UNIVERSITAT DE BARCELONA

Exploración y modelización de patrones socioecológicos y tecnoculturales en sociedades preindustriales de zonas áridas afro-eurasiáticas

Una aproximación multidisciplinar desde métodos cuantitativos

Andreas Angourakis

Aquesta tesi doctoral està subjecta a la llicència ***Reconeixement- NoComercial 4.0. Espanya de Creative Commons***.

Esta tesis doctoral está sujeta a la licencia ***Reconocimiento - NoComercial 4.0. España de Creative* Commons**.

This doctoral thesis is licensed under the ***Creative Commons Attribution-NonCommercial 4.0. Spain License***.

Apéndice B: Software

B.1 Análisis de datos arqueométricos

- 'biplot2d3d' package documentation
- 'cerUB' package documentation

B.1.1 biplot2d3d

Este apartado presenta la documentación del 'biplot2d3d', un paquete R desarrollado para crear gráficos de tipo *biplot* altamente customizables. Este paquete se ha utilizado en las publicaciones que integran el capítulo 3. Además de su publicación en Zenodo (Angourakis 2017a), el código fuente se puede encontrar en GitHub (https://github.com/Andros-Spica/biplot2d3d/).

Aquest apartat presenta la documentació del 'biplot2d3d', un paquet R desenvolupat per crear gràfics de tipus *biplot* altament customitzables. Aquest paquet s'ha utilitzat en les publicacions que integren el capítol 3. A més de la publicació en Zenodo (Angourakis 2017a), el codi font es pot trobar a GitHub (https://github.com/Andros-Spica/biplot2d3d/).

This section presents the documentation of the 'biplot2d3d', an R package developed to create graphics of type *biplot* highly customizable. This package has been used in the publications that make up chapter 3. In addition to its publication in Zenodo (Angourakis 2017a), the source code can be found on GitHub (https://github.com/Andros-Spica/biplot2d3d/).

Package 'biplot2d3d'

October 15, 2017

Type Package

Title Create and Customize Ordination Plots (Biplots) in 2D and 3D

Version 1.0.1

Description This package allows the user to generate fully-customized two and three dimensional scatter plots, especially ordination plots or biplots. For example, it can be used for displaying the first two/three principal components of a Principal Components Analysis (points), together with the respective variable loadings (arrows). Mainly, the ade4 and rgl packages are used to built 2d and 3d plots, respectively. The biplots2d3d package centralizes the control over many options available in these packages (and others), organizing them exhastively into parameters of higher level functions, such as biplot_2d and biplot_3d.

License GPL-3

URL https://github.com/Andros-Spica/biplot2d3d

LazyData TRUE

RoxygenNote 6.0.1

Imports ade4, heplots, rgl, extrafont

Suggests knitr, rmarkdown, vegan, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Andreas Angourakis [aut, cre]

Maintainer Andreas Angourakis <andros.spica@gmail.com>

R topics documented:

1

`animation` *Create an animated GIF and a snapshot from the rgl device*

Description

Creates a sequence of images while rotating one or more axes and collapsing them into a GIF file. It uses `movie3d` so you must install ImageMagick (www.imagemagick.org) in order for it to work.

Usage

```
animation(directory = "", file_name = "animation", axis_spin = c(0, 1, 0),
  axis_spin_rpm = 5)
```

Arguments

`directory`	The directory within the working directory. For example, "MyFolder/".
`file_name`	The name of the output file.
`axis_spin`	a numeric vector of lenght 3 indicating around which axis should the plot spin. See `movie3d`.
`axis_spin_rpm`	the velocity of the spin. See `movie3d`.

`biplot_2d` *2D biplot*

Description

Generates a 2D biplot representing the default points/scores and loadings of an ordination object, such as a PCA produced by `princomp`.

Usage

```
biplot_2d(ordination_object, ordination_method = "PCA",
  biplot_type = "default", rows_over_columns = 0.5, groups = NULL,
  vips = NULL, detach_arrows = TRUE, show_grid = TRUE, show_axes = TRUE,
  show_group_legend = FALSE, show_vip_legend = TRUE, show_arrows = TRUE,
  show_fitAnalysis = TRUE, main_lwd = 2, grid_cex = 1, grid_font = 3,
  grid_adj = 0.5, invert_coordinates = c(FALSE, FALSE), xlim = NULL,
  ylim = NULL, x_title = "", y_title = "", x_title_cex = 1,
  x_title_font = 2, y_title_cex = 1, y_title_font = 2, subtitle = NULL,
  subtitle_position = "bottomleft", subtitle_cex = 1.2,
  point_type = "point", point_pch = 1, point_size = 1,
  point_label = NULL, point_label_cex = 1, point_label_font = 3,
```

```
  point_label_adj = c(0.5, 0.5), point_label_adj_override = NULL,
  arrow_color = "darkorange", arrow_mim_dist = 0, arrow_length = 0.2,
  arrow_cex = 0.1, arrow_lwd = 2, arrow_label_cex = 1,
  arrow_label_color = "black", arrow_label_font = 1,
  arrow_label_adj = c(0.5, 0.5), arrow_label_adj_override = NULL,
  group_color = "black", group_star_cex = 1, group_ellipse_cex = 1,
  group_ellipse_axes = FALSE, group_label_cex = 1,
  group_legend_title = "groups", group_legend_title_pos = c(0.5, 0.85),
  group_legend_title_cex = 1, group_legend_title_font = 3,
  group_legend_title_adj = 0.5, group_legend_box_color = "white",
  group_legend_key_pch = 15, group_legend_key_cex = 1,
  group_legend_key_lwd = 1, group_legend_key_margin = 0.15,
  group_legend_text_margin = 0.25, group_legend_text_color = "black",
  group_legend_text_cex = 1, group_legend_text_font = 1,
  group_legend_text_adj = 0, vip_color = "black", vip_pch = c(0, 1, 5, 2,
  6, 4, 3), vip_cex = c(5, 5, 5, 5, 5, 3, 3), vip_lwd = 5, vip_font = 1,
  vip_adj = c(0.5, 0.5), vip_legend_title = "VIPs",
  vip_legend_title_pos = c(0.5, 0.85), vip_legend_title_cex = 1,
  vip_legend_title_font = 3, vip_legend_title_adj = 0.5,
  vip_legend_box_color = "white", vip_legend_key_cex = 0.8,
  vip_legend_key_margin = 0.15, vip_legend_text_cex = 1,
  vip_legend_text_font = 1, vip_legend_text_adj = 0,
  vip_legend_text_margin = 0.25, fitAnalysis_lwd = 3,
  fitAnalysis_screePlot_color = c("grey", "white"),
  fitAnalysis_stress_cex = 1, fitAnalysis_stress_lab_cex = 1,
  fitAnalysis_stress_axis_cex = 1, fitAnalysis_stress_p_color = "darkgrey",
  fitAnalysis_stress_l_color = "black", test_text = NULL,
  test_spacing_paragraph = 0.8, test_spacing_line = 0.8, test_cex = 1,
  test_font = 1, test_adj = 0.5, fit_into_main = FALSE, main_fig = c(0,
  1, 0, 1), group_legend_fig = c(0.8, 0.99, 0.6, 0.9),
  vip_legend_fig = c(0.78, 0.99, 0.3, 0.55), arrow_fig = c(0.69, 0.99, 0.01,
  0.31), fitAnalysis_fig = c(0.02, 0.35, 0.06, 0.25), test_fig = c(0, 0.3,
  0.8, 0.99), x_title_fig = c(0.25, 1, 0.85, 1), y_title_fig = c(0.91, 1, 0,
  1), output_type = c("preview", "png"), open_new_device = TRUE,
  leave_device_open = FALSE, directory = "", file_name = "2D Biplot",
  width = 400, height = 400, family = "sans")
```

Arguments

`ordination_object`
: A R object containing a direct and named reference to default ordination outputs (i.e. `ordination_object$scores` or `ordination_object$points`, `ordination_object$loadings` and `ordination_object$sdev`) available for at least 2 dimensions. Alternatively, a data frame or matrix with at least two columns is accepted provided that `ordination_method = NULL`), which creates a two-dimensional scatter plot.

`ordination_method`
: Character, the ordination method that was used to generate the ordination object: "PCA" for Principal Components Analysis (default), "PCoA" for Principal Coordinates Analysis, "NMDS" for Non-metric Multidimensional Scaling, and "LDA" for Linear Discriminant Analysis.

`biplot_type`
: Character, indicating the type of biplot transformation of data: "default" and "pc.biplot", corresponding to the transformations performed in `biplot.princomp`

with pc.biplot = FALSE ("default") and pc.biplot = TRUE ("pc.biplot"). If NULL, no processing is performed, assuming that data within ordination_object was previously transformed.

rows_over_columns
: Numeric, the value defining the degree in which distances between observations have priority over distances between variables (0 = observation-focused, 1 = variable-focused). It corresponds to the argument scale in biplot.princomp. It will be ignored if biplot_type = NULL. The default is set at 0.5, which corresponds to a 'SQRT-Biplot', a compromise between the two representations.

groups
: A factor variable containing the group assignation of each point.

vips
: A list of logical (Boolean) vectors identifying "Very Important Points" under different methods or criteria.

detach_arrows
: Logical, wheter to display covariance arrows a independent miniature plot, overlapping the main plot and placed according to arrow_fig.

show_grid
: Logical, wheter to display the background grid.

show_axes
: Logical, wheter to display the axes.

show_group_legend
: Logical, whether to display a legend for groups.

show_vip_legend
: Logical, whether to display a legend for vip criteria.

show_arrows
: Logical, whether to show variable covariance arrows.

show_fitAnalysis
: Logical, whether to display the fit analysis plot corresponding to the ordination method given (Scree plot or Shepard plot).

main_lwd
: The line width to be used in the main plot. (lwd in (par).

grid_cex, grid_font, grid_adj
: The scale, font type, and text justification of the grid notation.

invert_coordinates
: Logical, vector of length two expressing which dimensions, if any, must be inverted before plotting (e.g. for aesthetical reasons).

xlim, ylim
: the ranges to be encompassed by the x and y axes, if NULL they are computed (See s.class).

x_title, y_title
: Character, texts to be placed in the x and y axes.

x_title_cex, x_title_font
: The text parameters of the x axis or horizontal title (par).

y_title_cex, y_title_font
: The text parameters of the y axis or vertical title (par).

subtitle
: Character, a subtitle to be displayed in the bottom-left corner of the plot (csub in s.class). If equals NULL and ordination_method = "PCA", a default subtitle is the R-Squared of the 2D fit respect to the original distances.

subtitle_position
: Character, value indicating the position of the subtitle in the main plot ("topleft", "topright", "bottomleft", and "bottomright"; see s.class).

subtitle_cex
: the font size of the subtitle (csub in s.class).

point_type
: A character input accepting three values: "point", "label" (displaying the content of point_label) and "point and label", placing both points and labels.

`point_pch` A number or a numerical vector given to `pch` in `par`.

`point_size` The size or scale given to `cpoint` in `s.class`.

`point_label` A character vector labelling every observation. It's length must be equal to the number of rows in the points/scores of the ordination object (`nrow(ordination_object$points) == length(point_label)`).

`point_label_cex, point_label_font, point_label_adj`
The text parameters of the points' labels (`text`).

`point_label_adj_override`
A data frame with x,y values to be passed to `adj`, overrinding the default value given in `point_label_adj`. The rows must be named exactly as points are referred in the ordination object.

`arrow_color` The color or colors to be used in covariance arrows.

`arrow_mim_dist` Numeric, the minimum distance of a arrow from the origin of arrows (i.e. zero covariance), in order for it to be displayed (range [0 = all arrows are displayed,1 = no arrow is displayed]).

`arrow_length` Numeric, the scalar factor applied to loadings to resize them respect to scores. If `detach_arrows = TRUE`, resizing is controlled with `arrow_fig`, so this value is ignored.

`arrow_cex, arrow_lwd, arrow_label_cex, arrow_label_color, arrow_label_font, arrow_label_adj`
Graphical parameters of the covariance arrows and their labels (see `arrows` and `par`). `arrow_cex` is actually equivalent to `length` in `arrows`.

`arrow_label_adj_override`
A data frame with x,y values to be passed to `adj`, overrinding the default value given in `arrow_label_adj`. The rows must be named exactly as variables are referred in the ordination object.

`group_color` A vector containing the colors to be used in each group (applied to points, labels, stars and ellipses). If NULL, automatically assign different colors from the `rainbow()` palette.

`group_star_cex, group_ellipse_cex, group_label_cex`
The size or scale of the stars, ellipses, and labels representing groups passed to `s.class` of the `ade4` package. Zero values render these elements invisible.

`group_ellipse_axes`
Logical, wheter to show the ellipses axes, passed to `axesell` in `s.class` of the `ade4` package.

`group_legend_title`
Character, the title of the group legend. If equal NULL or "", no title is displayed.

`group_legend_title_pos`
A numeric vector of length two, the xy position of the title within the group legend box.

`group_legend_title_cex, group_legend_title_font, group_legend_title_adj`
The text parameters to be applied in the group legend title (`par`).

`group_legend_box_color`
The background color of the group legend box.

`group_legend_key_pch, group_legend_key_cex, group_legend_key_lwd`
The type, size and line width of the keys in the group legend.

`group_legend_key_margin`
The x position of the keys within the group legend box. Values from 0 to 1.

`group_legend_text_margin`
: The x position of the text entries in the group legend. Values from 0 to 1.

`group_legend_text_color`
: The color or colors of the text entries in the group legend.

`group_legend_text_cex, group_legend_text_font, group_legend_text_adj`
: The text parameters of the text entries in the group legend.

`vip_color`
: The color or colors to be used in vip markings.

`vip_pch`
: A character vector containing the characters used for the vips markings under each criterion.

`vip_cex, vip_lwd, vip_font, vip_adj`
: The graphical parameters of the vips markings.

`vip_legend_title`
: Character, the title of the vips legend.

`vip_legend_title_pos`
: A numeric vector of length two, the xy position of the title within the vips legend box.

`vip_legend_title_cex, vip_legend_title_font, vip_legend_title_adj`
: The text parameters to be applied in the vips legend title (`par`).

`vip_legend_box_color`
: The background color of the vips legend box.

`vip_legend_key_cex`
: Numeric, the sizing factor of the keys in the vips legend respect to the vips marking in the plot.

`vip_legend_key_margin`
: Numeric, the x position of the keys within the vips legend box. Values from 0 to 1.

`vip_legend_text_cex, vip_legend_text_font, vip_legend_text_adj`
: The text parameters of the text entries in the vips legend.

`vip_legend_text_margin`
: Numeric, the x position of the text entries in the vips legend box. Values from 0 to 1.

`fitAnalysis_lwd, fitAnalysis_screePlot_color, fitAnalysis_stress_cex, fitAnalysis_stress_lab_cex`
: The graphical parameters of the plot for fit analysis correspondint to the ordination method (Scree plot for PCA, PCoA, LDA; Shepard or Stress plot for NMDS). (`par`, `stressplot` of the vegan package).

`test_text`
: A list of character vectors or expressions with the lines of text presenting the results of statistical tests. A example structure would be: `list(c("first line", "second line"), "s`

`test_spacing_paragraph, test_spacing_line`
: Numeric, relative spacing between paragraphs (list elements) and lines (character elements within a list element, if more than one).

`test_cex, test_font, test_adj`
: The parameters of the text with the test results (`par`).

`fit_into_main`
: Logical, wheter to fit all elements into the main plot. If TRUE, the 'fig' parameter of every element is interpreted as relative to `main_fig`.

`main_fig, group_legend_fig, vip_legend_fig, arrow_fig, fitAnalysis_fig, test_fig, x_title_fig, y`
: The `fig` parameter (`par`) to place in the display region of the graphics device, respectively, the main plot, the group and vip legends, the fit analysis plot, the tests results, and the x and y axes titles.

`output_type` A character vector indicating the output image types to be generated. Values accepted are: "png", "tiff", "jpeg", "eps", and "preview" (i.e. R graphics device).

`open_new_device` Logical, wheter to build the plot in a new graphics device. If FALSE, the biplot is draw in the current device.

`leave_device_open` Logical, wheter to leave the graphics device open, e.g. to continue the plot adding external elements. TRUE is only accepted if `output_type` has a single value. If `output_type = c("preview")`, the device is left open by default.

`directory` Character, the directory within the working directory. For example, "MyFolder/".

`file_name` Character, the name of the output file.

`width, height` Numeric, the dimensions of the output image.

`family` Character, the font family used in every text in the plot, (`par`).

Details

This function allows customising virtually every graphical parameter in a 2D biplot, including several extra elements that may be useful for multivariate explorations. It is focused mainly on improving basic visualization aspects of ordination methods through 'classical' biplots. There are several packages that address the creation of other variations of biplot: BiplotGUI, GGEBiplotGUI, multibiplotGUI, biplotbootGUI, NominalLogisticBiplot, OrdinalLogisticBiplot, ade4, vegan, MultBiplotR. When `biplot_type = "default"`, the biplot processing is done as in `biplot.princomp`, which follows the definition of Gabriel (1971). As in this method, when `biplot_type = "pc.biplot"`, this function creates biplots according with Gabriel and Odoroff (1990). Since there are several types of biplot transformations, it is possible to use 'scores' and 'loadings' that were already transformed, passing `biplot_type = NULL`. Groups can be represented as stars, ellipsoids, and/or colors, using (`s.class`), which can be tracked by a fully-customisable legend (`group_legend` arguments). Individual observations deemed exceptional (vip = Very Important Points) can be marked with custom symbols. Whenever is are more than one type of marking (e.g. different methods/criteria of outlier detection), the different symbols can be presented in a legend (`vip_legend` arguments). When desired, it is possible to display a Scree plot representing the eigenvalues (Principal Components Analysis, Principal Coordinates Analysis) or a Shepard or stress plot (Nonmetric Multidimensional Scaling, e.g. `metaMDS` in the vegan package) by enabling `show_fit_analysis`. It is also possible to display statistical test results by enabling `show.tests` and introducing the corresponding lines in `test_text`. The position of all elements (legend boxes, title and subtitle, fit analysis plot and tests) can be customized using the corresponding `fig` parameter.

