Package ‘itsadug’

October 13, 2022

Version 2.4.1

Date 2022-06-17

Title Interpreting Time Series and Autocorrelated Data Using GAMMs

Author Jacolien van Rij [aut, cre],
Martijn Wieling [aut],
R. Harald Baayen [aut],
Hedderik van Rijn [ctb]

Maintainer Jacolien van Rij <vanrij.jacolien@gmail.com>

Description GAMM (Generalized Additive Mixed Modeling; Lin & Zhang, 1999)
as implemented in the R package 'mgcv' (Wood, S.N., 2006; 2011) is a nonlinear
regression analysis which is particularly useful for time course data such as
EEG, pupil dilation, gaze data (eye tracking), and articulography recordings,
but also for behavioral data such as reaction times and response data. As time
course measures are sensitive to autocorrelation problems, GAMMs implements
methods to reduce the autocorrelation problems. This package includes functions
for the evaluation of GAMM models (e.g., model comparisons, determining regions
of significance, inspection of autocorrelational structure in residuals)
and interpreting of GAMMs (e.g., visualization of complex interactions, and
contrasts).

License GPL (>= 2)

LazyData true

Depends R (>= 4.0), mgcv (>= 1.8), plotfunctions (>= 1.4)

VignetteBuilder knitr

Suggests knitr, xtable, sp, data.table

RoxygenNote 7.2.0

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2022-06-17 15:40:02 UTC

R topics documented:

acf_n_plots *Generate N ACF plots of individual or aggregated time series.*

Description

Generate N ACF plots of individual or aggregated time series.

Usage

```
acf_n_plots(
  x,
  n = 5,
  split_by = NULL,
  cond = NULL,
  max_lag = NULL,
  fun = mean,
  plot = TRUE,
  random = F,
  mfrow = NULL,
  add = FALSE,
  print.summary = getOption("itsadug_print"),
  ...
)
```

Arguments

x	A vector with time series data, typically residuals of a regression model.
n	The number of plots to generate.
split_by	List of vectors (each with equal length as x) that group the values of x into trials or timeseries events. Generally other columns from the same data frame.
cond	Named list with a selection of the time series events specified in split_by. Default is NULL, indicating that all time series are being processed, rather than a selection.
max_lag	Maximum lag at which to calculate the acf. Default is the maximum for the longest time series.
fun	The function used when aggregating over time series (depending on the value of split_by).
plot	Logical: whether or not to produce plot. Default is TRUE.

`random`	Logical: determine randomly which `n` (aggregated) time series are plotted, or use the `quantile` function to find a range of different time series to plot. Default is FALSE (not random).
`mfrow`	A vector of the form c(nr, nc). The figures will be drawn in an nr-by-nc array on the device by rows.
`add`	Logical: whether to add the plots to an exiting plot window or not. Default is FALSE.
`print.summary`	Logical: whether or not to print summary. Default set to the print info messages option (see `infoMessages`).
`...`	Other arguments for plotting, see `par`.

Value

`n` ACF plots providing information about the autocorrelation in `x`.

Author(s)

Jacolien van Rij, R. Harald Baayen

See Also

Use `acf` for the original ACF function, and `acf_plot` for an ACF that takes into account time series in the data.

Other functions for model criticism: `acf_plot()`, `acf_resid()`, `derive_timeseries()`, `resid_gam()`, `start_event()`, `start_value_rho()`

Examples

```
data(simdat)

# Separate ACF for each time series:
acf_n_plots(simdat$Y, split_by=list(simdat$Subject, simdat$Trial))

# Average ACF per participant:
acf_n_plots(simdat$Y, split_by=list(simdat$Subject))

## Not run:
# Data treated as single time series. Plot is added to current window.
# Note: 1 time series results in 1 plot.
acf_n_plots(simdat$Y, add=TRUE)
# Plot 4 ACF plots doesn't work without splitting data:
acf_n_plots(simdat$Y, add=TRUE, n=4)

# Plot ACFs of 4 randomly selected time series:
acf_n_plots(simdat$Y, random=TRUE, n=4, add=TRUE,
    split_by=list(simdat$Subject, simdat$Trial))


## End(Not run)
```

```
#---------------------------------------------
# When using model residuals
#---------------------------------------------

## Not run:
# add missing values to simdat:
simdat[sample(nrow(simdat), 15),]$Y <- NA
# simple linear model:
m1 <- lm(Y ~ Time, data=simdat)

# This will generate an error:
# acf_n_plots(resid(m1), split_by=list(simdat$Subject, simdat$Trial))

# This should work:
el.na <- missing_est(m1)
acf_n_plots(resid(m1),
     split_by=list(simdat[-el.na,]$Subject, simdat[-el.na,]$Trial))

# This should also work:
simdat$res <- NA
simdat[!is.na(simdat$Y),]$res <- resid(m1)
acf_n_plots(simdat$res, split_by=list(simdat$Subject, simdat$Trial))

## End(Not run)

# see the vignette for examples:
vignette('acf', package='itsadug')
```

acf_plot — *Generate an ACF plot of an aggregated time series.*

Description

Generate an ACF plot of an aggregated time series.

Usage

```
acf_plot(
  x,
  split_by = NULL,
  max_lag = NULL,
  plot = TRUE,
  fun = mean,
  cond = NULL,
  return_all = FALSE,
  ...
)
```

Arguments

`x`	A vector with time series data, typically residuals of a regression model. (See examples for how to avoid errors due to missing values.)
`split_by`	List of vectors (each with equal length as `x`) that group the values of `x` into trials or timeseries events. Generally other columns from the same data frame.
`max_lag`	Maximum lag at which to calculate the acf. Default is the maximum for the longest time series.
`plot`	Logical: whether or not to plot the ACF. Default is TRUE.
`fun`	The function used when aggregating over time series (depending on the value of `split_by`).
`cond`	Named list with a selection of the time series events specified in `split_by`. Default is NULL, indicating that all time series are being processed, rather than a selection.
`return_all`	Returning acfs for all time series.
`...`	Other arguments for plotting, see `par`.

Value

An aggregated ACF plot and / or optionally a list with the aggregated ACF values.

Author(s)

Jacolien van Rij

See Also

Use `acf` for the original ACF function, and `acf_n_plots` for inspection of individual time series.

