install if not installed
Cls2=ProjectionBasedClustering::interactiveClustering(genUmatrixList$Umatrix,

```

```

genUmatrixList$Bestmatches, Cls)

## End(Not run)

```

---

DBSclustering

*Databonic swarm clustering (DBS)*


---

## Description

DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm [Pswarm](#), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations [Thrun/Ultsch, 2021]. The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors [GeneratePswarmVisualization](#), called the generalized U-matrix. The third module is a clustering method with no sensitive parameters [DBSclustering](#) (see [Thrun, 2018, p. 104 ff]). The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

The [DBSclustering](#) function applies the automated Clustering approach of the Databonic swarm using abstract U distances, which are the geodesic distances based on high-dimensional distances combined with low dimensional graph paths by using ShortestGraphPathsC.

## Usage

```

DBSclustering(k, DataOrDistance, BestMatches, LC, StructureType = TRUE,
PlotIt = FALSE, ylab,main, method = "euclidean",...)

```

## Arguments

k	number of clusters, how many to you see in the topographic map (3D landscape)?
DataOrDistance	Either [1:n,1:d] Matrix of Data (n cases, d dimensions) that will be used. One DataPoint per row or symmetric Distance matrix [1:n,1:n]
BestMatches	[1:n,1:2] Matrix with positions of Bestmatches or ProjectedPoints, one matrix line per data point
LC	grid size c(Lines,Columns), please see details
StructureType	Optional, bool; = TRUE: compact structure of clusters assumed, =FALSE: connected structure of clusters assumed. For the two options for Clusters, see [Thrun, 2018] or Handl et al. 2006
PlotIt	Optional, bool, Plots Dendrogramm
ylab	Optional, character vector, ylabel of dendrogramm
main	Optional, character vctor, title of dendrogramm

method	Optional, one of 39 distance methods of parDist of package parallelDist, if Data matrix is chosen above
...	Further arguments passed on to the parDist function, e.g. user-defined distance functions

## Details

The input of the LC parameter depends on the choice of Bestmatches input argument. Usually as the name of the argument states, the Bestmatches of the [GeneratePswarmVisualization](#) function are used which is define in the notation of self-organizing map. In this case please see example one.

However, as written above, clustering and visualization can be applied independently of each other. In this case the places of Lines L and Columns C are switched because Lines is a value slightly above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of ProjectedPoint. Hence, one should give [DBSclustering](#) the argument LC[2, 1] as shown in example 2.

Often it is better to mark the outliers manually after the prozess of clustering and sometimes a clustering can be improved through human interaction [Thrun/Ultsch,2017] <DOI:10.13140/RG.2.2.13124.53124>; use in this case the visualization [plotTopographicMap](#) of the package GeneralizedUmatrix. If you would like to mark the outliers interactivly in the visualization use the **ProjectionBasedClustering** package with the function interactiveClustering(), or for full interactive clustering IPBC(). The package is available on CRAN. An example is shown in case of interactiveClustering() function in the third example.

## Value

[1:n] numerical vector of numbers defining the classification as the main output of this cluster analysis for the n cases of data corresponding to the n bestmatches. It has k unique numbers representing the arbitrary labels of the clustering. You can use [plotTopographicMap\(Umatrix, Bestmatches, CIs\)](#) for verification.

## Note

If you want to verify your clustering result externally, you can use Heatmap or SilhouettePlot of the package **DataVisualizations** available on CRAN.

## Author(s)

Michael Thrun

## References

[Thrun/Ultsch, 2021] Thrun, M. C., and Ultsch, A.: Swarm Intelligence for Self-Organized Clustering, Artificial Intelligence, Vol. 290, pp. 103237, [doi:10.1016/j.artint.2020.103237](#), 2021.

## Examples

```
data("Lsun3D")
Data=Lsun3D$Data
InputDistances=as.matrix(dist(Data))
```

```

projection=Pswarm(InputDistances)
## Example One
genUmatrixList=GeneratePswarmVisualization(Data,

projection$ProjectedPoints,projection$LC)

Cls=DBScustering(k=3, Data,

genUmatrixList$Bestmatches, genUmatrixList$LC,PlotIt=TRUE)

## Example Two
#automatic Clustering without GeneralizedUmatrix visualization
Cls=DBScustering(k=3, Data,

projection$ProjectedPoints, projection$LC[c(2,1)],PlotIt=TRUE)

## Not run:
## Example Three
## Sometimes an automatic Clustering can be improved
## thorough an interactive approach,
## e.g. if Outliers exist (see [Thrun/Ultsch, 2017])
library(ProjectionBasedClustering)
Cls2=ProjectionBasedClustering::interactiveClustering(genUmatrixList$Umatrix,
genUmatrixList$Bestmatches, Cls)

## End(Not run)

```

---

DefaultColorSequence    *Default color sequence for plots*

---

## Description

Defines the default color sequence for plots made within the Projections package.

## Usage

```
data("DefaultColorSequence")
```

## Format

A vector with 562 different strings describing colors for plots.



---

Delaunay4Points*Adjacency matrix of the delaunay graph for BestMatches of Points*

---

**Description**

Calculates the adjacency matrix of the delaunay graph for BestMatches (BM's) in tiled form if BestMatches are located on a toroid grid.

**Usage**

```
Delaunay4Points(Points, IsToroid = TRUE, LC, PlotIt=FALSE,  
Gabriel=FALSE)
```

**Arguments**

Points	[1:n,1:3] matrix containing the BMKey, X and Y coordinates of the n, BestMatches NEED NOT to be UNIQUE, however, there is an edge in the Deaunay between duplicate points!
IsToroid	Optional, logical, indicating if BM's are on a toroid grid. Default is True
LC	Optional, A vector of length 2, containing the number of lines and columns of the Grid. Lines is a value slightly above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of Points.
PlotIt	Optional, bool, Plots the graph
Gabriel	Optional, bool, default: FALSE, If TRUE: calculates the gabriel graph instead of the delaunay graph

**Value**

Delaunay[1:n,1:n] adjacency matrix of the Delaunay-Graph

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

---

DelaunayClassificationError

*Delaunay Classification Error (DCE)*


---

### Description

DCE searches for the k-nearest neighbors of the first delaunay neighbors weighted by the Euclidean Distances of the Inputspace. DCE evaluates these neighbors in the Output space. A low value indicates a better two-dimensional projection of the high-dimensional Input space.

### Usage