Examples

```
## Not run:

# Use iris data
data("iris")

# get an ordination object
# ("PCA" is the default input of this function)
pca <- princomp(iris[, 1:4])

# Default plot using Species as the group factor
biplot_2d(pca, groups = iris$Species)

# Use the typical visualization,
```

```
# placing scores and loadings around the same origin
biplot_2d(pca, groups = iris$Species, detach_arrows = FALSE)

# Compare different versions of the classical biplot
# "default" vs. "pc.biplot"
biplot_2d(pca,
          output_type = "preview",
          leave_device_open = TRUE,
          x_title = 'biplot_type = "default"',
          x_title_fig = c(0, 1, 0.9, 1),
          fit_into_main = TRUE,
          main_fig = c(0, 0.499, 0, 1))
biplot_2d(pca,
          output_type = "preview",
          open_new_device = FALSE,
          biplot_type = "pc.biplot",
          x_title = 'biplot_type = "pc.biplot"',
          x_title_fig = c(0, 1, 0.9, 1),
          fit_into_main = TRUE,
          main_fig = c(0.5099, 1, 0, 1))

# varying focus on representing distances
# between observations or between variables
biplot_2d(pca,
output_type = "png",
leave_device_open = TRUE,
rows_over_columns = 1,
x_title = 'observation-focused\nrows_over_columns = 1',
x_title_fig = c(0, 1, 0.9, 1),
fit_into_main = TRUE,
main_fig = c(0, 0.329, 0, 1),
width = 1200)
biplot_2d(pca,
          open_new_device = FALSE,
          leave_device_open = TRUE,
          rows_over_columns = 0.5,
          x_title = 'compromise\nrows_over_columns = 0.5',
          x_title_fig = c(0, 1, 0.9, 1),
          fit_into_main = TRUE,
          main_fig = c(0.331, 0.659, 0, 1))
biplot_2d(pca,
          open_new_device = FALSE,
          rows_over_columns = 0,
          biplot_type = "pc.biplot",
          x_title = 'variable-focused\nrows_over_columns = 0',
          x_title_fig = c(0, 1, 0.9, 1),
          fit_into_main = TRUE,
          main_fig = c(0.661, 1, 0, 1))

# ---------------------------------------------------------
# Plot groups as different colors and point types (pch),
# make group star, ellipsis, and label invisible and
# add a group legend with a a custom title.
biplot_2d(pca,
          groups = iris$Species,
          group_color = NULL,
          point_pch = c(1, 3, 2),
```

```
          group_star_cex = 0,
          group_ellipse_cex = 0,
          group_label_cex = 0,
          show_group_legend = T,
          group_legend_title = "Species")

# ------------------------------------------------------------
# Polish covariance arrows
# Abbreviate variables names
dimnames(pca$loadings)[[1]] <- c("SL", "SW", "PL", "PW")
# Set a specific justification (adj) for each variable label
arrow_label_adj_override <- rbind(c(-0.1, 0),
                                  c(-0.1, 0.5),
                                  c(0.5, 1.3),
                                  c(0.5, 1.3))
row.names(arrow_label_adj_override) <-
   dimnames(pca$loadings)[[1]]
# Plot: arrows with different colors and
# without the background grid
biplot_2d(pca,
          groups = iris$Species,
          point_pch = c(1, 3, 2),
          group_star_cex = 0,
          group_ellipse_cex = 0,
          group_label_cex = 0,
          show_group_legend = T,
          group_legend_title = "Species",
          arrow_color = c("orange",
                          "blue",
                          "red",
                          "green"),
          arrow_label_adj_override = arrow_label_adj_override,
          show_grid = FALSE)

# ------------------------------------------------------------
# Get arbitrary Very Important Points
irisVIP <- list(setosa = (1:nrow(iris) == 16 |
                          1:nrow(iris) == 42),
                versicolor=(1:nrow(iris) == 61),
                virginica=(1:nrow(iris) == 107 |
                           1:nrow(iris) == 118 |
                           1:nrow(iris) == 132))

# Plot observations using their names and group by Species using only color.
# Mark the VIP and add the respective legend with custom characters.
biplot_2d(pca,
          groups = iris$Species,
          point_type = "label",
          point_label = row.names(iris),
          group_color = c("red", "blue", "green"),
          group_star_cex = 0,
          group_ellipse_cex = 0,
          group_label_cex = 0,
          show_group_legend = TRUE,
          group_legend_title = "",
          vips = irisVIP,
          vip_pch = c("X", "O", "+"),
```

```
          vip_cex = c(2, 2, 3),
          vip_legend_fig = c(0.01, 0.25, 0.7, 0.99),
          show_axes = FALSE)

# ------------------------------------------------------------
# Test the setosa separation
irisDist <- dist(iris[, 1:4])
setosaSeparation <- iris$Species == "setosa"
## multivariate test for the setosa separation
require(vegan)
irisTests <- NULL
irisTests$permanova <- adonis(irisDist~setosaSeparation)
irisTests$permdisp2 <- permutest(betadisper(irisDist,setosaSeparation),
                                 pairwise = TRUE)
# This function prepares a list of character vectors containing the test results
getTestText <- function(tests){
  permanova_F <- as.character(round(tests$permanova$aov.tab$F.Model[1], 3))
  permanova_pvalue <- as.character(round(tests$permanova$aov.tab$"Pr(>F)"[1], 3))
  permanova_rSquared <- as.character(round(tests$permanova$aov.tab$R2[1], 3))
  permdisp2_F <- as.character(round(tests$permdisp2$tab$F[1], 3))
  permdisp2_pvalue <- as.character(round(tests$permdisp2$tab$"Pr(>F)"[1], 3))
  text <- list(c(paste("PERMANOVA:\n   F = ", permanova_F,
                       " (p = ", permanova_pvalue, ")\n", sep = ""),
                 c(expression(paste("   ", R^2, " =", sep = "")),
                   paste("          ", permanova_rSquared, sep = ""))),
               paste("PERMDISP2:\n   F = ", permdisp2_F,
                     " (p = ", permdisp2_pvalue,")", sep = ""))
  return(text)
}

# Plot observations using points and groups as colored stars with no labels.
# Place tests results in the top left corner and give a custom horizontal title.
biplot_2d(pca,
          groups = iris$Species,
          group_color = NULL,
          group_ellipse_cex = 0,
          group_label_cex = 0,
          show_group_legend = TRUE,
          group_legend_title = "",
          test_text = resultText(irisTests),
          test_fig = c(0.01,0.5,0.6,0.95),
          show_axes = FALSE,
          x_title = "testing setosa separation",
          x_title_cex = 1.5)


## End(Not run)
```

`biplot_3d` *3D biplot*

Description

Generates a 3D biplot using a rgl device, representing the default points/scores and loadings of an ordination object, such as a PCA produced by `princomp`).

Usage

```
biplot_3d(ordination_object, ordination_method = "PCA",
  biplot_type = "default", rows_over_columns = 0.5, groups = NULL,
  vips = NULL, detach_arrows = TRUE, show_group_legend = FALSE,
  show_vip_legend = FALSE, show_arrows = TRUE, show_fitAnalysis = TRUE,
  show_axes = c("X", "Y", "Z"), show_planes = NULL, show_bbox = FALSE,
  invert_coordinates = c(FALSE, FALSE, FALSE), aspect = c(1, 1, 1),
  symmetric_axes = FALSE, adapt_axes_origin = TRUE,
  axes_colors = "darkgrey", axes_head_size = 3, axes_titles = "",
  axes_titles_cex = 2, axes_titles_font = 2, axes_titles_adj = list(x =
  c(-0.1, 0), y = c(0.5, -0.5), z = c(1.1, 0)), axes_titles_alpha = 1,
  planes_colors = c("lightgrey", "lightgrey", "lightgrey"),
  planes_textures = NULL, planes_alpha = 0.5, planes_lit = TRUE,
  planes_shininess = 50, bbox_color = c("#333377", "black"),
  bbox_alpha = 0.5, bbox_shininess = 5, bbox_xat = NULL,
  bbox_xlab = NULL, bbox_xunit = 0, bbox_xlen = 3, bbox_yat = NULL,
  bbox_ylab = NULL, bbox_yunit = 0, bbox_ylen = 3, bbox_zat = NULL,
  bbox_zlab = NULL, bbox_zunit = 0, bbox_zlen = 3, bbox_marklen = 15,
  bbox_marklen_rel = TRUE, bbox_expand = 1, bbox_draw_front = FALSE,
  title = "", title_color = "black", title_line = -2, title_size = 3,
  title_font = 20, title_adj = 0.5, subtitle = NULL,
  subtitle_color = "black", subtitle_position = c(0.03, 0.005),
  subtitle_cex = 2, subtitle_font = 2, subtitle_adj = 0,
  point_type = "point", point_label = NULL, point_size = 5,
  point_alpha = 1, point_label_cex = 0.8, point_label_font = 3,
  point_label_adj = c(0.5, 0.5), point_label_alpha = 1,
  group_color = NULL, group_representation = NULL,
  ellipsoid_type = "wire and shade", ellipsoid_level = 0.95,
  ellipsoid_singleton_color = NULL, ellipsoid_singleton_radius = 0.1,
  ellipsoid_wire_alpha = 0.2, ellipsoid_wire_lit = FALSE,
  ellipsoid_shade_alpha = 0.1, ellipsoid_shade_lit = FALSE,
  ellipsoid_label_cex = 1, ellipsoid_label_font = 2,
  ellipsoid_label_adj = c(-0.25, 0.5), ellipsoid_label_alpha = 1,
  star_centroid_radius = 0.005, star_centroid_alpha = 0.5,
  star_link_width = 1, star_link_alpha = 1, star_label_cex = 1,
  star_label_font = 2, star_label_adj = c(-0.25, 0.5),
  star_label_alpha = 1, group_legend_title = "Groups",
  group_legend_title_pos = c(0.5, 0.85), group_legend_title_cex = 2,
  group_legend_title_font = 3, group_legend_title_adj = 0.5,
  group_legend_box_color = "white", group_legend_key_pch = 15,
  group_legend_key_lwd = 1, group_legend_key_margin = 0.15,
  group_legend_key_cex = 3, group_legend_text_margin = 0.25,
  group_legend_text_color = "black", group_legend_text_cex = 1.5,
  group_legend_text_font = 1, group_legend_text_adj = 0, vip_pch = c("/",
  "\\", "o", "_", "|", "O"), vip_cex = c(2, 2, 2, 2, 2, 2),
  vip_colors = "black", vip_font = 3, vip_adj = c(0.5, 0.5),
  vip_alpha = 0.8, vip_legend_title = "Outliers",
  vip_legend_title_pos = c(0.5, 0.85), vip_legend_title_cex = 2,
  vip_legend_title_font = 3, vip_legend_title_adj = 0.5,
  vip_legend_box_color = "white", vip_legend_key_margin = 0.15,
  vip_legend_key_cexFactor = 0.8, vip_legend_text_margin = 0.25,
  vip_legend_text_cex = 1, vip_legend_text_font = 1,
```

```
vip_legend_text_adj = 0, arrow_color = "darkorange", arrow_min_dist = 0,
arrow_center_pos = c(1, 0, 1), arrow_head_shape_theta = pi/6,
arrow_head_shape_n = 3, arrow_head_size = 0.1, arrow_body_length = 0.2,
arrow_body_width = 1, arrow_label_color = "black",
arrow_label_cex = 0.8, arrow_label_font = 2, arrow_label_adj = 0.5,
arrow_label_alpha = 1, fitAnalysis_cex = 3, fitAnalysis_lwd = 3,
fitAnalysis_screePlot_color = "white",
fitAnalysis_stress_p_color = "darkgrey",
fitAnalysis_stress_l_color = "black", test_text = NULL,
test_spacing_paragraph = 0.8, test_spacing_line = 0.8, test_cex = 1,
test_font = 1, test_adj = 0.5, group_legend_fig = c(0.73, 0.99, 0.6,
0.9), vip_legend_fig = c(0.03, 0.25, 0.1, 0.3), fitAnalysis_fig = c(0.02,
0.35, 0.08, 0.3), test_fig = c(0, 0.3, 0.8, 0.99), new_device = FALSE,
bg_color = "white", view_theta = 15, view_phi = 20, view_fov = 60,
view_zoom = 0.8, width = 800, height = 600, family = "sans")
```

Arguments

`ordination_object`
: A R object containing a direct and named reference to default ordination outputs (i.e. `ordination_object$scores` or `ordination_object$points`, `ordination_object$loadings` and `ordination_object$sdev`) available for at least 3 dimensions. Alternatively, a data frame or matrix with three columns is accepted, provided that `ordination_method = NULL`), which will create a three-dimensional scatter plot.

`ordination_method`
: Character, the ordination method that was used to generate the ordination object: "PCA" for Principal Components Analysis (default), "PCoA" for Principal Coordinates Analysis, "NMDS" for Non-metric Multidimensional Scaling, and "LDA" for Linear Discriminant Analysis.

`biplot_type`
: Character, indicating the type of biplot scalling of data: "default" and "pc.biplot", corresponding to the transformations performed in `biplot.princomp` with `pc.biplot = FALSE` ("default") and `pc.biplot = TRUE` ("pc.biplot"). If NULL, no processing is performed, assuming that data within `ordination_object` was previously prepared.

`rows_over_columns`
: Numeric, the value defining the degree in which distances between observations have priority over distances between variables (0 = variable-focused, 1 = observation-focused). It corresponds to the argument `scale` in `biplot.princomp`. It will be ignored if `biplot_type = NULL`.

`groups`
: A factor variable containing the group assignation of each point.

`vips`
: A list of logical (Boolean) vectors identifying the "Very Important Points" under different methods or criteria.

`detach_arrows`
: Logical, wheter to display covariance arrows a independent miniature plot, overlapping the main plot and placed according to `arrow_fig`.

`show_group_legend`
: Logical, whether to display a legend for groups?

`show_vip_legend`
: Logical, whether to display a legend for vip criteria?

`show_arrows`
: Logical, whether to show variable covariance arrows.

`show_fitAnalysis`
: Logical, whether to display the fit analysis plot corresponding to the ordination method given (Scree plot or Shepard plot).

`show_axes, show_planes`
: Character, vectors indicating which axes and planes to draw (See `rgl_format`).

`show_bbox`
: Logical, wheter to display a bounding box (See `rgl_format`).

`invert_coordinates`
: Logical, vector of length three expressing which dimensions, if any, must be inverted before plotting (e.g. for aesthetical reasons).

`aspect, symmetric_axes, adapt_axes_origin, axes_colors, axes_head_size, axes_titles, axes_titles`
: The arguments passed to `rgl_format` to configure the space and create axes.

`planes_colors, planes_textures, planes_alpha, planes_lit, planes_shininess`
: The arguments passed to `rgl_format` to create dimensional planes.

`bbox_color, bbox_alpha, bbox_shininess, bbox_xat, bbox_xlab, bbox_xunit, bbox_xlen, bbox_yat, bb`
: The arguments passed to `rgl_format` to create a bounding box.

`title`
: Character, title to be placed in the fixed 2D canvas ('main' in `title`).

`title_line, title_color, title_size, title_font, title_adj`
: the line, color, size, font, and justification of the title (`line`, `col.main`, `cex.main`, `font.main`, and `adj` in `title`, `par`).

`subtitle`
: Character, subtitle to be placed in the fixed 2D canvas ('main' in `text`).

`subtitle_color, subtitle_cex, subtitle_font, subtitle_adj`
: the color, size, font, and justification of the subtitle ('col', 'cex', font and adj in `text`, `par`).

`subtitle_position`
: Numeric verctor of length two indicating the position of the subtitle in the fixed 2D canvas ('main' in `text`).

`point_type`
: Character, accepting three values: "point", the default `points3d`; "label", displaying the content of `point_label`; and "point and label", placing both points and labels.

`point_label`
: Character, vector labelling every observation. It's length must be equal to the number of rows in the points/scores of the ordination object. (`nrow(ordination_object$points) = length(point_label)`).

`point_size`
: The size or scale given to `size` in `points3d`

`point_alpha`
: The alpha of points given to `alpha` in `points3d`

`point_label_cex, point_label_font, point_label_adj, point_label_alpha`
: The text parameters and the alpha of the arrows' labels. (`text3d`, `rgl.material`).

`group_color`
: A vector containing the color or colors to be used in each group (applied to points, labels, stars and ellipsoids).

`group_representation`
: Character, indicating which group representation to draw: "stars", "ellipsoids", or "stars and ellipsoids". Neither stars or ellipsoids are drawn, if NULL is given instead.

`ellipsoid_type, ellipsoid_level, ellipsoid_singleton_color, ellipsoid_singleton_radius, ellipsoi`
: When ellipsoids are drawn, parameters given to `ellipsoids_3d`.

`star_centroid_radius, star_centroid_alpha, star_link_width, star_link_alpha, star_label_cex, sta`
: When stars are drawn, parameters given to `stars_3d`.

