Package 'recexcavAAR'

February 25, 2017

Type Package

Title 3D Reconstruction of Archaeological Excavations

Version 0.3.0

Maintainer Clemens Schmid <clemens@nevrome.de>

Description A toolset for 3D reconstruction and analysis of excavations. It provides methods to reconstruct natural and artificial surfaces based on field measurements. This allows to spatially contextualize documented subunits and features. Intended to be part of a 3D visualization workflow.

Date 2017-02-15

License GPL-2

LazyData TRUE

RoxygenNote 6.0.1

URL https://github.com/ISAAKiel/recexcavAAR

Imports Rcpp (>= 0.12.7)

Suggests devtools (>= 1.12.0), dplyr (>= 0.5.0), knitr (>= 1.15.1), magrittr (>= 1.5), rgl (>= 0.96.0), rmarkdown (>= 1.0), roxygen2 (>= 5.0.1), testthat (>= 1.0.2)

VignetteBuilder knitr

Depends R (>= 3.3.2), kriging (>= 1.1)

LinkingTo Rcpp

NeedsCompilation yes

Author Clemens Schmid [cre, cph, aut], Benjamin Serbe [aut]

Repository CRAN

Date/Publication 2017-02-25 23:55:29

R topics documented:

`cootrans` *Tool for transforming local metric coordinates*

Description

This function transforms local metric coordinates to absolute coordinates of referenced systems by use of a two dimensional four parameter Helmert transformation. This function does not cover the transformation of three dimensional points or transformation between two different datums.

Usage

```
cootrans(pair_matrix, pm_column, data_matrix, dm_column, checking = FALSE,
  checkplot = TRUE)
```

Arguments

`pair_matrix`	data.frame or matrix with pairs of local and corresponding absolute coordinates (Minimum two!)
`pm_column`	vector with numerical index of the columns in order: local x-value, local y-value, absolute x-value, absolute y-value
`data_matrix`	data.frame with local x- and y-values which schould be transformed.
`dm_column`	vector with numerical index of the columns in order: local x-value, local y-value.
`checking`	boolean switch to turn on the checking ability. Default: FALSE. If TRUE showes combined coordinate plots with indexed points and alters return of function.
`checkplot`	boolean switch to turn off the checking plot. Default: TRUE. Only matters if checking == TRUE.

Value

Original data.frame with additional columns containing the absolute x- and y-coordinates. In case of 'checking = TRUE' returns pair_matrix data.frame with additional columns of scale and rotation arc in degrees.

Examples

```
coord_data <- data.frame(
 loc_x = c(1,3,1,3),
 loc_y = c(1,1,3,3),
 abs_x = c(107.1,107,104.9,105),
 abs_y = c(105.1,107,105.1,106.9)
)

data_table <- data.frame(
 x = c(1.5,1.2,1.6,2),
 y = c(1,5,2.1,2),
 type = c("flint","flint","pottery","bone")
)

new_frame <- cootrans(coord_data, c(1,2,3,4), data_table, c(1,2))

check_data <- cootrans(coord_data, c(1,2,3,4), data_table, c(1,2), checking = TRUE)
```

---

draw_circle *Draws a circular point cloud (3D)*

Description

Draws a 2D circle on x- and y-plane around a center point in 3D space.

Usage

```
draw_circle(centerx, centery, centerz, radius, resolution = 30L)
```

Arguments

centerx	x axis value of circle center point
centery	y axis value of circle center point
centerz	z axis value of circle center point
radius	circle radius
resolution	amount of circle points (default = 30)

Value

data.frame with the spatial coordinates of the resulting points

Examples

```
draw_circle(
  centerx = 4,
  centery = 5,
  centerz = 1,
  radius = 3,
  resolution = 20
)

circ <- draw_circle(1,2,3,2)

plot(circ$x, circ$y)
```

`draw_sphere` *Draws a spherical point cloud (3D)*

Description

Draws a sphere around a center point in 3D space.

Usage

```
draw_sphere(centerx, centery, centerz, radius, phires = 10L, thetares = 10L)
```

Arguments

`centerx`	x axis value of sphere center point
`centery`	y axis value of sphere center point
`centerz`	z axis value of sphere center point
`radius`	sphere radius
`phires`	phi resolution (default = 10)
`thetares`	theta resolution (default = 10)

Value

data.frame with the spatial coordinates of the resulting points

Examples

```
sphere <- draw_sphere(
  centerx = 4,
  centery = 5,
  centerz = 1,
  radius = 3,
  phires = 20,
  thetares = 20
```

```
)

#library(rgl)
#plot3d(sphere)
```

`fillhexa` *Fills hexahedrons with a regular point raster (3D)*

Description

A hexahedron is a three dimensional shape that is defined by 6 faces and 8 corner points. `fillhexa` allows to fill such a shape with a regular point raster.