```
DelaunayClassificationError(Data,ProjectedPoints,Cls,LC,Gabriel=FALSE,
PlotIt=FALSE,Plotter = "native", Colors = NULL,LineColor= 'grey',
main = "Name of Projection", mainSize = 24,xlab = "X", ylab = "Y", xlim, ylim,
pch,lwd,Margin=list(t=50,r=0,l=0,b=0))
```

### Arguments

Data	[1:n,1:d] Numeric matrix with n cases and d variables
ProjectedPoints	[1:n,1:2] Numeric matrix with 2D points in cartesian coordinates
Cls	[1:n] Numeric vector with class labels
LC	Optional, Numeric vector of two values determining grid size of the underlying projection
Gabriel	Optional, Boolean: TRUE/FALSE => Gabriel/Delaunay graph (Default: FALSE => Delaunay)
PlotIt	Optional, Boolean: TRUE/FALSE => Plot/Do not plot (Default: FALSE)
Plotter	Optional, Character with plot technique (native or plotly)
Colors	Optional, Character vector of class colors for points
LineColor	Optional, Character of line color used for edges of graph
main	Optional, Character plot title
mainSize	Optional, Numeric size of plot title
xlab	Optional, Character name of x ax
ylab	Optional, Character name of y ax
xlim	Optional, Numeric vector with two values defining x ax range
ylim	Optional, Numeric vector with two values defining y ax range
pch	Optional, Numeric of point size (graphic parameter)
lwd	Optional, Numeric of linewidth (graphic parameter)
Margin	Optional, Margin of plotly plot

**Details**

Delaunay classification error (DCE) makes an unbiased evaluation of distance and densitybased structure which may be even non-linear separable. First, DCE utilizes the information provided by a prior classification to assess projected structures. Second, DCE applies the insights drawn from graph theory. Details are described in [Thrun/Ultsch, 2018]

**Value**

list of

DCE	DelaunayClassificationError NOTE the rest is just for development purposes
DCEperPoint	[1:n] unnormalized DCE of each point: $DCE = \text{mean}(DCEperPoint)$
nn	the number of points in a relevant neighborhood: $0.5 * 85\text{percentile}(\text{AnzNN})$
AnzNN	[1:n] the number of points with a delaunay graph neighborhood
NNdists	[1:n,1:nn] the distances within the relevant neighborhood, 0 for inner cluster distances
HD	[1:nn] $HD = \text{HarmonicDecay}(nn)$ i.e weight function for the NNdists: $DCEperPoint = HD * NNdists$

**Note**

see also chapter 6 of [Thrun, 2018]

**Author(s)**

Michael Thrun

**References**

[Thrun/Ultsch, 2018] Thrun, M. C., & Ultsch, A. : Investigating Quality measurements of projections for the Evaluation of Distance and Density-based Structures of High-Dimensional Data, Proc. European Conference on Data Analysis (ECDA), pp. accepted, Paderborn, Germany, 2018.

**Examples**

```
data(Hepta)

InputDistances=as.matrix(dist(Hepta$Data))
projection=Pswarm(InputDistances)
DelaunayClassificationError(Hepta$Data,projection$ProjectedPoints,Hepta$Cls,LC=projection$LC)$DCE
```

---

Delta3DWeightsC	<i>Intern function</i>
-----------------	------------------------

---

### Description

Implementation of the main equation for SOM, ESOM or the sESOM algorithms

### Usage

Delta3DWeightsC(vx,Datasample)

### Arguments

vx	array of weights [1:Lines,1:Columns,1:Weights]
Datasample	NumericVector of one Datapoint[1:n]

### Details

intern function in case of ComputeInR==FALSE in [GeneratePswarmVisualization](#), see chapter 5.3 of [Thrun, 2018] for generalized Umatrix and especially the sESOM4BMUs algorithm.

### Value

modified array of weights [1:Lines,1:Columns,1:]

### Author(s)

Michael Thrun

### References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

---

DijkstraSSSP	<i>Internal function: Dijkstra SSSP</i>
--------------	---

---

### Description

Dijkstra's SSSP (Single source shortest path) algorithm:

gets the shortest path (geodesic distance) from source vertice(point) to all other vertices(points) defined by the edges of the adjasency matrix

**Usage**

```
DijkstraSSSP(Adj, Costs, source)
```

**Arguments**

Adj	[1:n,1:n] 0/1 adjascency matrix, e.g. from delaunay graph or gabriel graph
Costs	[1:n,1:n] matrix, distances between n points (normally euclidean)
source	integer vertice(point) from which to calculate the geodesic distance to all other points

**Details**

Preallocating space for DataStructures accordingly to the maximum possible number of vertices which is fixed set at the number 10001. This is an internal function of [ShortestGraphPathsC](#), no errors or mis-usage is caught here.

**Value**

ShortestPaths[1:n] vector, shortest paths (geodesic) to all other vertices including the source vertice itself

**Note**

runs in  $O(E \cdot \log(V))$

**Author(s)**

Michael Thrun

**References**

uses a changed code which is inspired by Shreyans Sheth 28.05.2015, see <https://ideone.com/qkmt31>

---

```
findPossiblePositionsCsingle
```

*Intern function, do not use yourself*

---

**Description**

Finds all possible jumping position regarding a grid anda Radius for DataBots

**Usage**