`group_legend_title`
: Character, the title of the groups legend. If equal NULL or "", no title is displayed.

group_legend_title_pos
: A numeric vector of length two, the xy position of the title within the groups legend box.

group_legend_title_cex, group_legend_title_font, group_legend_title_adj
: The text parameters to be applied in the groups legend title (par).

group_legend_box_color
: The background color of the groups legend box.

group_legend_key_pch, group_legend_key_lwd, group_legend_key_cex
: The type, line width and sizing factor of the keys in the groups legend.

group_legend_key_margin
: The x position of the keys within the groups legend box. Values from 0 to 1.

group_legend_text_margin
: The x position of the text entries in the groups legend. Values from 0 to 1.

group_legend_text_color
: The color or colors of the text entries in the groups legend.

group_legend_text_cex, group_legend_text_font, group_legend_text_adj
: The text parameters of the text entries in the groups legend.

vip_pch
: A character vector containing the characters used for the vips markings under each criterion.

vip_cex, vip_colors, vip_font, vip_adj, vip_alpha
: The graphical parameters of the vips markings.

vip_legend_title
: Character, the title of the vips legend.

vip_legend_title_pos
: A numeric vector of length two, the xy position of the title within the vips legend box.

vip_legend_title_cex, vip_legend_title_font, vip_legend_title_adj
: The text parameters to be applied in the vips legend title (par).

vip_legend_box_color
: The background color of the vips legend box.

vip_legend_key_margin
: The x position of the keys within the vips legend box. Values from 0 to 1.

vip_legend_key_cexFactor
: The sizing factor of the keys in the vips legend respect to the vips marking in the plot.

vip_legend_text_margin
: The x position of the text entries in the vips legend box. Values from 0 to 1.

vip_legend_text_cex, vip_legend_text_font, vip_legend_text_adj
: The text parameters of the text entries in the vips legend.

arrow_color
: The color or colors to be used in covariance arrows (pass to radial_arrows_3d).

arrow_min_dist
: The minimum distance of a variable arrow from the origin of arrows, in order for it to be displayed (range [0 = all arrows are displayed, 1 = no arrows are displayed]).

arrow_center_pos
: A numeric vector of length three, containing the position of the origin of covariance arrows, expressed in relation to the 3D space represented (e.g. c(.5, .5, .5) will place the arrows in the center).

arrow_head_shape_theta, arrow_head_shape_n, arrow_head_size, arrow_body_width, arrow_body_length
: When the covariance arrows are displayed, parameters given to radial_arrows_3d.

`fitAnalysis_cex, fitAnalysis_lwd, fitAnalysis_screePlot_color, fitAnalysis_stress_p_color, fitAn`
: The graphical parameters of the plot for fit analysis of the ordination method (`par`, `stressplot` of the vegan package).

`test_text`
: A list of character vectors or expressions with the lines of text presenting the results of statistical tests. A example structure would be: `list(c("first line", "second line"), "s`

`test_spacing_paragraph, test_spacing_line`
: Numeric, relative spacing between paragraphs (list elements) and lines (character elements within a list element, if more than one).

`test_cex, test_font, test_adj`
: The parameters of the text with the test results (`par`).

`group_legend_fig, vip_legend_fig, fitAnalysis_fig, test_fig`
: The `fig` parameter (`par`) to place in the display region of the graphics device, respectively, the group and vip legends, the fit analysis plot, and the tests results.

`new_device, bg_color, view_theta, view_phi, view_fov, view_zoom, width, height`
: The arguments passed to `rgl_init`.

`family`
: The font family used in every text in the plot, (`par`).

Details

This function allows customising virtually every graphical parameter in a 3D biplot, including several extra elements that may be useful for multivariate explorations. It is focused mainly on improving basic visualization aspects of ordination methods through 'classical' biplots. There are several packages that address the creation of other variations of biplot: BiplotGUI, GGEBiplotGUI, multibiplotGUI, biplotbootGUI, NominalLogisticBiplot, OrdinalLogisticBiplot, ade4, vegan, MultBiplotR. When `biplot_type = "default"`, the biplot processing is done as in `biplot.princomp`, which follows the definition of Gabriel (1971). As in this method, when `biplot_type = "pc.biplot"`, this function creates biplots according with Gabriel and Odoroff (1990). Since there are several types of biplot transformations, it is possible to use 'scores' and 'loadings' that were already transformed, passing `biplot_type = NULL`. Groups can be represented as stars (`stars_3d`), ellipsoids (`ellipsoids_3d`), and/or colors, which can be tracked by a fully-customisable legend (`group_legend` arguments). Individual observations deemed exceptional (vip = Very Important Points) can be marked with custom characters. Whenever there are more than one type of marking (e.g. different methods/criteria of outlier detection), different characters can be presented in a legend (vip legend). When desired, it is possible to display a Scree plot representing the eigenvalues (Principal Components Analysis, Principal Coordinates Analysis) or a Shepard or stress plot (Nonmetric Multidimensional Scaling, `metaMDS` in the vegan package) by enabling `show_fit_analysis`. It is also possible to display statistical test results (enabling `show.tests` and introducing lines of text in `tests.text`). Every 2D element (legend boxes, title and subtitle, fit analysis plot and tests) are placed in a fixed 2D canvas (i.e. viewport) using `bgplot3d`.

References

Gabriel, K. R. (1971). The biplot graphical display of matrices with applications to principal component analysis. Biometrika, 58, 453-467.

Examples

```
## Not run:

# Use iris data
data("iris")
```

```
# get an ordination object
# ("PCA" is the default input of this function)
pca <- princomp(iris[, 1:4])

# Default plot using Species as the groups
biplot_3d(pca, groups = iris$Species)

# ---------------------------------------------------------
# Plot groups as ellipsoids, make group label invisible and
# add a groups legend with no title.
# Customize the covariance arrows default setting.
biplot_3d(pca,
          groups = iris$Species,
          group_representation = "ellipsoids",
          ellipsoid_label_alpha = 0,
          show_group_legend = TRUE,
          group_legend_title = "",
          arrow_center_pos = c(.5, 0, .5),
          arrow_body_length = 1,
          arrow_body_width = 2,
          view_theta = 0,
          view_zoom = 0.9)

# ---------------------------------------------------------
# Plot observations using their names and groups as
# stars but adding a legend instead of labels.
# Modify the aspect to normalize the variability
# of axes and do not show them. Zoom out a little.
biplot_3d(pca, groups = iris$Species,
          point_type = "label", point_label = row.names(iris),
          star_label_alpha = 0,
          show_group_legend = TRUE, group_legend_title = "",
          arrow_center_pos = c(.5, 0, .5),
          arrow_body_length = 2, arrow_body_width = 2,
          show_axes = FALSE, view_zoom = 1)

# ---------------------------------------------------------
# Get arbitrary Very Important Points
irisVIP <- list(setosa = (1:nrow(iris) == 16 |
                          1:nrow(iris) == 42),
                versicolor=(1:nrow(iris) == 61),
                virginica=(1:nrow(iris) == 107 |
                           1:nrow(iris) == 118 |
                           1:nrow(iris) == 132))

# Plot observations using their names and group by
# Species using only color. Mark the VIP and add the
# respective legend with custom characters.
# Rotate the theta view angle to fit the arrows
# in the default setting.
biplot_3d(pca,
          groups = iris$Species,
          point_type = "label",
          point_label = row.names(iris),
          group_representation = NULL,
          show_group_legend = TRUE,
```

```
          group_legend_title = "",
          vips = irisVIP,
          vip_pch = c("X", "O", "+"),
          vip_cex = c(2, 2, 3),
          show_axes = FALSE, view_theta = 340)

# ------------------------------------------------------------
# Test the setosa separation
irisDist <- dist(iris[, 1:4])

setosaSeparation <- iris$Species == "setosa"

## multivariate test for the setosa separation
require(vegan)
irisTests <- NULL
irisTests$permanova <- adonis(irisDist ~ setosaSeparation)
irisTests$permdisp2 <- permutest(betadisper(irisDist,
                                            setosaSeparation),
                                 pairwise = TRUE)

# The following function prepares a list of character vectors
# containing test results
getTestText <- function(tests){
  permanova_F <- as.character(round(tests$permanova$aov.tab$F.Model[1], 3))
  permanova_pvalue <- as.character(round(tests$permanova$aov.tab$"Pr(>F)"[1], 3))
  permanova_rSquared <- as.character(round(tests$permanova$aov.tab$R2[1], 3))
  permdisp2_F <- as.character(round(tests$permdisp2$tab$F[1], 3))
  permdisp2_pvalue <- as.character(round(tests$permdisp2$tab$"Pr(>F)"[1], 3))
  text <- list(c(paste("PERMANOVA:\n   F = ", permanova_F,
                       " (p = ", permanova_pvalue, ")\n",
                       sep = ""),
                 c(expression(paste("   ", R^2, " =",
                 sep = "")),
                   paste("          ", permanova_rSquared,
                   sep = ""))),
               paste("PERMDISP2:\n   F = ", permdisp2_F,
                     " (p = ", permdisp2_pvalue,")",
                     sep = ""))
  return(text)
}

# Plot observations using points and
# groups as stars with no labels.
# Place tests results in the bottom left corner
# and give a custom title.
biplot_3d(pca,
          groups = iris$Species,
          point_type = "point",
          star_label_alpha = 0,
          show_group_legend = TRUE,
          group_legend_title = "",
          test_text = getTestText(irisTests),
          test_cex = 1.5,
          test_fig = c(0.01, 0.5, 0.7, 1),
          show_axes = FALSE,
          view_theta = 340,
          title = "testing setosa separation")
```

```
## End(Not run)
```

`calculate_aspect` *Calculate aspect*

Description

Calculate the aspect or the relative scale of data in three dimensions.

Usage

```
calculate_aspect(x, y, z)
```

Arguments

`x, y, z`	numeric vectors representing point coordinates.

Value

Returns a vector with three numerical values equal or greater than 1.

Examples

```
## Not run:

calculate_aspect(0:100, 0:50, 0:10)


## End(Not run)
```

`ellipsoids_3d` *Draw ellipsoids per group*

Description

Compute and draw a labeled ellipsoid for each group in a rgl device. Singleton or groups with less than four observations are drawn as individual spheres.

Usage

```
ellipsoids_3d(x, y, z, groups, group_color = rainbow(nlevels((groups))),
  type = "wire and shade", level = 0.95, singleton_color = NULL,
  singleton_radius = 0.1, wire_color = NULL, wire_alpha = 0.2,
  wire_lit = FALSE, shade_color = NULL, shade_alpha = 0.1,
  shade_lit = FALSE, label_color = NULL, label_cex = 1,
  label_family = "sans", label_font = 2, label_adj = c(-0.25, 0.5),
  label_alpha = 1)
```

Arguments

`x, y, z`	Numeric vectors representing point coordinates.
`groups`	A factor vector of length `length(x)` containing the group assignation of each point.
`group_color`	A vector of length `nlevels(groups)` containing the colors to be used in each group.
`type`	a character representing the type of ellipsoid filling: "wire", "shade" or "wire and shade" (`wire3d`, `shade3d`).
`level`	the confidence level of a simultaneous confidence region (`ellipse3d`).
`singleton_color, wire_color, shade_color, label_color`	The specific colors to be used in each ellipsoid's elements for each group. If not NULL, they override the `colors` argument.
`singleton_radius`	the radius of the spheres (`spheres3d`) used to represent points of groups with three or less points.
`wire_alpha, wire_lit`	the wire alpha and lit parameters (`rgl.material`).
`shade_alpha, shade_lit`	the shade alpha and lit parameters (`rgl.material`).
`label_cex, label_family, label_font, label_adj, label_alpha`	the group labels text parameters (`text3d`, `rgl.material`). `label_adj` accepts a single numeric value (horizontal), a numeric vector of length two (horizontal, vertical) or a list of length `nlevels(groups)` containing the adj values for the specific groups.

See Also

`ellipsoid_3d`, `spheres3d`

Examples

```
## Not run:

# Use iris data
data("iris")

# introduce fictional singleton species
modIris <- iris
modIris$Species <- as.character(modIris$Species)
modIris <- rbind(modIris, list(6, 1, 2, 3, "outlier"))
modIris$Species <- factor(modIris$Species)

# Initializes the rgl device
rgl_init(theta = 60, phi = 45)

# add axes and bounding box
rgl_format(modIris$Sepal.Length, modIris$Sepal.Width, modIris$Petal.Length,
           axes_titles = c("Sepal length", "Sepal width", "Petal length"),
           show_planes = c("XZ", "XY", "YZ"))

# Add data points
```

```
points3d(modIris[modIris$Species == "setosa", 1],
         modIris[modIris$Species == "setosa", 2],
         modIris[modIris$Species == "setosa", 3],
         color = "green")
points3d(modIris[modIris$Species == "versicolor", 1],
         modIris[modIris$Species == "versicolor", 2],
         modIris[modIris$Species == "versicolor", 3],
         color = "red")
points3d(modIris[modIris$Species == "virginica", 1],
         modIris[modIris$Species == "virginica", 2],
         modIris[modIris$Species == "virginica", 3],
         color = "blue")

# Add ellipsoids
ellipsoids_3d(modIris[, 1], modIris[, 2], modIris[, 3],
              groups = modIris$Species,
              group_color = c("purple", "green", "red", "blue"))

remove(modIris)

## End(Not run)
```

`ellipsoid_3d` *Draw an labeled custom ellipsoid*

Description

Compute and draw a labeled ellipsoid in a rgl device.

Usage

```
ellipsoid_3d(x, y, z, color = "black", type = "wire and shade",
  level = 0.95, label = "", wire_color = NULL, wire_alpha = 0.2,
  wire_lit = FALSE, shade_color = NULL, shade_alpha = 0.1,
  shade_lit = FALSE, label_color = NULL, label_cex = 2,
  label_family = "sans", label_font = 2, label_adj = c(-0.25, 0.5),
  label_alpha = 1)
```

Arguments

`x, y, z`	Numeric vectors representing point coordinates.
`color`	The default color to be used in all elements.
`type`	Character, representing the type of ellipsoid filling: "wire", "shade" or "wire and shade" (`wire3d`, `shade3d`).
`level`	Numeric, the confidence level of a simultaneous confidence region (`ellipse3d`).
`label`	Character, The ellipsoid label.
`wire_color, shade_color, label_color`	The specific colors to be used in the ellipsoid's elements. If not NULL, they override the `color` argument.
`wire_alpha, wire_lit`	Numeric, the wire alpha and lit parameters (`rgl.material`).

shade_alpha, shade_lit
: Numeric, the shade alpha and lit parameters (rgl.material).

label_cex, label_family, label_font, label_adj, label_alpha
: The group labels text parameters (text3d, rgl.material).

See Also

ellipsoid_3d, rgl_init, ellipse3d, wire3d, shade3d, texts3d

Examples

```
## Not run:

# Use iris data
data("iris")

# get setosa
setosa <- iris[iris$Species == "setosa",]

# Initializes the rgl device
rgl_init(zoom = 0.75)

# add axes and bounding box
rgl_format(setosa$Sepal.Length, setosa$Sepal.Width, setosa$Petal.Length,
           axes_titles = c("Sepal length", "Sepal width", "Petal length"))

# Add data points
points3d(setosa[, 1], setosa[, 2], setosa[, 3], color = "black")

# Add ellipsoid
ellipsoid_3d(setosa[, 1], setosa[, 2], setosa[, 3], label = "setosa",
            wire_color = "green", shade_color = "red",
            label_color = "blue", label_adj = c(-1, 0.5))

remove(setosa)


## End(Not run)
```

---

filter_arrows *Filter covariance arrows for biplots*

Description

Filter the variables most represented in the firsts dimensions of an ordination object.

Usage

```
filter_arrows(loadings, min_dist = 0.5, dimensions = 2)
```

Arguments

loadings	Data frame, loadings or the covariances between variables and the coordinates calculated.
min_dist	Numeric, ratio representing the minimun distante of arrow's points to their origin. 1 return no variables.
dimensions	Numeric, the number of dimensions where arrows will be projected.

---

get_colors *Get colors for the different levels of a factor variable*

Description

Get colors for the different levels of a factor variable

Usage

```
get_colors(groups, color_palette = palette())
```

Arguments

groups	A factor variable containing the group of each observation.
color_palette	The palette or vector of colors to be used.

Value

Return a vector of colors matching the levels of groups.

---

get_lambda *Calculate lambda (biplot)*

Description

Calculate the lambda factor to scale biplot dimensions. The code is a snippet of biplot.princomp).

Usage

```
get_lambda(sdev, n.obs, dimensions = 2, scale = 1, pc.biplot = FALSE)
```

Arguments

sdev	Data frame containing the stardand deviations along n dimensions
n.obs	number of observations (rows) in the original data
dimensions	The number of dimensions
scale	Numeric, the value defining the degree in which distances between observations have priority over distances between variables (0 = observation-focused, 1 = variable-focused). It corresponds to the argument scale in biplot.princomp.
pc.biplot	Character, indicating the type of biplot transformation of data: "default" and "pc.biplot", corresponding to the transformations performed in biplot.princomp with pc.biplot = FALSE ("default") and pc.biplot = TRUE ("pc.biplot"). If NULL, no processing is performed, assuming that data within ordination_object was previously transformed.

`radial_arrows_3d` *Places a set of arrows in a rgl device*

Description

Places a set of arrows in a rgl device

Usage

```
radial_arrows_3d(x, y, z, variable_names = names(x), center_pos = c(0, 0,
  0), color = "darkorange", head_shape_theta = pi/6, head_shape_n = 3,
  head_size = 0.1, body_length = 1, body_width = 1,
  label_color = "black", label_cex = 1, label_family = "serif",
  label_font = 2, label_adj = 0, label_alpha = 1)
```

Arguments

`x, y, z`	Numeric vectors representing point coordinates where arrows are pointing to.
`variable_names`	Character vector with the names of the variables to be plotted as arrow labels.
`center_pos`	Numeric vector of length three containing the 3D position from which arrows are drawn.
`color`	The color or colors to be used in arrows.
`head_shape_theta`	Numeric, the angle of the barb of the arrows head ('theta' in `arrow3d`).
`head_shape_n`	Numeric, the number of barbs in the arrows head ('n' in `arrow3d`).
`head_size`	Numeric, the size of the arrows head ('s' in `arrow3d`).
`body_length`	Numeric, the length of the arrows body, as a fraction of the true distance from the origin.
`body_width`	Numeric, the width of the arrows body ('lwd' in `arrow3d`).
`label_color`	The color or colors of arrows' labels.
`label_cex, label_family, label_font, label_adj, label_alpha`	The text parameters and the alpha of the arrows' labels (`text3d`, `rgl.material`).

