Package 'OSMscale'

September 21, 2016

Title Add a Scale Bar to 'OpenStreetMap' Plots

Version 0.3.5

Date 2016-09-20

Author Berry Boessenkool

Maintainer Berry Boessenkool <berry-b@gmx.de>

Description

Functionality to handle and project lat-long coordinates, easily download background maps and add a correct scale bar to 'OpenStreetMap' plots in any map projection.

Imports OpenStreetMap, berryFunctions (>= 1.12.0), sp

URL https://github.com/brry/OSMscale

License GPL-3

Encoding UTF-8

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-09-21 19:27:31

R topics documented:

OSMscale-package *Add a Scalebar to OpenStreetMap Plots*

Description

Functionality to handle and project lat-long coordinates, easily download background maps and add a correct scale bar to 'OpenStreetMap' plots in any map projection. There are some other spatially related miscellaneous functions as well.

Note

Get the most recent code updates at https://github.com/brry/OSMscale

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, June 2016

See Also

scaleBar, pointsMap, projectPoints

Examples

```
d <- read.table(sep=",", header=TRUE, text=
"lat, long
55.685143, 12.580008
52.514464, 13.350137
50.106452, 14.419989
48.847003, 2.337213
51.505364, -0.164752")

# zoom set to 3 to speed up tests. automatic zoom determination is better.
map <- pointsMap(lat, long, data=d, type="maptoolkit-topo",
                 utm=TRUE, scale=FALSE, zoom=3, pch=16, col=2)
scaleBar(map, abslen=500, y=0.8, cex=0.8)
lines(projectPoints(d$lat, d$long), col="blue", lwd=2)
```

biketrack *GPS recorded bike track*

Description

My daily bike route, recorded with the app OSMtracker on my Samsung Galaxy S5

Format

'data.frame': 254 obs. of 4 variables:
$ lon : num 13 13 13 13 13 ...
$ lat : num 52.4 52.4 52.4 52.4 52.4 ...
$ time: POSIXct, format: "2016-05-18 07:53:22" "2016-05-18 07:53:23" ...
$ ele : num 66 66 66 67 67 67 68 69 69 69 ....

Source

GPS track export from OSMtracker App

Examples

```
data(biketrack)
plot(biketrack[,1:2])
# see equidistPoints
```

---

checkLL *lat-long coordinate check*

Description

check lat-long coordinates for plausibility

Usage

```
checkLL(lat, long, data, fun = stop, trace = TRUE, ...)
```

Arguments

`lat, long`	Latitude (North/South) and longitude (East/West) coordinates in decimal degrees
`data`	Optional: data.frame with the columns `lat` and `long`
`fun`	One of the functions `stop`, `warning`, or `message`. DEFAULT: stop
`trace`	Logical: Add function call stack to the message? DEFAULT: TRUE WARNING: in do.call settings with large objects (like `map` in `scaleBar`), tracing may take a lot of computing time.
`...`	Further arguments passed to `fun`

Value

Invisible T/F vector showing which of the coordinates is violated in the order: minlat, maxlat, minlong, maxlong. Only returned if check is passed or fun != stop

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Aug 2016

See Also

pointsMap, putm, berryFunctions::checkFile

Examples

```
checkLL(lat=52, long=130)
checkLL(130, 52, fun=message)
checkLL(85:95, 0, fun=message)

d <- data.frame(x=0, y=0)
checkLL(y,x, d)

## Not run:
checkLL(85:95, 0, fun="message")
checkLL(170,35) # throws an informative error
checkLL(85:95, 0, trace=FALSE)

## End(Not run)

mustfail <- function(expr) stopifnot(berryFunctions::is.error(expr))
mustfail( checkLL(100)         )
mustfail( checkLL(100, 200)    )
mustfail( checkLL(-100, 200)   )
mustfail( checkLL(90.000001, 0)   )
```

---

degree *decimal degree coordinate conversion*

Description

Convert latitude-longitude coordinates between decimal representation and degree-minute-second notation

Usage

```
degree(lat, long, data, todms = !is.character(lat), digits = 1,
  drop = FALSE)
```

