Package ‘pathmapping’

March 22, 2017

Type Package

Title Compute Deviation and Correspondence Between Spatial Paths

Version 1.0.2

Date 2017-03-22

Author Shane T. Mueller & Brandon S. Perelman

Maintainer Shane T. Mueller <shanem@mtu.edu>

Description Functions to compute and display the area-based deviation between spatial paths and to compute a mapping based on minimizing area and distance-based cost. For details, see: Mueller, S. T., Perelman, B. S., & Veinott, E. S. (2016) <DOI:10.3758/s13428-015-0562-7>.

License GPL-2

URL https://sites.google.com/a/mtu.edu/mapping/,
https://github.com/stmueller/pathmapping/

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-03-22 16:41:16 UTC

R topics documented:

`pathmapping-package` *Compute Deviation and Correspondence Between Spatial Paths*

Description

Functions to compute and display the area-based deviation between spatial paths and to compute a mapping based on minimizing area and distance-based cost. For details, see: "Mueller, S. T., Perelman, B. S., & Veinott, E. S. (2016). An optimization approach for mapping and measuring the divergence and correspondence between paths. Behavior research methods, 48(1), 53-71."

Details

Package:	pathmapping
Type:	Package
Version:	1.0.1
Date:	2016-03-15
License:	GPL 2.0

Computing the least-area mapping between paths.

Author(s)

Shane T. Mueller and Brandon Perelman

Maintainer: Shane T. Mueller <shanem@mtu.edu>

References

See Mueller et al., (2016).
`https://sites.google.com/a/mtu.edu/mapping/`

`ClosestPoint` *Find Closest Point*

Description

Find the the point on a line segment (x1,y1) (x2,y2) that is closest to point (px,py).

Usage

```
ClosestPoint(px, py, x1, y1, x2, y2)
```

Arguments

`px`	x coordinate of point
`py`	y coordinate of point
`x1`	x coordinate of one end of a segment
`y1`	y coordinate of one end of a segment
`x2`	x coordenite of other end of a segment
`y2`	y coordinate of other end of a segment

Details

This function finds the the point on a line segment (x1,y1) (x2,y2) that is closest to point (px,py). If the line perpendicular to the line segment does not intersect the segment, the function will return an end point of the segment, otherwise, it will return a point on the line segment where the perpendicular line intersects the segment.

Value

a point-pair (x,y)

Note

Uses LineMagnitude, also supplied by the pathmapping package.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/`

See Also

`LineMagnitude`, `DistancePointSegment`

Examples

```
ClosestPoint(1,10,2,0,5,0)    #returns end point 2 0
ClosestPoint(20,10,2,0,5,0)   #returns other end point 5 0
ClosestPoint(4.5,10,2,0,5,0)  #returns closest point 4.5 0
```

`connected` *Determine whether two nodes are connected.*

Description

Determines whether two nodes in the planar graph describing the mapping between two adjacent node-node mappings. That is, given a two correspondences between a nodes on two paths, it determines whether there is a legal transition between them. This can be seen as two elements of the mapping matrix, each specified by a row and column.

Usage

```
connected(r1, c1, r2, c2)
```

Arguments

`r1`	row of first node
`c1`	column of first node
`r2`	row of second node
`c2`	column of second node.

Details

r1,c1 should specify a node to the left/above r2,c2. The outcome depends on whether r,c is a node or segment on the path. Point-point mappings can transition to the next point-segment mappings, or the next point-point segment.

Value

returns `T` or `F`

Note

The outcome of this does not depend on the actual paths–it is simply a logical computation based on transitions between points and segments.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., 2016 https://sites.google.com/a/mtu.edu/mapping/

Examples

```
connected(3,5,2,4)
connected(3,3,1,1)
```

`Cost` *Compute area-based cost*

Description

This function computes an area associated with a transition between two correspondences on two paths. A number of cost functions can be specified, but the most reasonable are Cost.quadratic (the default) and Cost.area. other cost functions are not robust to choices of segmentation.

Usage

```
Cost(xy1, xy2, i, j, pi, pj, opposite, costfn)
```

Arguments

`xy1`	A path of x,y coordinates (a matrix in two columns)
`xy2`	A second path of x,y coordinates (a matrix in two columns)
`i`	Index of Node i of path xy1
`j`	Index of Node j of path xy2
`pi`	Index of node previous to node i on path xy1
`pj`	Index of node previous to node j on path xy2
`opposite`	matrix specifying whether points on one path are 'opposite' points on another path, and if so the proportion between the two points where the orthogonal line falls.
`costfn`	Specify a cost function to use. By default, Cost.quadratic is used, although Cost.area is also reasonable. Other cost functions are provided but may be sensitive to segmentation.