See Also

`arrow3d`,`biplot_3d`

Examples

```
## Not run:

# Use iris data
data("iris")

# Principal Components Analysis
pca <- princomp(iris[, 1:4])

# Initializes the rgl device
rgl_init(theta = 330)
```

```
# add axes and bounding box
rgl_format(x = pca$scores[, 1], y = pca$scores[, 2], z = pca$scores[, 3],
           axis_titles = c("PCA-1", "PCA-2", "PCA-3"))

# Add data points
points3d(x = pca$scores[iris$Species=="setosa", 1],
         y = pca$scores[iris$Species=="setosa",2],
         z = pca$scores[iris$Species=="setosa", 3],
         color = "green")
points3d(x = pca$scores[iris$Species=="versicolor", 1],
         y = pca$scores[iris$Species=="versicolor",2],
         z = pca$scores[iris$Species=="versicolor", 3],
         color = "red")
points3d(x = pca$scores[iris$Species=="virginica", 1],
         y = pca$scores[iris$Species=="virginica",2],
         z = pca$scores[iris$Species=="virginica", 3],
         color = "blue")

# Add covariance arrows
radial_arrows_3d(x = pca$loadings[,1],
                 y = pca$loadings[,2],
                 z = pca$loadings[,3],
                 body_width = 5)


## End(Not run)
```

---

rgl_format *Set up a rgl device*

Description

rgl_format set up the elements of a rgl device (axes, aspect, bounding box, projection planes) according to the space of the input data.

Usage

```
rgl_format(x, y, z, aspect = c(1, 1, 1), symmetric_axes = FALSE,
  show_axes = c("X", "Y", "Z"), show_planes = NULL, show_bbox = FALSE,
  adapt_axes_origin = TRUE, axes_colors = "darkgrey", axes_head_size = 3,
  axes_titles = "", axes_titles_cex = 2, axes_titles_font = 2,
  axes_titles_family = "sans", axes_titles_adj = list(x = c(-0.1, 0), y =
  c(0.5, -0.5), z = c(1.1, 0)), axes_titles_alpha = 1,
  planes_colors = c("lightgrey", "lightgrey", "lightgrey"),
  planes_textures = NULL, planes_alpha = 0.5, planes_lit = TRUE,
  planes_shininess = 50, bbox_color = c("#333377", "black"),
  bbox_alpha = 0.5, bbox_shininess = 5, bbox_xat = NULL,
  bbox_xlab = NULL, bbox_xunit = 0, bbox_xlen = 3, bbox_yat = NULL,
  bbox_ylab = NULL, bbox_yunit = 0, bbox_ylen = 3, bbox_zat = NULL,
  bbox_zlab = NULL, bbox_zunit = 0, bbox_zlen = 3, bbox_marklen = 15,
  bbox_marklen_rel = TRUE, bbox_expand = 1, bbox_draw_front = FALSE)
```

Arguments

`x, y, z`	The numeric vectors corresponding to the 3D coordinates of points.
`aspect`	A vector of length three indicating the relative scale of each dimension, to be past to `aspect3d`. By default, aspect is balanced (i.e. c(1, 1, 1)). If equals NULL, the "true" aspect is calculated with `calculate_aspect`.
`symmetric_axes`	Logical, whether the axes should be drawn symmetrically.
`show_axes`	Character vector, indicating which axes to draw: "X","Y","Z". NULL draws no axes.
`show_planes`	Character vector, indicating which planes to draw: "XZ","XY","YZ". NULL draws no plane.
`show_bbox`	Logical, indicating whether to draw a bounding box.
`adapt_axes_origin`	Logical, whether to adapt the axes origin.
`axes_colors`	The axes colors.
`axes_head_size`	The size of the head (end point) of the axes.
`axes_titles`	The axes titles.
`axes_titles_cex, axes_titles_font, axes_titles_family, axes_titles_adj, axes_titles_alpha`	The text parameters and the alpha of the titles (`text3d`). `axes_title_adj` accepts a single numeric value (horizontal), a numeric vector of length two (horizontal, vertical) or a list of length three (with elements named x, y and z) containing the adj values for the specific axes.
`planes_colors, planes_textures`	The colors and textures to be used in the planes. A vector can be given, following the order "XY","XZ" and "YZ". At least one color must be given.
`planes_alpha, planes_lit, planes_shininess`	The graphical parameters of the planes (`rgl.material`).
`bbox_color, bbox_alpha, bbox_shininess, bbox_xat, bbox_yat, bbox_zat, bbox_xlab, bbox_ylab, bbox`	arguments passed to create a bounding box (`bbox3d`).

Note

This function is based on the tutorial "A complete guide to 3D visualization device system in R - R software and data visualization" available in sthda.com Web site and last accessed in may 24 2016 (http://www.sthda.com/english/wiki/a-complete-guide-to-3d-visualization-device-system-in-r-r-so

See Also

`calculate_aspect`, `rgl_init`, `aspect3d`, `axis3d`, `bbox3d`

Examples

```
## Not run:

# Use iris data
data("iris")

# Initializes the rgl device
rgl_init(theta = 60)
```

```
# add axes and bounding box
rgl_format(iris$Sepal.Length, iris$Sepal.Width, iris$Petal.Length,
           aspect = c(1, 1, 1),
           axes_titles = c("Sepal length", "Sepal width", "Petal length"),
           show_planes = c("XY", "XZ", "YZ"))

# Add data points
points3d(iris[iris$Species == "setosa", 1],
         iris[iris$Species == "setosa", 2],
         iris[iris$Species == "setosa", 3], color = "green")
points3d(iris[iris$Species == "versicolor", 1],
         iris[iris$Species == "versicolor", 2],
         iris[iris$Species == "versicolor", 3], color = "red")
points3d(iris[iris$Species == "virginica", 1],
         iris[iris$Species == "virginica", 2],
         iris[iris$Species == "virginica", 3], color = "blue")

## End(Not run)
```

`rgl_init` *Initializes a rgl device*

Description

Initializes a new rgl device using specific settings.

Usage

```
rgl_init(new_device = FALSE, bg_color = "white", view_theta = 15,
  view_phi = 20, view_fov = 60, view_zoom = 0.8, width = 800,
  height = 600)
```

Arguments

`new_device`	If there is a rgl device open, should yet another be opened?
`bg_color`	The background color (`bg3d`).
`view_theta, view_phi, view_fov, view_zoom`	The theta and phi angles (polar coordinates), the field-of-view angle, and the zoom level of the viewpoint (`view3d`).
`width, height`	The width and height of the rgl device window in pixels (`par3d`).

Note

This function is based on the tutorial "A complete guide to 3D visualization device system in R - R software and data visualization" available in sthda.com Web site and last accessed in may 24 2016 (http://www.sthda.com/english/wiki/a-complete-guide-to-3d-visualization-device-system-in-r-r-so

See Also

`bg3d`, `view3d`, `par3d`, `rgl.open`, `rgl.close`, `rgl.clear`

Examples

```
## Not run:

rgl_init()
rgl_init(bg = "black")
rgl_init(width = 400, height = 300)


## End(Not run)
```

`scale_to_main` *Scale a 'fig' argument to fit another*

Description

Scale a numeric vector design to be used as `fig` in `par` to express a proportion of another 'fig' vector. This function enables to easily map several independent elements, with their own 'fig' arguments in the unit space (0 to 1 values), within a common plot area that is also controlled by a 'fig' argument.

Usage

```
scale_to_main(element_fig, main_fig)
```

Arguments

`element_fig`	Numeric, the 'fig' vector delimitating the plot area of a given element, using the unity as a reference.
`main_fig`	Numeric, the 'fig' vector delimitating the common plot area, within which limits the given element should be plot.

Examples

```
## Not run:

# graphics device is divided horizontally in two plots
plot1Fig <- c(0, 0.4999, 0, 1)
plot2Fig <- c(0.5099, 1, 0, 1)

# each plot has two independent elements
# placed in a position relatively to a plot area
myElement1Fig <- c(0.85, 1, 0.4, 0.6)
myElement2Fig <- c(0.3, 0.85, 0.7, 0.9)

# put plot fig arguments in a named list for convenience
plotsFigs <- list(plot1 = plot1Fig, plot2 = plot2Fig)

# save the par() configuration
current_par <- par(no.readonly = TRUE)

# iterate over the number of plots
```

```
for (i in 1:length(plotsFigs)) {

  # set plot area
  par(fig = plotsFigs[[i]])
  # create main plot
  plot(1:10, 1:10)

  # create first element
  par(fig = scale_to_main(myElement1Fig, plotsFigs[[i]]),
      new = T,
      mar = c(0, 0, 0, 0))
  plot.new()
  rect(0, 0, 1, 1)
  text(0.5, 0.5, labels = paste("myElement1\n", names(plotsFigs)[i], sep = ""))

  # create second element
  par(fig = scale_to_main(myElement2Fig, plotsFigs[[i]]),
      new = T,
      mar = c(0, 0, 0, 0))
  plot.new()
  rect(0, 0, 1, 1)
  text(0.5, 0.5, labels = paste("myElement2\n", names(plotsFigs)[i], sep = ""))

  par(current_par)

  par(new = T)
}
par(current_par)


## End(Not run)
```

`stars_3d` *Draw stars per group*

Description

Compute and draw stars (centroid with links) for each group in a rgl device.

Usage

```
stars_3d(x, y, z, groups, group_color = rainbow(nlevels((groups))),
  centroid_color = NULL, centroid_radius = 0.05, centroid_alpha = 0.5,
  link_color = NULL, link_width = 1, link_alpha = 1, label_color = NULL,
  label_cex = 1, label_family = "sans", label_font = 2,
  label_adj = c(-0.25, 0.5), label_alpha = 1)
```

Arguments

`x, y, z`	Numeric vectors representing point coordinates.
`groups`	A factor vector of length `length(x)` containing the group assignation of each point.

`group_color` A vector of length `nlevels(groups)` containing the colors to be used in each group.

`centroid_color` A color or a vector of colors to be used in group centroid. If NULL, `group_color` is used.

`centroid_radius` Numeric, the radius of the spheres used to represent group centroids (`spheres3d`).

`centroid_alpha` Numeric, the centroids alpha (`rgl.material`).

`link_color` A color or a vector of colors to be used in links between observations and centroid in each group. If NULL, `group_color` is used.

`link_width, link_alpha` Numeric, the link width and alpha parameters (`segments3d`, `rgl.material`).

`label_color` A color or a vector of colors to be used in group labels. If NULL, `group_color` is used.

`label_cex, label_family, label_font, label_adj, label_alpha` The text parameters and the alpha of the group labels (`text3d`, `rgl.material`). `label_adj` accepts a single numeric value (horizontal), a numeric vector of length two (horizontal, vertical) or a list of length `nlevels(groups)` containing the adj values for the specific groups.

See Also

`star_3d`

Examples

```
## Not run:

# Use iris data
data("iris")

# introduce fictional singleton species
modIris <- iris
modIris$Species <- as.character(modIris$Species)
modIris <- rbind(modIris, list(6, 1, 2, 3, "outlier"))
modIris$Species <- factor(modIris$Species)

# Initializes the rgl device
rgl_init(theta = 60, phi = 45, zoom = 0.75)

# add axes and bounding box
rgl_format(modIris$Sepal.Length, modIris$Sepal.Width, modIris$Petal.Length,
           axes_titles = c("Sepal length", "Sepal width", "Petal length"),
           show_planes = c("XZ", "XY", "YZ"))

# Add data points
points3d(modIris[modIris$Species == "setosa", 1],
         modIris[modIris$Species == "setosa", 2],
         modIris[modIris$Species == "setosa", 3],
         color = "green")
points3d(modIris[modIris$Species == "versicolor", 1],
         modIris[modIris$Species == "versicolor", 2],
         modIris[modIris$Species == "versicolor", 3],
         color = "red")
```

```
points3d(modIris[modIris$Species == "virginica", 1],
         modIris[modIris$Species == "virginica", 2],
         modIris[modIris$Species == "virginica", 3],
         color = "blue")

# Add stars
stars_3d(modIris[,1], modIris[,2], modIris[,3],
              groups = modIris$Species,
              group_color = c("purple","green","red","blue"),
              label_adj = list(c(-0.25, 0.5),
                               c(-0.5, 1.5),
                               c(-0.3, 1.2),
                               c(-0.3, 1.5)))

remove(modIris)


## End(Not run)
```

`star_3d` *Draw star*

Description

Compute and draw a star (centroid with links) in a rgl device.

Usage

```
star_3d(x, y, z, color = "black", label = "", centroid_color = NULL,
  centroid_radius = 0.05, centroid_alpha = 0.5, link_color = NULL,
  link_width = 1, link_alpha = 1, label_color = NULL, label_cex = 2,
  label_family = "sans", label_font = 2, label_adj = c(-0.25, 0.5),
  label_alpha = 1)
```

Arguments

`x, y, z`	Numeric vectors representing point coordinates.
`color`	The default color to be used in all elements.
`label`	The label placed at the centroid of the distribution.
`centroid_color`	The color of centroid point of the distribution.
`centroid_radius`	Numeric, the radius of the sphere used to represent the centroid of the distribution (`spheres3d`).
`centroid_alpha`	Numeric, the centroids alpha (`rgl.material`).
`link_color`	The color of the links connecting centroid and points.
`link_width, link_alpha`	Numeric, the link width and alpha parameters (`segments3d`, `rgl.material`).
`label_color`	The color of the centroid label.
`label_cex, label_family, label_font, label_adj, label_alpha`	The text parameters and the alpha of the centroid label (`text3d`, `rgl.material`).

See Also

stars_3d, rgl_init, spheres3d, segments3d, texts3d

Examples

```
## Not run:

# Use iris data
data("iris")

# get setosa
setosa <- iris[iris$Species == "setosa",]

# Initializes the rgl device
rgl_init(zoom = 0.8)

# add axes and bounding box
rgl_format(setosa$Sepal.Length, setosa$Sepal.Width, setosa$Petal.Length,
           axes_titles = c("Sepal length", "Sepal width", "Petal length"),
           show_planes = c("XZ", "XY", "YZ"))

# Add data points
points3d(setosa[, 1], setosa[, 2], setosa[, 3], color = "black")

# Add star
star_3d(setosa[, 1], setosa[, 2], setosa[, 3], label = "setosa",
       centroid_color = "green", link_color = "red", label_color = "blue")

remove(setosa)


## End(Not run)
```

Index

B.1.2 cerUB

Este apartado presenta la documentación del 'cerUB', un paquete R desarrollado para facilitar el análisis multivariante sobre datos de arqueometría cerámica. Este paquete se ha utilizado en las publicaciones que integran el capítulo 3. Además de su publicación en Zenodo (Angourakis y Martínez Ferreras 2017), el código fuente se puede encontrar en GitHub (https://github.com/Andros-Spica/cerUB/).

Aquest apartat presenta la documentació del 'cerUB', un paquet R desenvolupat per facilitar l'anàlisi multivariant sobre dades de arqueometria ceràmica. Aquest paquet s'ha utilitzat en les publicacions que integren el capítol 3. A més de la publicació en Zenodo (Angourakis y Martínez Ferreras 2017), el codi font es pot trobar a GitHub (https://github.com/Andros-Spica/cerUB/).

This section presents the documentation of the 'cerUB', an R package developed to facilitate the multivariate analysis of ceramic archaeometry data. This package has been used in the publications that make up chapter 3. In addition to its publication in Zenodo (Angourakis y Martínez Ferreras 2017), the source code can be found on GitHub (https://github.com/Andros-Spica/cerUB/).

Package 'cerUB'

October 15, 2017

Type Package

Title Protocols for exploring archaeometric data

Version 1.0.0

Description This package allows the user to apply four protocols of multivariate statistics for exploring archaeometric data, including geochemical and mineralogical compositions, and semi-quantitative petrographic characterizations. Protocols wrap several methods used in Geology and Ecology, relying on ade4 and vegan packages.

License GPL-3

URL https://github.com/Andros-Spica/cerUB

LazyData TRUE

RoxygenNote 6.0.1

Imports ade4, vegan, dbscan, pcaPP, robCompositions, setRNG, stringr

Suggests knitr, rmarkdown

NeedsCompilation no

Author Andreas Angourakis [aut, cre],
Veronica Martinez Ferreras [aut]

Maintainer Andreas Angourakis <andros.spica@gmail.com>

R topics documented:

`amphorae` *Wine Roman amphorae from Catalonia, NE Spain*

Description

A dataset containing petrographic, mineralogical, and geochemical data of wine Roman amphorae from Catalonia, NE Spain.