```
findPossiblePositionsCsingle(RadiusPositionsschablone,  
jumplength, alpha, Lines)
```

**Arguments**

RadiusPositionsschablone	NumericMatrix, see <a href="#">setPolarGrid</a>
jumplength	double radius of databots regarding neighborhood, they can jump to
alpha	double, zu streichen
Lines	double, jumplength has to smaller than Lines/2 and Lines/2 has to yield to a integer number.

**Details**

Algorithm is described in [Thrun, 2018, p. 95, Listing 8.1].

**Value**

OpenPositions    NumericMatrix, indizes of open positions

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

**See Also**

[setPolarGrid](#)

---

GeneratePswarmVisualization

*Generates the Umatrix for Pswarm algorithm*

---

**Description**

DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm [Pswarm](#), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors [GeneratePswarmVisualization](#), called the generalized U-matrix. The third module is a clustering method with no sensitive parameters [DBScustering](#). The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

The [GeneratePswarmVisualization](#) function generates the special case (please see [Thrun, 2018]) of the generalized Umatrix with the help of an unsupervised neural network (simplified emergent

self-organizing map published in [Thrun/Ultsch, 2020]). From the generalized Umatrix a topographic map with hypsometric tints can be visualized. To see this visualization use [plotTopographicMap](#) of the package **GeneralizedUmatrix**.

## Usage

```
GeneratePswarmVisualization(Data,ProjectedPoints,LC,PlotIt=FALSE,
ComputeInR=FALSE,Parallel=TRUE)
```

## Arguments

Data	[1:n,1:d] array of data: n cases in rows, d variables in columns
ProjectedPoints	matrix, ProjectedPoints[1:n,1:2] n by 2 matrix containing coordinates of the Projection: A matrix of the fitted configuration. See output of <a href="#">Pswarm</a> for further details
LC	size of the grid c(Lines,Columns), number of Lines and Columns automatic calculated by <a href="#">setGridSize</a> in <a href="#">Pswarm</a> Sometimes is better to choose a different grid size, e.g. to to reduce computational effort contrary to SOM, here the grid size defined only the resolution of the visualizations. The real grid size is predefined by Pswarm, but you may choose a factor x*res\$LC if you so desire. Therefore, The resulting grid size is given back in the Output.
PlotIt	Optional, default(FALSE), If TRUE than uses <a href="#">plotTopographicMap</a> of the package <b>GeneralizedUmatrix</b> is plotted as a topview in the tiled option, see details for explanation.
ComputeInR	Optional, =TRUE: Rcode, =FALSE C++ implementation
Parallel	Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation

## Details

Tiled: The topographic map is visualized 4 times because the projection is toroidal. The reason is that there are no border in the visualizations and clusters (if they exist) are not disrupted by borders of the plot.

If you used [Pswarm](#) with distance matrix instead of a data matrix (in the sense that you do not have any data matrix available), you may transform your distances into data by using MDS of the **ProjectionBasedClustering** package in order to use the [GeneratePswarmVisualization](#) function. The correct dimension can be found through the Sheppard diagram or kruskals stress.

## Value

list of	
Bestmatches	Numeric matrix [1:n,1:2], BestMatches of the Umatrix, contrary to ESOM they are always fixed, because predefined by GridPoints.
Umatrix	Numeric matrix [1:Lines,1:Columns],

WeightsOfNeurons	Numeric 3D array [1:Lines,1:Columns,1:d], d is the dimension of the weights, the same as in the ESOM algorithm
GridPoints	Integer matrix [1:n,1:2], quantized projected points: projected points now lie on a predefined grid.
LC	c(Lines,Columns), normally equal to grid size of Pswarm, sometimes it a better or a lower resolution for the visualization is better. Therefore here the grid size of the neurons is given back.
PlotlyHandle	If PlotIt=FALSE: NULL, otherwise plotly object for plotting topview of topographic map.

**Note**

If you used pswarm with distance matrix instead of a data matrix you can mds transform your distances into data (see the MDS function of the ProjectionBasedClustering package.). The correct dimension can be found through the Sheppard diagram or kruskals stress.

**Note**

The extraction of an island out of the generalized Umatrix can be performed using the interactiveGeneralizedUmatrixIsland function in the package **ProjectionBasedClustering**.

The main code of both functions GeneralizedUmatrix and GeneratePswarmVisualization is the same C++ function sESOM4BMUs which is described in [Thrun/Ultsch, 2020].

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

[Thrun/Ultsch, 2020] Thrun, M. C., & Ultsch, A.: Uncovering High-Dimensional Structures of Projections from Dimensionality Reduction Methods, MethodsX, Vol. 7, pp. 101093, [doi:10.1016/j.mex.2020.101093](#), 2020.

**See Also**

[Pswarm](#) and [plotTopographicMap](#) and [GeneralizedUmatrix](#) of the package **GeneralizedUmatrix**

**Examples**

```
data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))

projList=Pswarm(InputDistances)
```



```

genUmatrixList=GeneratePswarmVisualization(Data,
  projList$ProjectedPoints,projList$LC,
  Parallel=FALSE)#CRAN guidelines do not allow =TRUE for testing
library(GeneralizedUmatrix)
plotTopographicMap(genUmatrixList$Umatrix,genUmatrixList$Bestmatches,Cls)

```

---

getCartesianCoordinates

*Intern function: Transformation of Databot indizes to coordinates*


---

## Description

Transforms Databot indizes to exact cartesian coordinates on an toroid two dimensional grid.

## Arguments

DataBotsPos	[1:N] complex vector Two Indizes per Databot describing its positions in an two dimensional grid
GridRadius	[Lines,Columns] Radii Matrix of all possible Positions of DataBots in Grid, see also documentation of <a href="#">setPolarGrid</a>
GridAngle	[Lines,Columns] Angle Matrix of all possible Positions of DataBots in Grid, see also documentation of <a href="#">setPolarGrid</a>
Lines	Defines Size of planar toroid two dimensional grid
Columns	Defines Size of planar toroid two dimensional grid
QuadOrHexa	Optional, FALSE=If DataPos on hexadiagonal grid, round to 2 decimals after value, Default=TRUE

## Details

Transformation is described in [Thrun, 2018, p. 93].

## Value

BestMatchingUnits[1:N,2] coordinates on an two dimensional grid for each databot excluding unique key, such that by using [GeneratePswarmVisualization](#) a visualization of the Pswarm projection is possible

## Author(s)

Michael Thrun

## References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

---

`getUmatrix4Projection` *depricated! see GeneralizedUmatrix() Generalisierte U-Matrix fuer Projektionsverfahren*

---

### Description

depricated! see GeneralizedUmatrix()

### Usage