Usage

```
fillhexa(hex, res)
```

Arguments

`hex`	dataframe with three columns and eight rows to define a hexahedron by its corner point coordinates x, y and z
`res`	numeric value > 0 and <= 1 for the resolution of the point raster

Details

See https://stackoverflow.com/questions/36115215/filling-a-3d-body-with-a-systematic-point-raster for a description of the function and how it was developed.

Value

data.frame with the spatial coordinates of the resulting points of the grid

Examples

```
hexatestdf <- data.frame(
  x = c(0,1,0,4,5,5,5,5),
  y = c(1,1,4,4,1,1,4,4),
  z = c(4,8,4,9,4,8,4,6)
)

cx = fillhexa(hexatestdf, 0.1)

#library(rgl)
#plot3d(
# cx[,1], cx[,2], cx[,3],
# type = "p",
# xlab = "x", ylab = "y", zlab = "z"
#)
```

kriglist *Apply kriging {kriging} to a list of data.frames*

Description

kriging {kriging} is a simple and highly optimized ordinary kriging algorithm to plot geographical data. This interface to the method allows to not just apply it to one data.frame but to a list of data.frames. The result is reduced to the data.frame with the predicted values. For a more detailed output kriging {kriging} has to be called for the individual input data.frames.

Usage

```
kriglist(plist, x = 1, y = 2, z = 3, rdup = TRUE, ...)
```

Arguments

plist	List of data.frames with point coordinates
x	index of data.frame column with x-axis spatial points. Defaults to 1
y	index of data.frame column with y-axis spatial points. Defaults to 2
z	index of data.frame column with z-axis spatial points. Defaults to 3
rdup	switch to activate removal of double values for single horizontal positions in the input data.frames. Defaults to TRUE
...	Arguments to be passed to method kriging {kriging}

Value

list with data.frames which contains the predicted values along with the coordinate covariates

Examples

```
df1 <- data.frame(
 x = rnorm(50),
 y = rnorm(50),
 z = rnorm(50) - 5
)

df2 <- data.frame(
 x = rnorm(50),
 y = rnorm(50),
 z = rnorm(50) + 5
)

lpoints <- list(df1, df2)

surfacelist <- kriglist(lpoints, lags = 3, model = "spherical")
```

KT_spits *KT_data: Niveau measurements from the fictional trench of a excavation KT*

Description

A dataset containing coordinates of niveau measurements of a fictional excavation KT with 4 spits.

Format

A data frame with 304 rows and 4 variables:

- id: IDs of individual measurements with the information about to which level they belong
- x: x axis coordinates of measurements
- y: y axis coordinates of measurements
- z: z axis coordinates of measurements

See Also

Other KT_data: KT_squarecorners, KT_vessel

KT_squarecorners *KT_data: Corner points of a 1m*1m raster within the trench of a fictional excavation KT*

Description

A dataset containing horizontal coordinates of corner points of a 1m*1m raster within the rectangular trench (corner points of squares).

Format

A data frame with 63 rows and 2 variables:

- x: x axis coordinates of corner points
- y: y axis coordinates of corner points

See Also

Other KT_data: KT_spits, KT_vessel

KT_vessel *KT_data: Information about individual sherds of a reconstructed vessel from the trench of a fictional excavation KT*

Description

A dataset containing spatial and contextual information for individual sherds of a single vessel. Some sherds were documented in the field with single find measurements. For the others only spit and square attribution is possible.

Format

A data frame with 7 rows and 7 variables:

- inv: Inventory numbers of sherds. KTF means single find with individual measurement, KTM means mass find without this precise information.
- spit: spits where the sherds were found
- square: squares where the sherds were found
- feature: features where the sherds were found
- x: x axis coordinates of sherds
- y: y axis coordinates of sherds
- z: z axis coordinates of sherds

See Also

Other KT_data: KT_spits, KT_squarecorners

pnp *Check if a point is within a polygon (2D)*

Description

pnp is able to determine if a point is within a polygon in 2D space. The polygon is described by its corner points. The points must be in a correct drawing order.

Based on this solution: Copyright (c) 1970-2003, Wm. Randolph Franklin http://wrf.ecse.rpi.edu/pmwiki/pmwiki.php/Main/Software#toc24

Usage

```
pnp(vertx, verty, testx, testy)
```

Arguments

vertx	vector of x axis values of polygon corner points
verty	vector of y axis values of polygon corner points
testx	x axis value of point of interest
testy	y axis value of point of interest

Details

For discussion see: http://stackoverflow.com/questions/217578/how-can-i-determine-whether-a-2d-point-is
2922778#2922778

Value

boolean value - TRUE, if the point is within the polygon. Otherwise FALSE.