Arguments

lat, long	Latitude (North/South) and longitude (East/West) coordinates in decimal degrees
data	Optional: data.frame with the columns lat and long

todms	Logical specifying direction of conversion. If FALSE, converts to decimal degree notation, splitting coordinates at the symbols for degree, minute and second (\U00B0, ', "). DEFAULT: !is.character(lat)
digits	Number of digits the seconds are rounded to. DEFAULT: 1
drop	Drop to lowest dimension? DEFAULT: FALSE

Value

data.frame with x and y as character strings or numerical values, depending on conversion direction

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Aug 2016

See Also

earthDist, projectPoints for geographical reprojection, sp::char2dms

Examples

```
# DECIMAL to DMS notation: ------------------------------------------------
degree(52.366360, 13.024181)
degree(c(52.366360, -32.599203), c(13.024181,-55.809601))
degree(52.366360, 13.024181, drop=TRUE) # vector
degree(47.001, -13.325731, digits=5)

# Use table with values instead of single vectors:
d <- read.table(header=TRUE, sep=",", text="
lat, long
 52.366360,  13.024181
-32.599203, -55.809601")
degree(lat, long, data=d)

# DMS to DECIMAL notation: ------------------------------------------------
# You can use the degree symbol and escaped quotation mark (\") as well.
degree("52'21'58.9"N", "13'1'27.1"E")
print(degree("52'21'58.9"N", "13'1'27.1"E"), digits=15)

d2 <- read.table(header=TRUE, stringsAsFactors=FALSE, text="
lat long
52'21'58.9"N 13'01'27.1"E
32'35'57.1"S 55'48'34.6"W") # columns cannot be comma-separated!
degree(lat, long, data=d2)

# Rounding error checks: --------------------------------------------------
oo <- options(digits=15)
d
degree(lat, long, data=degree(lat, long, d))
degree(lat, long, data=degree(lat, long, d, digits=3))
options(oo)
stopifnot(all(degree(lat,long,data=degree(lat,long,d, digits=3))==d))
```

earthDist *distance between lat-long coordinates*

Description

Great-circle distance between points at lat-long coordinates. (The shortest distance over the earth's surface). The distance of all the entries relative to the first one is computed.

Usage

```
earthDist(lat, long, data, r = 6371, trace = TRUE)
```

Arguments

lat, long	Latitude (North/South) and longitude (East/West) coordinates in decimal degrees
data	Optional: data.frame with the columns lat and long
r	radius of the earth. Could be given in miles. DEFAULT: 6371 (km)
trace	Logical: trace the coordinate check with checkLL? Should be set to FALSE in a do.call setting to avoid overhead computing time. DEFAULT: TRUE

Value

Vector with distance(s) in km

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Aug 2016. Angle formula from Diercke Weltatlas 1996, Page 245

See Also

degree for pre-formatting, http://www.movable-type.co.uk/scripts/latlong.html

Examples

```
d <- read.table(header=TRUE, sep=",", text="
lat, long
52.514687, 13.350012   # Berlin
35.685024, 139.753365  # Tokio
51.503162, -0.131082") # London
earthDist(lat, long, d) # 8922 and 928 km
map <- pointsMap(lat, long, d, zoom=2, abslen=5000, y=0.7)
scaleBar(map, y=0.5, abslen=5000)   # in mercator projections, scale bars are not
scaleBar(map, y=0.3, abslen=5000)   # transferable to other latitudes
# slightly different with other formulas:
# geosphere::distHaversine(as.matrix(d[1,2:1]), as.matrix(d[2,2:1])) / 1000
```

```
# compare with UTM distance
set.seed(42)
d <- data.frame(lat=runif(100, 47,54), long=runif(100, 6, 15))
d2 <- projectPoints(d$lat, d$long)
d_utm <- berryFunctions::distance(d2$x[-1],d2$y[-1], d2$x[1],d2$y[1])/1000
d_earth <- earthDist(lat,long, d)
plot(d_utm, d_earth) # distances in km
hist(d_utm-d_earth) # UTM distance slightly larger than earth distance
plot(d_earth, d_utm-d_earth) # correlates with distance
berryFunctions::colPoints(d2$x[-1], d2$y[-1], d_utm-d_earth, add=FALSE)
points(d2$x[1],d2$y[1], pch=3, cex=2, lwd=2)
```

`equidistPoints` *Evenly spaced points along path*

Description

Compute waypoints with equal distance to each other along a (curved) path or track given by coordinates