Details

This is the basic cost function for a transition between nodes. This function does not need to be used directly, but is called repeatedly by `CreateMap`

Value

returns a floating-point value descibing the area defined by the the two path transitions provided the transitions are legal (according to `connected`. Otherwise, it will return `Inf`.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

`CreateMap` *Create a mapping between paths*

Description

Given two paths, this creates a mapping that minimizes the

Usage

```
CreateMap(xy1.1, xy2.1, plotgrid = F,costfn=Cost.area,
         nondecreasingos = F, verbose = F, insertopposites = T)
```

Arguments

`xy1.1`	The first path (a matrix of x,y points).
`xy2.1`	The second path (a matrix of x,y points).
`plotgrid`	T/F variable; should the lattice grid be plotted? Defaults to F
`costfn`	Cost function to use to measure deviation between path segments.
`nondecreasingos`	T/F variable; defaults to F. If T, forces multiple consecutive mappings of points on one map to a single segment on the second map be monotonic. This will not necessarily find the optimal mapping; to do this, you must set insertopposites=T.
`verbose`	T/F; Whether to print out intermediate status information.
`insertopposites`	T/F; defaults to T. If T, it will insert points on each path when they are opposite a point on the other graph. This allows for an optimal monotonic mapping between paths, at the cost of (possibly substantial) efficiency cost.

Details

This finds the minimum-area mapping between two paths. It also produces a candidate minimum-area mapping.

