Package 'SimilarityMeasures'

February 19, 2015

Type Package

Title Trajectory Similarity Measures

Version 1.4

Date 2015-02-06

Author Kevin Toohey

Maintainer Kevin Toohey <kevintoohey@live.com>

Description Functions to run and assist four different similarity measures. The similarity measures included are: longest common subsequence (LCSS), Frechet distance, edit distance and dynamic time warping (DTW). Each of these similarity measures can be calculated from two n-dimensional trajectories, both in matrix form.

License GPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-06 06:19:21

R topics documented:

`SimilarityMeasures-package`

Implements Several Similarity Measures and Useful Functions

Description

This package contains functions to run and assist four different similarity measures. The similarity measures included are: longest common subsequence (LCSS), Frechet distance, edit distance and dynamic time warping (DTW). Each of these similarity measures can be calculated from two n-dimensional trajectories, both in matrix form.

Details

Package:	SimilarityMeasures
Type:	Package
Version:	1.4
Date:	2014-02-06
License:	GPL-3

Author(s)

Kevin Toohey

Maintainer: Kevin Toohey <kevintoohey@live.com>

References

Alt, H. and Godau, M. (1995) Computing the Frechet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, **5(01n02)**, 75–91.

Berndt, D. and Clifford, J. (1994) Using Dynamic Time Warping to Find Patterns in Time Series. Paper presented at the KDD workshop.

Chen, L., Ozsu, M. T. and Oria, V. (2005) Robust and fast similarity search for moving object trajectories. Paper presented at the Proceedings of the 2005 ACM SIGMOD international conference on Management of data, Baltimore, Maryland.

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

`AveTranslate` *Create an Average Point Translation Vector*

Description

A function to calculate a translation vector for trajectory2 using the average of the two trajectories. The vector created can be used to translate the average of trajectory2's points to that of trajectory1.

Usage

```
AveTranslate(traj1, traj2)
```

Arguments

`traj1`	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
`traj2`	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.

Details

The average of each dimension of both trajectories is calculated. The required translation is then calculated from the difference between them.

Value

A vector of length n is returned containing the translation in each dimension. If there is a problem this returns a string of information instead.

Author(s)

Kevin Toohey

See Also

`LCSSTranslation`, `StartEndTranslate`

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the translation.
AveTranslate(path1, path2)
```

`DistanceCheck` *Check if Two Points Lie Within some Distance in All Dimensions*

Description

A function to check whether two points lie within some distance in every dimension.

Usage

```
DistanceCheck(point1, point2, dist, dimensions=length(point1))
```

Arguments

`point1`	An n dimensional vector representing point1.
`point2`	An n dimensional vector representing point2.
`dist`	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
`dimensions`	An integer representing the number of dimensions being checked. This defaults to the length of the first vector.

Value

A boolean value is returned. The value is true if the points are within the distance in every dimension and false if not.

Author(s)

Kevin Toohey

Examples

```
# Creating two points.
point1 <- c(0, 2, 4, 6)
point2 <- c(0, 1, 2, 3)

# Running the check with distances 1 and 3 in 4 dimensions.
DistanceCheck(point1, point2, 1, 4)
DistanceCheck(point1, point2, 3, 4)
```

DistanceSq — *Calculate the Square Distance Between Two Points*

Description

A function to calculate the square of the distance between two points.

Usage

```
DistanceSq(point1, point2, dimensions=length(point1))
```

Arguments

point1	An n dimensional vector representing point1.
point2	An n dimensional vector representing point2.
dimensions	An integer representing the number of dimensions being used for the distance square calculation. This defaults to the length of the first vector.

Value

A floating point value is returned, representing the square of the distance between the two points.

Author(s)

Kevin Toohey

Examples

```
# Creating two points.
point1 <- c(0, 2, 4, 6)
point2 <- c(0, 1, 2, 3)

# Calculating the square distance between the two points
# in 4 dimensions.
DistanceSq(point1, point2, 4)
```

Dot — *Calculate the Dot Product Between Two Vectors*

Description

A function to calculate the dot product between two vectors. This is used in the Frechet algorithm and does not need to be called directly.

Usage

```
Dot(vect1, vect2, dimensions=length(vect1))
```

Arguments

vect1	An n dimensional vector representing the first vector.
vect2	An n dimensional vector representing the second vector.
dimensions	An integer representing the number of dimensions being used for the dot product calculation. This defaults to the length of the first vector.

Details

This function is called by the Frechet function.

Value

A floating point value is returned, representing the dot product between the two vectors.

Author(s)

Kevin Toohey

DTW *Run the Dynamic Time Warping Algorithm on Two Trajectories*

Description

A function to calculate the dynamic time warping value between two trajectories.

Usage

```
DTW(traj1, traj2, pointSpacing=-1)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.