Other functions for model criticism: `acf_n_plots()`, `acf_resid()`, `derive_timeseries()`, `resid_gam()`, `start_event()`, `start_value_rho()`

Examples

```
data(simdat)

# Default acf function:
acf(simdat$Y)
# Same plot with acf_plot:
acf_plot(simdat$Y)
# Average of ACFs per time series:
acf_plot(simdat$Y, split_by=list(simdat$Subject, simdat$Trial))
# Median of ACFs per time series:
acf_plot(simdat$Y, split_by=list(simdat$Subject, simdat$Trial), fun=median)

# extract value of Lag1:
lag1 <- acf_plot(simdat$Y,
   split_by=list(Subject=simdat$Subject, Trial=simdat$Trial),
   plot=FALSE)['1']
```

```
#---------------------------------------------
# When using model residuals
#---------------------------------------------

# add missing values to simdat:
simdat[sample(nrow(simdat), 15),]$Y <- NA
# simple linear model:
m1 <- lm(Y ~ Time, data=simdat)

## Not run:
# This will generate an error:
acf_plot(resid(m1), split_by=list(simdat$Subject, simdat$Trial))

## End(Not run)
# This should work:
el.na <- missing_est(m1)
acf_plot(resid(m1),
     split_by=list(simdat[-el.na,]$Subject, simdat[-el.na,]$Trial))

# This should also work:
simdat$res <- NA
simdat[!is.na(simdat$Y),]$res <- resid(m1)
acf_plot(simdat$res, split_by=list(simdat$Subject, simdat$Trial))

# see the vignette for examples:
vignette('acf', package='itsadug')
```

acf_resid — *Generate an ACF plot of model residuals. Works for lm, lmer, gam, bam, ...*

Description

Wrapper around `acf_plot` and `acf_n_plots` for regression models.

Usage

```
acf_resid(
  model,
  split_pred = NULL,
  n = 1,
  plot = TRUE,
  check.rho = NULL,
  main = NULL,
  ...
)
```

Arguments

model	A regression model generated by lm, glm, lmer, glmer, gam, or bam. (See examples for how to avoid errors due to missing values.)
split_pred	Vector with names of model predictors that determine the time series in the data, or should be used to split the ACF plot by. Alternatively, split_pred can be a named list as being used by acf_plot and acf_n_plots. Yet another option is to provide the text string 'AR.start', for a model that includes an AR1 model. The events are derived from the AR.start column if that is provided.
n	The number of plots to generate. If n=1 (default) then acf_plot is being called. If n>1 then acf_n_plots is being called.
plot	Logical: whether or not to produce plot. Default is TRUE.
check.rho	Numeric value: Generally leave at NULL. This value does not change anything, but it is used to check whether the model's AR1 coefficient matches the expected value of rho.
main	Text string, title of plot.
...	Other arguments as input for acf_plot or acf_n_plots.

Value

An aggregated ACF plot and / or optionally a list with the aggregated ACF values.

Author(s)

Jacolien van Rij

See Also

Use acf for the original ACF function, and acf_plot, or acf_n_plots.

Other functions for model criticism: acf_n_plots(), acf_plot(), derive_timeseries(), resid_gam(), start_event(), start_value_rho()

Examples

```
data(simdat)

# add missing values to simdat:
simdat[sample(nrow(simdat), 15),]$Y <- NA

## Not run:
# Run GAMM model:
m1 <- bam(Y ~ te(Time, Trial)+s(Subject, bs='re'), data=simdat)

# Using a list to split the data:
acf_resid(m1, split_pred=list(simdat$Subject, simdat$Trial))
# ...or using model predictors:
acf_resid(m1, split_pred=c('Subject', 'Trial'))

# Calling acf_n_plots:
```

```
acf_resid(m1, split_pred=c('Subject', 'Trial'), n=4)
# add some arguments:
acf_resid(m1, split_pred=c('Subject', 'Trial'), n=4, max_lag=10)

# This does not work...
m2 <- lm(Y ~ Time, data=simdat)
acf_resid(m2, split_pred=c('Subject', 'Trial'))
# ... but this is ok:
acf_resid(m2, split_pred=list(simdat$Subject, simdat$Trial))

# Using AR.start column:
simdat <- start_event(simdat, event=c('Subject', 'Trial'))
r1 <- start_value_rho(m1)
m3 <- bam(Y ~ te(Time, Trial)+s(Subject, bs='re'), data=simdat,
    rho=r1, AR.start=simdat$start.event)
acf_resid(m3, split_pred='AR.start')
# this is the same:
acf_resid(m3, split_pred=c('Subject', 'Trial'))
# Note: use model comparison to find better value for rho

## End(Not run)
# see the vignette for examples:
vignette('acf', package='itsadug')
```

`check_resid` *Inspect residuals of regression models.*

Description

Inspect residuals of regression models.

Usage

```
check_resid(
  model,
  AR_start = NULL,
  split_pred = NULL,
  ask = TRUE,
  select = 1:4
)
```

Arguments

`model`	A regression model, resulting from the functions `gam` or `bam`, or `lm`, `glm`, `lmer`, or `glmer`.
`AR_start`	Defaults to NULL. Only use this when the model was run in an old versions of package `mgcv` and the function cannot retrieve the used AR.start values from the `model`. When an error is shown with newer versions of `mgcv`, please check the column provided as values of AR.start. when using old versions of package `mgcv`. Function will give error when it cannot find AR.start.

`split_pred`	A names list indicating time series in the data.
`ask`	Logical: whether or not to show the plots one by one. Defaults to TRUE. When set to FALSE, make sure to have specified sufficient rows and columns to show the X plots. Alternatively, use `select` to plot only specific plots.
`select`	Vector or numeric value indicating which plots to return (see Notes). Defaults to 1:4 (all).

Note

- Plot 1: distribution of residuals with QQ norm plot.
- Plot 2: distribution of residuals with density plot.
- Plot 3: ACF plot of residuals. In case an AR1 model is included, the gray lines indicate standard residuals, and the thick black lines indicate AR1 corrected residuals.
- Plot 4 (optional): In case the `split_pred` predictors are specified an ACF plot averaged over the time series is produced. dashed lines indicate the maximum and minimum time series (w.r.t. lag 2), the solid lines the 25 mean of all time series.

See the examples on how to specify a selection of these plots.

Author(s)

Jacolien van Rij

See Also

Other Model evaluation: `diagnostics()`, `plot_modelfit()`

Examples

```
data(simdat)

## Not run:
# Add start event column:
simdat <- start_event(simdat, event=c('Subject', 'Trial'))
head(simdat)

# bam model with AR1 model (toy example, not serious model):
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group),
   data=simdat, rho=.5, AR.start=simdat$start.event)

# Warning, no time series specified:
check_resid(m1)

# Time series specified, results in a 'standard' ACF plot,
# treating all residuals as single time seriesand,
# and an ACF plot with the average ACF over time series:
check_resid(m1, split_pred=list(Subject=simdat$Subject, Trial=simdat$Trial))
# Note: residuals do not look very good.
# Alternative (results in the same, see help(acf_resid) ):
check_resid(m1, split_pred='AR.start')
```

```
# Define larger plot window (choose which line you need):
dev.new(width=8, height=8) # on windows or mac
quartz(,8,8)               # on mac
x11(width=8, height=8)     # on linux or mac

par(mfrow=c(2,2), cex=1.1)
check_resid(m1, split_pred='AR.start', ask=FALSE)

## End(Not run)
```

`compareML` *Function for comparing two GAMM models.*

Description

Function for comparing two GAMM models.