See Also

Other pnpfuncs: pnpmulti

Examples

```
df <- data.frame(
  x = c(1,1,2,2),
  y = c(1,2,1,2)
)

pnp(df$x, df$y, 1.5, 1.5)
pnp(df$x, df$y, 2.5, 2.5)

# caution: false-negatives in edge-cases:
pnp(df$x, df$y, 2, 1.5)
```

---

pnpmulti *Check if multiple points are within a polygon (2D)*

Description

pnpmulti works as pnp but for multiple points.

Usage

```
pnpmulti(vertx, verty, testx, testy)
```

Arguments

vertx	vector of x axis values of polygon corner points
verty	vector of y axis values of polygon corner points
testx	vector of x axis values of points of interest
testy	vector of y axis values of points of interest

Value

vector with boolean values - TRUE, if the respective point is within the polygon. Otherwise FALSE.

See Also

Other pnpfuncs: pnp

Examples

```
polydf <- data.frame(
  x = c(1,1,2,2),
  y = c(1,2,1,2)
)

testdf <- data.frame(
  x = c(1.5, 2.5),
  y = c(1.5, 2.5)
)

pnpmulti(polydf$x, polydf$y, testdf$x, testdf$y)
```

---

posdec *Multiple point position decision in relation to a set of stacked surfaces (3D)*

Description

posdec has the purpose to make a decision about the position of individual points in relation to a set of stacked surfaces in 3D space. The decision is made by comparing the mean z axis value of the four horizontally closest points of a surface to the z axis value of the point in question.

Usage

```
posdec(crdf, maplist)
```

Arguments

crdf	data.frame with the spatial coordinates of the points of interest. Must contain three columns with the x axis values, y axis values and z axis values of the points in the order x, y, z
maplist	list of data.frames which contain the points that make up the surfaces. The individual data.frames must have the same structure as crdf

Value

data.frame with the spatial coordinates of the points of interest and the respective position information

See Also

Other posdecfuncs: posdeclist

Examples

```
df1 <- data.frame(
  x = rnorm(50),
  y = rnorm(50),
  z = rnorm(50) - 5
)

df2 <- data.frame(
  x = rnorm(50),
  y = rnorm(50),
  z = rnorm(50) + 5
)

lpoints <- list(df1, df2)

maps <- kriglist(lpoints, lags = 3, model = "spherical")

finds <- data.frame(
  x = c(0, 1, 0.5, 0.7),
  y = c(0.5, 0, 1, 0.7),
  z = c(-10, 10, 0, 2)
)

posdec(finds, maps)
```

---

posdeclist *Multiple point position decision in relation to a set of stacked surfaces (3D) for multiple data.frames in a list*

Description

posdeclist works as posdec but not just for a single data.frame with individual points but for a list of data.frames

Usage

```
posdeclist(crdflist, maplist)
```

Arguments

crdflist	list of data.frames with the spatial coordinates of the points of interest (for details see posdec)
maplist	list of data.frames which contain the points that make up the surfaces

Value

list of data.frames with the spatial coordinates of the points of interest and the respective position information

See Also

Other posdecfuncs: posdec

Examples

```
df1 <- data.frame(
  x = rnorm(50),
  y = rnorm(50),
  z = rnorm(50) - 5
)

df2 <- data.frame(
  x = rnorm(50),
  y = rnorm(50),
  z = rnorm(50) + 5
)

lpoints <- list(df1, df2)

maps <- kriglist(lpoints, lags = 3, model = "spherical")

hexadf1 <- data.frame(
  x = c(0, 1, 0, 4, 5, 5, 5, 5),
  y = c(1, 1, 4, 4, 1, 1, 4, 4),
  z = c(1, 5, 1, 6, 1, 5, 1, 3)
)

hexadf2 <- data.frame(
  x = c(0, 1, 0, 4, 5, 5, 5, 5),
  y = c(1, 1, 4, 4, 1, 1, 4, 4),
  z = c(-1, -5, -1, -6, -1, -5, -1, -3)
```

```
)

cx1 <- fillhexa(hexadf1, 0.1)
cx2 <- fillhexa(hexadf2, 0.1)

cubelist <- list(cx1, cx2)

posdeclist(cubelist, maps)
```

`rescale` *Scales a point cloud (3D)*

Description

Scales a 3D point cloud on every axis.