Value

returns a mapping object

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

See Also

GetMinMap

Examples

```
##################################
## Example from appendix of Mueller, Perelman, & Veinott:

pathA <- rbind(c(0,0),c(5,0),c(10,0))
pathB <- rbind(c(1,1),c(2,-1),c(3,4),c(5,1),c(10,-3))
answer<- CreateMap(pathA,pathB,FALSE)
PlotMap(answer)
answer2 <- GetMinMap(answer)
PlotMap(answer2)





## Not run:
##################################
##Here is an example of two diagonal paths, a fixed number
##of units apart look at how different equivalent paths produce
## different mappings, but the same area

test2.a <- cbind(1:10*10,1:10*10)
test2.b <- cbind(1:10*10+10,1:10*10)

test2.outa <- CreateMap(test2.a,test2.b,FALSE)
test2.outb <- CreateMap(test2.b,test2.a,FALSE)
test2.outc <- CreateMap((test2.a[10:1,]),(test2.b[10:1,]),FALSE)
test2.outd <- CreateMap((test2.b[10:1,]),(test2.a[10:1,]),FALSE)

par(mfrow=c(2,2))
PlotMap(test2.outa)
PlotMap(test2.outb)
PlotMap(test2.outc)
PlotMap(test2.outd)

##################################
##Now, get the 'minimum-distance' mapping among these:

test2.mapa <- GetMinMap(test2.outa)
test2.mapb <- GetMinMap(test2.outb)
```

```
test2.mapc <- GetMinMap(test2.outc)
test2.mapd <- GetMinMap(test2.outd)

par(mfrow=c(2,2))
PlotMap(test2.mapa)
PlotMap(test2.mapb)
PlotMap(test2.mapc)
PlotMap(test2.mapd)

##################################
## Example: a loop and itself

test3.a <- rbind(c(102, 100),
      c(  120, 109),      c(  133, 124),
      c(  146, 138),      c(  158, 155),
      c(  174, 166),      c(  194, 170),
      c(  213, 173),      c(  233, 176),
      c(  251, 169),      c( 260, 151),
      c( 255, 132),      c( 245, 115),
      c( 235,  98),      c( 223,  82),
      c( 212,  65),      c( 194,  58),
      c( 175,  65),      c( 166,  82),
      c( 169, 101),      c(300,101))

test3.b <- test3.a

test3.out <- CreateMap(test3.a,test3.b)
PlotMap(test3.out)


########################################
##Example: A loop with an offset version of itself

test4.a <- test3.a
test4.b <- test3.a + 20
test4.out <- CreateMap(test4.a,test4.b,plotgrid=FALSE)
par(mfrow=c(1,2))
PlotMap(test4.out)
PlotMap(GetMinMap(test4.out))

#######################################
## Example:  a gentle curve, and a line.
test5.a <- cbind((-10):10*10,exp(-(-10:10*10)^2/500))
test5.b <- cbind(-10:10*10,-.5)
test5.a2 <- test5.a[21:1,]
test5.b2 <- test5.b[21:1,]


test5.out <- CreateMap(test5.b,test5.a,FALSE)
test5.outb <-CreateMap(test5.b2,test5.a2,FALSE)
par(mfrow=c(2,2))
PlotMap(test5.out)
PlotMap(test5.outb)
PlotMap(GetMinMap(test5.out))
PlotMap(GetMinMap(test5.outb))

## Note: the curved path gets 'shadow' opposite points inserted, and so
##the MinMap is a bit off.  In this case, we shouldn't need to insert
##opposites, so we can turn it off:

test5.out <- CreateMap(test5.b,test5.a,plotgrid=FALSE,insertopposites=FALSE)
test5.outb <-CreateMap(test5.b2,test5.a2,plotgrid=FALSE,insertopposites=FALSE)
par(mfrow=c(2,2))
PlotMap(test5.out)
PlotMap(test5.outb)

PlotMap(GetMinMap(test5.out))
PlotMap(GetMinMap(test5.outb))


########################################
##Cut off one part:
test5.b2<- test5.b[c(1,5,21),]
test5.out2 <- CreateMap(test5.a,test5.b2,FALSE)

PlotMap(test5.out2)
PlotMap(GetMinMap(test5.out2))


#######################################
## Example: a path with a bump.  Note that
## if we don't allow mapping points onto segments
## the area goes outside the polygon.

test6.a <- rbind(c(0,0),c(1,0),c(10,0))
test6.b <- rbind(c(0,1),c(4,1),c(5,9),c(6,1),c(10,1))

##true area should be 1x10 + 2*8/2 = 18.
test6.out <- CreateMap(test6.a,test6.b,FALSE)
PlotMap(test6.out)
PlotMap(GetMinMap(test6.out))


########################################
## Example: to lines, one with a bump
test7.a  <-  rbind(c(1,0),c(2,-1),c(3,0),c(4,0),c(5,0),c(6,0))
test7.b <- rbind(c(1,1),c(2,1),c(3,1),c(4,1),c(5,1),c(6,1))

test7.out <- CreateMap(test7.a,test7.b,FALSE)
test7.outr <- CreateMap(test7.b,test7.a,FALSE)

test7.outmin <- GetMinMap(test7.out)

par(mfrow=c(3,1),mar=c(3,2,2,0))
PlotMap(test7.out)
PlotMap(test7.outr)
PlotMap(GetMinMap(test7.out))

########################################
## Example: simplified case with a lot of 'opposites'
test8.a <- cbind(0:4+.5,0)
test8.b <- cbind(0:4,1)
test8.out <- CreateMap(test8.a,test8.b,FALSE)
par(mfrow=c(1,2))
PlotMap(test8.out)
PlotMap(GetMinMap(test8.out))

########################################
## Example: a crossover

test9.a <- rbind(c(0,0),c(1,0),c(10,0))
test9.b <- rbind(c(0,-1),c(4,-1),c(5,9),c(6,-1),c(10,-1))
test9.out <- CreateMap(test9.a,test9.b,FALSE)
PlotMap(test9.out)
PlotMap(GetMinMap(test9.out))

########################################
## Example: a variation on previous
test10.a <- test9.b
test10.b <- rbind(c(0,10),c(20,10))
test10.out <- CreateMap(test10.a,test10.b,FALSE)
test10.out2 <- CreateMap(test10.b,test10.a,FALSE)

PlotMap(test10.out)
PlotMap(test10.out2)
PlotMap(GetMinMap(test10.out))
PlotMap(GetMinMap(test10.out2) )


#######################################
##  Example: Appendix figures
pathA <- rbind(c(0,0),c(5,0),c(10,0))
pathB <- rbind(c(1,1),c(2,-1),c(3,4),c(5,1),c(10,-3))
map1 <- CreateMap(pathA,pathB,FALSE,insertopposites=FALSE)


##map2 is broken, or at least the display of map2:
map2 <- GetMinMap(map1)

par(mfrow=c(2,1))
PlotMap(map1)
PlotMap(map2)



############################################
## Example: another crossover
```

```
real.sub <- rbind(c(50,25),c(100,150),c(275,275))
mem.sub <- rbind(c(100,30),c(150,250), c(250,200))

xy1 <- real.sub
xy2 <- mem.sub

test10.out <- CreateMap(xy1,xy2,FALSE)
PlotMap(test10.out)
PlotMap(GetMinMap(test10.out))

## End(Not run)
```