Usage

```
amphorae
```

Format

A data frame with 238 rows and 148 variables:

Site_Name Archaeological site name (see `levels(amphorae$Site_Name)`)

LOCATION_SITE_INITIALS Initials indicating site location (see `levels(amphorae$LOCATION_SITE_INITIALS)`)

CHARAC Is the observation fully characterised? ("complete", "incomplete"; in this data set all are "complete")

FabricGroup Fabric group (see `levels(amphorae$FabricGroup)`)

ChemReferenceGroup Chemical reference group (see `levels(amphorae$ChemReferenceGroup)`)

INCLUS_DISTRIB Inclusions distribution ("poorly", "poorly to moderately", "moderately", "moderately to well", "well", "none")

INCLUS_ORIENT Inclusions orientation ("unparallel", "slightly parallel", "parallel", "none")

TEMP Estimated firing temperature in degrees Celsius ("unfired", "700-800oC", "800-900oC", "900-1000oC", "1000-1100oC")

ATM Firing atmosphere ("reducing", "reducing to oxidising", "oxidising", "indeterminate"; in this data set all are "oxidising")

POST_ATM Post-firing atmosphere ("reducing", "reducing to oxidising", "oxidising", "indeterminate"; in this data set all are either "oxidising" or "indeterminate")

VOID_OVERALL Overall void frequency ("none", "very few", "few", "common", "abundant", "very abundant")

VOID_X_Y Void frequency by shape and size ("none", "few", "frequent", "predominant")

COAR_FREQ Inclusions coarse fraction frequency ("none", "very few", "few", "common", "abundant", "very abundant")

COAR_GRAINSIZE Inclusions coarse fraction grain size ("none","very fine","very fine to fine","fine","fine to medium","medium","medium to coarse","coarse","coarse to very coarse","very coarse")

COAR_ROUNDNESS Inclusions coarse fraction roundness ("angular","angular to subangular", "subangular","subangular to subrounded","subrounded","subrounded to rounded","rounded","none")

COAR_FORM Inclusions coarse fraction form ("elongate","elongate to equidimensional","equidimensional","equidime to laminar","laminar", "none")

COAR_SPACING Inclusions coarse fraction spacing ("single-spaced","single to double-spaced","double-spaced","double to open-spaced","open-spaced","none")

COAR_SORTING Inclusions coarse fraction sorting ("poorly-sorted","poorly to moderately-sorted","moderately-sorted","moderately to well-sorted","well-sorted","none")

COAR_R_X Inclusions coarse fraction (rocks) frequency by type ("none", "few", "common", "frequent", "dominant", "predominant")

COAR_C_X Inclusions coarse fraction (crystals) frequency by type ("none", "few", "common", "frequent", "dominant", "predominant")

FINE_FREQ Inclusions fine fraction frequency ("none", "very few", "few", "common", "abundant", "very abundant")

FINE_GRAINSIZE Inclusions fine fraction grain size ("none","very fine silt","very fine to fine silt","fine silt","fine to medium silt","medium silt","medium to coarse silt","coarse silt","coarse silt to very fine sand")

FINE_FORM Inclusions fine fraction form ("elongate","elongate to equidimensional","equidimensional","equidimensi to laminar","laminar", "none")

FINE_C_X Inclusions fine fraction (crystals) frequency by type ("none", "few", "frequent", "predominant")

Fe2O3 Component mass (Fe2O3) ...

`apply_ordination` *Apply ordination procedures for multivariate statistical analysis*

Description

#' Applies a given ordination procedure to a data set and returns a ordination object.

Usage

```
apply_ordination(data, protocol = "1", dimensions = 2,
  exception_columns = NULL, variable_tags = NULL, coda_override = NULL,
  coda_transformation_method = "CLR", coda_alr_base = 1,
  coda_pca_method = "robust", init_seed = 0, coda_samples = 100)
```

Arguments

`data`	Data frame, including compositional and petrographic data.
`protocol`	Character, cerUB protocol to be applied. "1": Analysis of compositional data; "2a": Analysis of petrographic data (relative ranking difference); "2b": Analysis of petrographic data (neighbor interchange); "3": Analysis of compositional data and petrographic data (relative ranking); "4": Analysis of compositional data and petrographic data (relative ranking) to characterize provenance.
`dimensions`	Numeric, number of dimensions of the ordination object.
`exception_columns`	Numeric, the vector of variables names to be searched for exceptions.
`variable_tags`	Character, two-column data frame containing (1) the names of variables and (2) their tags.

`coda_override` Character, vector with the names of the compositional variables.

`coda_transformation_method`
Character, the log-ratio transformation to be applied: "ALR" for additive log-ratio, "CLR" for centered log-ratio, "ILR" for isometric log-ratio. Additionally, accepts "log" for applying logarithmic transformation and "std" for standardization (scaled and centred).

`coda_alr_base` Character/Numeric, the name/index of the variable to be used as divisor in additional log-ratio transformation.

`coda_pca_method`
Character, Principal Components Analysis (PCA) method: "standard" for standard PCA, "robust" for robust PCA.

`init_seed, coda_samples`
Numeric, arguments passed to `princomp_coda`.

Value

Ordination object containing the projection of observations (scores) and variables (loadings) in 'n' dimensions, the distance matrix used (dist_matrix), and an approximation of the fitness of projections.

`best_pcaCoDa` *Find the best 'pcaCoDa' (robCompositions)*

Description

Searches the best projection given by `pcaCoDa`.

Usage

```
best_pcaCoDa(dt, method = "robust", init_seed = 0, samples = 100)
```

Arguments

`dt`	Data frame containing compositional data
`method`	Character, "standard" for standard PCA, "robust" for robust PCA.
`init_seed`	Numeric, the seed for the random number generator used in `best_pcaCoDa`).
`samples`	Numeric, the number of iterations applying to samples in `best_pcaCoDa`) and maxiter in `PCAgrid`).

`clean_and_format` *Clean and format data for cerUB protocols*

Description

Cleaning and format procedures, including coercing variables as numeric or factor, excluding columns (constants, perturbed, unreliable) and rows (incomplete data, outliers).

Usage

```
clean_and_format(data, categorical_columns = c(), numerical_columns = c(),
  completion_variable = c("CHARAC", "complete"), as_na = c(NULL),
  method = "random", columns_to_exclude = c("variableNameToExclude"),
  rows_to_exclude = c("outlierObservation"))
```

Arguments

`data`	Data frame, a data frame to be prepared for applying cerUB protocols.
`categorical_columns`	Character/Numeric, vector with the names/indexes of the categorical variables.
`numerical_columns`	Character/Numeric, vector with the names/indexes of the numeric variables.
`completion_variable`	Character, vector with two elements (name, value) referencing the column that indicates wheter observations (rows) are completed.
`as_na`	Character, vector that specifies values to be considered as NA.
`method`	Character, method to be used in for replacing NA, if any (`replace_na`).
`columns_to_exclude`	Character/Numeric, vector with the names/indexes of columns to exclude.
`rows_to_exclude`	Character/Numeric, vector with the names/indexes of rows to exclude.

Examples

```
## Not run:

dt <- cbind("First" = c(1,2,2,3,5,1,6,0,4,10),
            "Second" = c("A","A","A","A","A","A","A","A","A","A"),
            "Third" = c("1","2","2","3","5","1","6","0","4","10"),
            "Fourth" = c("A","B","C","D","E","F","G","H","I","J"),
            "dummy" = c("bla","ble","bli","blo","blu","bla","ble",
                        "bli","blo","blu"),
            "checked" = c("yes","yes","no","yes","no","yes","yes",
                          "no","yes","yes"))
row.names(dt) <- 1:10
claen_and_format(dt,
                 categorical_columns = c("Second", "Fourth"),
                 numerical_columns = c("First", "Third"),
                 completion_variable = "checked",
                 as_na = c("0","D"),
```

```
                  method = "random",
                  columns_to_exclude = c("dummy"),
                  rows_to_exclude = c(1, 10)
                  )


## End(Not run)
```

`code_variables` *Codify petrographic variable names*

Description

Creates a code for each variable in a petrographic data frame using cerUB naming system, and return a two-column data frame relating original names with their codes.

Usage

```
code_variables(data)
```

Arguments

`data`	A petrographic data frame using cerUB naming system.

`detect_outliers` *Detect outliers*

Description

Detects outliers in a distance matrix using a certain method and following a certain criterion.

Usage

```
detect_outliers(distMatrix, method = "MD", criterion = "MAD", LOF_k = 2,
  MAD_trim = 2, boxplot_trim = 1.5)
```

Arguments

`distMatrix`	Numeric, distance matrix
`method`	Character, method for measuring separation of one point from all other points. The following are accepted: "MdD": median distance; "MD": average (mean) distance; "MAH": Mahalanobis distances (`outCoDa`); "LOF": Local Outlier Factor Score (`lof`).
`criterion`	Numeric/Character, the criterion used for separating outliers. The following are accepted: <Numeric, 0-1>: number between 0 and 1, sets a quantile type of threshold; "boxplot": outliers are those singled out as points in a boxplot; "MAD": threshold is given by Median Absolute Deviation.
`LOF_k`	Numeric, when method = "LOF", the size of the neighborhood. See `lof`.

`MAD_trim`	Numeric, when criterion = "MAD", the multipler of MAD to calculate a outlier threshold.
`boxplot_trim`	Numeric, when criterion = "boxplot", the multipler of the interquartile range (IQR) to calculate a outlier threshold.

Examples

```
## Not run:

pca <- princomp(iris[, 1:4])

irisOutliers_MD <- detect_outliers(dist(iris[, 1:4]),
                                   method = "MD",
                                   criterion = "MAD")
irisOutliers_LOF <- detect_outliers(dist(iris[, 1:4]),
                                   method = "LOF",
                                   criterion = "MAD")
plot(pca$scores[, 1:2], col = "black", main = "Outliers")
points(pca$scores[irisOutliers_MD$index, 1:2],
       col = "red", pch = 2, cex = 1.5)
points(pca$scores[irisOutliers_LOF$index, 1:2],
       col = "blue", pch = 6, cex = 1.5)
legend(0.65 * max(pca$scores[,1]), max(pca$scores[,2]),
       c("MD", "LOF"), pch = c(2, 6), col = c("red", "blue"))


## End(Not run)
```

`detect_outliers_per_group` *Detect outliers per group*

Description

Detects outliers per each group in a distance matrix using a certain method and following a certain criterion.

Usage

```
detect_outliers_per_group(distMatrix, groups, method = "MD",
  criterion = "MAD", LOF_k = 2, MAD_trim = 2, boxplot_trim = 1.5)
```

Arguments

`distMatrix`	Numeric, distance matrix
`groups`	Factor, vector containing the assignation of each observation in the matrix to a specific group.
`method`	Character, method for measuring separation of one point from all other points. The following are accepted: "MdD": median distance; "MD": average (mean) distance; "MAH": Mahalanobis distances (`outCoDa`); "LOF": Local Outlier Factor Score (`lof`).

criterion	Numeric/Character, the criterion used for separating outliers. The following are accepted: <Numeric, 0-1>: number between 0 and 1, sets a quantile type of threshold; "boxplot": outliers are those singled out as points in a boxplot; "MAD": threshold is given by Median Absolute Deviation.
LOF_k	Numeric, when method = "LOF", the size of the neighborhood. See lof.
MAD_trim	Numeric, when criterion = "MAD", the multipler of MAD to calculate a outlier threshold.
boxplot_trim	Numeric, when criterion = "boxplot", the multipler of the interquartile range (IQR) to calculate a outlier threshold.

Examples

```
## Not run:

pca <- princomp(iris[, 1:4])

irisSpeciesOutliers_MD <- detect_outliers_per_group(dist(iris[, 1:4]),
                                                    iris$Species,
                                                    method = "MD",
                                                    criterion = "MAD")
irisSpeciesOutliers_LOF <- detect_outliers_per_group(dist(iris[, 1:4]),
                                                     iris$Species,
                                                     method = "LOF",
                                                     criterion = "MAD")
plot(pca$scores[, 1:2],
    col = iris$Species,
    main = "Outliers per group")
points(pca$scores[irisOutliers_MD$index, 1:2],
       col = "purple", pch = 2, cex = 1.5)
points(pca$scores[irisOutliers_LOF$index, 1:2],
       col = "orange", pch = 6, cex = 1.5)
legend(0.65 * max(pca$scores[,1]), max(pca$scores[,2]),
       c("MD", "LOF"), pch = c(2, 6), col = c("purple", "orange"))


## End(Not run)
```

`dist.ktab_cerUB` *Mixed-variables coefficient of distance (cerUB version)*

Description

"The mixed-variables coefficient of distance generalizes Gower's general coefficient of distance to allow the treatment of various statistical types of variables when calculating distances. This is especially important when measuring functional diversity. Indeed, most of the indices that measure functional diversity depend on variables (traits) that have various statistical types (e.g. circular, fuzzy, ordinal) and that go through a matrix of distances among species." (From dist.ktab) This is a modified version that allows for 'exception values' in ordinal variables and weighting the class-wise data sets differently.

Usage

```
dist.ktab_cerUB(ktab_data, variable_classes, option = c("scaledBYrange",
  "scaledBYsd", "noscale"), scann = FALSE, tol = 1e-08,
  is_protocol2b = FALSE, dist.excep = 2, weight = NULL)
```

Arguments

ktab_data	Object of class ktab (created with ktab.list.df), including data frames of different data classes.
variable_classes	Vector that provide the type of each table in x. The possible types are "Q" (quantitative), "O" (ordinal), "N" (nominal), "D" (dichotomous), "F" (fuzzy, or expressed as a proportion), "B" (multichoice nominal variables, coded by binary columns), "C" (circular), "CODA" (compositional). Values in type must be in the same order as in x.
option, scann, tol	Arguments of dist.ktab.
is_protocol2b	Logical, whether the protocol 2b is being applied. Protocol 2b uses 'neighbour interexchange' to calculate distance in ordinal variables.
dist.excep	The distance from any value to an exception value, that should be previously transform into NA.
weight	Numeric, vector with the weights attributed to every data frame included in ktab_data.

---

extended_gower *Extended Gower with exceptions*

Description

Calculate the extended Gower coefficient for archaeometric data sets, considering exceptional values in ordinal variables, using a modified version of dist.ktab.

Usage

```
extended_gower(data, variable_sets, method = "RRD",
  exception_columns = NULL, exception_values = NULL,
  exception_distance = 0)
```

Arguments

data	Data frame containing ordinal variables and optionally compositional data (log-ratios).
variable_sets	List containing vectors with the numeric index of variables of different kind.
method	Character, "NI" for neighbour interexchange, "RRD" for relative ranking difference, "MM" for mixed mode.
exception_columns	Numeric, the vector of variables names to be searched for exceptions.
exception_values	Numeric, the value to be set with distance.

`exception_distance`
: Numeric, the index of the variable to be used as divisor in additive log-ratio transformation.

`get_coda` *Get indexes of compositional variables*

Description

Get indexes of compositional variables that were transformed using `transform_coda`.

Usage

```
get_coda(data, coda_variables, transformation_method = "")
```

Arguments

`data`
: Data frame containing petrographic variables that use the cerUB naming system.

`coda_variables`
: Character/Numeric, vector with the names/indexes of the compositional variables.

`transformation_method`
: Character, vector specifying a transformation: "ALR" -> additive log-ratio, "CLR" -> centered log-ratio, "ILR" -> isometric log-ratio. Additionally, accepts "log" for applying logarithmic transformation.

`get_petro` *Get indexes of petrographic variables*

Description

Get indexes of petrographic variables that use the cerUB naming system.

Usage

```
get_petro(data)
```

Arguments

`data`
: Data frame containing petrographic variables that use the cerUB naming system.

get_provenance *Get indexes of provenance-related variables*

Description

Get indexes of the compositional and petrographic variables indicative of provenance. Petrographic variables must use the cerUB naming system. Compositional variables must have been transformed using `transform_coda`.

Usage

```
get_provenance(data, coda_variables, transformation_method = "")
```

Arguments

`data`	Data frame containing petrographic variables that use the cerUB naming system.
`coda_variables`	Character/Numeric, vector with the names/indexes of the compositional variables.
`transformation_method`	Character, vector specifying a transformation: "ALR" -> additive log-ratio, "CLR" -> centered log-ratio, "ILR" -> isometric log-ratio. Additionally, accepts "log" for applying logarithmic transformation.

Value

List of two numeric vectors containing the column indexes of (1) petrographic and (2) compositional variables.

nmds *Non-metric Multidimensional Scaling (NMDS)*

Description

Apply Non-metric Multidimensional Scaling to a given distance matrix, calculate variable covariances, and the percent of variance explained by 2D and 3D projections.

Usage

```
nmds(distance_matrix, original_data, variable_tags = c(), dimensions = 2,
  init_seed = 0, trymax = 100, autotransform = FALSE)
```

Arguments

`distance_matrix`	distance or dissimilarity matrix
`original_data`	data frame containing the original data
`variable_tags`	Character, two-column data frame containing (1) the names of variables and (2) their tags.
`dimensions`	Numeric, number of dimensions of the projection equivalent to k in `metaMDS`.

`init_seed` Numeric, the seed for the random number generator used by `metaMDS`.

`trymax, autotransform`
Numeric, Maximum number of random starts in search of stable solution. Logical, whether to use simple heuristics for possible data transformation of typical community data (see below). If you do not have community data, you should probably set autotransform = FALSE. Arguments passed to `metaMDS`.

---

order_petro *Order petrographic ordinal variables*

Description

Order the values (levels) of petrographic ordinal variables (factors) that use the cerUB naming system.

Usage

```
order_petro(data)
```

Arguments

`data`	Data frame containing petrographic ordinal variables that use the cerUB naming system.

---

pcoa *Principal Coordinates Analysis*

Description

Apply Principal Coordinates Analysis to a given distance matrix, calculate variable covariances, and the percent of variance explained by 2D and 3D projections.

Usage

```
pcoa(distance_matrix, original_data, variable_tags = NULL, dimensions = 2)
```

Arguments

`distance_matrix`	distance or dissimilarity matrix
`original_data`	data frame containing the original data
`variable_tags`	Character, two-column data frame containing (1) the names of variables and (2) their tags.
`dimensions`	Numeric, number of dimensions of the projection equivalent to k in `cmdscale`

`princomp_coda` *Principal Components Analysis for compositional data*

Description

Principal Components Analysis (PCA) of compositional data after applying log-ratio transformation.