Details

The dynamic time warping algorithm (DTW) calculates the smallest warp path for the two trajectories. This is the DTW version discussed by Berndt & Clifford (1994). Several variations of calculating the warping cost exist. In this version, the warping path is equal to the sum of the distances at each point along the path. Please see the references for more information.

Value

A floating point value representing the smallest warp path is returned. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Berndt, D. and Clifford, J. (1994) Using Dynamic Time Warping to Find Patterns in Time Series. Paper presented at the KDD workshop.

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the dynamic time warping algorithm with point spacing
# set to 4.
DTW(path1, path2, 4)
```

EditDist *Run the Edit Distance Algorithm on Two Trajectories*

Description

A function to calculate the edit distance between two trajectories.

Usage

```
EditDist(traj1, traj2, pointDistance=20)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.

Details

The edit distance algorithm calculates the minimum number of edits required to allow the two trajectories to be considered equivalent. This function uses Edit Distance on Real sequence (EDR) as described by Chen et al. (2005). Please see the references for more information.

Value

An integer representing the minimum number of edits required is returned. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Chen, L., Ozsu, M. T. and Oria, V. (2005) Robust and fast similarity search for moving object trajectories. Paper presented at the Proceedings of the 2005 ACM SIGMOD international conference on Management of data, Baltimore, Maryland.

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the edit distance algorithm with point distance
# set to 2.
EditDist(path1, path2, 2)
```

Frechet *Run the Frechet Calculation Algorithm on Two Trajectories*

Description

A function to calculate the Frechet distance between two trajectories. The function can also be used to test leash values.

Usage

```
Frechet(traj1, traj2, testLeash=-1)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
testLeash	A numeric leash value, which if positive, checks whether the leash can be used between the two trajectories. If this value is negative, then it is not used and the standard calculation is performed.

Details

This algorithm calculates the Frechet distance. The Frechet metric (or distance) is generally described in the following way: A man is walking a dog on a leash, the man walks on one curve while the dog walks on the other (Alt & Godau, 1995). The dog and the man are able to vary their speeds, or even stop, but not go backwards. The Frechet metric is the minimum leash length required to complete the traversal of both curves. Please see the references for more information.

Value

A floating point value representing the Frechet distance is returned. If a test leash is given, then a boolean value is returned as true if the leash was successful and false if not. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Alt, H. and Godau, M. (1995) Computing the Frechet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, **5(01n02)**, 75–91.

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the Frechet distance algorithm.
Frechet(path1, path2)
```

FrechetCheck *Checks a Frechet Leash Distance*

Description

A function to check whether a Frechet leash distance is successful or not. This is used by the Frechet calculation and does not need to be called directly.

Usage

```
FrechetCheck(traj1, traj2, leash, dist1, dist2, distSq12)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
leash	A numeric leash value to be checked by the function.
dist1	A vector containing the distance between each successive two points in trajectory1.
dist2	A vector containing the distance between each successive two points in trajectory2.
distSq12	A matrix containing the distance between each pair of two points where 1 point lies in trajectory1 and the other in trajectory2.

Details

This function is required by Frechet and in general does not need to be called directly.

Value

A boolean value is returned. A value of true is returned if the leash is successful and false if not.

Author(s)

Kevin Toohey

References

Alt, H. and Godau, M. (1995) Computing the Frechet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, **5(01n02)**, 75–91.

See Also

Frechet

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the Frechet distance algorithm with a test leash of 2.
Frechet(path1, path2, 2)
```

LCSS *Run the LCSS Algorithm on Two Trajectories Allowing Translations*

Description

A function to calculate the longest common subsequence between two trajectories. This calculation automatically uses translations to find the largest value.

Usage

```
LCSS(traj1, traj2, pointSpacing=-1, pointDistance=20,
     errorMarg=2, returnTrans=FALSE)
```

Arguments

`traj1`	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
`traj2`	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
`pointSpacing`	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.
`pointDistance`	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
`errorMarg`	A floating point error margin used to scale the accuracy and speed of the calculation.
`returnTrans`	A boolean value to allow the best translation found to be returned as well as the LCSS value if set to true.

Details

The LCSS algorithm calculates the largest number of equivalent points between the two trajectories when traversed in a monotone way. Possible translations are calculated in each dimension and used to provide the maximum LCSS. The accuracy of the algorithm can be varied to provide faster or slower calculations. Please see the references for more information.

Value

An integer value is returned. This represents the maximum LCSS value obtained using the variables provided. If returnTrans is set to TRUE, then the LCSS value and the translations are returned as a vector. The first value of this vector is the LCSS value and the translations follow directly afterwards. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

See Also

LCSSCalc, LCSSRatio, LCSSRatioCalc, LCSSTranslation

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the LCSS algorithm on the trajectories.
LCSS(path1, path2, 2, 2, 0.5)

# Running the LCSS algorithm on the trajectories
# and returning the translation as well.
LCSS(path1, path2, 2, 2, 0.5, TRUE)
```

LCSSCalc *Run the LCSS Algorithm on Two Trajectories Without Translations*

Description

A function to calculate the longest common subsequence between two trajectories. This function does not calculate translations and only uses the given trajectories and optional translation.

