Package: DatabionicSwarm (via r-universe)

October 29, 2026

Type Package

License GPL-3

Title Swarm Intelligence for Self-Organized Clustering

Version 2.0.0

Date 2024-06-20

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, ABCanalysis, ggplot2

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

Author Michael Thrun [aut, cre, cph] (<https://orcid.org/0000-0001-9542-5543>), Quirin Stier [aut, rev] (<https://orcid.org/0000-0002-7896-4737>)

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

Config/pak/sysreqs make

Repository https://cran.r-universe.dev

Date/Publication 2024-06-21 02:41:57 UTC

RemoteUrl https://github.com/cran/DatabionicSwarm

RemoteRef HEAD

RemoteSha d80080cd8bb398ba375e0b61103187fe8b9a3481

Contents

DatabionicSwarm-package . . . 3
DBSclustering . . . 7
DefaultColorSequence . . . 9
Delaunay4Points . . . 10
Delta3DWeightsC . . . 11
DijkstraSSSP . . . 11
findPossiblePositionsCsingle . . . 12
GeneratePswarmVisualization . . . 13
getCartesianCoordinates . . . 16
getUmatrix4Projection . . . 17
Hepta . . . 18
Lsun3D . . . 19
plotSwarm . . . 20
ProjectedPoints2Grid . . . 20
Pswarm . . . 21

PswarmEpochsParallel 23
PswarmEpochsSequential 25
PswarmRadiusParallel 27
PswarmRadiusSequential 29
rDistanceToroidCsingle 30
RelativeDifference 31
RobustNorm_BackTrafo 32
RobustNormalization 33
sESOM4BMUs 35
setdiffMatrix 36
setGridSize 37
setPolarGrid 38
setRmin 39
ShortestGraphPathsC 40
trainstepC 41
trainstepC2 42
UniquePoints 43

Index **45**

`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 of help topics:

```
DBSclustering           Databonic swarm clustering (DBS)
DatabionicSwarm-package
                        Swarm Intelligence for Self-Organized
                        Clustering
DefaultColorSequence    Default color sequence for plots
Delaunay4Points         Adjacency matrix of the delaunay graph for
                        BestMatches of Points
Delta3DWeightsC         intern function, do not use yourself
DijkstraSSSP            Internal function: Dijkstra SSSP
GeneratePswarmVisualization
                        Generates the Umatrix for Pswarm algorithm
Hepta                   Hepta is part of the Fundamental Clustering
                        Problem Suit (FCPS) [Thrun/Ultsch, 2020].
Lsun3D                  Lsun3D is part of the Fundamental Clustering
                        Problem Suit (FCPS) [Thrun/Ultsch, 2020].
ProjectedPoints2Grid    Transforms ProjectedPoints to a grid
Pswarm                  A Swarm of Databots based on polar coordinates
                        (Polar Swarm).
PswarmEpochsParallel    Intern function, do not use yourself
PswarmEpochsSequential
                        Intern function, do not use yourself
PswarmRadiusParallel    Intern function, do not use yourself
PswarmRadiusSequential
                        intern function, do not use yourself
RelativeDifference      Relative Difference
RobustNorm_BackTrafo    Transforms the Robust Normalization back
RobustNormalization     RobustNormalization
ShortestGraphPathsC     Shortest GraphPaths = geodesic distances
UniquePoints            Unique Points
findPossiblePositionsCsingle
                        Intern function, do not use yourself
getCartesianCoordinates
                        Intern function: Transformation of Databot
                        indizes to coordinates
getUmatrix4Projection   depricated! see GeneralizedUmatrix()
                        Generalisierte U-Matrix fuer
                        Projektionsverfahren
plotSwarm               Intern function for plotting during the Pswarm
                        annealing process
rDistanceToroidCsingle
                        Intern function for 'Pswarm'
sESOM4BMUs              Intern function: Simplified Emergent
                        Self-Organizing Map
```

setGridSize	Sets the grid size for the Pswarm algorithm
setPolarGrid	Intern function: Sets the polar grid
setRmin	Intern function: Estimates the minimal radius for the Databot scent
setdiffMatrix	setdiffMatrix shortens Matrix2Curt by those rows that are in both matrices.
trainstepC	internal function for s-esom
trainstepC2	internal function for s-esom

Note

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

If you want to verifiy 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., Heinemann, 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=DBSclustering(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 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,Cls) for verification.