Usage

```
rescale(x, y, z, scalex = 1, scaley = 1, scalez = 1)
```

Arguments

`x`	vector of x axis values of scale point cloud
`y`	vector of y axis values of scale point cloud
`z`	vector of z axis values of scale point cloud
`scalex`	scaling factor on x axis (default = 1)
`scaley`	scaling factor on y axis (default = 1)
`scalez`	scaling factor on z axis (default = 1)

Value

data.frame with the spatial coordinates of the resulting points

Examples

```
s <- draw_sphere(1,1,1,3)

#library(rgl)
#plot3d(s)

s2 <- rescale(s$x, s$y, s$z, scalex = 4, scalez = 5)

#library(rgl)
#plot3d(s2)
```

rotate *Rotate a point cloud around a pivot point (3D)*

Description

Rotate a point cloud around a defined pivot point by defined angles. The default rotation angle around each axis is zero and the default pivot point is the center point of the point cloud (defined by mean())

Usage

```
rotate(x, y, z, degrx = 0, degry = 0, degrz = 0, pivotx = NA_real_,
  pivoty = NA_real_, pivotz = NA_real_)
```

Arguments

x	vector of x axis values of rotation point cloud
y	vector of y axis values of rotation point cloud
z	vector of z axis values of rotation point cloud
degrx	rotation angle around x axis in degree (default = 0)
degry	rotation angle around y axis in degree (default = 0)
degrz	rotation angle around z axis in degree (default = 0)
pivotx	x axis value of pivot point (default = mean(x))
pivoty	y axis value of pivot point (default = mean(y))
pivotz	z axis value of pivot point (default = mean(z))

Value

data.frame with the spatial coordinates of the resulting points

Examples

```
circ <- draw_circle(0,0,0,5)

#library(rgl)
#plot3d(
#  circ,
#  xlim = c(-6,6),
#  ylim = c(-6,6),
#  zlim = c(-6,6)
#)

rotcirc <- rotate(circ$x, circ$y, circ$z, degrx = 45)

#plot3d(
#  rotcirc,
```

```
#  xlim = c(-6,6),
#  ylim = c(-6,6),
#  zlim = c(-6,6)
#)
```

---

spatiallong *Transformation of numeric matrices from wide to long format*

Description

spatiallong transforms a set of two independent variables in vectors and a dependent variable in a wide matrix to a long matrix that combines the information. The result is exported as a data.frame.

Usage

```
spatiallong(x, y, z)
```

Arguments

x	vector of first independent variable. e.g. vector with x axis spatial points
y	vector of second independent variable. e.g. vector with y axis spatial points
z	matrix of dependent variable. e.g. matrix with z axis spatial points

Value

data.frame with three columns x, y and z

See Also

Other transfuncs: spatialwide

Examples

```
x <- c(1, 1, 1, 2, 2, 2, 3, 3, 4)
y <- c(1, 2, 3, 1, 2, 3, 1, 2, 3)
z <- c(3, 4, 2, 3, NA, 5, 6, 3, 1)

sw <- spatialwide(x, y, z, digits = 3)

spatiallong(sw$x, sw$y, sw$z)
```

`spatialwide` *Transformation of numeric matrices from long to wide format*

Description

Transforms a set of two independent and one dependent variables in vectors from a long to a wide format and exports this result as a list

Usage

```
spatialwide(x, y, z, digits)
```

Arguments

`x`	vector of first independent variable. e.g. vector with x-axis spatial points
`y`	vector of second independent variable. e.g. vector with y-axis spatial points
`z`	vector of dependent variable. e.g. vector with z-axis spatial points
`digits`	integer indicating the number of decimal places to be used for rounding the dependent variables `x` and `y`.

Value

List with three elements:

$x: vector with ascendingly sorted, unique values of the first independent variable `x`

$y: vector with ascendingly sorted, unique values of the second independent variable `y`

$z: matrix with the values of `z` for the defined combinations of `x` (columns) and `y` (rows)

See Also

Other transfuncs: `spatiallong`

Examples

```
x <- c(1, 1, 1, 2, 2, 2, 3, 3, 4)
y <- c(1, 2, 3, 1, 2, 3, 1, 2, 3)
z <- c(3, 4, 2, 3, NA, 5, 6, 3, 1)

spatialwide(x, y, z, digits = 3)
```

spitcenter *Center determination for hexahedrons*

Description

A hexahedron is a three dimensional shape that is defined by 6 faces and 8 corner points. spitcenter determines a center point for an input hexahedron by calculating the mean of the maximal extent on all three axis.