Usage

```
compareML(
  model1,
  model2,
  signif.stars = TRUE,
  suggest.report = FALSE,
  print.output = TRUE
)
```

Arguments

`model1`	First model.
`model2`	Second model.
`signif.stars`	Logical (default = TRUE). Whether or not to display stars indicating the level of significance on 95% confidence level.
`suggest.report`	Logical (default = FALSE). Whether or not to present a suggestion on how one could report the information. If `print.output` is set to FALSE, `suggest.report` will set to FALSE too. Please inspect yourself whether the label between square bracket fits your own standards. Note: the `X2` should be replaced by a proper Chi-Square symbol χ^2.
`print.output`	Logical: whether or not to print the output. By default set to true, even if the the messages are not allowed by a global package option using the function `infoMessages`.

Details

As an Chi-Square test is performed on two times the difference in minimized smoothing parameter selection score (GCV, fREML, REML, ML), and the difference in degrees of freedom specified in the model. The degrees of freedom of the model terms are the sum of 1) the number of estimated smoothing parameters for the model, 2) number of parametric (non-smooth) model terms including the intercept, and 3) the sum of the penalty null space dimensions of each smooth object.

This method is preferred over other functions such as `AIC` for models that include an AR1 model or random effects (especially nonlinear random smooths using `bs='fs'`). CompareML also reports the AIC difference, but that value should be treated with care.

Note that the Chi-Square test will result in a very low p-value when the difference in degrees of freedom approaches zero. Use common sense to determine if the difference between the two models is meaningful. A warning is presented when the difference in score is smaller than 5.

The order of the two models is not important. Model comparison is only implemented for the methods GCV, fREML, REML, and ML.

Value

Optionally returns the Chi-Square test table.

Notes

For suppressing the output and all warnings, set infoMessages to FALSE (`infoMessages('off')`), set the argument `print.output` to FALSE, and use the function `suppressWarnings` to suppress warning messages.

Author(s)

Jacolien van Rij. With many thanks to Simon N. Wood for his feedback.

See Also

For models without AR1 model or random effects `AIC` can be used.

Other Testing for significance: `plot_diff2()`, `plot_diff()`, `report_stats()`, `wald_gam()`

Examples

```
data(simdat)

## Not run:
infoMessages('on')
# some arbitrary models:
m1 <- bam(Y~Group + s(Time, by=Group), method='fREML', data=simdat)
m2 <- bam(Y~Group + s(Time), method='fREML', data=simdat)

compareML(m1, m2)

# exclude significance stars:
compareML(m1, m2, signif.stars=FALSE)
```

```
m3 <- bam(Y~Group + s(Time, by=Group, k=25), method='fREML',
    data=simdat)
compareML(m1, m3)

# do not print output, but save table for later use:
cml <- compareML(m1, m2, print.output=FALSE)$table
cml

# Use suppressWarnings to also suppress warnings:
suppressWarnings(cml <- compareML(m1, m2, print.output=FALSE)$table)


## End(Not run)
```

---

convertNonAlphanumeric

Prepare string for regular expressions (backslash for all non-letter and non-digit characters)

Description

Prepare string for regular expressions (backslash for all non-letter and non-digit characters)

Usage

```
convertNonAlphanumeric(text)
```

Arguments

`text`	A text string (smooth term label) that needs to be converted to a regular expression.

Value

A regular expression string.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)
# Model for illustrating coefficients:
m0 <- bam(Y ~ s(Time) + s(Subject, bs='re')
+ s(Time, Subject, bs='re'), data=simdat)

# get all coefficients:
coef(m0)
# to get only the Subject intercepts:
coef(m0)[grepl(convertNonAlphanumeric('s(Subject)'), names(coef(m0)))]
# to get only the Subject slopes:
coef(m0)[grepl(convertNonAlphanumeric('s(Time,Subject)'), names(coef(m0)))]
```

---

corfit *Calculate the correlation between the fitted model and data.*

Description

Calculate the correlation between the fitted model and data.

Usage

```
corfit(model)
```

Arguments

`model`	A fitted regression model (using gam, or bam).

Value

Numeric value: correlation between fitted model and data.

See Also

Other Utility functions: `convertNonAlphanumeric()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

# Fit simple GAM model:
gam1 <- bam(Y ~ s(Time), data=simdat, discrete=TRUE)
corfit(gam1)
```

derive_timeseries *Derive the time series used in the AR1 model.*

Description

Derive the time series used in the AR1 model.

Usage

```
derive_timeseries(model, AR.start = NULL)
```

Arguments

`model`	GAMM model that includes an AR1 model.
`AR.start`	Vector with AR.start information, necessary for the AR1 model. Optional, defaults to NULL.

Value

A vector with time series indication based on the AR1 model.

Author(s)

Jacolien van Rij

See Also

Other functions for model criticism: `acf_n_plots()`, `acf_plot()`, `acf_resid()`, `resid_gam()`, `start_event()`, `start_value_rho()`

Examples

```
data(simdat)

# add missing values to simdat:
simdat[sample(nrow(simdat), 15),]$Y <- NA
simdat <- start_event(simdat, event=c('Subject', 'Trial'))

## Not run:
# Run GAMM model:
m1 <- bam(Y ~ te(Time, Trial)+s(Subject, bs='re'), data=simdat,
    rho=.5, AR.start=simdat$start.event)
simdat$Event <- NA
simdat[!is.na(simdat$Y),]$Event <- derive_timeseries(m1)
acf_resid(m1, split_pred=list(Event=simdat$Event))

# And this works too:
simdat$Event <- derive_timeseries(simdat$start.event)
acf_resid(m1, split_pred=list(Event=simdat$Event))
```

```
# Note that acf_resid automatically makes use of derive_timeseries:
acf_resid(m1, split_pred='AR.start')

## End(Not run)
```

`diagnostics` *Visualization of the model fit for time series data.*

Description

Diagnostic plots for evaluating the model fit.