`DistancePointSegment` *Compute distance between a point and a segment*

Description

Compute distance between a point and a segment

Usage

```
DistancePointSegment(px, py, x1, y1, x2, y2)
```

Arguments

`px`	x coordinate of point
`py`	y coordinate of point
`x1`	x coordinate of one end of segment
`y1`	y coordinate of one end of segment
`x2`	x coordinate of other end of segment
`y2`	y coordinate of other end of segment

Details

Computes the distance between a point and a segment via the shortest line. This line will be perpendicular to the segment if the point opposes the line, or it will be attached directly to an endpoint.

Value

returns a scalar value measuring the distance

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

See Also

LineMagnitude,ClosestPoint

Examples

```
##select a random point and find its closest point
##on the segment.
x1 <- runif(1)*20
y1 <- runif(1)*20

s1x <- 5;s1y <- 5
s2x <- 0;s2y <- 15

d <- DistancePointSegment(x1,y1,s1x,s1y,s2x,s2y)
plot(c(s1x,s2x),c(s1y,s2y),pch=16,xlab="x",ylab="y",
      ylim=c(-1,20),xlim=c(-1,20),type="o")
points(x1,y1,col="red",pch=16,cex=2)

p2 <-  ClosestPoint(x1,y1,s1x,s1y,s2x,s2y)
segments(p2[1],p2[2],x1,y1,lty=3)
text(10,2,paste("Distance from line to point:",
       round(d,3)))
```

even *Is a number even?*

Description

Is a number even?

Usage

```
even(x)
```

Arguments

x	a number

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

GetMinMap *Gets least-distance mapping among the minimum-area mappings.*

Description

Finds minimum linear-distance mapping among the least-cost area-based mappings between paths.

Usage

```
GetMinMap(mapping, leftbias=T, verbose = F )
```

Arguments

`mapping`	An object computed via CreateMap()
`leftbias`	Boolean value determining whether to prefer left or upper node connections first.
`verbose`	Whether intermediate output should be printed.

Details

GetMinMap() finds the best mapping between two paths amongst those that have the smallest area-based dissimilarity. It adds several data structures to a mapping produced by CreateMap: $leastcostchain and $chainpath, and it sets the boolian $minmap from FALSE to TRUE.

Value

`$leastcostchain`	The complete matrix used to solve the minimization.
`$chainpath`	The sequence of consecutive nodes representing the path.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

See Also

See Also `CreateMap`,

`InsertIntersections` *Inserts points on paths where two paths intersect*

Description

This function does two rounds of insertion. First, it inserts a point on each path whenever path1 intersects path2. Next, it optionally inserts points on segments of one path that are 'opposite' points on the other path, to allow a monotonic mapping between the two paths.

Usage

```
InsertIntersections(path1, path2, insertopposites = T, verbose = F)
```

Arguments

`path1`	path1
`path2`	path2
`insertopposites`	T/F, whether points opposite points on the other path should be inserted
`verbose`	T/F, whether to print interim progress information

Value

A list of four data sequences are returned:

`newpath1`	New list 1 with new points inserted
`newpath2`	New list2 with new points inserted
`key1`	Set of indices mapping the points back to the original path 1. Inserted points are labeled -1
`key2`	Set of indices mapping the points back to the original path 2. Inserted points are labeled -1

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

`IntersectPoint` *Find where point opposite segment intersects segment.*

Description

Find where point opposite segment intersects segment. It gives a proportion of AB that the orthogonal line passing through C meets. If outside (0,1) it does not pass through AB

Usage

```
IntersectPoint(A, B, C)
```

Arguments

`A`	`(x,y)` point on one end of line segment
`B`	`(x,y)` point on other end of line segment
`C`	`(x,y)` point to compare to line segment