Note

If you want to verifiy 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=DBSclustering(k=3, Data, genUmatrixList$Bestmatches,
genUmatrixList$LC,PlotIt=TRUE)


## Example Two
#automatic Clustering without GeneralizedUmatrix visualization
Cls=DBSclustering(k=3, Data, projection$ProjectedPoints,projection$LC,
PlotIt=TRUE)

## Not run:
## Example Three
## Sometimes an automatic Clustering can be improved
## through 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 (BMs) 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
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, 2018.

`Delta3DWeightsC` *intern function, do not use yourself*

Description

Delta3DWeightsC

Usage

```
Delta3DWeightsC(vx, Datasample)
```

Arguments

`vx`	Array [1:n,1:m,1:d] of neuron weights on a nxm grid with d dimensional weights.
`Datasample`	One observation of a d-dimensional datapoint.

Details

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

Value

`vx`	Array [1:n,1:m,1:l]

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` *Internal function: Dijkstra SSSP*

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	int, 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*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 setPolarGrid
jumplength	double radius of databots regarding neighborhood, they can jump to
alpha	double, zu streichen
Lines	double, jumpinglength 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 DBSclustering. 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, Tiled = FALSE, DataPerEpoch = 1)
```

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 Pswarm for further details
LC	size of the grid c(Lines,Columns), number of Lines and Columns automatic calculated by setGridSize in Pswarm Sometimes is better to choose a different grid size, e.g. to to reduce computional effort contrary to SOM, here the grid size defined only the resolution of the visualizations The real grid size is predfined 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 plotTopographicMap of the package GeneralizedUmatrix 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 Sequential C++ implementation
Tiled	Optional, =TRUE: arrangement of four grids for better understanding of edge behaviour, =FALSE: single grid.
DataPerEpoch	Optional: Number between 0 and 1 stating the ratio of data per epoch for training of the generalized u-matrix approach.

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`	matrix [1:n,1:2], BestMatches of the Umatrix, contrary to ESOM they are always fixed, because predefined by GridPoints.
`Umatrix`	matrix [1:Lines,1:Columns],
`WeightsOfNeurons`	array [1:Lines,1:Columns,1:d], d is the dimension of the weights, the same as in the ESOM algorithm
`GridPoints`	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 ploting 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 `interactiveGeneralizedUmatrixIsl` 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)
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.

Usage

```
getCartesianCoordinates(DataBotsPosRe, DataBotsPosIm, GridRadius, GridAngle,
QuadOrHexa = TRUE)
```

Arguments

`DataBotsPosRe`	[1:N] real part of complex vector Two Indizes per Databot describing its positions in an two dimensional grid
`DataBotsPosIm`	[1:N] imaginary part of complex vector Two Indizes per Databot describing its positions in an two dimensional grid
`GridRadius`	[Columns, Lines] Radii Matrix of all possible Positions of DataBots in Grid, see also documentation of `setPolarGrid`
`GridAngle`	[Columns, Lines] Angle Matrix of all possible Positions of DataBots in Grid, see also documentation of `setPolarGrid`
`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] array of data: 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, defaut=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 *Lsun3D is part of the Fundamental Clustering Problem Suit (FCPS) [Thrun/Ultsch, 2020].*

Description

clearly defined clusters, different variances

Usage

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

Details

Size 404, Dimensions 3

Dataset defined discontinuites, 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 *Intern function for plotting during the Pswarm annealing process*

Description

Intern function, generates a scatter plot of the progess of the Pswarm algorithm after every nash equlibirum. 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 cartesianncoordinates of ProjectedPoints

Usage

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

Arguments

ProjectedPoints	[1:n,1:2] 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	Numeric vector containing the classification vector.

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, 2018.

See Also

GeneratePswarmVisualization

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

Description

This projetion method is a part of the databionic swarm which uses the nash equlibrium [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, Cls = NULL, QuadOrHexa = "Hexa", NumJumps = 4,
LC = NULL, Parallel = FALSE, NCores = "max", Verbose = 1,
PlotIt = FALSE, Debug = FALSE, DistanceMeasure = "euclidean",
Eps = 0.001)
```