```
getUmatrix4Projection(Data,ProjectedPoints,
PlotIt=TRUE,Cls=NULL,toroid=T,Tiled=F,ComputeInR=F)
```

### Arguments

Data	[1:n,1:d] Numeric matrix: n cases in rows, d variables in columns
ProjectedPoints	[1:n,2]n by 2 matrix containing coordinates of the Projection: A matrix of the fitted configuration.
PlotIt	Optional,bool, default=FALSE, if =TRUE: U-Marix of every current Position of Databots will be shown
Cls	Optional, For plotting, see plotUmatrix in package Umatrix
toroid	Optional, Default=FALSE, ==FALSE planar computation ==TRUE: toroid borderless computation, set so only if projection method is also toroidal
Tiled	Optional,For plotting see plotUmatrix in package Umatrix
ComputeInR	Optional, =T: Rcode, =F Cpp Code

### Value

List with

Umatrix	[1:Lines,1:Columns] (see ReadUMX in package DataIO)
EsomNeurons	[Lines,Columns,weights] 3-dimensional numeric array (wide format), not wts (long format)
Bestmatches	[1:n,OutputDimension] GridConverted Projected Points information converted by convertProjectionProjectedPoints() to predefined Grid by Lines and Columns
gplotres	Ausgabe von ggplot
unbesetztePositionen	Umatrix[unbesetztePositionen] = NA

### Author(s)

Michael Thrun

## References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, ISBN: 978-3-658-20539-3, Heidelberg, 2018.

## Examples

```
data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
res=cmdscale(d=InputDistances, k = 2, eig = TRUE, add = FALSE, x.ret = FALSE)
ProjectedPoints=as.matrix(res$points)
# Stress = KruskalStress(InputDistances, as.matrix(dist(ProjectedPoints)))
#resUmatrix=GeneralizedUmatrix(Data,ProjectedPoints)
#plotTopographicMap(resUmatrix$Umatrix,resUmatrix$Bestmatches,Cls)
```

---

Hepta

*Hepta is part of the Fundamental Clustering Problem Suit (FCPS)*  
*[Thrun/Ultsch, 2020].*

---

## Description

clearly defined clusters, different variances

## Usage

```
data("Hepta")
```

## Details

Size 212, Dimensions 3, stored in Hepta\$Data

Classes 7, stored in Hepta\$Cls

## References

[Thrun/Ultsch, 2020] Thrun, M. C., & Ultsch, A.: Clustering Benchmark Datasets Exploiting the Fundamental Clustering Problems, Data in Brief, Vol. 30(C), pp. 105501, DOI 10.1016/j.dib.2020.105501, 2020.

## Examples

```
data(Hepta)
str(Hepta)
```

---

Lsun3D	<i>Lsun3D is part of the Fundamental Clustering Problem Suit (FCPS) [Thrun/Ultsch, 2020].</i>
--------	---

---

**Description**

clearly defined clusters, different variances

**Usage**

```
data("Lsun3D")
```

**Details**

Size 404, Dimensions 3

Dataset defined discontinuities, where the clusters have different variances. Three main Clusters, and four Outliers (in Cluster 4). See for a more detailed description in [Thrun, 2018].

**References**

[Thrun/Ultsch, 2020] Thrun, M. C., & Ultsch, A.: Clustering Benchmark Datasets Exploiting the Fundamental Clustering Problems, Data in Brief, Vol. 30(C), pp. 105501, DOI 10.1016/j.dib.2020.105501, 2020.

**Examples**

```
data(Lsun3D)
str(Lsun3D)
Cls=Lsun3D$Cls
Data=Lsun3D$Data
```

---

plotSwarm	<i>Intern function for plotting during the Pswarm annealing process</i>
-----------	---

---

**Description**

Intern function, generates a scatter plot of the progress of the Pswarm algorithm after every nash equilibrium. Every point symbolizes a Databot. If a prior classification is given (Cls) then the Databots have the colors defined by the class labels.

**Usage**

```
plotSwarm(Points,Cls,xlab,ylab,main)
```

**Arguments**

Points	ProjectedPoints or DataBot positions in cartesian coordinates
Cls	optional, Classification as a numeric vector, if given
xlab	= 'X', optional, string
ylab	= 'Y', optional, string
main	= "DataBots", optional, string

**Author(s)**

Michael Thrun

**See Also**

[Pswarm](#) with PlotIt=TRUE

---

ProjectedPoints2Grid    *Transforms ProjectedPoints to a grid*

---

**Description**

quantized xy cartesian coordinates of ProjectedPoints

**Usage**

```
ProjectedPoints2Grid(ProjectedPoints, Lines, Columns, PlotIt=FALSE, Cls)
```

**Arguments**

ProjectedPoints	[1:n,1:2] numeric matrix of cartesian xy coordinates
Lines	double, length of small side of the rectangular grid
Columns	double, length of big side of the rectangular grid
PlotIt	optional, bool, shows the result if TRUE
Cls	[1:n] numeric vector of classes for each projected point

**Details**

intern function, described in [Thrun, 2018, p.47]

**Value**

BestMatches[1:n,1:3] columns in order: Key, Lines, Columns

**Author(s)**

Michael Thrun

## References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

## See Also

[GeneratePswarmVisualization](#)

---

Pswarm

*A Swarm of Databots based on polar coordinates (Polar Swarm).*

---

## Description

This projection method is a part of the databionic swarm which uses the nash equilibrium [Thrun/Ultsch, 2021]. Using polar coordinates for agents (here Databots) in two dimensions has many advantages, for further details see [Thrun, 2018] and [Thrun/Ultsch, 2021].

## Usage