Value

returns a value which is the proportion of the length of `AB` where `AC` proects onto the line defined by AB. If the return value is between 0 and 1, the point is opposite the line segment. If negative, it falls on the `A` side of `AB`; if greater than 1, it falls to the `B` side of `AB`.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/`

See Also

`LineMagnitude`,`ClosestPoint`,`DistancePointSegment`

`ldist` *Compute line length*

Description

Computes line length

Usage

```
ldist(p1,p2)
```

Arguments

`p1`	(x,y) point on one end of line segment
`p2`	(x,y) point on other end of line segment

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/`

`LineMagnitude` *Compute line length*

Description

Computes line length

Usage

```
LineMagnitude(x1, y1, x2, y2)
```

Arguments

`x1`	x coordinate of one end of line
`y1`	y coordinate of one end of line
`x2`	x coordinate of other end of line
`y2`	y coordinate of other end of line

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

`linesIntersect` *Checks whether two line segments intersect.*

Description

Checks whether two line segments intersect.

Usage

```
linesIntersect(A1, A2, B1, B2)
```

Arguments

`A1`	one end of line A (x,y) pair
`A2`	other end of line A (x,y) pair
`B1`	one end of line B (x,y) pair
`B2`	other end of line B (x,y) pair

Value

returns a boolean value indicating whether there is an intersection.

Note

Results may not be consistent if intersection happens exactly at one end of a segment, due to rounding error.

`LinkCost` *Computes the distance of a particular link between paths*

Description

This computes the distance between two paths for a particular pair of nodes (points or segments). This is primarily a helper function for the pathmapping library, and typically does not need to be used by end users.

Usage

```
LinkCost(xy1, xy2, i, j)
```

Arguments

`xy1`	Path1
`xy2`	Path2
`i`	Index of path1. This is not the row of path1, but the implied node, where the first node is the first point, the second node is the first segment, the third node is the second point, and so on.
`j`	Index of path2. This is not the row of path1, but the implied node, where the first node is the first point, the second node is the first segment, the third node is the second point, and so on.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/`

`livenodes` *Which nodes are legal mappings?*

Description

Which nodes are legal mappings?

`LLbeta` *Compute Latecki/Lakaemper beta*

Description

Compute Latecki/Lakaemper beta

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016).
L. J. Latecki and R. Lakaemper. Convexity Rule for Shape Decomposition Based on Discrete Contour Evolution. Computer Vision and Image Understanding, vol. 73, pp. 441-454, 1999.
`https://sites.google.com/a/mtu.edu/mapping/`

`LLKscore` *Compute Latecki/Lakaemper K score*

Description

Compute Latecki/Lakaemper K score

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016).
L. J. Latecki and R. Lakaemper. Convexity Rule for Shape Decom- a position Based on Discrete Contour Evolution. Computer Vision and Image Understanding, vol. 73, pp. 441-454, 1999.
`https://sites.google.com/a/mtu.edu/mapping/`

`odd` *Is a number odd?*

Description

Is a number odd?

Usage

```
odd(x)
```

Arguments

`x`	a number

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/`

PathDist *Compute Length of a path*

Description

This function computes the length of a path. It does so by summing each consecutive line segment.

Usage

```
PathDist(path)
```

Arguments

`path`	A two-column matrix describing the x,y coordinates of a path.

Value

returns a scalar value indicating path length.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

PathOverlap *Compute the proportion overlap of two paths.*

Description

This computes the proportion of the two paths that are mapped onto one another. The outcome is a number between 0 and 1.0, measuring how much of each path corresponds to a non-endpoint of the other map. This does not by itself measure path similarity, because two paths that are highly dissimilar that happen to start and end at similar spots would have a value close to 1.0. This is sensitive to 'partial' paths. If on path is a sub-path of the other, then it should measure the average of the proportion of path1 that is in path2 and the proportion of path2 that is in path1.

Usage

```
PathOverlap(mapping)
```

Arguments

`mapping`	`mapping` is the output of `CreateMap()`.

Details

`PathOverlap` works with a basic mapping and computes 'minimal' mappings, both left-biased and right-biased, to arrive at two best mappings (those involving the least distance between corresponding points). These will typically be identical, but there are non-degenerate cases where they can differ.

Once this mapping is arrived at, the algorithm identifies the core of the mapping–the central segment of both paths that are mapped onto the core of the other path, by identifying the segments of each path (on both ends) that are mapped onto the endpoint of the other path. For each path, the length of its core is subtracted from the total length of the path, and these two values are averaged together for the returned proportion value.