Arguments

DataOrDistance	Numeric matrix nxd. Two cases here: d=n => assuming distance matrix d!=n => assuming data matrix with n cases and d features implying the need to compute the distance matrix internally.
Cls	Numeric vector [1:n] with class labels for each observation in DataOrDistance.
QuadOrHexa	Optional, Boolean indicating the geometry of tiles the 2D projection plane is built with.
NumJumps	Integer indicating the number of jumps to be considered for each single databot selected for jumping.
LC	Optional, grid size c(Columns, Lines), sometimes it is better to call setGridSize separately.
Parallel	Optional, Boolean: TRUE = parallel execution, FALSE = single thread execution.
NCores	Character or integer: choice of number of cores of CPU (in case). Can be 'max' or a number. The max will always be 'all available cores - 1', to avoid core overload.
PlotIt	Optional, bool, default=FALSE, If =TRUE, Plots the projection during the computation prozess after every nash equlibirum.
Debug	Optional, Debug, default=FALSE, =TRUE results in various console messages, depricated for CRAN, because cout is not allowed.
DistanceMeasure	Optional, one of 39 distance methods of parDist of package parallelDist, if Data matrix is chosen above
Verbose	optional, integer stating the degree of textual feedback. 0 = no output, 1 = basic notifications, 2 = progress bar, 3 = details.
Eps	optional, double: Stop criterion for convergence of each epoche.

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 DBSclustering. 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)
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, Quirin Stier

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.

[Stier/Thrun, 2024] Stier, Q. and Thrun, M. C.: An efficient multicore CPU implementation of the DatabionicSwarm, 18th conference of the International Federation of Classification Societies (IFCS), San José, Costa Rica, July 14-19, 2024.

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)
```

PswarmEpochsParallel *Intern function, do not use yourself*

Description

Finds the weak Nash equilibirium of the data bots for one epoch depending on a radius, which requires the setting of constants, grid, and so on in, see `Pswarm`.

Usage

```
PswarmEpochsParallel(AllDataBotsPosRe, AllDataBotsPosIm, MyDistanceMatrix,
AllFreePosR0, GridRadii, GridAngle, JumpsPerRadius, NumJumps, NumAllDB, Lines,
Columns, Origin, Happiness, QuadOrHexa, RadiusVector, Rmin, Rmax, Cls, Debug,
pp, PlotIt = FALSE, Verbose = 1, Eps = 0.0001)
```

Arguments

AllDataBotsPosRe	Numeric vector [1:n] of the current positions for the databots on first of two dimensions.
AllDataBotsPosIm	Numeric vector [1:n] of the current positions for the databots on second of two dimensions.
MyDistanceMatrix	Numeric vector with vectorized distance matrix of the datapoints in the original (high-dimensional) data space
AllFreePosR0	NumericMatrix, see AllallowedDBPosR0 in setPolarGrid
GridRadii	Numeric matrix with radius information of polar transformation for each grid position
GridAngle	Numeric matrix with angle information of polar transformation for each grid position
JumpsPerRadius	Numeric Vector of possible positions of the 1st coordinate.
NumJumps	Integer number of jumps.
NumAllDB	Integer total number of databots
Lines	Integer stating the number of Lines the polar grid consists of.
Columns	Integer stating the number of columns the polar grid consists of.
Origin	Numeric origin of the positions of grid in two dimensions
Happiness	Numeric value indicating the global happiness over all databots
QuadOrHexa	optional, bool: If TRUE prints status every 100 iterations
RadiusVector	Numeric vector stating all moving radius in a descending order (cooling down scheme).
Rmin	Integer stating minimum radius.
Rmax	Integer stating maximum radius.
Cls	Integer vector stating the classification vector for each datapoints/databots.
Debug	optional, bool: If TRUE prints information for debugging.
pp	Numeric vector stating ratio of number of jumping simultaneously DataBots of one eppoch (per nash-equilibirum), this vector is linearly monotonically decreasing.
PlotIt	optional, bool: If TRUE creates plot of projection after each epoch.
Verbose	optional, integer stating degree of textual feedback. 0 = no output, 1 = basic notifications, 2 = progress bar, 3 = details.
Eps	optional, double: Stop criterion for convergence of each epoche.

Details

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

Value

list of

`AllDataBotsPosRe`
: Numeric vector [1:n] of the current positions for the databots on first of two dimensions.

`AllDataBotsPosIm`
: Numeric vector [1:n] of the current positions for the databots on second of two dimensions.

`CourseOfHappiness`
: NumericVector, states the global happiness value per epoch.

`RadiusPerEpoch`
: NumericVector, stating the radius used per epoch in order of computation.

Author(s)

Quirin Stier

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.

[Stier/Thrun, 2024] Stier, Q. and Thrun, M. C.: An efficient multicore CPU implementation of the DatabionicSwarm, 18th conference of the International Federation of Classification Societies (IFCS), San José, Costa Rica, July 14-19, 2024.

PswarmEpochsSequential

Intern function, do not use yourself

Description

Finds the weak Nash equilibirium of the data bots for one epoch depending on a radius, which requires the setting of constants, grid, and so on in, see `Pswarm`.

Usage