```
Pswarm(DataOrDistance,PlotIt=FALSE,Cls=NULL,Silent=TRUE,
Debug=FALSE,LC=c(NULL,NULL),method= "euclidean",Parallel=FALSE,...)
```

## Arguments

DataOrDistance	Numeric matrix [1:n,1:n]: symmetric matrix of dissimilarities, if variable un-symmetric (Numeric matrix [1:d,1:n]) it is assumed as a dataset and the euclidean distances are calculated of d variables and n cases.
PlotIt	Optional, bool, default=FALSE, If =TRUE, Plots the projection during the computation prozess after every nash equilbirum.
Cls	Optional, numeric vector [1:n], given Classification in numbers, only for plotting if PlotIt=TRUE, irrelevant for computations.
Silent	Optional, bool, default=FALSE, If =TRUE results in various console messages
Debug	Optional, Debug, default=FALSE, =TRUE results in various console messages, depricated for CRAN, because cout is not allowed.
LC	Optional, grid size c(Lines, Columns), sometimes it is better to call <a href="#">setGridSize</a> separately.
method	Optional, one of 39 distance methods of parDist of package parallelDist, if Data matrix is chosen above
Parallel	Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation
...	Further arguments passed on to the parDist function, e.g. user-defined distance functions

## Details

DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm [Pswarm](#), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors [GeneratePswarmVisualization](#), called the generalized U-matrix. The third module is a clustering method with no sensitive parameters [DBScustering](#). The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

## Value

List with

ProjectedPoints

[1:n,1:2] xy cartesian coordinates of projection

LC

number of Lines and Columns in c(Lines,Columns). Lines is a value slightly above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of ProjectedPoints

Control

List, only for intern debugging

## Note

LC is now automatically estimated; LC is the size of the grid c(Lines,Columns), number of Lines and Columns, default c(NULL,NULL) and automatic calculation by [setGridSize](#)

## Author(s)

Michael Thrun

## References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

[Thrun/Ultsch, 2021] Thrun, M. C., and Ultsch, A.: Swarm Intelligence for Self-Organized Clustering, Artificial Intelligence, Vol. 290, pp. 103237, [doi:10.1016/j.artint.2020.103237](#), 2021.

## Examples

```
data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
#If not called separately setGridSize() is called in Pswarm
LC=setGridSize(InputDistances)
res=Pswarm(InputDistances,LC=LC,Cls=Cls,PlotIt=TRUE)
```

---

PswarmCurrentRadiusC2botsPositive

*intern function, do not use yourself*


---

## Description

Finds the weak Nash equilibrium for DataBots in one epoch(Radius), requires the setting of constants, grid, and so on in [Pswarm](#)

## Usage

```
PswarmCurrentRadiusC2botsPositive( AllDataBotsPosOld, Radius, DataDists,
IndPossibleDBPosR, RadiusPositionsschablone, pp, Nullpunkt, Lines, Columns,
nBots, limit, steigungsverlaufind, StressConstAditiv, debug)
```

## Arguments

AllDataBotsPosOld	ComplexVector [1:n,1], DataBots position in the last Nash-Equilibrium
Radius	double, Radius of payoff function, neighborhood, where other DatsBots can be smelled
DataDists	NumericMatrix, Inputdistances[1:n,1:n]
IndPossibleDBPosR	ComplexVector, see output of <a href="#">findPossiblePositionsCsingle</a>
RadiusPositionsschablone	NumericMatrix, see AllallowedDBPosR0 in <a href="#">setPolarGrid</a>
pp	NumericVector, number of jumping simultaneously DataBots of one epoch (per nash-equilibrium), this vector is linearly monotonically decreasing
Nullpunkt	NumericVector, equals which(AllallowedDBPosR0==0,arr.ind=T), see see AllallowedDBPosR0 in <a href="#">setPolarGrid</a>
Lines	double, small edge length of rectangulare grid
Columns	double, big edge length of rectangulare grid
nBots	double, intern constant, equals round(pp[Radius]*DBAnzahl)
limit	int, intern constant, equals ceiling(1/pp[Radius])
steigungsverlaufind	int, intern constant
StressConstAditiv	double, intern constant, sum of payoff of all databots in random condition before the algorithm starts
debug	optional, bool: If TRUE prints status every 100 iterations

## Details

Algorithm is described in [Thrun, 2018, p. 95, Listing 8.1].



**Value**

list of

AllDataBotsPos ComplexVector, indizes of DataBot Positions after a weak Nash equilibrium is found

stressverlauf NumericVector, intern result, for debugging only

fokussiertlaufind

NumericVector, intern result, for debugging only

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

---

rDistanceToroidCsingle

*Intern function for Pswarm*


---

**Description**

toroid distance calculation

**Usage**

```
rDistanceToroidCsingle(AllDataBotsPosX, AllDataBotsPosY, AllallowedDBPosR0,
Lines, Columns, Nullpunkt)
```

**Arguments**

AllDataBotsPosX

NumericVector [1:n,1], positions of on grid

AllDataBotsPosY

NumericVector [1:n,1], positions of on grid

AllallowedDBPosR0

NumericMatrix

Lines

double

Columns

double

Nullpunkt

NumericVector

**Details**

Part of the algorithm described in [Thrun, 2018, p. 95, Listing 8.1].

**Value**

numeric matrix of toroid Distances[1:n,1:n]

**Note**

do not use yourself

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

**See Also**

[Pswarm](#)

---

RelativeDifference	<i>Relative Difference</i>
--------------------	----------------------------

---

**Description**

Calculates the difference between positive x and y values

**Usage**

```
RelativeDifference(X, Y, epsilon = 10^-10,  
na.rm=FALSE,Silent=FALSE)
```

**Arguments**

- |         |   |
|---------|---|
| X       | either a value or numerical vector of [1:n]   |
| Y       | either a value or numerical vector of [1:n]   |
| epsilon | Optional, If both x and y are approximatly zero the output is also zero   |
| na.rm   | Optional, function does not work with non finite values. If these cases should be automatically removed, set parameter TRUE |
| Silent  | Optional, if TRUE message abput values below epsilon is not given back  |

**Details**

Contrary to other approaches in this cases the range of values lies between [-2,2]. The approach is only valid for positive values of X and Y. The relative difference R is defined with

$$R = \frac{Y - X}{0.5 * (X + Y)}$$

Negative value indicate that X is higher than Y and positive values that X is lower than Y.

**Value**

R

**Note**

It can be combined with the `GabrielClassificationError` if a clear baseline is defined.

**Author(s)**

Michael Thrun

**References**

Ultsch, A.: Is Log Ratio a Good Value for Measuring Return in Stock Investments? GfKl 2008, pp, 505-511, 2008.

**See Also**

[GabrielClassificationError](#)

**Examples**