The result of this can be used to weigh distance between paths in terms of overall path similarity. A reasonable measure of path similarity might be the (area between paths) / (average length of two paths) / overlap, so that two paths with low overlap get their total distance inflated by that proportion.

Value

Return value is a number between 0 and 1. Values below 0.5 are difficult, because the average overlap of the two paths is found. Thus, even if one path has 0 overlap, the other is likely to have an overlap near 1.0, resulting in an average of 0.5.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

`PlotGrid` *Plot the grid associated with a particular mapping problem.*

Description

This function plots a grid associated with the mapping between two paths. This will plot n*2-1 by m*2-1 nodes, associated with each point and segment in each path. This gets quite substantial when the paths have more than a handful of points (5-10), and is only really useful for display and debugging purposes. This is called by `CreateMap`, which overlays costs over the map on its own.

Usage

```
PlotGrid(path1,path2)
```

Arguments

`path1`	The first path
`path2`	The second path

Details

Warning–do not use for anything but very small paths, as it will take a long time to draw and be uninterpretable.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). https://sites.google.com/a/mtu.edu/mapping/

`PlotMap` *Plot the paths and their mapping.*

Description

Plots the paths and the mappings between paths.

Usage

```
PlotMap(mapping, cols = c("grey40"), linecol = "grey25",
          xlim= NA, ylim = NA, ...)
```

Arguments

`mapping`	A map object produced by CreateMap() or GetMinMap()
`cols`	A set of colors to shade consecutive polygons.
`linecol`	Color for lines connecting paths
`xlim`	two values that override default x range.
`ylim`	two values that override default y range.
`...`	other graphical arguments

`SimplifyPath` *Simplify a path*

Description

This function takes a path and removes points that impact the shape of the path less than a tolerance value, which is a scaled measure of degree deviation from removing a path. It is based on an algorithm described by Latecki & Lakaemper.

Usage

```
SimplifyPath(path,
             tolerance = 0.075,
             truncate=F,
             faster = T,
             verbose = F, plot = F)
```

Arguments

`path`	a 2-column matrix of x,y points
`truncate`	Number of digits to round values to. If FALSE (the default), there is no rounding. If another value, the value is used as the `digits` argument of `round` to use to truncate the precision of the path. This can help simplify the path by rounding to a known precision level.
`tolerance`	a tolerance threshold; any point that does not impact the shape of the path more than this is removed. For this, smaller values remove fewer points. Depending on the complexity of the path, values from 0.01 to 0.1 may be good starting points.
`faster`	If `TRUE`, this uses a faster point-trimming method. If `T`, it removes all points that have a K value equal to the minimum value, which can be many (for example, a bunch that are equal to 0). If `F`, it will only remove one one each round, which can be very slow.
`verbose`	if `TRUE`, will print out intermediate information during the shape evolution.
`plot`	if `TRUE`, will create a plot of the path and overlay the evolving simplified path.

Value

Returns a path with redundant points removed.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016).
and
L. J. Latecki and R. Lakaemper. Convexity Rule for Shape Decomposition Based on Discrete Contour Evolution. Computer Vision and Image Understanding, vol. 73, pp. 441-454, 1999.
https://sites.google.com/a/mtu.edu/mapping/

Examples

```
path <- cbind(1:100,exp(-(1:100-50)^2/80))
path2 <- SimplifyPath(path)


## Not run:
plot(path)
plot(path2)
plot(path)
points(SimplifyPath(path,tolerance=.1),type="o",
        col="red",cex=1.2,lwd=2)
plot(path)
points(SimplifyPath(path,tolerance=.01,plot=TRUE),type="o",
        col="red",cex=1.2,lwd=2)
plot(path)
points(SimplifyPath(path,tolerance=.005,plot=TRUE),type="o",
        col="red",cex=1.2,lwd=2)

## End(Not run)
```

`SummarizeMapping` *Summarize the mapping obtained by GetMinMap*

Description

Returns a data frame outlining specifically the mapping between two paths, including all the inferred points, identifying the original points, and the distances between corresponding points/segments.

Usage

```
SummarizeMapping(mapping)
```

Arguments

`mapping`	The output of `GetMinMapping()`

Details

This provides a detailed analysis of the outcome of a mapping, in the form of a seven-column data frame. An original path is transformed into a multi-segment path by adding intersections and 'opposite' points lying on segments.