```
PswarmEpochsSequential(AllDataBotsPos, MyDistanceMatrix, IndPossibleDBPosR,
AllFreePosR0, NumAllDB, Lines, Columns, Origin, Happiness, GridRadii, GridAngle,
QuadOrHexa, RadiusVector, Rmin, Rmax, Cls, Debug, pp, PlotIt = FALSE,
Verbose = 1)
```

Arguments

AllDataBotsPos	Complex vector [1:n] of the current positions for the databots on a 2d and real plane in complex numbers.
MyDistanceMatrix	Numeric vector with vectorized distance matrix of the datapoints in the original (high-dimensional) data space
IndPossibleDBPosR	Numeric vector containing the possible positions around a databot dependent on the radius.
AllFreePosR0	NumericMatrix, see AllallowedDBPosR0 in setPolarGrid.
NumAllDB	Integer total number of databots
Lines	Integer stating the number of Lines the polar grid consists of.
Columns	Integer stating the number of columns the polar grid consists of.
Origin	Numeric origin of the positions of grid in two dimensions
Happiness	Numeric value indicating the global happiness over all databots
GridRadii	Numeric matrix with radius information of polar transformation for each grid position
GridAngle	Numeric matrix with angle information of polar transformation for each grid position
QuadOrHexa	optional, bool: If TRUE prints status every 100 iterations
RadiusVector	Numeric vector stating all moving radius in a descending order (cooling down scheme).
Rmin	Integer stating minimum radius.
Rmax	Integer stating maximum radius.
Cls	Integer vector stating the classification vector for each datapoints/databots.
Debug	optional, bool: If TRUE prints information for debugging.
pp	Numeric vector stating ratio of number of jumping simultaneously DataBots of one eppoch (per nash-equilibirum), this vector is linearly monotonically decreasing.
PlotIt	optional, bool: If TRUE creates plot of projection after each epoch.
Verbose	optional, integer stating degree of textual feedback. 0 = no output, 1 = basic notifications, 2 = progress bar, 3 = details.

Details

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

Value

list of

AllDataBotsPosRe	Numeric vector [1:n] of the current positions for the databots on first of two dimensions.

`AllDataBotsPosIm`
: Numeric vector [1:n] of the current positions for the databots on second of two dimensions.

`CourseOfHappiness`
: NumericVector, states the global happiness value per epoch.

`RadiusPerEpoch`
: NumericVector, stating the radius used per epoch in order of computation.

Author(s)

Quirin Stier

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.

`PswarmRadiusParallel` *Intern function, do not use yourself*

Description

Finds the weak Nash equilibirium of the data bots for one epoch depending on a radius, which requires the setting of constants, grid, and so on in, see `Pswarm`.

Usage