Usage

```
diagnostics(
  model,
  plot = "all",
  ask = TRUE,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`model`	A lm or gam object, produced by `gam` or `bam`, `lm`, `glm`.
`plot`	A text string or numeric vector indicating which diagnostic plots to show. Default is 'all'. See Section 'Details' for the different options.
`ask`	Logical: whether the user is prompted before starting a new page of output. Defaults to TRUE.
`print.summary`	Logical: whether or not to print summary. Default set to the print info messages option (see `infoMessages`).

Details

When `plot='all'`, the following plots are generated:

1. Residuals by fitted values. Used for inspection of general trends in the residuals.
2. Residuals ordered by predictor. Useful for checking how the trends of individual predictors are captured by the model.
3. Distribution of residuals. QQ plot that compares the distribution of the residuals with the normal distribution.
4. ACF of residuals. Inspection of autocorrelation in the residuals. See also `acf_resid`.
5. Trends in the random smooths. Be careful with the interpretation of the 'fixed' effects and interactions when the random smooths show trends. See examples below.
6. Printing distributions of numeric predictors.

Author(s)

Jacolien van Rij

See Also

Other Model evaluation: `check_resid()`, `plot_modelfit()`

Examples

```
data(simdat)
## Not run:
# no random smooths:
m1 <- bam(Y ~ Group + s(Time, by=Group) + s(Trial) + s(Subject, bs='re'), data=simdat)
diagnostics(m1)

# only plot residuals by predictor:
diagnostics(m1, plot=2)

# without prompts:
par(mfrow=c(2,2))
diagnostics(m1, plot=1:4, ask=FALSE)

# only plot random smooths:
diagnostics(m1, plot=5)
# Note: the plot does not change,
# because there are no random smooths to plot.

# with random smooths
m2 <- bam(Y ~ Group + s(Time, by=Group) + s(Time, Subject, bs='fs', m=1), data=simdat)
diagnostics(m2)

## INSPECTION OF RANDOM SMOOTHS
## ----------------------------

# In this underspecified model (too much smoothing for the interaction)
# part of the effect of Time is captured by the random smooths:
m3 <- bam(Y ~ te(Time, Trial, k=c(3,3)) + s(Time, Subject, bs='fs', m=1), data=simdat)

# The plot shows a clear trend in the average of the random smooths,
# and the amplitude of the mean (!) curve is almost as large as the
# amplitude of the 'fixed' effect of Time:
diagnostics(m3, plot=5, ask=FALSE)

# Compare with the following models:
m4 <- bam(Y ~ te(Time, Trial, k=c(10,5)) + s(Time, Subject, bs='fs', m=1), data=simdat)
diagnostics(m4, plot=5, ask=FALSE)

m5 <- bam(Y ~ s(Time) + s(Trial) + ti(Time, Trial)
    + s(Time, Subject, bs='fs', m=1), data=simdat)
diagnostics(m5, plot=5, ask=FALSE)
```

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

`diff_terms` *Compare the formulas of two models and return the difference(s).*

Description

Compare the formulas of two models and return the difference(s).

Usage

```
diff_terms(model1, model2)
```

Arguments

`model1`	A fitted regression model (using lm, glm, gam, or bam).
`model2`	A fitted regression model (using lm, glm, gam, or bam).

Value

A list with model terms that are not shared by both models.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

# Fit simple GAM model:
gam1 <- bam(Y ~ s(Time), data=simdat)
gam2 <- bam(Y ~ Group+s(Time), data=simdat)
diff_terms(gam1, gam2)
```

`dispersion` *Calculate the dispersion of the residuals*

Description

Calculate the dispersion of the residuals

Usage

```
dispersion(model)
```

Arguments

`model`	A fitted regression model (using gam, or bam).

Value

Numeric value: dispersion of the residuals.

See Also

Other Functions for model inspection: `fvisgam()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `plot_topo()`, `pvisgam()`

Examples

```
trial <- function(f=.95){
    x <- rep(1-f,101)
    x[round(runif(1,1,50)):length(x)] <- f
    return(rbinom(101,1,x))
}
set.seed(123)
dat <- data.frame(Time=rep( seq(0,1,length=101),100),
    y = unlist(replicate(100, trial(f=1), simplify=FALSE)),
    stringsAsFactors=FALSE)
# under dispersion:
gam1 <- gam(y ~ s(Time), data=dat, family=binomial)
summary(gam1)
dispersion(gam1)
# but not here:
gam2 <- gam(y ~ 1, data=dat, family=binomial)
summary(gam2)
dispersion(gam2)
# and not here:
dat <- data.frame(Time=rep( seq(0,1,length=101),100),
    y = unlist(replicate(100, trial(f=.75), simplify=FALSE)),
    stringsAsFactors=FALSE)
gam3 <- gam(y ~ s(Time), data=dat, family=binomial)
summary(gam3)
```

```
dispersion(gam3)
```

`eeg` *Raw EEG data, single trial, 50Hz.*

Description

A dataset containing a single EEG trial.

Usage

```
eeg
```

Format

A data frame with 1504 rows and 5 variables:

`Electrode` Electrode that recorded the EEG.

`Time` Time, time measure from onset of the stimulus.

`Ampl` EEG amplitude, recorded by 32 electrodes.

`X` Approximation of electrode position, relative to Cz. Left is negative.

`Y` Approximation of electrode position, relative to Cz. Back is negative.

Author(s)

Jacolien van Rij

`fadeRug` *Fade out the areas in a surface without data.*

Description

Add a transparency Rug to a contour plot or image.

Usage

```
fadeRug(
  x,
  y,
  n.grid = 30,
  too.far = 0.03,
  col = "white",
  alpha = 1,
  use.data.range = TRUE
)
```

Arguments

`x`	Observations on x-axis.
`y`	Observations on y-axis.
`n.grid`	Resolution of Rug. Defaults to 30, which means that the x- and y-axis are divided in 30 bins. A two-value vector ould be used to specify different bins for x- and y-axis.
`too.far`	plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded. Based on `exclude.too.far` of Simon N. Wood.
`col`	Color representing missing data. Defaults to 'white'.
`alpha`	Transparency, number between 0 (completely transparent) and 1 (non-transparent). Defaults to 1.
`use.data.range`	Logical value, indicating whether `x` and `y` are the data that the plot is based on. Defaults to TRUE.

Value

Plots a shaded image over the contour plot or image.

Warning

On Linux `x11` devices may not support transparency. In that case, a solution might be to write the plots immediately to a file using functions such as `pdf`, or `png`.

