Package ‘gaussplotR’

July 22, 2025

Type Package

Title Fit, Predict and Plot 2D Gaussians

Version 0.2.5

Description Functions to fit two-dimensional Gaussian functions, predict values from fits, and produce plots of predicted data via either 'ggplot2' or base R plotting.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

Imports ggplot2 (>= 3.3.0), metR (>= 0.7.0), rgl, viridisLite

Suggests lattice, knitr, rmarkdown, testthat

Depends R (>= 3.3.0)

URL https://github.com/vbaliga/gaussplotR

BugReports https://github.com/vbaliga/gaussplotR/issues

VignetteBuilder knitr

NeedsCompilation no

Author Vikram B. Baliga [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-9367-8974>)

Maintainer Vikram B. Baliga <vbaliga87@gmail.com>

Repository CRAN

Date/Publication 2021-05-02 20:10:02 UTC

Contents

`autofit_gaussian_2D` *Automatically determine the best-fitting 2D-Gaussian for a data set*

Description

Automatically determine the best-fitting 2D-Gaussian for a data set

Usage

```
autofit_gaussian_2D(
  data,
  comparison_method = "rmse",
  maxiter = 1000,
  simplify = TRUE
)
```

Arguments

`data` A data.frame that contains the raw data (generally rectilinearly gridded data, but this is not a strict requirement). Columns must be named `"X_values"`, `"Y_values"` and `"response"`.

`comparison_method` One of "rmse", "rss", or "AIC"; what metric should be used to determine the "best-fitting" Gaussian?

`maxiter` Default 1000. A positive integer specifying the maximum number of iterations allowed. See `stats::nls.control()` for more details.

`simplify` TRUE or FALSE. If TRUE, return only the coefficients, model, model_error_stats, and fit_method for the best-fitting model. If FALSE, a model comparison table is also included in the returned list as `$model_comparison`. This table is obtained via `compare_gaussian_fits()`.

Details

This function runs `fit_gaussian_2D()` three times: once for each of the "main" types of models: 1) elliptical, unconstrained; 2) elliptical, log; 3) circular. In all three cases, amplitudes and orientations are unconstrained. The function `compare_gaussian_fits()` is then used to determine which of these three models is the best-fitting, using the `comparison_method` argument to make the decision.

Value

If `simplify = TRUE`, a list with the components:

- "coefs" A data.frame of fitted model parameters.
- "model" The model object, fitted by `stats::nls()`.
- "model_error_stats" A data.frame detailing the rss, rmse, deviance, and AIC of the fitted model.
- "fit_method" A character vector that indicates which method and orientation strategy was used by this function.

If `simplify = FALSE`, a model comparison table is also included in the returned list as `$model_comparison`. This table is obtained via `compare_gaussian_fits()`.

Author(s)

Vikram B. Baliga

Examples

```
if (interactive()) {
}
```

`characterize_gaussian_fits` *Characterize the orientation of fitted 2D-Gaussians*

Description

The orientation and partial correlations of Gaussian data are analyzed according to Levitt et al. 1994 and Priebe et al. 2003. Features include computation of partial correlations between response variables and independent and diagonally-tuned predictions, along with Z-difference scoring.

Usage

```
characterize_gaussian_fits(
  fit_objects_list = NULL,
  data = NULL,
  constrain_amplitude = FALSE,
  ...
)
```

Arguments

`fit_objects_list`
: A list of outputs from `fit_gaussian_2D()`. See Details for more. This is the preferred input object for this function.

`data`
: A data.frame that contains the raw data (generally rectilinearly gridded data, but this is not a strict requirement). Columns must be named `"X_values"`, `"Y_values"` and `"response"`. See `fit_gaussian_2D()` for details.

`constrain_amplitude`
: Default FALSE; should the amplitude of the Gaussian be set to the maximum value of the `"response"` variable (`TRUE`), or should the amplitude fitted by `stats::nls()` (`FALSE`)? See `fit_gaussian_2D()` for details.

`...`
: Additional arguments that can be passed to `fit_gaussian_2D()` if data are supplied.