```
PswarmRadiusParallel(DataBotsPos, DataDists, AllallowedDBPosR0, IndPossibleDBPosRe,
IndPossibleDBPosIm, Lines, Columns, Radius, NumAllDB, NumChoDB, NumFreeShape1,
NumJumps, Origin1, Origin2, Happiness, MinIterations, HappinessInclination, Eps, debug)
```

Arguments

`DataBotsPos`
: Numeric vector [1:NumJumps*n*2] containing the current positions and all positions for considered/possible jumps which can be computed (depending on number of jumps parameter NumJumps) for the databots on two dimensions.

`DataDists`
: Numeric vector with vectorized distance matrix of the datapoints in the original (high-dimensional) data space

`AllallowedDBPosR0`
: NumericMatrix, see `AllallowedDBPosR0` in `setPolarGrid`

`IndPossibleDBPosRe`
: Numeric Vector of possible positions of the 1st coordinate.

`IndPossibleDBPosIm`
: Numeric Vector of possible positions of the 2nd coordinate.

`Lines`
: Integer stating the number of Lines the polar grid consists of.

`Columns`
: Integer stating the number of columns the polar grid consists of.

`Radius`	Numeric (Integer) stating the moving radius of the databots
`NumAllDB`	Integer total number of databots
`NumChoDB`	Integer number of databots chosen for moving/jumps.
`NumFreeShape1`	Integer stating the first dimension of the numeric matrix book keeping the possible position grid
`NumJumps`	Integer number of jumps
`Origin1`	Numeric origin coordinate 1
`Origin2`	Numeric origin coordinate 2
`Happiness`	Numeric value indicating the global happiness over all databots
`MinIterations`	asdf
`HappinessInclination`	asdf
`Eps`	optional, double: Stop criterion for convergence of each epoche.
`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 equlibrium is found
`stressverlauf`	NumericVector, intern result, for debugging only
`fokussiertlaufind`	NumericVector, intern result, for debugging only

Author(s)

Quirin Stier

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.

[Stier/Thrun, 2024] Stier, Q. and Thrun, M. C.: An efficient multicore CPU implementation of the DatabionicSwarm, 18th conference of the International Federation of Classification Societies (IFCS), San José, Costa Rica, July 14-19, 2024.

PswarmRadiusSequential *intern function, do not use yourself*

Description

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

Usage

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

Arguments

AllDataBotsPosOld	ComplexVector [1:n,1], DataBots position in the last Nash-Equlibriuum
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 findPossiblePositionsCsingle
RadiusPositionsschablone	NumericMatrix, see AllallowedDBPosR0 in setPolarGrid
pp	NumericVector, number of jumping simultaneously DataBots of one eppoch (per nash-equilibirum), this vector is linearly monotonically decreasing
Nullpunkt	NumericVector, equals which(AllallowedDBPosR0==0,arr.ind=T), see see AllallowedDBPosR0 in setPolarGrid
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
Happiness	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 equlibrium 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, 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, 2018.

See Also

Pswarm

RelativeDifference *Relative Difference*

Description

Calculates the difference between positive x and y values

Usage

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

Details

Contrary to other approaches in this cases the range of values lies between [-2,2]. The approach is only valid for positive values ofX and Y. The realtive 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)
```

RobustNorm_BackTrafo *Transforms the Robust Normalization back*

Description

Transforms the Robust Normalization back if Capped=FALSE

Usage

```
RobustNorm_BackTrafo(TransformedData,

MinX,Denom,Center=0)
```

Arguments

TransformedData	[1:n,1:d] matrix
MinX	scalar
Denom	scalar
Center	scalar

Details

For details see RobustNormalization

Value

[1:n,1:d] Data matrix

Author(s)

Michael Thrun

See Also

RobustNormalization

Examples

```
data(Hepta)
Data = Hepta$Data
TransList = RobustNormalization(Data, Centered = TRUE, WithBackTransformation = TRUE)

HeptaData = RobustNorm_BackTrafo(TransList$TransformedData,
                                 TransList$MinX,
                                 TransList$Denom,
                                 TransList$Center)

sum(HeptaData - Data) #<e-15
```

RobustNormalization *RobustNormalization*

Description

RobustNormalization as described in [Milligan/Cooper, 1988].