Usage

```
equidistPoints(x, y, z, data, n, nint = 30, mid = FALSE, ...)
```

Arguments

`x, y, z`	Vectors with coordinates. z is optional and can be left empty
`data`	Optional: data.frame with the column names as given by x,y (and z)
`n`	Number of segments to create along the path (=number of points-1)
`nint`	Number of points to interpolate between original coordinates (with `approx2`). Larger numbers give more precisely equidistant points, but increase computing time. `int=1` to not do any interpolation. DEFAULT: 30
`mid`	Logical: Should centers of segments be returned instead of their ends?
`...`	Further arguments passed to `approx`

Value

Dataframe with the coordinates of the final points. ATTENTION: The columns are named x,y,z, not with the original names from the function call.

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, May 2016

See Also

`berryFunctions::distance` and `approx2`

Examples

```
library(berryFunctions) # distance, colPoints etc
x <- c(2.7, 5, 7.8, 10.8, 13.7, 15.8, 17.4, 17.7, 16.2, 15.8, 15.1, 13.1, 9.3, 4.8, 6.8, 12.2)
y <- c(2.3, 2.1, 2.6, 3.3, 3.7, 4.7, 7.6, 11.7, 12.4, 12.3, 12.3, 12.3, 12, 12.1, 17.5, 19.6)
eP <- equidistPoints(x,y, n=10) ; eP
plot(x,y, type="o", pch=4)
points(equidistPoints(x,y, n=10), col=4, pch=16)
points(equidistPoints(x,y, n=10, nint=1), col=2) # from original point set
round(distance(eP$x, eP$y), 2) # the 2.69 instead of 4.50 is in the sharp curve
# These points are quidistant along the original track

plot(x,y, type="o", pch=16, col=2)
round(sort(distance(x,y)), 2)
xn <- equidistPoints(x,y, n=10)$x
yn <- equidistPoints(x,y, n=10)$y
lines(xn,yn, type="o", pch=16)
round(sort(distance(xn,yn)), 2)
for(i in 1:8)
{
xn <- equidistPoints(xn,yn, n=10)$x
yn <- equidistPoints(xn,yn, n=10)$y
lines(xn,yn, type="o", pch=16)
print(round(sort(distance(xn,yn)), 2))
} # We may recursively get closer to equidistant along track _and_ air,
# but never actually reach it.

# Real dataset:
data(biketrack)
colPoints(lon, lat, ele, data=biketrack, add=FALSE, asp=1, pch=4, lines=TRUE)
points(equidistPoints(lon, lat, data=biketrack, n=25), pch=3, lwd=3, col=2)
bt2 <- equidistPoints(lon, lat, ele, data=biketrack, n=25)
bt2$dist <- distance(bt2$x, bt2$y)*1000
colPoints(x, y, z, data=bt2, legend=FALSE)
# in curves, crow-distance is shorter sometimes
plot(lat~lon, data=biketrack, asp=1, type="l")
colPoints(x, y, dist, data=bt2, Range=c(2.5,4), add=TRUE, asp=1, pch=3, lwd=5)
lines(lat~lon, data=biketrack)
```

---

pointsMap *Get map for lat-long points*

Description

Download and plot map with the extend of a dataset with lat-long coordinates

Usage

```
pointsMap(lat, long, data, ext = 0.07, fx = 0.05, fy = fx, type = "osm",
```

```
zoom = NULL, minNumTiles = 9L, mergeTiles = TRUE, map = NULL,
utm = FALSE, proj = putm(long = long), plot = TRUE, add = FALSE,
scale = TRUE, quiet = FALSE, pch = 3, col = "red", cex = 1,
pargs = NULL, ...)
```