Author(s)

Jacolien van Rij, based on Simon N. Wood's `exclude.too.far`

See Also

`rug`, `contour`, `image`

Other Functions for plotting: `rug_model()`

Examples

```
data(simdat)

# Introduce extreme values:
set.seed(123)
newdat <- simdat[sample(which(simdat$Time < 1500),
    size=round(.5*length(which(simdat$Time < 1500)))),]
newdat <- rbind(newdat,
    simdat[sample(which(simdat$Time > 1500),
    size=5),])
# Some simple GAM with tensor:
m1 <- bam(Y ~ te(Time, Trial), data=newdat)
```

```
# plot summed effects:
fvisgam(m1, view=c('Time', 'Trial'), zlim=c(-15,15))
fadeRug(newdat$Time, newdat$Trial)
# check with data points:
points(newdat$Time, newdat$Trial, pch=16, col=alpha(1))

# compare with default rug:
fvisgam(m1, view=c('Time', 'Trial'), zlim=c(-15,15))
rug(newdat$Time)
rug(newdat$Trial, side=2)
fadeRug(newdat$Time, newdat$Trial)
# and compare with too.far:
fvisgam(m1, view=c('Time', 'Trial'), zlim=c(-15,15),
    too.far=.03)
vis.gam(m1, view=c('Time', 'Trial'), zlim=c(-15,15),
    too.far=.03, plot.type='contour', color='topo')

# in case fade rug overlaps with color legend:
fvisgam(m1, view=c('Time', 'Trial'), zlim=c(-15,15),
     add.color.legend=FALSE)
fadeRug(newdat$Time, newdat$Trial, alpha=.75)
gradientLegend(c(-15,15), pos=.875)

# change x- and y-grid, and color:
fvisgam(m1, view=c('Time', 'Trial'), zlim=c(-15,15))
points(newdat$Time, newdat$Trial)
fadeRug(newdat$Time, newdat$Trial, n.grid=c(100,10), col='gray')
```

`find_difference` *Return the regions in which the smooth is significantly different from zero.*

Description

Return the regions in which the smooth is significantly different from zero.

Usage

```
find_difference(mean, se, xVals = NULL, f = 1, as.vector = FALSE)
```

Arguments

`mean`	A vector with smooth predictions.
`se`	A vector with the standard error on the smooth predictions.
`xVals`	Optional vector with x values for the smooth. When `xVals` is provided, the regions are returned in terms of x- values, otherwise as indices.
`f`	A number to multiply the `se` with, to convert the `se` into confidence intervals. Use 1.96 for 95% CI and 2.58 for 99%CI.
`as.vector`	Logical: whether or not to return the data points as vector, or not. Default is FALSE, and a list with start and end points will be returned.

Value

The function returns a list with start points of each region (`start`) and end points of each region (`end`). The logical `xVals` indicates whether the returned values are on the x-scale (TRUE) or indices (FALSE).

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

# Use aggregate to calculate mean and standard deviation per timestamp:
avg <- aggregate(simdat$Y, by=list(Time=simdat$Time),
    function(x){c(mean=mean(x), sd=sd(x))})
head(avg)
# Note that column x has two values in each row:
head(avg$x)
head(avg$x[,1])

# Plot line and standard deviation:
emptyPlot(range(avg$Time), c(-20,20), h0=0)
plot_error(avg$Time, avg$x[,'mean'], avg$x[,'sd'],
   shade=TRUE, lty=3, lwd=3)

# Show difference with 0:
x <- find_difference(avg$x[,'mean'], avg$x[,'sd'], xVals=avg$Time)
# Add arrows:
abline(v=c(x$start, x$end), lty=3, col='red')
arrows(x0=x$start, x1=x$end, y0=-5, y1=-5, code=3, length=.1, col='red')
```

`fvisgam` *Visualization of nonlinear interactions, summed effects.*

Description

Produces perspective or contour plot views of gam model predictions of the additive effects interactions. The code is based on the script for `vis.gam`, but allows to cancel random effects.

Usage

```
fvisgam(
  x,
  view = NULL,
  cond = list(),
  n.grid = 30,
  too.far = 0,
  col = NA,
  color = "terrain",
  contour.col = NULL,
  add.color.legend = TRUE,
  se = -1,
  sim.ci = FALSE,
  plot.type = "contour",
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  nCol = 50,
  rm.ranef = TRUE,
  print.summary = getOption("itsadug_print"),
  transform = NULL,
  transform.view = NULL,
  hide.label = FALSE,
  dec = NULL,
  show.diff = FALSE,
  col.diff = 1,
  alpha.diff = 0.5,
  f = 1.96,
  ...
)
```

Arguments

`x`	A gam object, produced by `gam` or `bam`.
`view`	A two-value vector containing the names of the two main effect terms to be displayed on the x and y dimensions of the plot. Note that variables coerced to factors in the model formula won't work as view variables.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`too.far`	Plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded.
`col`	The colors for the facets of the plot.

`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray' or 'bw'. Alternatively a vector with some colors can be provided for a custom color palette (see examples).
`contour.col`	sets the color of contours when using plot.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`plot.type`	one of 'contour' or 'persp' (default is 'contour').
`xlim`	A two item array giving the lower and upper limits for the x- axis scale. NULL to choose automatically.
`ylim`	A two item array giving the lower and upper limits for the y- axis scale. NULL to choose automatically.
`zlim`	A two item array giving the lower and upper limits for the z- axis scale. NULL to choose automatically.
`nCol`	The number of colors to use in color schemes.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.
`print.summary`	Logical: whether or not to print a summary. Default set to the print info messages option (see `infoMessages`).
`transform`	Function for transforming the fitted values. Default is NULL.
`transform.view`	List with two functions for transforming the values on the x- and y-axis respectively. If one of the axes need to be transformed, set the other to NULL (no transformation). See examples below.
`hide.label`	Logical: whether or not to hide the label (i.e., 'fitted values'). Default is FALSE.
`dec`	Numeric: number of decimals for rounding the color legend. When NULL, no rounding (default). If -1, automatically determined. Note: if value = -1, rounding will be applied also when `zlim` is provided.
`show.diff`	Logical: whether or not to indicate the regions that are significantly different from zero. Note that these regions are just an indication and dependent on the value of `n.grid`. Defaults to FALSE.
`col.diff`	Color to shade the nonsignificant areas.

`alpha.diff`	Level of transparency to mark the nonsignificant areas.
`f`	Scaling factor to determine the CI from the se, for marking the difference with 0. Only applies when `se` is smaller or equal to zero and `show.diff` is set to TRUE.
`...`	other options to pass on to persp, image or contour. In particular ticktype='detailed' will add proper axes labeling to the plots.

Warning

When the argument `show.diff` is set to TRUE a shading area indicates where the confidence intervals include zero. Or, in other words, the areas that are not significantly different from zero. Be careful with the interpretation, however, as the precise shape of the surface is dependent on model constraints such as the value of `choose.k` and the smooth function used, and the size of the confidence intervals are dependent on the model fit and model characteristics (see `vignette('acf', package='itsadug')`). In addition, the value of `n.grid` determines the precision of the plot.

Author(s)

Jacolien van Rij and Martijn Wieling. Modification of `vis.gam` from package `mgcv` of Simon N. Wood.

See Also

`vis.gam`, `plot.gam`

Other Functions for model inspection: `dispersion()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `plot_topo()`, `pvisgam()`

Examples

```
data(simdat)

## Not run:
# Model with random effect and interactions:
m1 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)

# Plot summed effects:
vis.gam(m1, view=c('Time', 'Trial'), plot.type='contour', color='topo')
# Same plot:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=FALSE)
# Without random effects included:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE)