Usage

```
princomp_coda(dt, transformation_method = "ILR", method = "robust",
  init_seed = 0, samples = 100, alr_base = 1)
```

Arguments

`dt`	Data frame containing compositional data
`transformation_method`	Character, the log-ratio transformation to be applied. "ALR" -> additive log-ratio, "CLR" -> centered log-ratio, "ILR" -> isometric log-ratio. Additionally, accepts "log" for applying logarithmic transformation and "std" for standardization (scaled and centred).
`method`	Character, "standard" for standard PCA, "robust" for robust PCA.
`init_seed`	Numeric, the seed for the random number generator used in `best_pcaCoDa`).
`samples`	Numeric, the number of iterations applying to samples in `best_pcaCoDa`) and maxiter in `PCAgrid`).
`alr_base`	Character/Numeric, the name/index of the variable to be used as divisor in additional log-ratio transformation.

`replace_na` *Replaces NA values*

Description

Replaces NA values in a vector according to a given method. Specific non-NA values can be considered as NA, if given in as_na.

Usage

```
replace_na(x, as_na = c(NULL), method = NULL)
```

Arguments

`x`	Numeric or Character/factor vector.
`as_na`	Numeric or Character, specifies values that may be considered as NA.
`method`	Character, the method used for choosing the replacement ("NULL", "random", "mode", "normal"). Accepts numeric value if x is numeric. Method must be compatible with x class.

Examples

```
## Not run:

replace_na(c(1, 2, NA, 4, 5, 6), method = 3)
replace_na(c(1, 3.5, 9, 2.8, 5.6, 10.4, 0.7, 2.4, 5.5, NA), method = "normal")
replace_na(c(1, 3.5, 9, 2.8, 5.6, 10.4, 0.7, 2.4, 5.5, NA), method = "random")
replace_na(c("A", "B", "A", "F", "K", "B", "O", "A"), method = "mode")


## End(Not run)
```

---

simplify_coda_names *Simplify transformed Compositional data names in ordination object*

Description

Replace composite name of type "transformationMethod-component", such as CLR-Fe2O3, with a shorter version, such as "Fe2O3".

Usage

```
simplify_coda_names(ordination_object)
```

Arguments

ordination_object	Ordination object, as generated by apply_ordination, containing a "loadings" data frame with transformed CoDa variables.

---

test_groups *Perform Group separation and uniformity tests*

Description

Perform four tests (anosim, betadisper, permdisp2, and permanova) that assess the separation and uniformity of the given group factor. Additionally, creates a generative text with the results of PERMANOVA and PERMDISP2 test results.

Usage

```
test_groups(distMatrix, groups)
```

Arguments

distMatrix	Numeric, distance matrix
groups	Factor, vector containing the assignation of each observation in the matrix to a specific group.

`transform_coda` *Transform compositional data*

Description

Transform compositional data in a given data frame and replace it.

Usage

```
transform_coda(data, coda_variables, method = c("CLR"), alr_base = 1,
  raw_filename = NULL, trans_filename = NULL, final_filename = NULL)
```

Arguments

`data`	Data frame containing compositional data.
`coda_variables`	Numeric/Character, vector containing the names/indexes of the compositional variables.
`method`	Character, the log-ratio transformation to be applied. "ALR" -> additive log-ratio, "CLR" -> centered log-ratio, "ILR" -> isometric log-ratio. Additionally, accepts "log" for applying logarithmic transformation and "std" for standardization (scaled and centred).
`alr_base`	Character/Numeric, the name/index of the variable to be used as divisor in additional log-ratio transformation. in additive log-ratio transformation.
`raw_filename, trans_filename, final_filename`	Character, file names for saving the raw (complete), tranformed (only coda), and final (complete) data sets.

Index

B.2 Modelización y simulación computacional

- 'Musical Chairs model' (NetLogo code)
- 'Nice Musical Chairs model'(NetLogo code)

B.2.1 Musical Chairs

A continuación, se presenta el código del modelo Musical Chairs en lenguaje de NetLogo, correspondiente a la versión publicada en Angourakis et al. (2014) y discutida en Angourakis (2014). No se incluye aquí el código completo del archivo ".nlogo". La copia completa de ésta y cualquier versión posterior se puede encontrar en Angourakis (2016) (https://www.openabm.org/model/4880/) y en GitHub (https://github.com/Andros-Spica/MusicalChairs). Todas las versiones de este modelo están bajo la Licencia Pública General Reducida de GNU, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

A continuació, es presenta el codi del model Musical Chairs en llenguatge de NetLogo, corresponent a la versió publicada en Angourakis et al. (2014) i discutida en Angourakis (2014). No s'inclou aquí el codi complet de l'arxiu ".nlogo". La còpia completa d'aquesta i qualsevol versió posterior es pot trobar a Angourakis (2016) (https://www.openabm.org/model/4880/) i en GitHub (https://github.com/Andros-Spica/MusicalChairs). Totes les versions d'aquest model estan sota la Llicència Pública General Reduïda de GNU, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

Next, the code of the Musical Chairs model is presented in NetLogo language, corresponding to the version published in Angourakis et al. (2014) and discussed in Angourakis (2014). The complete code of the ".nlogo" file is not included here. The full copy of this and any later version can be found at Angourakis (2016) (https://www.openabm.org/model/4880/) and on GitHub (https://github.com/Andros-Spica/MusicalChairs). All versions of this model are licensed under GNU General Public License, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

```
globals
[
  ;;; modified parameters
  f_int h_int f_ext h_ext
  init_h init_f
  h_r_m_i h_intg f_intg

  ;;; sets of agents that exists at the end of each run
  settled_farming_agents stable_herding_agents

  ;;; unlucky agents, their respective number of helpers and their intensity
  unlucky_F farming_support farming_intensity
  unlucky_H herding_support herding_intensity

  ;;; variables used in resolve_conflict
  index_of_opportunity ratio_of_intensities
  incentives_to_relinquish

  ;;; counters and final measures
  dilemma_events oasis_degression_events herding_success_ratio
  farming_growth farming_deterrence
  herding_growth herding_deterrence farming_balance herding_balance
  farming_histo_intensity_0_025
  farming_histo_intensity_025_05
  farming_histo_intensity_05_075
  farming_histo_intensity_075_1
  farming_histo_independence_0_025
  farming_histo_independence_025_05
  farming_histo_independence_05_075
  farming_histo_independence_075_1
  herding_histo_intensity_0_025
  herding_histo_intensity_025_05
  herding_histo_intensity_05_075
  herding_histo_intensity_075_1
```

```
  herding_histo_independence_0_025
  herding_histo_independence_025_05
  herding_histo_independence_05_075
  herding_histo_independence_075_1
  mean_fint mean_find mean_hint mean_hind
]

;;; establishes the classes of agents and their variables

breed [ farming_agents farming_agent ]

breed [ herding_agents herder_agent ]

farming_agents-own [ intensity independence ]

herding_agents-own [ intensity independence ]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SETUP ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to setup

  clear-all

  ;;; setup parameters depending on the type of experiment
  set f_int 0
  ifelse (random_exp? = true)
  [
    set f_int random-float farming_intrinsic_growth_rate
  ]
  [ set f_int farming_intrinsic_growth_rate ]
  set h_int 0
  ifelse (random_exp? = true)
```

```
[
  set h_int random-float herding_intrinsic_growth_rate
]
[ set h_int herding_intrinsic_growth_rate ]

set h_r_m_i 0
ifelse (random_exp? = true)
[
  set h_r_m_i
    ((1 + (random-float (herding_relative_max_intensity - 1)) ) /
      (1 + (random-float (herding_relative_max_intensity - 1)) ))
]
[set h_r_m_i herding_relative_max_intensity]

set h_ext 0
ifelse (random_exp? = true)
[set h_ext random-float herding_extrinsic_growth_rate]
[set h_ext herding_extrinsic_growth_rate]

set f_ext 0
ifelse (random_exp? = true)
[set f_ext random-float farming_extrinsic_growth_rate]
[set f_ext farming_extrinsic_growth_rate]

set h_intg 0
ifelse (random_exp? = true)
[set h_intg random-float herding_integration]
[set h_intg herding_integration]

set f_intg 0
ifelse (random_exp? = true)
[set f_intg random-float farming_integration]
[set f_intg farming_integration]
```

```
  ;;; create agents according to the parameter setting
  ;;; (position is arbitrary and has no consequence)
  ask patch 0 0
  [
    set init_h 0
    ifelse (random_exp? = true)
    [set init_h random init_herding]
    [set init_h init_herding]
    sprout-herding_agents init_h [ initialize-an-agent ]
  ]
  ask patch max-pxcor 0
  [
    set init_f 0
    ifelse (random_exp? = true)
    [set init_f random init_farming]
    [set init_f init_farming]
    sprout-farming_agents init_f [ initialize-an-agent ]
  ]

  ;;; initialize visualization
  ask patches [ set pcolor yellow ]

  update_visualization

  set dilemma_events 0 ;reset dilemma_events counter
  set oasis_degression_events 0 ;reset oasis_degression_events counter

  reset-ticks

end

to initialize-an-agent

  set hidden? true
```

```
  ifelse (breed = farming_agents)
  [ set intensity random-float farming_max_intensity ]
  [ set intensity random-float (farming_max_intensity * h_r_m_i) ]
  set independence (random-float 1)

end

to go

  farming_expansion

  herding_expansion

  ;update_land_use

  ;reset dilemma_events and oasis_degression_events counters
  set dilemma_events 0
  set oasis_degression_events 0

  check_competition

  update_visualization

  tick

end

to farming_expansion

  ;;; set of farming_agents currently present
  set settled_farming_agents turtle-set farming_agents

  ;;; reset counters
  set farming_growth 0
```

```
set farming_deterrence 0

;;; Intrinsic growth
ask farming_agents
[ if ( random-float 1 <= f_int )
  [
    hatch 1
    set farming_growth farming_growth + 1
  ]
]

;;; Extrinsic growth
ask patch max-pxcor 0
[ sprout-farming_agents (round (f_ext * (count herding_agents) ) )
  [
    initialize-an-agent
    set farming_growth farming_growth + 1
  ]
]

;;; Fit-to-maximum exclusion
ask farming_agents
[
  if (not member? self settled_farming_agents)
  [
  ; new farming_agents exit the territory in a random order
  ; whenever there is no more land to use
  if (count farming_agents > count patches)
  [ set farming_deterrence farming_deterrence + 1 die ]
  ]
]

;;; Density-dependent exclusion
ask farming_agents
```

```
  [
    if (not member? self settled_farming_agents)
    [
      ;new farming_agents exit the territory with a probability
      ; proportional to the density of land currently used for
      ; agriculture (i.e. proxy of the quality of the remaining land)
      if (random-float 1 < ((count farming_agents) / (count patches) ) )
      [ set farming_deterrence farming_deterrence + 1 die ]
    ]
  ]

  ;;; Volition-opportunity exclusion
  ask farming_agents
  [
    if (not member? self settled_farming_agents)
    [
      ;new farming_agents exit the territory if they are not bold enough to colonize t
      if ( ( (count herding_agents) / (count patches) ) > independence )
      [ set farming_deterrence farming_deterrence + 1 die ]
    ]
  ]

end

to herding_expansion

  ;;; set of herding_agents currently present
  set stable_herding_agents turtle-set herding_agents

  ;;; reset counters
  set herding_growth 0
  set herding_deterrence 0

  ;;; Intrinsic growth
```

```
ask herding_agents
[ if ( random-float 1 <= h_int )
  [
    hatch 1
    set herding_growth herding_growth + 1
  ]
]

;;; Extrinsic growth
let attraction ( (count patches) - (count stable_herding_agents) )
ask patch 0 0
[ sprout-herding_agents (round (h_ext * attraction))
  [
    initialize-an-agent
    set herding_growth herding_growth + 1
  ]
]

;;; Fit-to-maximum exclusion
ask herding_agents
[
  if (not member? self stable_herding_agents)
  [
    ;new herding_agents exit the territory in a random
    ;order whenever there is no more land to use
    if (count herding_agents > count patches)
    [ set herding_deterrence herding_deterrence + 1 die ]
  ]
]

;;; Density-dependent exclusion
ask herding_agents
[
  if (not member? self stable_herding_agents)
```

```
    [
      ;new herding_agents exit the territory with a probability
      ;proportional to the density of pastures currently in use
      ;(i.e. proxy of the quality of the remaining land)
      if (random-float 1 < ( (count herding_agents) / (count patches) ) )
      [ set herding_deterrence herding_deterrence + 1 die ]
    ]
  ]

end

to check_competition

  if ((count patches) - (count farming_agents) < count herding_agents)
  [ resolve_competition ]

end

to resolve_competition

  ;;; set competition conditions

  ;;;;; select one farming agent and its supporters, calculate the
  ;;;;;intensity of the farming land use involved in a land use unit
  set unlucky_F one-of farming_agents
  set farming_support round (f_intg * ((count farming_agents) - 1))
  let p 0
  if (farming_support > 0)
  [
    set p
      sum [intensity] of n-of farming_support farming_agents
        with [self != unlucky_F]
  ]
  set farming_intensity ([intensity] of unlucky_F + p )
```

```
;;;;; select one herding agent and its supporters,
;;;;; calculate the intensity of the herding land use
;;;;; to be involved in the same land use unit
set unlucky_H one-of herding_agents
set herding_support round (h_intg * ((count herding_agents) - 1))
let q 0
if (herding_support > 0)
[
  set q
    sum [intensity] of n-of herding_support herding_agents
      with [self != unlucky_H] ]
set herding_intensity ([intensity] of unlucky_H + q )

;;;;; calculate the ratio of intensities, the index of opportunity
;;;;; and the incentives for relinquish, all taken from the
;;;;; perspective of herding land use
set ratio_of_intensities
  (herding_intensity /
    (farming_intensity + herding_intensity))
set index_of_opportunity ((count farming_agents) / (count patches))
set incentives_to_relinquish
  (1 - (ratio_of_intensities * index_of_opportunity))

;;; resolve the competitive situation
ask unlucky_H
[
  ;;; Does the competitive situation evolves into a dilemma event?
  ifelse ( independence < incentives_to_relinquish)

  ;;; No. The herding agent exit the territory.
  [ set herding_deterrence herding_deterrence + 1 die ]

  ;;; Yes. A dilemma event is produced for there are two
```

```
    ;;; variants to be realized in a single land use unit.
    [
      set dilemma_events (dilemma_events + 1)

      ;;; Does the dilemma event evolves into a oasis degression event?
      ifelse (random-float 1 < ratio_of_intensities)

      ;;; Yes. The unlucky farming agent exits the territory.
      [
        ask unlucky_F
        [ set farming_deterrence farming_deterrence + 1 die ]
        set oasis_degression_events (oasis_degression_events + 1)
      ]

      ;;; No. The unlucky herding agent exits the territory.
      [ set herding_deterrence herding_deterrence + 1 die ]
    ]
  ]

  ;;; re-check the presence of competitive situations
  check_competition

end

to update_visualization

  update_patches

  set farming_balance (farming_growth - farming_deterrence)
  set herding_balance (herding_growth - herding_deterrence)

  ifelse (dilemma_events > 0)
  [
    set herding_success_ratio
```

```
    (oasis_degression_events / dilemma_events)
]
[ set herding_success_ratio 0 ]

set farming_histo_intensity_0_025 (count farming_agents
  with [intensity <= ( 0.25 * farming_max_intensity)])
set farming_histo_intensity_025_05 (count farming_agents
  with [intensity <= ( 0.5 * farming_max_intensity)])
set farming_histo_intensity_025_05
  (farming_histo_intensity_025_05 -
  farming_histo_intensity_0_025)
set farming_histo_intensity_05_075
  (count farming_agents
    with [intensity <= ( 0.75 * farming_max_intensity)])
set farming_histo_intensity_05_075
  (farming_histo_intensity_05_075 -
  farming_histo_intensity_0_025 -
  farming_histo_intensity_025_05)
set farming_histo_intensity_075_1
  ((count farming_agents) -
  farming_histo_intensity_0_025 -
  farming_histo_intensity_025_05 -
  farming_histo_intensity_05_075)
ifelse (count farming_agents > 0)
[
  set mean_fint
    ((farming_histo_intensity_0_025 * 1 / (count farming_agents)) +
    (farming_histo_intensity_025_05 * 2 / (count farming_agents)) +
    (farming_histo_intensity_05_075 * 3 / (count farming_agents)) +
    (farming_histo_intensity_075_1 * 4 / (count farming_agents))) ]
[ set mean_fint 0 ]

set farming_histo_independence_0_025
  (count farming_agents with [independence <= 0.25])
```

```
set farming_histo_independence_025_05
  (count farming_agents with [independence <= 0.5])
set farming_histo_independence_025_05
  (farming_histo_independence_025_05 -
  farming_histo_independence_0_025)
set farming_histo_independence_05_075
  (count farming_agents with [independence <= 0.75])
set farming_histo_independence_05_075
  (farming_histo_independence_05_075 -
  farming_histo_independence_0_025 -
  farming_histo_independence_025_05)
set farming_histo_independence_075_1
  ((count farming_agents) -
  farming_histo_independence_0_025 -
  farming_histo_independence_025_05 -
  farming_histo_independence_05_075)
ifelse (count farming_agents > 0)
[
  set mean_find
    ((farming_histo_independence_0_025 * 1 / (count farming_agents)) +
    (farming_histo_independence_025_05 * 2 / (count farming_agents)) +
    (farming_histo_independence_05_075 * 3 / (count farming_agents)) +
    (farming_histo_independence_075_1 * 4 / (count farming_agents)))
]
[ set mean_find 0 ]

set herding_histo_intensity_0_025
  (count herding_agents
    with [intensity <= (0.25 * (h_r_m_i * farming_max_intensity))])
set herding_histo_intensity_025_05
  (count herding_agents
    with [intensity <= (0.5 * (h_r_m_i * farming_max_intensity))])
set herding_histo_intensity_025_05
  (herding_histo_intensity_025_05 -
```

```
  herding_histo_intensity_0_025)
set herding_histo_intensity_05_075
  (count herding_agents
    with [intensity <= (0.75 * (h_r_m_i * farming_max_intensity))])
set herding_histo_intensity_05_075
  (herding_histo_intensity_05_075 -
  herding_histo_intensity_0_025 -
  herding_histo_intensity_025_05)
set herding_histo_intensity_075_1
  ((count herding_agents) -
  herding_histo_intensity_0_025 -
  herding_histo_intensity_025_05 -
  herding_histo_intensity_05_075)
ifelse (count herding_agents > 0)
[
  set mean_hint
    ((herding_histo_intensity_0_025 * 1 / (count herding_agents)) +
    (herding_histo_intensity_025_05 * 2 / (count herding_agents)) +
    (herding_histo_intensity_05_075 * 3 / (count herding_agents)) +
    (herding_histo_intensity_075_1 * 4 / (count herding_agents)))
]
[ set mean_hint 0 ]

set herding_histo_independence_0_025
  (count herding_agents with [independence <= 0.25])
set herding_histo_independence_025_05
  count herding_agents with [independence <= 0.5])
set herding_histo_independence_025_05
  (herding_histo_independence_025_05 -
  herding_histo_independence_0_025)
set herding_histo_independence_05_075
  (count herding_agents with [independence <= 0.75])
set herding_histo_independence_05_075
  (herding_histo_independence_05_075 -
```

```
    herding_histo_independence_0_025 -
    herding_histo_independence_025_05)
  set herding_histo_independence_075_1
    ((count herding_agents) -
    herding_histo_independence_0_025 -
    herding_histo_independence_025_05 -
    herding_histo_independence_05_075)
  ifelse (count herding_agents > 0)
  [
    set mean_hind
      ((herding_histo_independence_0_025 * 1 / (count herding_agents)) +
      (herding_histo_independence_025_05 * 2 / (count herding_agents)) +
      (herding_histo_independence_05_075 * 3 / (count herding_agents)) +
      (herding_histo_independence_075_1 * 4 / (count herding_agents)))
  ]
  [ set mean_hind 0 ]

end

to update_patches

  ifelse ( count farming_agents > (count patches with [pcolor = green]) )
  [
    let to-paint
      (count farming_agents -
      (count patches with [pcolor = green]))
    repeat to-paint
    [
      ifelse (count patches with [pcolor = green] < (max-pycor + 1) )
      [ ask one-of patches with [pxcor = max-pxcor] [ set pcolor green] ]
      [
        ask one-of patches
          with [(pcolor = yellow) and
          (count neighbors with [pcolor = green] > 2)]
```

```
          [ set pcolor green ]
      ]
    ]
  ]
  [
    let to-paint
      ((count patches with [pcolor = green]) - count farming_agents)
    repeat to-paint
    [
      ifelse (any? patches with [(pcolor = green) and
        (count neighbors with [pcolor = yellow] > 2)])
      [
        ask one-of patches
        with [(pcolor = green) and
        (count neighbors with [pcolor = yellow] > 2)]
        [ set pcolor yellow ]
      ]
      [
        ask one-of patches with [pxcor = min-pxcor]
        [set pcolor yellow]
      ]
    ]
  ]

end
```