Arguments

lat, long	Latitude (North/South) and longitude (East/West) coordinates in decimal degrees
data	Optional: data.frame with the columns lat and long
ext	Extension added in each direction if a single coordinate is given. DEFAULT: 0.07
fx, fy	Extend factors (additional map space around actual points) passed to custom version of extendrange. DEFAULT: 0.05
type	Tile server in openmap
zoom, minNumTiles, mergeTiles	Arguments passed to openmap
map	Optional map object. If given, it is not downloaded again. Useful to project maps in a second step. DEFAULT: NULL
utm	Logical: Convert map to UTM (or other proj)? Consumes some extra time. DEFAULT: FALSE
proj	proj4 character string or CRS object to project to. Only used if utm=TRUE. DEFAULT: putm(long=long)
plot	Logical: Should map be plotted and points added? DEFAULT: TRUE
add	Logical: add points to existing map? DEFAULT: FALSE
scale	Logical: should scaleBar be added? DEFAULT: TRUE
quiet	Logical: suppress progress messages? DEFAULT: FALSE
pch, col, cex	Arguments passed to points. DEFAULT: 3, "red", 1
pargs	List of arguments passed to points like lwd, type, cex...
...	Further arguments passed to scaleBar like abslen, ndiv, ...

Value

Map returned by openmap

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Jun 2016

See Also

projectPoints, OpenStreetMap::openmap

Examples

```
if(interactive()){
d <- read.table(sep=",", header=TRUE, text=
"lat, long # could e.g. be copied from googleMaps, rightclick on What's here?
43.221028, -123.382998
43.215348, -123.353804
43.227785, -123.368694
43.232649, -123.355895")

map <- pointsMap(lat, long, data=d)
map_utm <- pointsMap(lat, long, d, map=map, utm=TRUE)
axis(1); axis(2) # now in meters
projectPoints(d$lat, d$long)
scaleBar(map_utm, x=0.2, y=0.8, unit="mi", type="line", col="red", length=0.25)
pointsMap(lat, long, d[1:2,], map=map_utm, add=TRUE, col="red", pch=3, pargs=list(lwd=3))

d <- data.frame(long=c(12.95, 12.98, 13.22, 13.11), lat=c(52.40,52.52, 52.36, 52.45))
map <- pointsMap(lat,long,d, type="bing") # aerial map
}
```

`proj` *CRS of various PROJ.4 projections*

Description

coordinate reference system (CRS) Object for several proj4 character strings. `posm` and `pll` are taken directly from `OpenStreetMap::osm` and `longlat`.

Usage

```
putm(long, zone = mean(long, na.rm = TRUE)%/%6 + 31)

posm()

pll()
```

Arguments

`long`	Vector of decimal longitude coordinates (East/West values). Not needed of `zone` is given.
`zone`	UTM (Universal Transverse Mercator) zone, see e.g. https://upload.wikimedia.org/wikipedia/commons/e/ed/Utm-zones.jpg. DEFAULT: UTM zone at mean of `long`

Value

sp::CRS objects for one of:
- UTM projection with given zone
- Open street map (and google) mercator projection
- Latitude Longitude projection

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Aug 2016

See Also

projectPoints, degree

Examples

```
posm()
str(posm())
posm()@projargs
pll()
putm(5:14) # Germany
putm(zone=33) # Berlin
```

---

projectPoints *Project lat-lon points*

Description

Project long lat points to e.g. UTM projection. Basics copied from OpenStreetMap::projectMercator

Usage

```
projectPoints(lat, long, data, from = pll(), to = putm(long = long),
  spout = FALSE, dfout = TRUE, drop = FALSE, quiet = FALSE)
```

Arguments

lat, long	Latitude (North/South) and longitude (East/West) coordinates in decimal degrees
data	Optional: data.frame with the columns lat and long
from	Original Projection CRS (do not change for latlong-coordinates). DEFAULT: pll() = sp::CRS("+proj=longlat +datum=WGS84")
to	target projection CRS (Coordinate Reference System) Object. Other projections can be specified as sp::CRS("your_proj4_character_string"). DEFAULT: putm(long=long)

spout	Return the original spTransform output instead of coordinates only? DEFAULT: FALSE
dfout	Convert output to data.frame to allow easier indexing? DEFAULT: TRUE
drop	Drop to lowest dimension? DEFAULT: FALSE (unlike projectMercator)
quiet	Suppress warning about NA coordinates? DEFAULT: FALSE

Value

data.frame (or matrix, if dfout=FALSE) with points in new projection

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Jun 2016

See Also

scaleBar, OpenStreetMap::projectMercator, http://gis.stackexchange.com/a/74723, http://spatialreference.org on proj4strings