```
x=c(1:5)
y=runif(5,min=1,max=10)
RelativeDifference(x,y)
```

---

sESOM4BMUs

*Intern function: Simplified Emergent Self-Organizing Map*


---

**Description**

Intern function for the simplified ESOM (sESOM) algorithm for fixed `BestMatchingUnits`.

**Usage**

```
sESOM4BMUs(BMUs,Data, esom, toroid, CurrentRadius,
ComputeInR=FALSE,Parallel=TRUE)
```

**Arguments**

BMUs	[1:Lines,1:Columns], BestMAatchingUnits generated by ProjectedPoints2Grid()
Data	[1:n,1:d] array of data: n cases in rows, d variables in columns
esom	[1:Lines,1:Columns,1:weights] array of NeuronWeights, see ListAsEsomNeurons()
toroid	TRUE/FALSE - topology of points
CurrentRadius	number between 1 to x
ComputeInR	=T: Rcode, =F Cpp Code
	number between 1 to x
Parallel	Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation

**Details**

Algorithm is described in [Thrun, 2018, p. 48, Listing 5.1].

**Value**

esom	numeric array [1:Lines,1:Columns,1:d], d is the dimension of the weights, the same as in the ESOM algorithm. modified esomneuros regarding a predefined neighborhood defined by a radius
------	--

**Note**

Usually not for seperated usage!

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, doi:[10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

**See Also**

[GeneratePswarmVisualization](#)

---

setdiffMatrix	<i>setdiffMatrix shortens Matrix2Curt by those rows that are in both matrices.</i>
---------------	--

---

### Description

setdiffMatrix shortens Matrix2Curt by those rows that are in both matrices.

### Arguments

Matrix2Curt	[n,k] matrix, which will be shortened by x rows
Matrix2compare	[m,k] matrix whose rows will be compared to those of Matrix2Curt x rows in Matrix2compare equal rows of Matrix2Curt (order of rows is irrelevant). Has the same number of columns as Matrix2Curt.

### Value

V\$CurtedMatrix[n-x,k] Shortened Matrix2Curt

### Author(s)

CL,MT 12/2014

---

setGridSize	<i>Sets the grid size for the Pswarm algorithm</i>
-------------	--

---

### Description

Automatically sets the size of the grid, formula see [Thrun, 2018, p. 93-94].

### Usage

```
setGridSize(InputDistances,minp=0.01,maxp=0.99,alpha=4)
```

### Arguments

InputDistances	[1:n,1:n] symmetric matrix of input distances
minp	default value: 0.01,see <a href="#">quantile</a> , first value in the vector of probs estimates robust minimum of distances
maxp	default value: 0.99, see <a href="#">quantile</a> , last value of the vector of probs estimates robust maximum of distances
alpha	Do not change! Intern parameter, Only if Java Version of Pswarm instead of C++ version is used.

**Details**

grid is set such that minimum and maximum distances can be shown on the grid

**Value**

LC=c(Lines,Columns) size of the grid for Pswarm

**Author(s)**

Michael Thrun, Florian Lerch

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, doi:[10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

**See Also**

automatic choice of LC for [Pswarm](#)

**Examples**

```
data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
#If not called separately setGridSize() is called in Pswarm
LC=setGridSize(InputDistances)
```

---

setPolarGrid

*Intern function: Sets the polar grid*


---

**Description**

Sets a polar grid for a swarm in an rectangular shape

**Usage**

```
setPolarGrid(Lines,Columns,QuadOrHexa,PlotIt,global)
```

**Arguments**

Lines	Integer, hast to be able to be divided by 2
Columns	Integer, with Columns>=Lines
QuadOrHexa	bool, default(TRUE) If False Hexagonal grid, default quad grid
PlotIt	bool, default(FALSE)
global	bool, default(TRUE), intern parameter, how shall the radii be calculated?

Details

Part of the Algorithm described in [Thrun, 2018, p. 95, Listing 8.1].

Value

list of	
GridRadii	matrix [1:Lines,1:Columns], Radii Matrix of all possible Positions of DataBots in Grid
GridAngle	matrix [1:Lines,1:Columns], Angle Matrix of all possible Positions of DataBots in Grid
AllallowedDBPosR0	matrix [1:Lines+1,1:Columns+1], Matrix of radii in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump
AllallowedDBPosPhi0	matrix [1:Lines+1,1:Columns+1], # V\$AllallowedDBPosPhi0[Lines+1,Lines+1] Matrix of angle in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump

Author(s)

Michael Thrun

References

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

See Also

[Pswarm](#)

---

setRmin	<i>Intern function: Estimates the minimal radius for the Databot scent</i>
---------	--

---

Description

estimates the minimal radius on apolar grid in the automated annealing process of Pswarm, details of how can be read in [Thrun, 2018, p. 97]

**Arguments**

Lines	x-value determining the size of the map, i.e. how many open places for DataBots will be available on the 2-dimensional grid BEWARE: has to be able to be divided by 2
Columns	y-value determining the size of the map, i.e. how many open places for DataBots will be available on the 2-dimensional grid Columns>Lines
AllallowedDBPosR0	[1:Lines+1,1:Lines+1]Matrix of radii in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump
p	percent of gitterpositions, which should be considered

**Value**

Rmin Minimum Radius

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

---

ShortestGraphPathsC     *Shortest GraphPaths = geodesic distances*

---

**Description**

Dijkstra's SSSP (Single source shortest path) algorithm, from all points to all points

**Usage**

ShortestGraphPathsC(Adj, Cost)

**Arguments**

Adj	[1:n,1:n] 0/1 adjascency matrix, e.g. from delaunay graph or gabriel graph.
Cost	[1:n,1:n] matrix, distances between n points (normally euclidean)

**Details**

Vertices are the points, edges have the costs defined by weights (normally a distance). The algorithm runs in runs in  $O(n \cdot E \cdot \log(V))$ , see also [Jungnickel, 2013, p. 87]. Further details can be found in [Jungnickel, 2013, p. 83-87] and [Thrun, 2018, p. 12].



**Value**

ShortestPaths[1:n,1:n] vector, shortest paths (geodesic) to all other vertices including the source vertex itself from all vertices to all vertices, stored as a matrix

**Note**

require C++11 standard (set flag in Compiler, if not set automatically)

**Author(s)**

Michael Thrun

**References**

[Dijkstra,1959] Dijkstra, E. W.: A note on two problems in connexion with graphs, Numerische mathematik, Vol. 1(1), pp. 269-271. 1959.

[Jungnickel, 2013] Jungnickel, D.: Graphs, networks and algorithms, (4th ed ed. Vol. 5), Berlin, Heidelberg, Germany, Springer, ISBN: 978-3-642-32278-5, 2013.

[Thrun/Ultsch, 2017] Thrun, M.C., Ultsch, A.: Projection based Clustering, Conf. Int. Federation of Classification Societies (IFCS),DOI:10.13140/RG.2.2.13124.53124, Tokyo, 2017.

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](https://doi.org/10.1007/9783658205409), 2018.

**See Also**

[DijkstraSSSP](#)

---

trainstepC

---

*Internal function for sESOM*


---

**Description**

Does the training for fixed bestmatches in one epoch of the sESOM algorithm (see [Thrun, 2018] for details).

**Usage**