# Notes on the color legend:
# Labels can easily fall off the plot, therefore the numbers can be
# automatically rounded.
# To do the rounding, set dec=-1:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
     dec=-1)
# For custom rounding, set dec to a value:
```

```
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
     dec=0)
# To increase the left marging of the plot (so that the numbers fit):
oldmar <- par()$mar
par(mar=oldmar + c(0,0,0,1) ) # add one line to the right
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
     dec=3)
par(mar=oldmar) # restore to default settings

# changing the color palette:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
    color='terrain')
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
    color=c('blue', 'white', 'red'), col=1)

# Using transform
# Plot log-transformed dependent predictor on measurement scale:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE, transform=exp)

# Notes on transform.view:
# This will generate an error, because x-values <= 0 will result in NaN:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
   transform.view=list(log, NULL))
# adjusting the x-axis helps:
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE,
   xlim=c(1,2000), transform.view=list(log, NULL))

# too.far:
n <- which(simdat$Time > 1500 & simdat$Trial > 5)
simdat[n,]$Y <- NA
simdat[simdat$Trial == -3,]$Y <- NA
m1 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
fvisgam(m1, view=c('Time', 'Trial'), rm.ranef=TRUE, too.far=.03)


## End(Not run)
# see the vignette for examples:
vignette('inspect', package='itsadug')
```

gamtabs *Convert model summary into Latex/HTML table for knitr/R Markdown reports.*

Description

Convert model summary into Latex/HTML table for knitr/R Markdown reports.

Usage

```
gamtabs(
  model,
  caption = " ",
  label = "tab.gam",
  pnames = NA,
  snames = NA,
  ptab = NA,
  stab = NA,
  ...
)
```

Arguments

`model`	A GAM(M) model build in the package `mgcv` using `gam` or `bam`. Alternatively, a summary of a GAMM model could be provided.
`caption`	A string with the caption for the table.
`label`	A string for the label to refer to the table in the markdown document.
`pnames`	A vector with labels to relabel the rows in the parametric part of the summary.
`snames`	A vector with labels to relabel the rows in the smooth part of the summary.
`ptab`	A vector with labels to relabel the column names of the parametric summary.
`stab`	A vector with labels to relabel the column names of the smooth summary.
`...`	Optional additional arguments which are passed to `xtable` (see 'help(xtable)').

Value

A vector with color values.

Note

This function is useful for markdown documents using the package `knitr` to integrate R code with Latex and Sweave. This function requires the package `xtable`.

Author(s)

R. Harald Baayen

See Also

`summary.gam`, `gam`, `bam`.

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `plot_topo()`, `pvisgam()`

Examples

```
data(simdat)
## Not run:
# Model with random effect and interactions:
m1 <- bam(Y ~ Group+te(Time, Trial, by=Group),
    data=simdat)
summary(m1)
gamtabs(m1, caption='Summary of m1')

## End(Not run)
# See for more examples:
vignette('inspect', package='itsadug')
```

---

get_coefs — *Get coefficients for the parametric terms (intercepts and random slopes).*

Description

Wrapper around the function `coef`, and loosely based on `summary.gam`. This function provides a much faster alternative for `summary(model)$p.table`. The function `summary.gam`) may take considerably more time for large models, because it additionally needs to calculate estimates for the smooth term table.

Usage

```
get_coefs(model, se = TRUE)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`se`	Logical: whether or not to return the standard errors.

Value

The coefficients of the parametric terms.

Author(s)

Jacolien van Rij

See Also

Other Model predictions: `get_difference()`, `get_fitted()`, `get_modelterm()`, `get_predictions()`, `get_random()`

Examples

```
data(simdat)

# Condition as factor, to have a random intercept
# for illustration purposes:
simdat$Condition <- as.factor(simdat$Condition)

# Model with random effect and interactions:
m1 <- bam(Y ~ Group * Condition + s(Time),
    data=simdat)

# extract all parametric coefficients:
get_coefs(m1)
# calculate t-values:
test <- get_coefs(m1)
test <- cbind(test, test[,1] / test[,2] )
colnames(test)[3] <- 't-value'
test

# get_coefs returns the same numbers as shown in the parametric summary:
summary(m1)
# get_coefs is based on the function coef. This function returns
# values of all coefficients, and does not provide SE:
coef(m1)
```

`get_difference` *Get model predictions for differences between conditions.*

Description

Get model predictions for differences between conditions.

Usage

```
get_difference(
  model,
  comp,
  cond = NULL,
  rm.ranef = TRUE,
  se = TRUE,
  sim.ci = FALSE,
  f = 1.96,
  return.n.posterior = 0,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`comp`	A named list with the two levels to compare.
`cond`	A named list of the values to use for the other predictor terms. Variables omitted from this list will have the closest observed value to the median for continuous variables, or the reference level for factors.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE. Alternatively a vector of numbers with the mdoelterm number of the random effect(s) to remove. (See notes.)
`se`	Logical: whether or not to return the confidence interval or standard error around the estimates.
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. In addition, make sure to specify at least 200 points for each smooth for the simulations when using simultaneous CI. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`f`	A number to scale the standard error. Defaults to 1.96, resulting in 95% confidence intervals. For 99% confidence intervals use a value of 2.58.
`return.n.posterior`	Numeric: N samples from the posterior distribution of the fitted model are returned. Default value is 0 (no samples returned). Only workes when `sim.ci=TRUE`.
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).

Value

Returns a data frame with the estimates of the difference and optionally the confidence intervals around that estimate.

Notes

Other, not specified effects and random effects are generally canceled out, when calculating the difference. When the predictors that specify the conditions to compare are involved in other interactions or included as random slopes, it may be useful to specify the values of other predictors with `cond` or remove the random effects with `rm.ranef`.

Author(s)

Jacolien van Rij, Martijn Wieling

See Also

Other Model predictions: `get_coefs()`, `get_fitted()`, `get_modelterm()`, `get_predictions()`, `get_random()`

Examples

```
data(simdat)

# first fit a simple model:
m1 <- bam(Y ~ Group+te(Time, Trial, by=Group), data=simdat)

# get difference estimates:
diff <- get_difference(m1, comp=list(Group=c('Adults', 'Children')),
    cond=list(Time=seq(0,500,length=100)))
head(diff)
```

`get_fitted` *Get model all fitted values.*

Description

Get model all fitted values.

Usage

```
get_fitted(
  model,
  se = 1.96,
  rm.ranef = NULL,
  as.data.frame = FALSE,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`se`	A number to scale the standard error. Defaults to 1.96, resulting in 95% confidence intervals. For 99% confidence intervals use a value of 2.58.
`rm.ranef`	Logical: whether or not to remove random effects. Default is FALSE. Alternatively a string (or vector of strings) with the name of the random effect(s) to remove.
`as.data.frame`	Logical: return values as data frame or as vector. Default is FALSE (as vector).
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).