Details

This function accepts either a list of objects output from `fit_gaussian_2D()` (preferred) or a data.frame that contains the raw data.

The supplied fit_objects_list must be a list that contains objects returned by `fit_gaussian_2D()`. This list must contain exactly three models. All three models must have been run using `method = "elliptical_log"`. The models must be: 1) one in which orientation is unconstrained, 2) one in which orientation is constrained to Q = 0 (i.e. a diagonally-oriented Gaussian), and 3) one in which orientation is constrained to Q = -1 (i.e. a horizontally-oriented Gaussian). See this function's Examples for guidance.

Should raw data be provided instead of the fit_objects_list, the `characterize_gaussian_fits()` runs `fit_gaussian_2D()` internally. This is generally not recommended, as difficulties in fitting models via `stats::nls()` are more easily troubleshot by the optional arguments in `fit_gaussian_2D()`. Nevertheless, supplying raw data instead of a list of fitted models is feasible, though your mileage may vary.

Value

A list with the following:

- "model_comparison" A model comparison output (i.e. what is produced by `compare_gaussian_fits()`), which indicates the relative preference of each of the three models.
- "Q_table" A data.frame that provides information on the value of Q from the best-fitting model, along with the 5-95% confidence intervals of this estimate.
- "r_i" A numeric, the correlation of the data with the independent (Q = -1) prediction.
- "r_s" A numeric, the correlation of the data with the diagonally- oriented (Q = 0) prediction.
- "r_is" A numeric, the correlation between the independent (Q = -1) prediction and the the diagonally-oriented (Q = 0) prediction.
- "R_indp" A numeric, partial correlation of the response variable with the independent (Q = -1) prediction.
- "R_diag" A numeric, partial correlation of the response variable with the diagonally-oriented (Q = 0) prediction.

- "ZF_indp" A numeric, the Fisher Z-transform of the R_indp coefficient. See Winship et al. 2006 for details.
- "ZF_diag" A numeric, the Fisher Z-transform of the R_diag coefficient. See Winship et al. 2006 for details.
- "Z_diff" A numeric, the Z-difference between ZF_indp and ZF_diag. See Winship et al. 2006 for details.

Author(s)

Vikram B. Baliga

References

Levitt JB, Kiper DC, Movshon JA. Receptive fields and functional architecture of macaque V2. J Neurophysiol. 1994 71:2517–2542.

Priebe NJ, Cassanello CR, Lisberger SG. The neural representation of speed in macaque area MT/V5. J Neurosci. 2003 Jul 2;23(13):5650-61. doi: 10.1523/JNEUROSCI.23-13-05650.2003.

Winship IR, Crowder N, Wylie DRW. Quantitative reassessment of speed tuning in the accessory optic system and pretectum of pigeons. J Neurophysiol. 2006 95(1):546-551. doi: 10.1152/jn.00921.2005

Examples

```
if (interactive()) {
  library(gaussplotR)

  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]

  ## Fit the three required models
  gauss_fit_uncn <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_amplitude = FALSE,
      constrain_orientation = "unconstrained"
    )

  gauss_fit_diag <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_amplitude = FALSE,
      constrain_orientation = 0
    )

  gauss_fit_indp <-
```

```
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_amplitude = FALSE,
      constrain_orientation = -1
    )

  ## Combine the outputs into a list
  models_list <-
    list(
      gauss_fit_uncn,
      gauss_fit_diag,
      gauss_fit_indp
    )

  ## Now characterize
  out <-
    characterize_gaussian_fits(models_list)
  out

  ## Alternatively, the raw data itself can be supplied.
  ## This is less preferred, as fitting of models may fail
  ## internally.
  out2 <-
    characterize_gaussian_fits(data = samp_dat)

  ## This produces the same output, assuming models are fit without error
  identical(out, out2)
}
```

`compare_gaussian_fits` *Compare fitted 2D-Gaussians and determine the best-fitting model*

Description

Compare fitted 2D-Gaussians and determine the best-fitting model

Usage

```
compare_gaussian_fits(fit_objects_list, comparison_method = "rmse")
```

Arguments

`fit_objects_list`
: A list of outputs from `fit_gaussian_2D()`. See Details for more

`comparison_method`
: One of "rmse", "rss", or "AIC"; what metric should be used to determine the "best-fitting" Gaussian?