Usage

```
LCSSCalc(traj1, traj2, pointSpacing=-1, pointDistance=20,
         trans=rep(0, (dim(traj1)[2])))
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
trans	A vector containing translations in each dimension to be applied to trajectory2 in this calculation.

Details

The LCSS algorithm calculates the largest number of equivalent points between the two trajectories when traversed in a monotone way. If a translation is given then this is applied before the calculation is done. This function is used by all of the the LCSS functions. Please see the references for more information.

Value

An integer value is returned. This represents the maximum LCSS value obtained using the variables provided. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

See Also

LCSS, LCSSRatio, LCSSRatioCalc, LCSSTranslation

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the LCSS algorithm on the trajectories.
LCSSCalc(path1, path2, 2, 2, c(0, 3))
```

LCSSRatio *Find the LCSS Ratio using Two Trajectories Allowing Translations*

Description

A function to calculate the longest common subsequence ratio using two trajectories. This calculation automatically uses translations to find the largest value.

Usage

```
LCSSRatio(traj1, traj2, pointSpacing=-1, pointDistance=20,
          errorMarg=2, returnTrans=FALSE)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
errorMarg	A floating point error margin used to scale the accuracy and speed of the calculation.
returnTrans	A boolean value to allow the best translation found to be returned as well as the LCSS value if set to true.

Details

The LCSS algorithm calculates the largest number of equivalent points between the two trajectories when traversed in a monotone way. The ratio of this value to the smallest number of points out of the two trajectories is then calculated. Possible translations are calculated in each dimension and used to provide the maximum LCSS. The accuracy of the algorithm can be varied to provide faster or slower calculations. Please see the references for more information.

Value

A floating point value is returned. This represents the maximum LCSS ratio obtained using the variables provided. If returnTrans is set to TRUE, then the LCSS ratio and the translations are returned as a vector. The first value of this vector is the LCSS ratio and the translations follow directly afterwards. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

See Also

LCSSCalc, LCSS, LCSSRatioCalc, LCSSTranslation

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the LCSS ratio algorithm on the trajectories.
LCSSRatio(path1, path2, 2, 2, 0.5)

# Running the LCSS ratio algorithm on the trajectories
# and returning the translation as well.
LCSSRatio(path1, path2, 2, 2, 0.5, TRUE)
```

LCSSRatioCalc *Find the LCSS Ratio using Two Trajectories Without Translations*

Description

A function to calculate the longest common subsequence ratio using two trajectories. This function does not calculate translations and only uses the given trajectories and optional translation.

Usage

```
LCSSRatioCalc(traj1, traj2, pointSpacing=-1, pointDistance=20,
              trans=rep(0.0, (dim(traj1)[2])))
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.

pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
trans	A vector containing translations in each dimension to be applied to trajectory2 in this calculation.

Details

The LCSS algorithm calculates the largest number of equivalent points between the two trajectories when traversed in a monotone way. The ratio of this value to the smallest number of points out of the two trajectories is then calculated. If a translation is given then this is applied before the calculation is done. Please see the references for more information.

Value

A floating point value is returned. This represents the maximum LCSS ratio obtained using the variables provided. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

See Also

LCSS, LCSSRatio, LCSSCalc, LCSSTranslation

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the LCSS ratio algorithm on the trajectories.
LCSSRatioCalc(path1, path2, 2, 2, c(0, 3))
```

LCSSTranslation *Create a Translation Vector Using LCSS*

Description

A function to return the best translation calculated using the LCSS method. The vector created can be used to translate trajectory2's points to the position of the maximum LCSS found with trajectory1.

Usage

```
LCSSTranslation(traj1, traj2, pointSpacing=-1, pointDistance=20,
                errorMarg=2)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation. A negative value sets the point spacing to unlimited.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
errorMarg	A floating point error margin used to scale the accuracy and speed of the calculation.

Details

The LCSS function is called using the two trajectories along with the given variables. The optimal translation vector is then returned from this result. Please see the references for more information.

Value

A vector of length n is returned containing the translation in each dimension. If a problem occurs, then a string containing information about the problem is returned.

Author(s)

Kevin Toohey

References

Vlachos, M., Kollios, G. and Gunopulos, D. (2002) Discovering similar multidimensional trajectories. Paper presented at the Data Engineering, 2002. Proceedings. 18th International Conference on.

See Also

AveTranslate, StartEndTranslate

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the translation.
LCSSTranslation(path1, path2, 1, 1, 0.5)
```

SimLoop *Loop Over and Test Trajectories With Different Translations*

Description

Function to loop over and test the trajectories using the different translations in each dimension. This is used by the LCSS function to test all of the n dimensional translations. Do not call this function directly.