Value

A data frame with estimates and optionally errors.

Author(s)

Jacolien van Rij

See Also

Other Model predictions: `get_coefs()`, `get_difference()`, `get_modelterm()`, `get_predictions()`, `get_random()`

Examples

```
data(simdat)
## Not run:
m1 <- bam(Y ~ Group + s(Time, by=Group)+ s(Subject, bs='re'),
    data=simdat)

# as.data.frame FALSE and rm.ranef=NULL results in fitted():
all( get_fitted(m1) == fitted(m1) )

# now fitted values without random effects:
all( get_fitted(m1, rm.ranef=TRUE) == fitted(m1) )
head(get_fitted(m1, rm.ranef=TRUE))

# without summary:
infoMessages('off')
head(get_fitted(m1, rm.ranef=TRUE))
infoMessages('on')

## End(Not run)
```

`get_modelterm` *Get estimated for selected model terms.*

Description

Get estimated for selected model terms.

Usage

```
get_modelterm(
  model,
  select,
  cond = NULL,
  n.grid = 30,
  se = TRUE,
  f = 1.96,
  as.data.frame = TRUE,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`select`	A number, indicating the model term to be selected.
`cond`	A named list of the values to restrict the estimates for the predictor terms. When NULL (default) values are automatically selected. Only relevant for complex interactions, which involve more than two dimensions.
`n.grid`	Number of data points estimated for each random smooth.
`se`	Logical: whether or not to return the confidence interval or standard error around the estimates.
`f`	A number to scale the standard error. Defaults to 1.96, resulting in 95% confidence intervals. For 99% confidence intervals use a value of 2.58.
`as.data.frame`	Logical: whether or not to return a data.frame. Default is TRUE, and a list will be returned.
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).

Value

A data frame with estimates for the selected smooth term. Or a list with two or more elements:

- `fit`: Numeric vector with the fitted values;
- `se.fit`: Optionally, only with `se=TRUE`. Numeric vector with the error or confidence interval values (f*SE);
- `f`: The multiplication factor for generating the confidence interval values;
- `terms`: Numeric vector (for 1-dimensional smooth) or data frame (more 2- or more dimensional surfaces) with values of the modelterms.
- `title`: String with name of the model term.
- `xlab`, `ylab`, or `labels`: Labels for x-axis and optionally y-axis. Precise structure depends on type of smooth term: for 1-dimensional smooth only x-label is provided, for 2-dimensional smooths x-label and y-label are provided, for more complex smooths a vector of of labels is provided.

Author(s)

Jacolien van Rij

See Also

Other Model predictions: `get_coefs()`, `get_difference()`, `get_fitted()`, `get_predictions()`, `get_random()`

Examples

```
data(simdat)

## Not run:
# Model with random effect and interactions:
m1 <- bam(Y ~ s(Time) + s(Trial)
+ti(Time, Trial)
+s(Time, Subject, bs='fs', m=1),
data=simdat)

# Get data frame with predictions:
p <- get_modelterm(m1, select=1)
emptyPlot(range(p$terms), range(p$fit), h=0)
plot_error(p$terms, p$fit, p$se.fit, shade=TRUE, xpd=TRUE)

# Plot random effects in separate panels:
pp <- get_modelterm(m1, select=4, as.data.frame=TRUE)
require(lattice)
lattice::xyplot(fit~Time|Subject,
    data=pp, type='l',
    xlab='Time', ylab='Partial effect')

# Plot selection of subjects:
pp <- get_modelterm(m1, select=4,
    cond=list(Subject=c('a01', 'a03', 'c16')),
    as.data.frame=TRUE)
lattice::xyplot(fit~Time|Subject,
    data=pp, type='l',
    xlab='Time', ylab='Partial effect')

# Or using the package ggplot2:
require(ggplot2)
pp <- get_modelterm(m1, select=4, as.data.frame=TRUE)
pg <- ggplot2::qplot(Time, fit, data = pp,
    geom = c('point', 'line'), colour = Subject)
pg + ggplot2::guides(col = guide_legend(nrow = 18))

## End(Not run)
```

`get_pca_predictions` *Return PCA predictions.*

Description

Produces perspective or contour plot views of gam model predictions of the additive effects interactions. The code is based on the script for `vis.gam`, but allows to cancel random effects.

Usage

```
get_pca_predictions(
  x,
  pca.term = NULL,
  weights = NULL,
  view = NULL,
  cond = list(),
  select = NULL,
  n.grid = 30,
  se = 1.96,
  xlim = NULL,
  ylim = NULL,
  partial = TRUE,
  rm.ranef = NULL,
  as.data.frame = TRUE,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`x`	A gam object, produced by `gam` or `bam`.
`pca.term`	Text string, name of model predictor that represents a principle component.
`weights`	Named list with the predictors that are combined in the PC and their weights. See examples.
`view`	A two-value vector containing the names of the two terms to plot. The two terms should be part of the PC. Note that variables coerced to factors in the model formula won't work as view variables.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`select`	A number, selecting a single model term for printing. e.g. if you want the plot for the second smooth term set select=2.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
`xlim`	A two item array giving the lower and upper limits for the x- axis scale. NULL to choose automatically.
`ylim`	A two item array giving the lower and upper limits for the y- axis scale. NULL to choose automatically.
`partial`	Logical value: whether or not to plot the partial effect (TRUE) or the summed effect (FALSE, default).
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.

`as.data.frame`	Logical: whether the output is returned as data frame (TRUE, default) or as list (FALSE).
`print.summary`	Logical: whether or not to print a summary. Default set to the print info messages option (see `infoMessages`).

Author(s)

Jacolien van Rij

See Also

`plot_pca_surface`, `prcomp`

Other Functions for PCA interpretation: `plot_pca_surface()`

Examples

```
data(simdat)
# add hypothetical correlated term:
simdat$predictor <-  (simdat$Trial+10)^.75 + rnorm(nrow(simdat))
# principal components analysis:
pca <- prcomp(simdat[, c('Trial', 'predictor')])
# only first PC term contributes:
summary(pca)
# get rotation (weights of predictors in PC):
pcar <- pca$rotation
# add PC1 to data:
simdat$PC1 <- pca$x[,1]

## Not run:
# model:
m1 <- bam(Y ~ Group + te(Time, PC1, by=Group)
    + s(Time, Subject, bs='fs', m=1, k=5), data=simdat)
# inspect surface:
fvisgam(m1, view=c('Time', 'PC1'), cond=list(Group='Children'),
    rm.ranef=TRUE)
# how does Trial contribute?
p <- get_pca_predictions(m1, pca.term='PC1', weights=pcar[,'PC1'],
    view=c('Time', 'Trial'), cond=list(Group='Children'),
    rm.ranef=TRUE, partial=FALSE)
# Note that the range of Trial is estimated based on the values of PC1.
# A better solution is to specify the range:
p <- get_pca_predictions(m1, pca.term='PC1', weights=pcar[,'PC1'],
    view=list(Time=range(simdat$Time), Trial=range(simdat$Trial)),
    cond=list(Group='Children'),rm.ranef=TRUE, partial=FALSE)
# plotting of the surface:
plot_pca_surface(m1, pca.term='PC1', weights=pcar[,'PC1'],
    view=c('Time', 'Trial'), cond=list(Group='Children'),rm.ranef=TRUE)

## End(Not run)
```

`get_predictions` *Get model predictions for specific conditions.*

Description

Get model predictions for specific conditions.