Details

For the argument `fit_objects_list`, a list of fitted model objects (output from `fit_gaussian_2D()`) can simply be combined via `list()`. Naming the list is optional; should you supply names, the output of `compare_gaussian_fits()` will refer to specific models by these names.

Value

A list with the components:

- "preferred_model" A character indicating the name of the preferred model (or if a named list was not provided, a model number is given in the order of the original supplied list).
- "comparison_table" A data.frame detailing the rss, rmse, deviance, and AIC of the fitted models. The data.frame is sorted by the comparison_method that was selected.

Author(s)

Vikram B. Baliga

Examples

```
if (interactive()) {
  library(gaussplotR)

  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]

  ## Fit a variety of different models
  gauss_fit_ue <-
    fit_gaussian_2D(samp_dat)
  gauss_fit_uel <-
    fit_gaussian_2D(samp_dat, method = "elliptical_log")
  gauss_fit_cir <-
    fit_gaussian_2D(samp_dat, method = "circular")

  ## Combine the outputs into a list
  models_list <-
    list(
      unconstrained_elliptical = gauss_fit_ue,
      unconstrained_elliptical_log = gauss_fit_uel,
      circular = gauss_fit_cir
    )

  ## Compare via rmse
  models_compared <-
    compare_gaussian_fits(
      fit_objects_list = models_list,
      comparison_method = "rmse" ## the default
```

```
    )
}
```

fit_gaussian_2D   *Determine the best-fit parameters for a specific 2D-Gaussian model*

Description

Determine the best-fit parameters for a specific 2D-Gaussian model

Usage

```
fit_gaussian_2D(
  data,
  method = "elliptical",
  constrain_amplitude = FALSE,
  constrain_orientation = "unconstrained",
  user_init = NULL,
  maxiter = 1000,
  verbose = FALSE,
  print_initial_params = FALSE,
  ...
)
```

Arguments

`data` A data.frame that contains the raw data (generally rectilinearly gridded data, but this is not a strict requirement). Columns must be named `"X_values"`, `"Y_values"` and `"response"`.

`method` Choice of `"elliptical"`, `"elliptical_log"`, or `"circular"`. Determine which specific implementation of 2D-Gaussian to use. See Details for more.

`constrain_amplitude` Default FALSE; should the amplitude of the Gaussian be set to the maximum value of the `"response"` variable (`TRUE`), or should the amplitude fitted by `stats::nls()` (`FALSE`)?

`constrain_orientation` If using `"elliptical"` or `method = "elliptical_log"`, should the orientation of the Gaussian be unconstrained (i.e. the best-fit orientation is returned) or should it be pre-set by the user? See Details for more. Defaults to `"unconstrained"`.

`user_init` Default NULL; if desired, the user can supply initial values for the parameters of the chosen model. See Details for more.

`maxiter` Default 1000. A positive integer specifying the maximum number of iterations allowed. See `stats::nls.control()` for more details.

`verbose` TRUE or FALSE; should the trace of the iteration be printed? See the `trace` argument of `stats::nls()` for more detail.

print_initial_params
: TRUE or FALSE; should the set of initial parameters supplied to `stats::nls()` be printed to the console? Set to FALSE by default to avoid confusion with the fitted parameters attained after using `stats::nls()`.

... Additional arguments passed to `stats::nls.control()`

Details

`stats::nls()` is used to fit parameters for a 2D-Gaussian to the supplied data. Each method uses (slightly) different sets of parameters. Note that for a small (but non-trivial) proportion of data sets, nonlinear least squares may fail due to singularities or other issues. Most often, this occurs because of the starting parameters that are fed in. By default, this function attempts to set default parameters by making an educated guess about the major aspects of the supplied data. Should this strategy fail, the user can make use of the `user_init` argument to supply an alternate set of starting values.