```
trainstepC(vx,vy, DataSampled,BMUSampled,Lines,Columns, Radius, toroid)
```

**Arguments**

vx	array [1:Lines,1:Columns,1:Weights], WeightVectors that will be trained, internally transformed von NumericVector to cube
vy	array [1:Lines,1:Columns,1:2], meshgrid for output distance computation
DataSampled	NumericMatrix, n cases shuffled Dataset[1:n,1:d] by sample
BMUsampled	NumericMatrix, n cases shuffled BestMatches[1:n,1:2] by sample in the same way as DataSampled
Lines	double, Height of the grid
Columns	double, Width of the grid
Radius	double, The current Radius that should be used to define neighbours to the bm
toroid	bool, Should the grid be considered with cyclically connected borders?

**Details**

Algorithm is described in [Thrun, 2018, p. 48, Listing 5.1].

**Value**

WeightVectors, array[1:Lines,1:Columns,1:weights] with the adjusted Weights

**Note**

Usually not for seperated usage!

**Author(s)**

Michael Thrun

**References**

[Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm Intelligence, doctoral dissertation 2017, Springer, Heidelberg, ISBN: 978-3-658-20539-3, [doi:10.1007/9783658205409](#), 2018.

---

UniquePoints

*Unique Points*


---

**Description**

return only the unique points in Datapoints

**Usage**

```
UniquePoints(Datapoints, Cls, Eps=1e-10)
```

**Arguments**

Datapoints	[1:n,1:d] numeric matrix of Datapoints points of dimension d, the points are in the rows
Cls	[1:n] numeric vector of classes for each datapoint.
Eps	Optional, scalar above zero that defines minimum non-identical euclidean distance between two points

**Details**

Euclidean distance is computed and used within. Setting Eps to a very small number results in the identification of unique data points. Setting epsilon to a higher number results in the definition of mesh points within an d-dimensional R-ball graph.

**Value**

List with	
Unique	[1:k,1:d] Datapoints points without duplicate points
UniqueInd	[1:k] index vector such that Unique == Datapoints[UniqueInd,], it has k non-consecutive numbers or labels, each label defines a row number within Datapoints[1:n,1:d] of a unique data point
Uniq2DatapointsInd	[1:n] index vector. It has k unique index numbers representing the arbitrary labels. Each labels is mapped uniquely to a point in Unique. Logically in a way such that Datapoints == Unique[Uniq2DatapointsInd,] (will not work directly in R this way)
NewUniqueInd	[1:k] index vector stating the index of the newly defined datastructure Unique.
NewUniq2DataIdx	[1:k] index vector such that Unique[NewUniq2DataIdx,] == Datapoints[Uniq2DatapointsInd,], it has n non-consecutive numbers or labels, each label defines a row number within Unique[1:k,1:d] of a unique data point
IsDuplicate	[1:n,1:n] matrix, for $i \neq j$ IsDuplicate[i,j]== 1 if Datapoints[i,] == Datapoints[j,] IsDuplicate[i,i]==0
Eps	Numeric stating the neighborhood radius around unique points.

**Author(s)**

Michael Thrun

**Examples**

```
Datapoints = rbind(c(0,0), c(1,1), c(2,2))
Datapoints2 = rbind(Datapoints, Datapoints+0.001)
Datapoints3 = rbind(Datapoints2, c(1,1)-0.001)