Usage

```
RobustNormalization(Data,Centered=FALSE,Capped=FALSE,

na.rm=TRUE,WithBackTransformation=FALSE,

pmin=0.01,pmax=0.99)
```

Arguments

Data	[1:n,1:d] data matrix of n cases and d features
Centered	centered data around zero by median if TRUE
Capped	TRUE: outliers are capped above 1 or below -1 and set to 1 or -1.
na.rm	If TRUE, infinite vlaues are disregarded
WithBackTransformation	If in the case for forecasting with neural networks a backtransformation is required, this parameter can be set to 'TRUE'.
pmin	defines outliers on the lower end of scale
pmax	defines outliers on the higher end of scale

Details

Normalizes features either between -1 to 1 (Centered=TRUE) or 0-1 (Centered=TRUE) without changing the distribution of a feature itself. For a more precise description please read [Thrun, 2018, p.17].

"[The] scaling of the inputs determines the effective scaling of the weights in the last layer of a MLP with BP neural netowrk, it can have a large effect on the quality of the final solution. At the outset it is besto to standardize all inputs to have mean zero and standard deviation 1 [(or at least the range under 1)]. This ensures all inputs are treated equally in the regularization prozess, and allows to choose a meaningful range for the random starting weights."[Friedman et al., 2012]

Value

if `WithBackTransformation=FALSE`: TransformedData[1:n,1:d] i.e., normalized data matrix of n cases and d features

if `WithBackTransformation=TRUE`: List with

TransformedData	[1:n,1:d] normalized data matrix of n cases and d features
MinX	[1:d] numerical vector used for manual back-transformation of each feature
MaxX	[1:d] numerical vector used for manual back-transformation of each feature
Denom	[1:d] numerical vector used for manual back-transformation of each feature
Center	[1:d] numerical vector used for manual back-transformation of each feature

Author(s)

Michael Thrun

References

[Milligan/Cooper, 1988] Milligan, G. W., & Cooper, M. C.: A study of standardization of variables in cluster analysis, Journal of Classification, Vol. 5(2), pp. 181-204. 1988.

[Friedman et al., 2012] Friedman, J., Hastie, T., & Tibshirani, R.: The Elements of Statistical Learning, (Second ed. Vol. 1), Springer series in statistics New York, NY, USA:, ISBN, 2012.

[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

RobustNorm_BackTrafo

Examples

```
Scaled = RobustNormalization(rnorm(1000, 2, 100), Capped = TRUE)
hist(Scaled)

m = cbind(c(1, 2, 3), c(2, 6, 4))
List = RobustNormalization(m, FALSE, FALSE, FALSE, TRUE)
TransformedData = List$TransformedData

mback = RobustNorm_BackTrafo(TransformedData, List$MinX, List$Denom, List$Center)

sum(m - mback)
```

---

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], BestMAtchingUnits 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 betweeen 1 to x
ComputeInR	=T: Rcode, =F Cpp Code.
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, 2018.

See Also

`GeneratePswarmVisualization`

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

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 *Sets the grid size for the Pswarm algorithm*

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, Verbose = 0)
```

Arguments

InputDistances	[1:n,1:n] symmetric matrix of input distances
minp	default value: 0.01,see quantile, first value in the vector of probs estimates robust minimum of distances
maxp	default value: 0.99, see quantile, 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.
Verbose	optional, integer stating degree of textual feedback. 0 = no output, 1 = basic notifications, 2 = progress bar, 3 = details.

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, 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 *Intern function: Estimates the minimal radius for the Databot scent*

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, 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*E*Log(V)), see also [Jungnickel, 2013, p. 87]. Further details can be foubd 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 vertice itself from al 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, 2018.

See Also

DijkstraSSSP

`trainstepC` *internal function for s-esom*

Description

Does the training for fixed bestmatches in one epoch of the sESOM.

Usage

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

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?
`NoCases`	int, number of samples in the given non-sampled dataset

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.

`trainstepC2` *internal function for s-esom*

Description

Does the training for fixed bestmatches in one epoch of the sESOM.

Usage