Examples

```
library("OpenStreetMap")
lat <- runif(100, 6, 12)
lon <- runif(100, 48, 58)
plot(lat,lon)
plot(projectMercator(lat,lon), main="Mercator")
plot(projectPoints(lat,lon), main="UTM32")
stopifnot(all( projectPoints(lat,lon, to=posm()) == projectMercator(lat,lon) ))

projectPoints(c(52.4,NA),      c(13.6,12.9))
projectPoints(c(52.4,NA),      c(13.6,12.9), quiet=TRUE)
projectPoints(c(52.4,52.3,NA), c(13.6,12.9,13.1))
projectPoints(c(52.4,52.3,NA), c(13.6,NA  ,13.1))
projectPoints(c(52.4,52.3,NA), c(NA  ,12.9,13.1))

# Reference system ETRS89 with GRS80-Ellipsoid (common in Germany)
set.seed(42)
d <- data.frame(N=runif(50,5734000,6115000), E=runif(50, 33189000,33458000))
d$VALUES <- berryFunctions::rescale(d$N, 20,40) + rnorm(50, sd=5)
head(d)
c1 <- projectPoints(lat=d$N, long=d$E-33e6, to=pll(),
          from=sp::CRS("+proj=utm +zone=33 +ellps=GRS80 +units=m +no_defs") )
c2 <- projectPoints(y, x, data=c1, to=posm() )
head(c1)
head(c2)


map <- pointsMap(y,x, c1, plot=FALSE)
pdf("ETRS89.pdf")
par(mar=c(0,0,0,0))
plot(map)
```

```
rect(par("usr")[1], par("usr")[3], par("usr")[2], par("usr")[4],
     col=berryFunctions::addAlpha("white", 0.7))
scaleBar(map, y=0.2, abslen=100)
points(c2)
berryFunctions::colPoints(c2$x, c2$y, d$VALUE )
dev.off()
system2("open", "ETRS89.pdf")
#unlink("ETRS89.pdf")
```

`randomPoints` *Distanced random points*

Description

Arranges points in square randomly, but with certain minimal distance to each other

Usage

```
randomPoints(xmin, xmax, ymin, ymax, number, mindist, plot = TRUE, ...)
```

Arguments

`xmin`	Minimum x coordinate
`xmax`	Upper limit x values
`ymin`	Ditto for y
`ymax`	And yet again: Ditto.
`number`	How many points should be randomly + uniformly distributed
`mindist`	Minimum DIstance each point should have to others
`plot`	Plot the result? DEFAULT: TRUE
`...`	Further arguments passed to plot

Value

data.frame with x and y coordinates.

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, 2011/2012

See Also

`distance`, the package RandomFields (https://cran.r-project.org/package=RandomFields)

Examples

```
P <- randomPoints(xmin=200,xmax=700, ymin=300,ymax=680, number=60,mindist=10, asp=1)
rect(xleft=200, ybottom=300, xright=700, ytop=680, col=NA, border=1)

format( round(P,4), trim=FALSE)

for(i in 1:10)
{
rp <- randomPoints(xmin=0,xmax=20, ymin=0,ymax=20, number=20, mindist=3, plot=FALSE)
plot(rp, las=1, asp=1, pch=16)
abline(h=0:30*2, v=0:30*2, col=8); box()
for(i in 1:nrow(rp))
    berryFunctions::circle(rp$x[i],rp$y[i], r=3, col=rgb(1,0,0,alpha=0.2), border=NA)
}
```

`scaleBar` *scalebar for OSM plots*

Description

Add a scalebar to default or (UTM)-projected OpenStreetMap plots

Usage

```
scaleBar(map, x = 0.1, y = 0.9, length = 0.4, abslen = NA,
  unit = c("km", "m", "mi", "ft", "yd"), label = unit, type = c("bar",
  "line"), ndiv = NA, field = "rect", fill = NA, adj = c(0.5, 1.5),
  cex = par("cex"), col = c("black", "white"), targs = NULL, lwd = 7,
  lend = 1, bg = "transparent", mar = c(2, 0.7, 0.2, 3), ...)
```