Datapoints = rbind(c(0,0), c(0,0.015), c(0,0.01), c(0,0.015))
```

```
V1 = UniquePoints(Datapoints = Datapoints, Eps = 0.01)
V2 = UniquePoints(Datapoints = Datapoints2, Eps = 0.01)
V3 = UniquePoints(Datapoints = Datapoints3, Eps = 0.01)
```

# Index

- \* **Classification Error**
  - DelaunayClassificationError, [10](#)
- \* **DBS**
  - DatabionicSwarm-package, [3](#)
  - DBSclustering, [6](#)
  - Pswarm, [22](#)
- \* **DCE**
  - DelaunayClassificationError, [10](#)
- \* **DR**
  - DatabionicSwarm-package, [3](#)
  - GeneratePswarmVisualization, [14](#)
  - Pswarm, [22](#)
- \* **DataBionic**
  - DatabionicSwarm-package, [3](#)
- \* **Databionic swarm**
  - GeneratePswarmVisualization, [14](#)
  - Pswarm, [22](#)
- \* **Databionic**
  - DatabionicSwarm-package, [3](#)
- \* **Databonic swarm**
  - DBSclustering, [6](#)
- \* **Delaunay Classification Error**
  - DelaunayClassificationError, [10](#)
- \* **Delaunay Graph**
  - Delaunay4Points, [9](#)
- \* **Delaunay**
  - Delaunay4Points, [9](#)
- \* **Dijkstra's SSSP**
  - DijkstraSSSP, [12](#)
- \* **Dijkstra**
  - DijkstraSSSP, [12](#)
- \* **Dimensionality Reduction**
  - DatabionicSwarm-package, [3](#)
- \* **ESOM**
  - GeneratePswarmVisualization, [14](#)
  - sESOM4BMUs, [27](#)
- \* **Evaluation of projection methods**
  - DelaunayClassificationError, [10](#)
- \* **FCPS**
  - Hepta, [19](#)
  - Lsun3D, [20](#)
- \* **GeneratePswarmVisualization**
  - ProjectedPoints2Grid, [21](#)
  - sESOM4BMUs, [27](#)
- \* **Hepta**
  - Hepta, [19](#)
- \* **Lsun3D**
  - Lsun3D, [20](#)
- \* **Pswarm**
  - Pswarm, [22](#)
- \* **Points**
  - Delaunay4Points, [9](#)
- \* **Polar Swarm**
  - Pswarm, [22](#)
- \* **Pswarm**
  - DatabionicSwarm-package, [3](#)
  - plotSwarm, [20](#)
- \* **QM**
  - DelaunayClassificationError, [10](#)
- \* **Quality measurement**
  - DelaunayClassificationError, [10](#)
- \* **Quality measure**
  - DelaunayClassificationError, [10](#)
- \* **RelativeDifference**
  - RelativeDifference, [26](#)
- \* **SOM**
  - GeneratePswarmVisualization, [14](#)
- \* **SSSP**
  - DijkstraSSSP, [12](#)
- \* **ShortestGraphPaths**
  - ShortestGraphPathsC, [32](#)
- \* **ShortestPaths**
  - ShortestGraphPathsC, [32](#)
- \* **Single source shortest path**
  - DijkstraSSSP, [12](#)
- \* **U-matrix**
  - GeneratePswarmVisualization, [14](#)
- \* **Umatrix**

- GeneratePswarmVisualization, 14
- \* **Voronoi**
  - Delaunay4Points, 9
- \* **cluster analysis**
  - DatabionicSwarm-package, 3
  - DBScustering, 6
  - GeneratePswarmVisualization, 14
  - Pswarm, 22
- \* **clustering**
  - DatabionicSwarm-package, 3
  - DBScustering, 6
- \* **cluster**
  - DBScustering, 6
- \* **data points**
  - UniquePoints, 34
- \* **datasets**
  - Lsun3D, 20
- \* **dataset**
  - Hepta, 19
- \* **difference**
  - RelativeDifference, 26
- \* **dimensionality reduction**
  - Pswarm, 22
- \* **distances**
  - rDistanceToroidCsingle, 25
- \* **emergence**
  - DatabionicSwarm-package, 3
- \* **equilibrium**
  - DatabionicSwarm-package, 3
- \* **game theory**
  - DatabionicSwarm-package, 3
  - Pswarm, 22
- \* **generalized Umatrix**
  - GeneratePswarmVisualization, 14
- \* **generlized Umatrix**
  - ProjectedPoints2Grid, 21
- \* **graph**
  - Delaunay4Points, 9
- \* **grid**
  - setPolarGrid, 30
- \* **hexagonal**
  - setPolarGrid, 30
- \* **nash**
  - DatabionicSwarm-package, 3
  - PswarmCurrentRadiusC2botsPositive, 24
- \* **pSwarm**
  - Pswarm, 22
- \* **package**
  - DatabionicSwarm-package, 3
- \* **points**
  - UniquePoints, 34
- \* **polar**
  - PswarmCurrentRadiusC2botsPositive, 24
- \* **positions**
  - findPossiblePositionsCsingle, 13
- \* **projection method**
  - DatabionicSwarm-package, 3
  - Pswarm, 22
- \* **projection**
  - DatabionicSwarm-package, 3
  - Pswarm, 22
- \* **pswarm**
  - Pswarm, 22
- \* **relative**
  - RelativeDifference, 26
- \* **sESOM**
  - GeneratePswarmVisualization, 14
- \* **self-organization**
  - DatabionicSwarm-package, 3
- \* **self-organizing-map**
  - GeneratePswarmVisualization, 14
- \* **swarm intelligence**
  - DatabionicSwarm-package, 3
  - Pswarm, 22
- \* **swarms**
  - findPossiblePositionsCsingle, 13
  - Pswarm, 22
- \* **swarm**
  - DatabionicSwarm-package, 3
  - DBScustering, 6
  - GeneratePswarmVisualization, 14
  - plotSwarm, 20
  - Pswarm, 22
  - PswarmCurrentRadiusC2botsPositive, 24
  - setPolarGrid, 30
- \* **toroid**
  - rDistanceToroidCsingle, 25
- \* **unique**
  - UniquePoints, 34
- \* **visualization**
  - DatabionicSwarm-package, 3
  - GeneratePswarmVisualization, 14
  - Pswarm, 22

DatabionicSwarm  
    (DatabionicSwarm-package), 3  
DatabionicSwarm-package, 3  
DBSclustering, 6, 6, 7, 14, 23  
DefaultColorSequence, 8  
Delaunay4Points, 9  
DelaunayClassificationError, 10  
Delta3DWeightsC, 12  
DijkstraSSSP, 12, 33  
  
findPossiblePositionsCsingle, 13, 24  
  
GabrielClassificationError, 27  
GeneralizedUmatrix, 16  
GeneratePswarmVisualization, 6, 7, 12, 14,  
    14, 15, 17, 22, 23, 28  
getCartesianCoordinates, 17  
getUmatrix4Projection, 18  
  
Hepta, 19  
  
Lsun3D, 20  
  
plotSwarm, 20  
plotTopographicMap, 7, 15, 16  
ProjectedPoints2Grid, 21  
Pswarm, 6, 14–16, 21, 22, 23–26, 30, 31  
pswarm (Pswarm), 22  
pswarmCpp (Pswarm), 22  
PswarmCurrentRadiusC2botsPositive, 24  
  
quantile, 29  
  
rDistanceToroidCsingle, 25  
RelativeDifference, 26  
  
sESOM4BMUs, 27  
setdiffMatrix, 29  
setGridSize, 15, 22, 23, 29  
setPolarGrid, 14, 17, 24, 30  
setRmin, 31  
ShortestGraphPathsC, 13, 32  
  
trainstepC, 33  
  
UniquePoints, 34