Value

The first two columns indicate the index node of these two lengthened paths that correspond to one another. Odd nodes indicate points, and even nodes indicate segments (segments can be skipped). These correspond to rows and columns of the optimization matrix also returned by `CreateMap` and `GetMinMapping`.

The next two columns indicate how these rows are mapped back onto the original paths. 0 indicates the row was not an original point, non-zero integers indicate specific elements of the paths.

The next two column indicate the x,y coordinates on path 1 of each point (original or inferred). The following two columns indicate the x,y, coordinates on path 2. The final column indicates the euclidean distance corresponding to the particular mapping indicated.

Author(s)

Shane T. Mueller and Brandon Perelman

`surveyors` *The Surveyor's Formula*

Description

Computes the area of a polygon using the so-called Surveyor's formula, with special-purpose faster versions for 3- and 4-gons, and a compiled version implemented via the 'shoelace' formula.

Usage

```
surveyors(poly,usedet=FALSE)
surveyors.3(poly)
surveyors.4(poly)
shoelace(poly)
```

Arguments

`poly`	A 2-column matrix containing the vertices of a polygon (x and y coordinates).
`usedet`	TRUE/FALSE variable, if T, will force the use of the true surveyor's formula, which is 15-20x slower than the special-purpose triangle/4-gon code. This is only really good for testing/validating things.

Details

This computes the area of a polygon using the so-called 'surveyor's' formula. It computes the sum of the determinants of each edge, which results in the area of the polygon, provided the polygon is regular (does not intersect itself, etc.). If the polygon is not regular, it will not measure the area, because negative areas will be subtracted from positive areas.

The general surveyor's formula is pretty inefficient. We primarily (probably exclusively) use it for 2-, 3-, and 4-gons, and so there are special-purpose functions defined as `surveyors.3` and `surveyors.4` that are used in these cases that are about 15-20x faster than the general one, using the so-called 'shoelace' formula.

Calls to this function account for a lot of the efficiency of the entire algorithm. Currently, surveyors tests the size of the n-gon and routes to the specialized function, which appears to add 20% overhead. If you know that you are dealing with a trigon or a quadrilateral, you can cut down time by a small amount. even more efficiency.

The shoelace function is a compiled c function that implements the surveyor's formula for polygons of any size. It is not restricted to 3- or 4-gons like surveyors.3 and surveyors.4, but is still slightly slower than these functions for 3- and 4-gons because of the overhead of calling the compiled function. It takes about 1.5x as long as the special-purpose surveyor's formula for 3 and 4 points, but it is not restricted on the number of points, so is a fast replacement for the much slower surveyors() function. Consequently, it is not currently used in the library directly. Note that like all surveyor's formula implementations, it will not handle cross-over paths appropriately.

Value

returns a measure of area.

Author(s)

Shane T. Mueller and Brandon Perelman

References

See Mueller et al., (2016). `https://sites.google.com/a/mtu.edu/mapping/` and B. Braden. "The Surveyor's Area Formula". The College Mathematics Journal, vol. 17, no. 4, pp. 326-337, 1986.

Examples

```
poly <- rbind(c(1,1),c(10,1),c(5,3))
surveyors(poly)

## Profiling test for 3-gon
  poly <- rbind(c(1.1,1.2),c(2.1,3.3),c(4.1,1.2))
#system.time(for(i in 1:50000)surveyors(poly,usedet=TRUE))
#system.time(for(i in 1:50000)surveyors(poly))
#system.time(for(i in 1:50000)surveyors.3(poly))
#system.time(for(i in 1:50000)shoelace(poly))
# Profiling Test for 4-gon
poly2 <- rbind(c(1.1,1.2),c(2.2,1.3),c(4.0,4.25),c(1.3,3.9))
#system.time(for(i in 1:50000)surveyors(poly2,usedet=TRUE))
#system.time(for(i in 1:50000)surveyors(poly2))
#system.time(for(i in 1:50000)surveyors.4(poly2))
#system.time(for(i in 1:50000)shoelace(poly2))

poly3 <- cbind(runif(20),runif(20))
#system.time(for(i in 1:50000)surveyors(poly3,usedet=TRUE))
#system.time(for(i in 1:50000)shoelace(poly3))
```

Index