B.2.2 Nice Musical Chairs

A continuación, se presenta el código del modelo Nice Musical Chairs en lenguaje de NetLogo, correspondiente a la versión publicada en Angourakis et al. (2017). No se incluye aquí el código completo del archivo ".nlogo". La copia completa de ésta y cualquier versión posterior se puede encontrar en Angourakis (2017b) (https://www.openabm.org/model/4885/) y en GitHub (https://github.com/Andros-Spica/NiceMusicalChairs). Todas las versiones de este modelo están bajo la Licencia Pública General Reducida de GNU, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

A continuació, es presenta el codi del model Nice Musical Chairs en llenguatge de NetLogo, corresponent a la versió publicada en Angourakis et al. (2017). No s'inclou aquí el codi complet de l'arxiu ".nlogo". La còpia completa d'aquesta i qualsevol versió posterior es pot trobar a Angourakis (2017b) (https://www.openabm.org/model/4885/) i en GitHub (https://github.com/Andros-Spica/NiceMusicalChairs). Totes les versions d'aquest model estan sota la Llicència Pública General Reduïda de GNU, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

Next, the code of the Musical Chairs model is presented in NetLogo language, corresponding to the version published in Angourakis et al. (2017). The complete code of the ".nlogo" file is not included here. The full copy of this and any later version can be found at Angourakis (2017b) (https://www.openabm.org/model/4885/) and on GitHub (https://github.com/Andros-Spica/NiceMusicalChairs). All versions of this model are licensed under GNU General Public License, v.3 (GPL-3, https://www.gnu.org/licenses/lgpl-3.0.en.html).

```
;;;;;;;;;;;;;;;;;
;;;;; BREEDS ;;;;
;;;;;;;;;;;;;;;;;

breed [ groups group ]

breed [ pointers pointer ]

breed [ labelpositions labelposition ]

;;;;;;;;;;;;;;;;;
;;; VARIABLES ;;;
;;;;;;;;;;;;;;;;;

globals
[
  totalPatches

  ;;; modified parameters
  initH initF
  baseIntGrowth maxExtGrowth
  initGroups
  effectivenessGr
  maxGroupChangeRate
  opt optimalGrowthIncrease

  group_management group_pasture_tenure pairing

  ;;; variables used in resolve_conflict
  defender contender

  ;;; counters and final measures
  countLandUseF countLandUseH
  numberGroups
```

```
  FFcompetitions HHcompetitions HFcompetitions FHcompetitions
  landUseChangeEvents managementEvents
  farmingDemand farmingGrowth farmingDeterrence farmingBalance
  herdingDemand herdingGrowth herdingDeterrence herdingBalance
  meanGroupSize bigGroupSize
  meanGroupEffectiveness bigGroupEffectiveness
  bigTargetFarmingRatio meanTargetFarmingRatio
  meanFarmingIntegration meanHerdingIntegration meanMixedIntegration
]

groups-own
[
  groupSize groupEffectiveness
  intGrowthF intGrowthH
  farmingRatio targetFarmingRatio
  ;;; helpers
  groupSizeF groupSizeH
  groupDemandF groupDemandH
  groupDemandRemain
]

patches-own
[
  landUse myGroup
  contendersF contendersH
  withinIntegration betweenIntegration
]

pointers-own [ value ]

labelpositions-own [ name ]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SETUP ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to setup

  ;;; This procedure initializes the model

  clear-all

  set totalPatches count patches

  ;;; setup parameters depending on the type of experiment
  if (typeOfExperiment = "random")
  [
    ; set random seed
    let aSeed new-seed
    random-seed aSeed
    set seed aSeed


    ;;; randomly choose one scenario
    let listOfScenarios
    (list
      "Ao - open access, simple group dynamics"
      "Bo - open access, pairing"
      "Co - open access, management"
      "Do - open access, pairing and management"
      "Ar - restricted access, simple group dynamics"
      "Br - restricted access, pairing"
      "Cr - restricted access, management"
      "Dr - restricted access, pairing and management"
    )
    let randomIndex random 8
    set scenario item randomIndex listOfScenarios
    set baseIntGrowth 0.01 + random-float base_intrinsic_growth_rate
```

```
  set maxExtGrowth 0.001 + random-float max_extrinsic_growth_rate
  set opt random-float optimum
  set optimalGrowthIncrease random-float optimal_growth_increase
  set initGroups 1 + random initial_number_of_groups
  set maxGroupChangeRate random-float max_group_change_rate
  set effectivenessGr random-float totalPatches
  set initH random round ((init_herding / 100) * totalPatches)
  set initF random round ((init_farming / 100) * totalPatches)
]
if (typeOfExperiment = "defined by GUI")
[
  ; set random seed
  random-seed seed

  set baseIntGrowth base_intrinsic_growth_rate
  set maxExtGrowth max_extrinsic_growth_rate
  set opt optimum
  set optimalGrowthIncrease optimal_growth_increase
  set initGroups initial_number_of_groups
  set maxGroupChangeRate max_group_change_rate
  set effectivenessGr effectiveness_gradient
  set initH round ((init_herding / 100) * totalPatches)
  set initF round ((init_farming / 100) * totalPatches)
]
if (typeOfExperiment = "defined by expNumber")
[
  ; set random seed
  let aSeed new-seed
  random-seed aSeed
  set seed aSeed

  load-experiment
]
```

```
set pairing true
if (
  scenario = "Ao - open access, simple group dynamics" OR
  scenario = "Co - open access, management" OR
  scenario = "Ar - restricted access, simple group dynamics" OR
  scenario = "Cr - restricted access, management"
  )
[
  set optimalGrowthIncrease 0
  set pairing false
]

set group_pasture_tenure false
if (
  scenario = "Ar - restricted access, simple group dynamics" OR
  scenario = "Br - restricted access, pairing" OR
  scenario = "Cr - restricted access, management" OR
  scenario = "Dr - restricted access, pairing and management"
  )
[
  set group_pasture_tenure true
]

set group_management false
if (
  scenario = "Co - open access, management" OR
  scenario = "Do - open access, pairing and management" OR
  scenario = "Cr - restricted access, management" OR
  scenario = "Dr - restricted access, pairing and management"
  )
[
  set group_management true
]
```

```
ask patch 0 0
[
  sprout-groups initGroups
]

;;; set land use according to the parameter setting
;;; (position is arbitrary and has no consequence)
ask patches
[
  set landUse "N"
  set myGroup nobody
  set contendersF (turtle-set)
  set contendersH (turtle-set)
]
ask n-of initF patches
[
  set landUse "F"
]
ask n-of initH patches with [landUse = "N"]
[
  set landUse "H"
]
initialize-patches-and-groups

;;; initialize visualization

ask patch
  (min-pxcor + round ((max-pxcor - min-pxcor) * 0.97) )
  (min-pycor + round ((max-pycor - min-pycor) * 0.97) )
[
  sprout-labelpositions 1
  [ set name "scenario" set label scenario set shape "invisible" ]
]
if (display_details = true)
```

```
  [
    ask patch
      (min-pxcor + round ((max-pxcor - min-pxcor) * 0.97) )
      (min-pycor + round ((max-pycor - min-pycor) * 0.03) )
    [
      sprout-labelpositions 1
      [ set name "time" set label "time: 0" set shape "invisible" ]
    ]
    ask patch
      (min-pxcor + round ((max-pxcor - min-pxcor) * 0.3) )
      (min-pycor + round ((max-pycor - min-pycor) * 0.03) )
    [
      sprout-labelpositions 1
      [ set name "farming" set shape "invisible" ]
    ]
    ask patch
      (min-pxcor + round ((max-pxcor - min-pxcor) * 0.72) )
      (min-pycor + round ((max-pycor - min-pycor) * 0.03) )
    [
      sprout-labelpositions 1
      [ set name "bigGroupSize" set shape "invisible" ]
    ]

  ]

  update-visualization

  reset-ticks

end

to initialize-patches-and-groups

  ;;; This procedure initializes patch and group variables
```

```
  ask patches
  [
      if (landUse != "N") [ set myGroup (one-of groups) ]
  ]

  ask groups
  [
    set hidden? true
    move-to one-of patches
      with [any? groups-here = false and (pxcor > 2) and (pycor > 2) and
        (pxcor < max-pxcor - 2) and (pycor < max-pycor - 2)]
    set targetFarmingRatio random-float 1
    ;set targetFarmingRatio FarmingRatio ;;; alternative initialization
    update-group
  ]

  ask patches
  [
    update-landUnits
  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CYCLE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to go

  ;;; This procedure is the cycle of the model (what happens during one "tick").

  reset-counters
```

```
  growth

  landUse-expansion

  check-competitions

  change-groups

  if (group_management = true) [ group-management ]

  update-visualization

  tick
  if (display_details = true)
  [ ask labelpositions with [ name = "time" ]
  [ set label (word "time: " ticks) ] ]
  if ticks > endSimulation [stop]

end

to reset-counters

  ;;; This procedure reset all counters which are used
  ;;; either during the cycle or summarized at the
  ;;; "update-visualization" procedure.

  set farmingGrowth 0
  set farmingDeterrence 0
  set herdingGrowth 0
  set herdingDeterrence 0
  set FFcompetitions 0
  set HHcompetitions 0
  set HFcompetitions 0
  set FHcompetitions 0
```

```
  set landUseChangeEvents 0
  set managementEvents 0

  ask groups [ set groupDemandF 0 set groupDemandH 0 ]

end

to growth

  ;;; This procedure calculates the groups demands for each
  ;;; land use class, based on both the intrinsic and
  ;;; extrinsic growth rates of each of them.
  ;;; Note that growth rates are dependent on parameters,
  ;;; but also on the context, and may vary from one "tick" to another.

  ask groups [ set groupDemandF 0 set groupDemandH 0 ]
  ;;; FARMING
  ;;; Intrinsic Demand
  ask groups with [groupSize > 0]
  [
    let myLand count patches with [landUse = "F" and myGroup = myself]
    repeat myLand
    [
      if ( random-float 1 <= intGrowthF )
      [
        set groupDemandF groupDemandF + 1
      ]
    ]
  ]
  ;;; Extrinsic Demand
  let extF (round (maxExtGrowth * ( totalPatches - countLandUseF ) ) )
  repeat extF
  [
    ask one-of groups
```

```
    [
      set groupDemandF groupDemandF + 1
    ]
  ]
  ;;; HERDING
  ;;; Intrinsic Growth
  ask groups with [groupSize > 0]
  [
    let myLand count patches with [landUse = "H" and myGroup = myself]
    repeat myLand
    [
      if ( random-float 1 <= intGrowthH )
      [
        set groupDemandH groupDemandH + 1
      ]
    ]
  ]
  ;;; Extrinsic Growth
  let extH (round (maxExtGrowth * ( totalPatches - countLandUseH ) ) )
  repeat extH
  [
    ask one-of groups
    [
      set groupDemandH groupDemandH + 1
    ]
  ]

end

to landUse-expansion

  ;;; This procedure calls for the expansion procedures of
  ;;; farming and herding, intentionally in this order.
```

```
  farming-expansion
  herding-expansion

end

to farming-expansion

  ;;; In this procedure, groups attempt to assign patches to
  ;;; their new farming units.
  ;;; If there is no patch freely available, groups will randomly
  ;;; choose a patch, and
  ;;; if this patch belongs to another group (density-dependent
  ;;; growth), the group will generate a competitive situation and
  ;;; be accounted within "contendersF".
  ;;; In the case that the patch is used for "herding" and
  ;;; "group_pasture_tenure = false", the group will automatically
  ;;; occupy it and change its land use to farming.

  let growingGroupsF groups with [ groupDemandF > 0 ]
  ask growingGroupsF [ set groupDemandRemain groupDemandF ]

  repeat sum [groupDemandRemain] of growingGroupsF
  [
    ask one-of growingGroupsF with [groupDemandRemain > 0]
    [
      let me self
      ifelse (any? patches with [myGroup = nobody])
      [
        ;;; if the land is not saturated
        ask one-of patches with [myGroup = nobody]
        [
          if (landUse = "N")
          [ set landUseChangeEvents landUseChangeEvents + 1 ]
          set myGroup me set landUse "F"
```

```
  ]
]
[
  ;;; if the territory is saturated
  ;;; Choose a random patch
  ask one-of patches
  [
    ;;; if the patch is used by another group
    if (myGroup != me)
    [
      ifelse (landUse = "F") [
        if ( allow_within-class_competition = true )
        [
          ;;; If the patch is used for farming,
          ;;; F-F competition will be called later
          set contendersF (turtle-set contendersF me)
          set FFcompetitions (FFcompetitions + 1)
        ]
      ]
      [
        ifelse (group_pasture_tenure = true)
        [
          ;;; F-H competition will be called later
          set contendersF (turtle-set contendersF me)
          set FHcompetitions (FHcompetitions + 1)
        ]
        [
          ;;; farming will start using a former pasture
          set myGroup myself
          set landUSe "F"
          set landUseChangeEvents landUseChangeEvents + 1
          set farmingGrowth farmingGrowth + 1
          set herdingDeterrence herdingDeterrence + 1
        ]
```

```
            ]
          ]
        ]
      ]
      set groupDemandRemain groupDemandRemain - 1
    ]
  ]

end

to herding-expansion

  ;;; In this procedure, groups attempt to assign patches
  ;;; to all their herding units (if "group_pasture_tenure = false") or
  ;;; to their new herding units (group_pasture_tenure = true).
  ;;; If there is no patch freely available, groups will randomly
  ;;; choose a patch, and if this patch belongs to another group
  ;;; (density-dependent growth), the group will generate a
  ;;; competitive situation and be accounted within "contendersH"
  let groupsH nobody
  let herds 0
  ifelse (group_pasture_tenure = true)
  [
    set groupsH groups with [ groupDemandH > 0 ]
    ask groupsH [ set groupDemandRemain groupDemandH ]
    set herds sum [groupDemandRemain] of groupsH
  ]
  [
    set groupsH groups with [ farmingRatio < 1 ]
    ask groupsH
    [
      let me self
      set groupDemandRemain groupSizeH + groupDemandH
    ]
```

```
  set herds sum [ groupDemandRemain ] of groups

  ;;; reset herding positions (herds go back not necessarily to
  ;;; the same patch)
  ask patches with [ landUse = "H"] [ set myGroup nobody ]
]

repeat herds
[
  ask one-of groupsH with [ groupDemandRemain > 0 ]
  [
    let me self
    ifelse (any? patches with [myGroup = nobody])
    [
      ;;; if the land is not saturated
      ask one-of patches with [myGroup = nobody]
      [
        if (landUse != "H")
        [
          set landUse "H"
          set landUseChangeEvents landUseChangeEvents + 1
          set herdingGrowth herdingGrowth + 1
        ]
        set myGroup me
      ]
    ]
    [
      ;;; if the territory is saturated
      ;;; Choose a random patch
      ask one-of patches
      [
        ;;; Fit-to-maximum exclusion, Density-dependent exclusion
        if (myGroup != me)
        [
```

```
            ifelse (landUse = "F")
            [
              ;;; a H-F competition will be called later
              set contendersH (turtle-set contendersH me)
              set HFcompetitions (HFcompetitions + 1)
            ]
            [
              if (allow_within-class_competition = true)
              [
                ;;; a H-H competition will be called later
                set contendersH (turtle-set contendersH me)
                set HHcompetitions (HHcompetitions + 1)
              ]
            ]
          ]
        ]
      ]
      set groupDemandRemain groupDemandRemain - 1
    ]
  ]

  ;;; rangelands not claimed will be considered free land (no land use)
  if (any? patches with [landUse = "H" and myGroup = nobody] )
  [
    ask patches with [landUse = "H" and myGroup = nobody]
    [ set landUse "N" ]
  ]

end

to check-competitions

  ;;; This procedure calls, in a particular sequence,
  ;;; for the resolution of all competitive situations generated by
```

```
  ;;; farming and herding expansions.

  ;;; Due to their sedentary condition, farming contenders will
  ;;; act first (F-F and F-H -> H-H and H-F)

  ;;; Farming stakeholders prefer to acquire other groups' farmlands (F-F),
  ;;; rather than investing in new infraestructures (F-H)

  check-FFcompetitions
  check-FHcompetitions

  ;;; Herding stakeholders prefer to acquire other groups' pastures (H-H),
  ;;; rather than converting farmlands by  violence or negotiation (H-F)

  check-HHcompetitions
  check-HFcompetitions

  ask groups [ update-group ]

end

to check-FFcompetitions

  ;;; farming-farming competition

  ask patches with [ landUse = "F" and any? contendersF ]
  [
;    print "F-F"