Arguments

`map`	Map object with map$tiles[[1]]$projection to get the projection from.
`x, y`	Relative position of left end of scalebar. DEFAULT: 0.1, 0.9
`length`	Approximate relative length of bar. DEFAULT: 0.4
`abslen`	Absolute length in `units`. DEFAULT: NA (computed internally from `length`)
`unit`	Unit for computation and label. Possible are kilometer and meter as well as miles, feet and yards. Note that the returned absolute length is in m. DEFAULT: "km"
`label`	Unit label in plot. DEFAULT: `unit`
`type`	Scalebar type: simple `'line'` or classical black & white `'bar'`. DEFAULT: "bar"

ndiv	Number of divisions if type="bar". DEFAULT: NA (computed internally) Internal selection of ndiv is based on divisibility of abslen (modulo) with 1:6. For ties, preferation order is 5>4>3>2>6>1. For maps with abslen=4000, this means 5 will be chosen, even though 4 is more appealing. if abslen is also missing (or in a certain set), a better default is chosen.
field, fill, adj, cex	Arguments passed to textField
col	Vector of (possibly alternating) colors passed to segments or rect. DEFAULT: c("black","white")
targs	List of further arguments passed to textField like font, col (to differ from bar color), etc. DEFAULT: NULL
lwd, lend	Line width and end style passed to segments. DEFAULT: 5,1, which works well in pdf graphics.
bg	Background color, e.g. addAlpha(White). DEFAULT: "transparent" to suppress background.
mar	Background margins approximately in letter width/height. DEFAULT: c(2,0.7,0.2,3)
...	Further arguments passed to segments like lty. (Color for segments is the first value of col). Passed to rect if type="bar", like lwd.

Details

This uses a hack to get the right distance in the default mercator projected maps. There, the axes are not in meters, but rather ca 0.7m units (for NW Germany area maps with 20km across). Accordingly, other packages plot wrong bars:
SDMTools::Scalebar(x=1442638,y=6893871,distance=10000)
raster::scalebar(d=5000, xy=c(1442638,6893871))
mapmisc::scaleBar(map$tiles[[1]]$projection, seg.len=10, pos="center", bg="transparent")
I suppose this function works for other projections as well, but haven't tried yet. You might need to specify abslen manually with other projections where the axes do not resemble meters at all.

Value

invisible: coordinates of scalebar and label

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, Jun 2016

See Also

pointsMap, projectPoints

Examples

```
if(interactive()){
d <- data.frame(long=c(12.95, 12.98, 13.22, 13.11), lat=c(52.40,52.52, 52.36, 52.45))
map <- pointsMap(lat,long,d, scale=FALSE, zoom=9)
coord <- scaleBar(map)  ; coord
```

```
scaleBar(map, bg=berryFunctions::addAlpha("white", 0.7))
scaleBar(map, 0.3, 0.05, unit="m", length=0.45, type="line")
scaleBar(map, 0.3, 0.5, unit="km", abslen=5, col=4:5, lwd=3)
scaleBar(map, 0.3, 0.8, unit="mi", col="red", targ=list(col="blue", font=2), type="line")

# I don't like subdivisions, but if you wanted them, you could use:
scaleBar(map, 0.12, 0.28, abslen=10, adj=c(0.5, -1.5)  )
scaleBar(map, 0.12, 0.28, abslen=4, adj=c(0.5, -1.5), targs=list(col="transparent"), label="" )
}

## Not run:  ## Too much downloading time, too error-prone
# Tests around the world
par(mfrow=c(1,2), mar=rep(1,4))
long <- runif(2, -180, 180) ;  lat <- runif(2, -90, 90)
map <- pointsMap(lat, long)
map2 <- pointsMap(lat, long, map=map, utm=TRUE)

## End(Not run)
```

`triangleArea` *Area of a triangle*

Description

calculate Area of a planar triangle

Usage

```
triangleArea(x, y, digits = 3)
```

Arguments

`x`	Vector with 3 values (x coordinates of triangle corners)
`y`	Ditto for y.
`digits`	Number of digits the result is rounded to. DEFAULT: 3)

Value

Numeric

Author(s)

Berry Boessenkool, <berry-b@gmx.de>, 2011

See Also

`berryFunctions::distance`

Examples

```
a <- c(1,5.387965,9); b <- c(1,1,5)
plot(a[c(1:3,1)], b[c(1:3,1)], type="l", asp=1)#; grid()

triangleArea(a,b)
#triangleArea(a,b[1:2])
```

Index