```
trainstepC2(esomwts,aux, DataSampled,BMUsampled,Lines,Columns, Weights, Radius,
toroid, NoCases)
```

Arguments

`esomwts`	array [1:Lines,1:Columns,1:Weights], WeightVectors that will be trained, internally transformed von NumericVector to cube
`aux`	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
`Weights`	double, number of weights
`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?
`NoCases`	int, number of samples in the given non-sampled dataset

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, Eps=1e-10)
```

Arguments

`Datapoints`	[1:n,1:d] matrix of Datapoints points of dimension d, the points are in the rows
`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
`IsDuplicate`	[1:n,1:n] matrix,for i!=j IsDuplicate[i,j]== 1 if Datapoints[i,] == Datapoints[j,] IsDuplicate[i,i]==0
`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)

Author(s)

Michael Thrun

Examples

```
Datapoints2D=rbind(c(1,2),c(1,2),c(1,3),c(3,1))
V=UniquePoints(Datapoints2D)
```

Index

∗ **BackTransformation_RobustNormalization**
RobustNorm_BackTrafo, 32
∗ **Cluster Analysis**
Pswarm, 21
∗ **Clustering**
Pswarm, 21
∗ **DBS**
DatabionicSwarm-package, 3
DBSclustering, 7
∗ **DR**
DatabionicSwarm-package, 3
GeneratePswarmVisualization, 13
∗ **Data-driven**
Pswarm, 21
∗ **DataBionic**
DatabionicSwarm-package, 3
∗ **Databionic Swarm**
Pswarm, 21
∗ **Databionic swarm**
GeneratePswarmVisualization, 13
∗ **Databionic**
DatabionicSwarm-package, 3
Pswarm, 21
∗ **Databonic swarm**
DBSclustering, 7
∗ **Delaunay Graph**
Delaunay4Points, 10
∗ **Delaunay**
Delaunay4Points, 10
∗ **Dijkstra's SSSP**
DijkstraSSSP, 11
∗ **Dijkstra**
DijkstraSSSP, 11
∗ **Dimensionality Reduction**
DatabionicSwarm-package, 3
Pswarm, 21
∗ **ESOM**
GeneratePswarmVisualization, 13
sESOM4BMUs, 35
∗ **Equilibrium**
Pswarm, 21
∗ **FCPS**
Hepta, 18
Lsun3D, 19
∗ **Game Theory**
Pswarm, 21
∗ **GeneratePswarmVisualization**
ProjectedPoints2Grid, 20
sESOM4BMUs, 35
∗ **Hepta**
Hepta, 18
∗ **Human in the Loop**
Pswarm, 21
∗ **Lsun3D**
Lsun3D, 19
∗ **Machine Learning**
Pswarm, 21
∗ **PSwarm**
Pswarm, 21
∗ **Points**
Delaunay4Points, 10
∗ **Polar Swarm**
Pswarm, 21
∗ **Projection Method**
Pswarm, 21
∗ **Projection**
Pswarm, 21
∗ **Pswarm**
DatabionicSwarm-package, 3
plotSwarm, 20
∗ **RelativeDifference**
RelativeDifference, 31
∗ **RobustNorm_BackTrafo**
RobustNorm_BackTrafo, 32
∗ **RobustNormalization**
RobustNorm_BackTrafo, 32
RobustNormalization, 33