The simplest method is `method = "circular"`. Here, the 2D-Gaussian is constrained to have a roughly circular shape (i.e. spread in X- and Y- are roughly equal). If this method is used, the fitted parameters are: Amp (amplitude), X_peak (x-axis peak location), Y_peak (y-axis peak location), X_sig (spread along x-axis), and Y_sig (spread along y-axis).

A more generic method (and the default) is `method = "elliptical"`. This allows the fitted 2D-Gaussian to take a more ellipsoid shape (but note that `method = "circular"` can be considered a special case of this). If this method is used, the fitted parameters are: A_o (a constant term), Amp (amplitude), theta (rotation, in radians, from the x-axis in the clockwise direction), X_peak (x-axis peak location), Y_peak (y-axis peak location), a (width of Gaussian along x-axis), and b (width of Gaussian along y-axis).

A third method is `method = "elliptical_log"`. This is a further special case in which log2-transformed data may be used. See Priebe et al. 2003 for more details. Parameters from this model include: Amp (amplitude), Q (orientation parameter), X_peak (x-axis peak location), Y_peak (y-axis peak location), X_sig (spread along x-axis), and Y_sig (spread along y-axis).

If using either `method = "elliptical"` or `method = "elliptical_log"`, the `"constrain_orientation"` argument can be used to specify how the orientation is set. In most cases, the user should use the default "unconstrained" setting for this argument. Doing so will provide the best-fit 2D-Gaussian (assuming that the solution yielded by `stats::nls()` converges on the global optimum).

Setting `constrain_orientation` to a numeric (e.g. `constrain_orientation = pi/2`) will force the orientation of the Gaussian to the specified value. Note that this is handled differently by `method = "elliptical"` vs `method = "elliptical_log"`. In `method = "elliptical"`, the theta parameter dictates the rotation, in radians, from the x-axis in the clockwise direction. In contrast, the `method = "elliptical_log"` procedure uses a Q parameter to determine the orientation of the 2D-Gaussian. Setting `constrain_orientation = 0` will result in a diagonally-oriented Gaussian, whereas setting `constrain_orientation = -1` will result in horizontal orientation. See Priebe et al. 2003 for more details.

The `user_init` argument can also be used to supply a vector of initial values for the A, Q, X_peak, Y_peak, X_var, and Y_var parameters. If the user chooses to make use of `user_init`, then a vector containing all parameters must be supplied in a particular order.

Additional arguments to the `control` argument in `stats::nls()` can be supplied via `...`.

Value

A list with the components:

- "coefs" A data.frame of fitted model parameters.
- "model" The model object, fitted by `stats::nls()`.
- "model_error_stats" A data.frame detailing the rss, rmse, deviance, and AIC of the fitted model.
- "fit_method" A character vector that indicates which method and orientation strategy was used by this function.

Author(s)

Vikram B. Baliga

References

Priebe NJ, Cassanello CR, Lisberger SG. The neural representation of speed in macaque area MT/V5. J Neurosci. 2003 Jul 2;23(13):5650-61. doi: 10.1523/JNEUROSCI.23-13-05650.2003.

Examples

```
if (interactive()) {
  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]


  #### Example 1: Unconstrained elliptical ####
  ## This fits an unconstrained elliptical by default
  gauss_fit <-
    fit_gaussian_2D(samp_dat)

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  library(ggplot2); library(metR)
```

```
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)


  #### Example 2: Constrained elliptical_log ####
  ## This fits a constrained elliptical, as in Priebe et al. 2003
  gauss_fit <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_orientation = -1
    )

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)
}
```

---

gaussplot_sample_data *Sample data set*

Description

A `data.frame` of raw data and fitted 2D-Gaussian parameters; intended for use with `predict_gaussian_2D()`

Usage

```
gaussplot_sample_data
```

Format

A data frame with 36 rows and 11 variables:

X_values vector of numeric values for the x-axis

Y_values vector of numeric values for the y-axis

response vector of numeric values for the response variable

norm_g_resp normalized values from the 2D-Gaussian fit

g_resp values from the 2D-Gaussian fit

A amplitude of 2D-Gaussian (repeated)

X_peak location of peak x-axis value (repeated)