    ;;; the center assigned is the one that is effectively using the land
    set defender myGroup
    repeat count contendersF
    [
      set contender one-of contendersF
      ;;; remove contender from the respective contenders agent-set
```

```
      set contendersF contendersF with [self != contender]
      ;print (word "contendersF after: " contendersF)
      resolve-competition "FF"
    ]
  ]

end

to check-FHcompetitions

  ;;; farming-herding competition

  ask patches with [ landUse = "H" and any? contendersF ]
  [
;    print "F-H"
    set defender myGroup
    repeat count contendersF
    [
      set contender one-of contendersF
      ;;; remove contender from the respective contenders agent-set
      set contendersF contendersF with [self != contender]
;      print (word "contendersH after: " contendersH)
      resolve-competition "FH"
    ]
  ]

end

to check-HHcompetitions

  ;;; herding-herding competition

  ask patches with [ landUse = "H" and any? contendersH ]
  [
```

```
;    print "H-H"
    ;;; Since their schedule may vary, a herding center is
    ;;; assigned randomly among the contenders to be the one
    ;;; arriving first (defender)
    set defender myGroup
    repeat count contendersH
    [
      set contender one-of contendersH
      ;;; remove contender from the respective contenders agent-set
      set contendersH contendersH with [self != contender]
;      print (word "contendersH after: " contendersH)
      ;;; check if any of contenders still exists.
      ;;; If so, then resolve competition
      if ([groupSize] of contender > 0)
      [ resolve-competition "HH"]
    ]
  ]

end

to check-HFcompetitions

  ;;; farming-herding competition

  ask patches with [ landUse = "F" and any? contendersH ]
  [
;    print "H-F"
    set defender myGroup
    repeat count contendersH
    [
      set contender one-of contendersH
      ;;; remove contender from the respective contenders agent-set
      set contendersH contendersH with [self != contender]
;      print (word "contendersH after: " contendersH)
```

```
      resolve-competition "HF"
    ]
  ]

end

to resolve-competition [ typeOfComp ]

  ;;; This procedure resolves the current competitive situation,
  ;;; and calculate the consequences of contenders success according to
  ;;; "typeOfComp" ("FF"=farming-farming, "FH"=farming-herding,
  ;;; "HH"=herding-herding, "HF"=herding-farming).

  ;;; set competition conditions
  ; define intensities
  let supportDef get-group-influence defender
  let supportCon get-group-influence contender
;  print (word defender " vs " contender ")
;  print (word "supportDef: " supportDef " ; supportCon: " supportCon)

  ;;; the contender is discarted if both defender and
  ;;; contender have zero strength at this patch
  if (supportCon + supportDef > 0)
  [
    ;;; a contender is the one attempting to expand,
    ;;; thus it is the one to make a informed decision
    let ratio_of_intensities  (supportCon /(supportCon + supportDef))

    ;;; Does the competitive situation evolves into land use change event?
    if ( random-float 1 < ratio_of_intensities)
    [
      ;;; extending whichever land use is encouraged
      set myGroup contender
```

```
      ;;; update landUse
      if (typeOfComp = "HF")
      [
  ;      print "herding wins"
        set landUse "H"
        ;;; Hence, there is land use change
        set landUseChangeEvents landUseChangeEvents + 1
        set herdingGrowth herdingGrowth + 1
        set farmingDeterrence farmingDeterrence + 1
      ]
      if (typeOfComp = "FH")
      [
  ;      print "farming wins"
        set landUse "F"
        ;;; Hence, there is land use change
        set landUseChangeEvents landUseChangeEvents + 1
        set farmingGrowth farmingGrowth + 1
        set herdingDeterrence herdingDeterrence + 1
      ]
    ]
  ]


end

to change-groups

  ;;; In this procedure, every patch of every group test their
  ;;; particular probability of changing to another group,
  ;;; which may be an existing group or a new one collecting all
  ;;; the defecting patches of a group (fission).
  ;;; The criterium to leave and choose a group is the competitive
  ;;; strength or influence that groups have in the patch at hand
  ;;; (size * effectiveness)
```

```
ask groups
[
  ;;; each patch of a group will assess their will (maxGroupChangeRate)
  ;;; and their freedom, which is inversely related to the group
  ;;; effectiveness (1 - ([groupEffectiveness] of myGroup) ),
  ;;; to change groups, possibly forming a new group.
  let me self
  let myLand patches with [myGroup = me]
  let defectingPatches (patch-set nobody)
  ask myLand
  [
    if ( random-float 1 < maxGroupChangeRate * (1 - ([groupEffectiveness] of myGroup
    [
      set defectingPatches (patch-set defectingPatches self)
    ]
  ]
  if (any? defectingPatches)
  [
    ;;; if there are any patches defecting from this group...
    ;;; the viability of the possible new group is calculated
    ;;; for each patch and compared to the most influent group
    let newGroup nobody
    let influenceNewGroup
      (count defectingPatches) *
      e ^ ( - (count defectingPatches) /
          (effectivenessGr * totalPatches) )
    let mostInfluentGroup
      max-one-of groups [groupSize * groupEffectiveness]
    let influenceOtherGroup get-group-influence mostInfluentGroup
    ifelse (influenceOtherGroup > influenceNewGroup)
    [
      ask defectingPatches [ set myGroup mostInfluentGroup ]
    ]
```

```
[
  if (newGroup = nobody)
  [
    ifelse (any? groups with [groupSize = 0])
    [
      ask one-of groups with [groupSize = 0]
      [
        set targetFarmingRatio ([targetFarmingRatio] of me)
        set newGroup self
      ]
    ]
    [
      hatch-groups 1
      [
      ;;; new groups inherit the traits of the original group
      ;;; *** or modify them given a mutation parameter
      set hidden? true
      move-to one-of patches
        with [
          any? groups-here = false and (pxcor > 2) and
          (pycor > 2) and (pxcor < max-pxcor - 2) and
          (pycor < max-pycor - 2)
        ]
      ;;; random mutation
      ;set targetFarmingRatio
        min (list
          1
          max (list
            0
            (([targetFarmingRatio] of me) + (0.1 - random-float 0.2))
          )
        )
      ;;; following the optimal
      ;set targetFarmingRatio
```

```
              ([targetFarmingRatio] of me) +
              0.1 * (opt - targetFarmingRatio)
            ;;; following tradition
            set targetFarmingRatio ([targetFarmingRatio] of me)
            set newGroup self
;            print (word
                   " Group fission: " me
                   " (groupSize=" count myLand
                   ") splits into " me
                   " (groupSize=" (count myLand - count defectingPatches)
                   ") and " newGroup
                   " (groupSize=" count defectingPatches ")"
                   )
            ]
          ]
        ]
        ask defectingPatches [ set myGroup newGroup ]
      ]
    ]
    update-group
  ]

end

to-report get-group-influence [ theGroup ]

  report [groupSize * groupEffectiveness] of theGroup

end

to group-management

  ;;; In this procedure, groups with more than one member
  ;;; calculate the difference between their "farmingRatio"
```

```
  ;;; and their "targetFarmingRatio",
  ;;; and attempt to change the land use of the respective
  ;;; number of patches (note that "floor" is used),
  ;;; with a success proportional to their "groupEffectiveness".

  ask groups
  [
    if (groupSize > 1)
      [
        let dif ((farmingRatio - targetFarmingRatio) * groupSize)
        let num floor (abs dif * groupEffectiveness)
;        print (word
                self
                " -> farmingRatio: " precision farmingRatio 4
                " Â¦Â¦ targetFarmingRatio: " precision targetFarmingRatio 4
                " Â¦Â¦ groupSize: " groupSize
                " Â¦Â¦ groupEffectiveness: " precision groupEffectiveness 4
                " Â¦Â¦ dif: " dif " Â¦Â¦ num: " num
                )
        if (num > 0)
        [
          ;;; if it is greater than target
          ifelse ( dif > 0 )
          [
            ask n-of num patches
              with [landUse ="F" and myGroup = myself]
            [
              set landUse "H"
              ;;; Hence, there is land use change
              set landUseChangeEvents landUseChangeEvents + 1
              set herdingGrowth herdingGrowth + 1
              set farmingDeterrence farmingDeterrence + 1
              set managementEvents managementEvents + 1
            ]
```

```
          ]
          [
            ;;; if it is smaller than target
            if ( dif < 0 )
            [
              ask n-of num patches
                with [landUse ="H" and myGroup = myself]
              [
                set landUse "F"
                ;;; Hence, there is land use change
                set landUseChangeEvents landUseChangeEvents + 1
                set farmingGrowth farmingGrowth + 1
                set herdingDeterrence herdingDeterrence + 1
                set managementEvents managementEvents + 1
              ]
            ]
          ]
        ]
      ]
      update-group
  ]

end

to update-group

  ;;; This procedure updates group variables
  ;;; (groupSize, groupEffectiveness,
  ;;; farmingRatio, intGrowthF, intGrowthH).

  set farmingRatio 0
  set groupSize count patches with [myGroup = myself]
  set groupSizeF count patches
    with [myGroup = myself and landUse = "F"]
```

```
  set groupSizeH count patches
    with [myGroup = myself and landUse = "H"]
  if (groupSize > 0)
  [
    set farmingRatio
      ( count patches
        with [landUse = "F" and myGroup = myself] / groupSize )
  ]
  set groupEffectiveness
    e ^ ( - groupSize / (effectivenessGr * totalPatches) )

  ;;; calculate modified growth out of the group distance from the optimal
  let d 0
  ifelse (farmingRatio < opt)
  [
    set d (farmingRatio / opt)
    set intGrowthF baseIntGrowth * (1 + (optimalGrowthIncrease / 100))
    set intGrowthH baseIntGrowth * (1 + (optimalGrowthIncrease / 100) * d)
  ]
  [
    ifelse (opt = 1)
    [ set d 1 ]
    [ set d 1 - ((farmingRatio - opt) / (1 - opt)) ]
    set intGrowthH baseIntGrowth * (1 + (optimalGrowthIncrease / 100))
    set intGrowthF baseIntGrowth * (1 + (optimalGrowthIncrease / 100) * d)
  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; VISUALIZATION ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to update-landUnits
```

```
  ;;; This procedure updates the patches' "withinIntegration"
  ;;; and "betweenIntegration".

  ask patches with [myGroup != nobody]
  [
    ifelse (landUse = "F")
    [
      set withinIntegration [farmingRatio] of myGroup
      set betweenIntegration 1 - [farmingRatio] of myGroup
    ]
    [
      set withinIntegration 1 - [farmingRatio] of myGroup
      set betweenIntegration [farmingRatio] of myGroup
    ]
  ]

end

to update-visualization

  ;;; this procedure updates the display and all global output variables.

  if (display_mode = "land use proportion")
  [
    update-patches
    if (display_details = true)
    [ ask labelpositions [set label-color black] ]
  ]
  if (display_mode = "groups")
  [
    update-network
    if (display_details = true)
    [ ask labelpositions [set label-color white] ]
```

```
]

set numberGroups count groups with [groupSize > 0]
set countLandUseF count patches with [ landUse = "F" ]
set countLandUseH count patches with [ landUse = "H" ]

set farmingBalance (farmingGrowth - farmingDeterrence)
set herdingBalance (herdingGrowth - herdingDeterrence)

set meanGroupSize
  mean [[groupSize] of myGroup] of patches with [ landUse != "N" ]
set bigGroupSize
  [groupSize] of max-one-of groups [groupSize]
set meanGroupEffectiveness
  mean [[groupEffectiveness] of myGroup] of patches with [ landUse != "N" ]
set bigGroupEffectiveness
  [groupEffectiveness] of max-one-of groups [groupSize]

set meanTargetFarmingRatio
  mean [[targetFarmingRatio] of myGroup] of patches with [ landUse != "N" ]
set bigTargetFarmingRatio
  [targetFarmingRatio] of max-one-of groups [groupSize]

update-landUnits

ifelse (any? patches with [landUse = "F"])
[
  set meanFarmingIntegration
    mean [withinIntegration] of patches with [landUse = "F"]
]
[ set meanFarmingIntegration -0.01 ]
ifelse (any? patches with [landUse = "H"])
[
  set meanHerdingIntegration
```

```
      mean [withinIntegration] of patches with [landUse = "H"]
  ]
  [ set meanHerdingIntegration -0.01 ]

  set meanMixedIntegration
    mean [betweenIntegration] of patches with [ landUse != "N" ]

  ifelse (display_details = true)
  [
    ask labelpositions with [ name = "farming" ]
    [
      set label
        (word
          "farming(%): "
          (precision (100 * countLandUseF / totalPatches) 2)
        )
    ]
    ask labelpositions with [ name = "bigGroupSize" ]
    [
      set label
        (word
          "biGroupSize(%): "
          (precision (100 * bigGroupSize / totalPatches) 2)
        )
    ]
  ]
  [
    ask labelpositions with [ name = "time" ] [ set label "" ]
    ask labelpositions with [ name = "farming" ] [ set label "" ]
    ask labelpositions with [ name = "bigGroupSize" ] [ set label "" ]
  ]

end
```

```
to update-patches

  ;;; this procedure updates the "land use proportion" display mode.

  ask pointers [die]
  ask groups [set hidden? true]
  ask patches
  [
    set pcolor brown
    if (landUse = "F")
    [ set pcolor green ]
    if (landUse = "H")
    [ set pcolor yellow ]
  ]

end

to update-network

  ;;; this procedure updates the "groups" display mode.

  ask pointers [die]
  ask groups
  [
    ifelse (groupSize > 0)
    [
      set hidden? false
      set color red
      set shape "circle"
      set size 0.5
    ]
    [ set hidden? true ]
    create-links-with other groups [ set color black ]
  ]
```

```
  layout-spring groups links 0.18 9 1.2
  ask links [die]
  ask patches
  [
    set pcolor black
    let thisPatch self
    if (landUse != "N" )
    [
      sprout-pointers 1 [
        set shape "circle" set size 0.2
        ifelse ([landUse] of patch-here = "F")
        [ set color green ]
        [ set color yellow ]
        create-link-with myGroup [ set color grey]
        move-to myGroup
      ]
    ]
  ]
  ask groups
  [
    let num groupSize
    repeat groupSize [
      ask one-of link-neighbors [
        rt 360 * who
        fd 0.1 * num * e ^ (- num / 60)
      ]
      set num num - 1
    ]
  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;; Parametrization from file ;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to load-experiment

  ;;; this procedure loads the values of each (explored)
  ;;; parameter from a csv file.
  ;;; Note that the setup will use the value set by the user
  ;;; for any other parameter (e.g. scenario).

  ;;; create folders in the model's directory
  ;;; before trying to load experiments
  let FilePath
    "SensAnalysis//exp//"
  ;;; the parameter setting of experiments must be saved
  ;;; as ".csv" files named "exp_<NUMBER>.csv"
  let filename
    (word FilePath "exp_" expNumber ".csv")
  file-open filename
  while [not file-at-end?]
    [
      ;;; the values of the file must follow this same order

      set initH round ((file-read / 100) * totalPatches)
      set initF round ((file-read / 100) * totalPatches)
      set baseIntGrowth file-read
      set maxExtGrowth file-read
      set initGroups file-read
      set effectivenessGr file-read
      set maxGroupChangeRate file-read
      set opt file-read
      set optimalGrowthIncrease file-read

      ;; use this to cut down the time of simulation
```

```
      ;; (e.g. if the file reads 2000)
      set endSimulation file-read
    ]
  file-close

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;; movie generation ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to generate-animation

  ;;; this procedure generates a video sequencing the displays of
  ;;; a simulation (using the current parameter configuration).

  setup
  ;;; you can add more information in the name of the file
  ;;; (here, only scenario is used)
  movie-start (word scenario ".mov")
  repeat endSimulation [ go movie-grab-view ]
  movie-close

end
```