∗ **SOM**
GeneratePswarmVisualization, 13
∗ **SSSP**
DijkstraSSSP, 11
∗ **Segregation**
Pswarm, 21
∗ **ShortestGraphPaths**
ShortestGraphPathsC, 40
∗ **ShortestPaths**
ShortestGraphPathsC, 40
∗ **Single source shortest path**
DijkstraSSSP, 11
∗ **Swarm Intelligence**
Pswarm, 21
∗ **Swarms**
Pswarm, 21
∗ **Swarm**
Pswarm, 21
∗ **U-matrix**
GeneratePswarmVisualization, 13
∗ **Umatrix**
GeneratePswarmVisualization, 13
∗ **Visual Analytics**
Pswarm, 21
∗ **Visualization**
Pswarm, 21
∗ **Voronoi**
Delaunay4Points, 10
∗ **cluster analysis**
DatabionicSwarm-package, 3
DBSclustering, 7
GeneratePswarmVisualization, 13
∗ **clustering**
DatabionicSwarm-package, 3
DBSclustering, 7
∗ **cluster**
DBSclustering, 7
∗ **data points**
UniquePoints, 43
∗ **datasets**
Lsun3D, 19
∗ **dataset**
Hepta, 18
∗ **difference**
RelativeDifference, 31
∗ **distances**
rDistanceToroidCsingle, 30
∗ **emergence**
DatabionicSwarm-package, 3
∗ **equilibrium**
DatabionicSwarm-package, 3
∗ **game theory**
DatabionicSwarm-package, 3
∗ **generalized Umatrix**
GeneratePswarmVisualization, 13
∗ **generlized Umatrix**
ProjectedPoints2Grid, 20
∗ **graph**
Delaunay4Points, 10
∗ **grid**
setPolarGrid, 38
∗ **hexagonal**
setPolarGrid, 38
∗ **nash**
DatabionicSwarm-package, 3
Delta3DWeightsC, 11
PswarmEpochsParallel, 23
PswarmEpochsSequential, 25
PswarmRadiusParallel, 27
PswarmRadiusSequential, 29
∗ **package**
DatabionicSwarm-package, 3
∗ **points**
UniquePoints, 43
∗ **polar**
Delta3DWeightsC, 11
PswarmEpochsParallel, 23
PswarmEpochsSequential, 25
PswarmRadiusParallel, 27
PswarmRadiusSequential, 29
∗ **positions**
findPossiblePositionsCsingle, 12
∗ **projection method**
DatabionicSwarm-package, 3
∗ **projection**
DatabionicSwarm-package, 3
∗ **relative**
RelativeDifference, 31
∗ **sESOM**
GeneratePswarmVisualization, 13
∗ **self-organization**
DatabionicSwarm-package, 3
∗ **self-organizing-map**
GeneratePswarmVisualization, 13
∗ **swarm intelligence**
DatabionicSwarm-package, 3

∗ **swarms**
findPossiblePositionsCsingle, 12
∗ **swarm**
DatabionicSwarm-package, 3
DBSclustering, 7
Delta3DWeightsC, 11
GeneratePswarmVisualization, 13
plotSwarm, 20
PswarmEpochsParallel, 23
PswarmEpochsSequential, 25
PswarmRadiusParallel, 27
PswarmRadiusSequential, 29
setPolarGrid, 38
∗ **toroid**
rDistanceToroidCsingle, 30
∗ **unique**
UniquePoints, 43
∗ **visualization**
DatabionicSwarm-package, 3
GeneratePswarmVisualization, 13

DatabionicSwarm
*(*DatabionicSwarm-package*)*, 3
DatabionicSwarm-package, 3
DBSclustering, 7, *7, 8*, *13*, *22*
DefaultColorSequence, 9
Delaunay4Points, 10
Delta3DWeightsC, 11
DijkstraSSSP, 11, *41*

findPossiblePositionsCsingle, 12, *29*

GabrielClassificationError, *32*
GeneralizedUmatrix, *15*
GeneratePswarmVisualization, *7, 8*, 13, *13, 14*, *17*, *21, 22*, *36*
getCartesianCoordinates, 16
getUmatrix4Projection, 17

Hepta, 18

Lsun3D, 19

plotSwarm, 20
plotTopographicMap, *8*, *14, 15*
ProjectedPoints2Grid, 20
Pswarm, *7*, *13–15*, *20*, 21, *22, 23*, *25*, *27*, *29–31*, *37*, *39*
PswarmEpochsParallel, 23
PswarmEpochsSequential, 25
PswarmRadiusParallel, 27
PswarmRadiusSequential, 29

quantile, *37*

rDistanceToroidCsingle, 30
RelativeDifference, 31
RobustNorm_BackTrafo, 32, *35*
RobustNormalization, 33, *33*

sESOM4BMUs, 35
setdiffMatrix, 36
setGridSize, *14*, *22, 23*, 37
setPolarGrid, *13*, *16*, *24*, *26, 27*, *29*, 38
setRmin, 39
ShortestGraphPathsC, *12*, 40

trainstepC, 41
trainstepC2, 42

UniquePoints, 43