X_var variance in x (repeated)

Q orientation parameter of the gaussian (repeated)

Y_peak location of peak y-axis value (repeated)

Y_var variance in y (repeated)

`get_volume_gaussian_2D` *Compute volume under 2D-Gaussian*

Description

Compute volume under 2D-Gaussian

Usage

```
get_volume_gaussian_2D(X_sig, Y_sig)
```

Arguments

`X_sig`	numeric value(s) of the x-axis spread (sigma)
`Y_sig`	numeric value(s) of the y-axis spread (sigma)

Details

Volume under the 2D-Gaussian is computed as: `2 * pi * sqrt(abs(X_sig)) * sqrt(abs(Y_sig))`

Numeric vectors can be supplied to `X_sig` and `Y_sig`. If vectors of length greater than 1 are given, the function computes volume for each sequential pair of `X_sig`, `Y_sig` values. The lengths of these supplied vectors must be identical.

Value

Numeric value(s) indicating the computed volume(s)

Author(s)

Vikram B. Baliga

Examples

```
library(gaussplotR)

get_volume_gaussian_2D(5, 3) #24.33467
```

`ggplot_gaussian_2D` *Plot a 2D-Gaussian via ggplot*

Description

Plot a 2D-Gaussian via ggplot

Usage

```
ggplot_gaussian_2D(
  gauss_data,
  normalize = TRUE,
  contour_thickness = 0.04,
  contour_color = "black",
  bins = 15,
  viridis_dir = 1,
  viridis_opt = "B",
  x_lab = "X values",
  y_lab = "Y values",
  axis.text = element_text(size = 6),
  axis.title = element_text(size = 7),
  axis.ticks = element_line(size = 0.3),
  plot.margin = unit(c(0.1, 0.1, 0.1, 0.1), "cm"),
  ...
)
```

Arguments

`gauss_data`	Data.frame with X_values, Y_values, and predicted_values, e.g. exported from `predict_gaussian_2D()`
`normalize`	Default TRUE, should predicted_values be normalized on a 0 to 1 scale?
`contour_thickness`	Thickness of contour lines
`contour_color`	Color of the contour lines
`bins`	Number of bins for the contour plot
`viridis_dir`	See "direction" in scale_fill_viridis_c()
`viridis_opt`	See "option" in scale_fill_viridis_c()
`x_lab`	Arguments passed to xlab()
`y_lab`	Arguments passed to ylab()

axis.text	Arguments passed to axis.text
axis.title	Arguments passed to axis.title
axis.ticks	Arguments passed to axis.ticks
plot.margin	Arguments passed to plot.margin
...	Other arguments supplied to `ggplot2::theme()`

Value

A ggplot object that uses metR::geom_contour_fill() to display the 2D-Gaussian

Author(s)

Vikram B. Baliga

Examples

```
if (interactive()) {
  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]


  #### Example 1: Unconstrained elliptical ####
  ## This fits an unconstrained elliptical by default
  gauss_fit <-
    fit_gaussian_2D(samp_dat)

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  library(ggplot2); library(metR)
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)
```

```
  #### Example 2: Constrained elliptical_log ####
  ## This fits a constrained elliptical, as in Priebe et al. 2003
  gauss_fit <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_orientation = -1
    )

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)
}
```

`predict_gaussian_2D` *Predict values from a fitted 2D-Gaussian*

Description

Predict values from a fitted 2D-Gaussian

Usage

```
predict_gaussian_2D(fit_object, X_values, Y_values, ...)
```

Arguments

`fit_object`	Either the output of `gaussplotR::fit_gaussian_2D()` or a list that contains coefficients and fit methods (see Details).
`X_values`	vector of numeric values for the x-axis
`Y_values`	vector of numeric values for the y-axis
`...`	Additional arguments

Details

This function assumes Gaussian parameters have been fitted beforehand. No fitting of parameters is done within this function; these can be supplied via the object created by `gaussplotR::fit_gaussian_2D()`.