Usage

```
SimLoop(traj1, traj2, pointSpacing, pointDistance, spacing,
        similarity, translations, dimensions,
        dimLeft=dimensions, currentTrans=rep(0, dimensions))
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.
spacing	The integer spacing between each translation that will be tested.
similarity	A vector containing the current best similarity and translations calculated.
translations	A list of vectors containing the translations in each dimension.
dimensions	An integer representing the number of dimensions being used for the calculation.
dimLeft	An integer number of dimensions which have not been looped over yet.
currentTrans	A vector containing the current translation being tested.

Details

This function is used to loop over the n dimensions for the LCSS function. This function should not be called directly.

Value

Returns the current best LCSS value and the translations that created this as a vector.

Author(s)

Kevin Toohey

See Also

LCSS, LCSSRatio, LCSSRatioCalc, LCSSTranslation, LCSSCalc

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the LCSS algorithm on the trajectories.
LCSS(path1, path2, 2, 2, 0.5)
```

SinglePointCalc *Calculate Frechet Distance With a Single Point Trajectory*

Description

A function to calculate the Frechet distance between two trajectories. This function is called by the frechet function when one trajectory consists of a single point. This function does not need to be directly called.

Usage

```
SinglePointCalc(traj1, traj2)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.

Details

This calculates the longest distance to the single point trajectory. This is the Frechet distance for such a pair of trajectories. This function is automatically called if required by the Frechet function, therefore it does not need to be called directly.

Value

A floating point value representing the Frechet distance is returned.

Author(s)

Kevin Toohey

See Also

Frechet

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1), 1)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the Frechet distance algorithm which will
# automatically call this function.
Frechet(path1, path2)
```

StartEndTranslate *Translate a Trajectory Based on Start and End Points*

Description

A function to translate, rotate and scale the points of trajectory2 using trajectory1. The new trajectory will have the same start and end points as trajectory1.

Usage

```
StartEndTranslate(traj1, traj2)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.

Details

Every point of trajectory2 is rotated, scaled and translated so that the start and end points of the two trajectories match. The new variation of trajectory2 is returned as a matrix.

Value

An m x n matrix containing the new variation of trajectory2 is returned. Here m is the number of points and n is the dimension of the points.

Author(s)

Kevin Toohey

See Also

LCSSTranslation, AveTranslate

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)

# Running the translation.
StartEndTranslate(path1, path2)
```

TrajCheck *Checking Two Trajectories are Matrices of N Dimensional Points*

Description

This function checks whether two trajectories are compatible with the functions provided. They are first checked if they are represented as matrices. They are then checked to ensure that points in both trajectories have the same dimensions.

Usage

```
TrajCheck(traj1, traj2)
```

Arguments

traj1	An m x n matrix containing trajectory1. Here m is the number of points and n is the dimension of the points.
traj2	A k x n matrix containing trajectory2. Here k is the number of points and n is the dimension of the points. The two trajectories are not required to have the same number of points.

Details

This function is useful for determining if the trajectories are compatible with the many functions provided. This function is performed at the start of most other functions to ensure that no major errors can occur and that the results make sense.

Value

If there is a problem with one of the checks then a string containing information is returned. If all of the checks pass then -1 is returned.

Author(s)

Kevin Toohey

Examples

```
# Creating two trajectories.
path1 <- matrix(c(0, 1, 2, 3, 0, 1, 2, 3), 4)
path2 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 4)
path3 <- matrix(c(0, 1, 2, 3, 4, 5, 6, 7), 2)

# Running the trajectory check on the trajectories.
TrajCheck(path1, path2)
TrajCheck(path1, path3)
```

TranslationSubset *Calculate the Subset of Translations for LCSS*

Description

A function for calculating the subsets of translations to be tested using the LCSS methods.

Usage

```
TranslationSubset(traj1, traj2, pointSpacing, pointDistance)
```

Arguments

traj1	A vector containing one dimension of trajectory1.
traj2	A vector containing one dimension of trajectory2.
pointSpacing	An integer value of the maximum index difference between trajectory1 and trajectory2 allowed in the calculation.
pointDistance	A floating point number representing the maximum distance in each dimension allowed for points to be considered equivalent.

Details

This function is called by the LCSS method to calculate the translations required.

Value

A vector of floating point numbers is returned containing the translations calculated. This vector is sorted in ascending order.

Author(s)

Kevin Toohey

See Also

LCSS

Index