Usage

```
spitcenter(hex)
```

Arguments

hex	dataframe with three columns and eight rows to define a hexahedron by its corner point coordinates x, y and z

Value

vector with the spatial coordinates of the center point of the input hexahedron

See Also

Other centerdetfuncs: spitcenternatlist, spitcenternat

Examples

```
hexatestdf <- data.frame(
  x = c(0,1,0,4,5,5,5,5),
  y = c(1,1,4,4,1,1,4,4),
  z = c(4,8,4,9,4,8,4,6)
)

center <- spitcenter(hexatestdf)

#library(rgl)
#plot3d(
# hexatestdf$x, hexatestdf$y, hexatestdf$z,
# type = "p",
# xlab = "x", ylab = "y", zlab = "z"
#)
#plot3d(
#  center[1], center[2], center[3],
#  type = "p",
#  col = "red",
#  add = TRUE
#)
```

`spitcenternat` *Center determination for rectangles whose tops and bottoms are defined by irregular surfaces (3D)*

Description

`spitcenternat` first of all calculates the horizontal center of an input rectangle. Then it determines the vertical positions of the center points in relation to a surface stack.

Usage

```
spitcenternat(hex, maplist)
```

Arguments

`hex`	data.frame with the 2D corners of the rectangle defined by four points
`maplist`	list of data.frames which contain the points that make up the surfaces

Value

data.frame with the spatial coordinates of the center points

See Also

Other centerdetfuncs: `spitcenternatlist`, `spitcenter`

Examples

```
df1 <- data.frame(
  x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
  y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
  z = c(0.9+0.05*rnorm(6), 0.9+0.05*rnorm(14), 1.3+0.05*rnorm(14), 1.2+0.05*rnorm(6))
)

df2 <- data.frame(
    x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
    y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
    z = c(0.6+0.05*rnorm(6), 0.6+0.05*rnorm(14), 1.0+0.05*rnorm(14), 0.9+0.05*rnorm(6))
)

df3 <- data.frame(
    x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
    y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
    z = c(0.3+0.05*rnorm(6), 0.3+0.05*rnorm(14), 0.7+0.05*rnorm(14), 0.6+0.05*rnorm(6))
)

lpoints <- list(df1, df2, df3)

maps <- kriglist(lpoints, lags = 3, model = "spherical")
```

```
hexatestdf <- data.frame(
    x = c(1, 1, 1, 1, 2, 2, 2, 2),
    y = c(0, 1, 0, 1, 0, 1, 0, 1)
)

spitcenternat(hexatestdf, maps)
```

`spitcenternatlist` *Center determination for rectangles whose tops and bottoms are defined by irregular surfaces (3D) for multiple data.frames in a list*

Description

`spitcenternatlist` works as `spitcenternat` but not just for a single data.frame but for a list of data.frames

Usage

```
spitcenternatlist(hexlist, maplist)
```

Arguments

`hexlist`	list of data.frames with the 2D corners of the rectangles
`maplist`	list of data.frames which contain the points that make up the surfaces

Value

list of data.frames with the spatial coordinates of the center points

See Also

Other centerdetfuncs: `spitcenternat`, `spitcenter`

Examples

```
df1 <- data.frame(
x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
  y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
  z = c(0.9+0.05*rnorm(6), 0.9+0.05*rnorm(14), 1.3+0.05*rnorm(14), 1.2+0.05*rnorm(6))
)

df2 <- data.frame(
    x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
    y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
    z = c(0.6+0.05*rnorm(6), 0.6+0.05*rnorm(14), 1.0+0.05*rnorm(14), 0.9+0.05*rnorm(6))
)
```

```
df3 <- data.frame(
    x = c(rep(0, 6), seq(0.2, 2.8, 0.2), seq(0.2, 2.8, 0.2), rep(3,6)),
    y = c(seq(0, 1, 0.2), rep(0, 14), rep(1, 14), seq(0, 1, 0.2)),
    z = c(0.3+0.05*rnorm(6), 0.3+0.05*rnorm(14), 0.7+0.05*rnorm(14), 0.6+0.05*rnorm(6))
)

lpoints <- list(df1, df2, df3)

maps <- kriglist(lpoints, lags = 3, model = "spherical")

hexatestdf1 <- data.frame(
  x = c(1, 1, 1, 1, 2, 2, 2, 2),
  y = c(0, 1, 0, 1, 0, 1, 0, 1)
)

hexatestdf2 <- data.frame(
  x = c(0, 0, 0, 0, 1, 1, 1, 1),
  y = c(0, 1, 0, 1, 0, 1, 0, 1)
)

hexs <- list(hexatestdf1, hexatestdf2)

spitcenternatlist(hexs, maps)
```

Index