If `fit_object` is not an object created by `gaussplotR::fit_gaussian_2D()`, `predict_gaussian_2D()` attempts to parse `fit_object` as a list of two items. The coefficients of the fit must be supplied as a one-row, named data.frame within `fit_object$coefs`, and details of the methods for fitting the Gaussian must be contained as a character vector in `fit_object$fit_method`. This character vector in `fit_object$fit_method` must be a named vector that provides information about the method, amplitude constraint choice, and orientation constraint choice, using the names `method`, `amplitude`, and `orientation`. `method` must be one of: `"elliptical"`, `"elliptical_log"`, or `"circular"`. `amplitude` and `orientation` must each be either `"unconstrained"` or `"constrained"`. For example, `c(method = "elliptical", amplitude = "unconstrained", orientation = "unconstrained")`. One exception to this is when `method = "circular"`, in which case `orientation` must be `NA`, e.g.: `c(method = "circular",amplitude = "unconstrained", orientation = NA)`.

Value

A data.frame with the supplied `X_values` and `Y_values` along with the predicted values of the 2D-Gaussian (`predicted_values`)

Author(s)

Vikram B. Baliga

Examples

```
if (interactive()) {
  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]


  #### Example 1: Unconstrained elliptical ####
  ## This fits an unconstrained elliptical by default
  gauss_fit <-
    fit_gaussian_2D(samp_dat)

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
```

```
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  library(ggplot2); library(metR)
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)


  #### Example 2: Constrained elliptical_log ####
  ## This fits a constrained elliptical, as in Priebe et al. 2003
  gauss_fit <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_orientation = -1
    )

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)
}
```

---

rgl_gaussian_2D *Produce a 3D plot of the 2D-Gaussian via rgl*

Description

Produce a 3D plot of the 2D-Gaussian via rgl

Usage

```
rgl_gaussian_2D(
```

```
    gauss_data,
    normalize = TRUE,
    viridis_dir = 1,
    viridis_opt = "B",
    x_lab = "X values",
    y_lab = "Y values",
    box = FALSE,
    aspect = TRUE,
    ...
  )
```

Arguments

gauss_data	Data.frame with X_values, Y_values, and predicted_values, e.g. exported from predict_gaussian_2D()
normalize	Default TRUE, should predicted_values be normalized on a 0 to 1 scale?
viridis_dir	See "direction" in scale_fill_viridis_c()
viridis_opt	See "option" in scale_fill_viridis_c()
x_lab	Arguments passed to xlab()
y_lab	Arguments passed to ylab()
box	Whether to draw a box; see rgl::plot3d()
aspect	Whether to adjust the aspect ratio; see rgl::plot3d()
...	Other arguments supplied to rgl::plot3d()

Value

An rgl object (i.e. of the class 'rglHighlevel'). See rgl::plot3d() for details.

Author(s)

Vikram B. Baliga

Examples

```
if (interactive()) {
  ## Load the sample data set
  data(gaussplot_sample_data)

  ## The raw data we'd like to use are in columns 1:3
  samp_dat <-
    gaussplot_sample_data[,1:3]


  #### Example 1: Unconstrained elliptical ####
  ## This fits an unconstrained elliptical by default
  gauss_fit <-
    fit_gaussian_2D(samp_dat)
```

```
  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  library(ggplot2); library(metR)
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)


  #### Example 2: Constrained elliptical_log ####
  ## This fits a constrained elliptical, as in Priebe et al. 2003
  gauss_fit <-
    fit_gaussian_2D(
      samp_dat,
      method = "elliptical_log",
      constrain_orientation = -1
    )

  ## Generate a grid of x- and y- values on which to predict
  grid <-
    expand.grid(X_values = seq(from = -5, to = 0, by = 0.1),
                Y_values = seq(from = -1, to = 4, by = 0.1))

  ## Predict the values using predict_gaussian_2D
  gauss_data <-
    predict_gaussian_2D(
      fit_object = gauss_fit,
      X_values = grid$X_values,
      Y_values = grid$Y_values,
    )

  ## Plot via ggplot2 and metR
  ggplot_gaussian_2D(gauss_data)

  ## Produce a 3D plot via rgl
  rgl_gaussian_2D(gauss_data)
}
```

Index