Usage

```
get_predictions(
  model,
  cond = NULL,
  rm.ranef = TRUE,
  se = TRUE,
  sim.ci = FALSE,
  f = 1.96,
  return.n.posterior = 0,
  print.summary = getOption("itsadug_print")
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`cond`	A named list of the values to use for the predictor terms. Variables omitted from this list will have the closest observed value to the median for continuous variables, or the reference level for factors.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE. Alternatively a vector with numbers (modelterms) of the random effect(s) to remove.
`se`	Logical: whether or not to return the confidence interval or standard error around the estimates.
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. In addition, make sure to specify at least 200 points for each smooth for the simulations when using simultaneous CI. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`f`	A number to scale the standard error. Defaults to 1.96, resulting in 95% confidence intervals. For 99% confidence intervals use a value of 2.58.
`return.n.posterior`	Numeric: N samples from the posterior distribution of the fitted model are returned. Default value is 0 (no samples returned). Only workes when `sim.ci=TRUE`.
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).

Value

A data frame with estimates and optionally errors.

Author(s)

Jacolien van Rij

See Also

Other Model predictions: `get_coefs()`, `get_difference()`, `get_fitted()`, `get_modelterm()`, `get_random()`

Examples

```
data(simdat)

## Not run:
m1 <- bam(Y ~ Group + s(Time, by=Group), data=simdat)

# Time value is automatically set:
pp <- get_predictions(m1, cond=list(Group='Adults'))
head(pp)

# Range of time values:
pp <- get_predictions(m1,
    cond=list(Group='Adults', Time=seq(0,500,length=100)))
# plot:
emptyPlot(500, range(pp$fit), h=0)
plot_error(pp$Time, pp$fit, pp$CI, shade=TRUE, xpd=TRUE)

# Warning: also unrealistical values are possible
range(simdat$Time)
pp <- get_predictions(m1,
    cond=list(Group='Adults', Time=seq(-500,0,length=100)))
# plot of predictions that are not supported by data:
emptyPlot(c(-500,0), range(pp$fit), h=0)
plot_error(pp$Time, pp$fit, pp$CI, shade=TRUE, xpd=TRUE)

m2 <- bam(Y ~ Group + s(Time, by=Group)
    + s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
# Simultaneous CI vs pointwise CI
# NOTE: USE AT LEST 200 DATAPOINTS FOR SIMULTANEOUS CI
pp <- get_predictions(m2,
    cond=list(Group='Adults', Time=seq(0,2000,length=200)),
    rm.ranef=TRUE, sim.ci=TRUE)
head(pp)
# plot:
emptyPlot(2000, range(pp$fit), h=0)
plot_error(pp$Time, pp$fit, pp$CI, shade=TRUE, xpd=TRUE)
plot_error(pp$Time, pp$fit, pp$sim.CI, shade=FALSE, col=2, xpd=TRUE)
```

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

get_random — *Get coefficients for the random intercepts and random slopes.*

Description

Get coefficients for the random intercepts and random slopes.

Usage

```
get_random(model, cond = NULL, print.summary = getOption("itsadug_print"))
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`cond`	A named list of the values to restrict the estimates for the random predictor terms. When NULL (default) all levels are returned. Only relevant for complex interactions, which involve more than two dimensions.
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).

Value

The coefficients of the random intercepts and slopes.

Author(s)

Jacolien van Rij

See Also

Other Model predictions: `get_coefs`(), `get_difference`(), `get_fitted`(), `get_modelterm`(), `get_predictions`()

Examples

```
data(simdat)

## Not run:
# Condition as factor, to have a random intercept
# for illustration purposes:
simdat$Condition <- as.factor(simdat$Condition)

# Model with random effect and interactions:
```

```
m2 <- bam(Y ~ s(Time) + s(Trial)
+ ti(Time, Trial)
+ s(Condition, bs='re')
+ s(Time, Subject, bs='re'),
data=simdat)

# extract all random effects combined:
newd <- get_random(m2)
head(newd)

# extract coefficients for the random intercept for Condition:
# Make bar plot:
barplot(newd[[1]])
abline(h=0)

# or select:
get_random(m2, cond=list(Condition=c('2','3')))

## End(Not run)
```

---

info *Information on how to cite this package*

Description

Information on how to cite this package

Usage

```
info(input = NULL)
```

Arguments

`input`	Optional parameter. Normally (NULL) the citation info is printed. If value 'version' then only the version is printed.

See Also

`citation`, `R.version`, `sessionInfo`

Other Functions for package use: `infoMessages()`

Examples

```
info()
info('version')
citation(package='itsadug')
# To get info about R version:
R.version.string
```

`infoMessages` *Turn on or off information messages.*

Description

Turn on or off information messages.

Usage

```
infoMessages(input)
```

Arguments

`input`	Input variable indicating to print info messages ('on', or 1, or TRUE) or not ('off', 0, or FALSE).

See Also

Other Functions for package use: `info()`

Examples

```
# To turn on the info messages (all the same):
infoMessages('on')
infoMessages(1)
infoMessages(TRUE)
# To turn off the info messages (all the same):
infoMessages('off')
infoMessages(0)
infoMessages(FALSE)
# checking output:
(out <- infoMessages(FALSE))
```

`inspect_random` *Inspection and interpretation of random factor smooths.*

Description

Inspection and interpretation of random factor smooths.

Usage

```
inspect_random(
  model,
  select = 1,
  fun = NULL,
  cond = NULL,
  n.grid = 30,
  print.summary = getOption("itsadug_print"),
  plot = TRUE,
  add = FALSE,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ylim = NULL,
  h0 = 0,
  v0 = NULL,
  eegAxis = FALSE,
  ...
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`select`	A number, indicating the model term to be selected.
`fun`	A string or function description to apply to the random effects estimates. When NULL (default), the estimates for the random effects are returned.
`cond`	A named list of the values to restrict the estimates for the random predictor terms. When NULL (default) all levels are returned.
`n.grid`	Number of data points estimated for each random smooth.
`print.summary`	Logical: whether or not to print a summary of the values selected for each predictor. Default set to the print info messages option (see `infoMessages`).
`plot`	Logical: whether or not to plot the random effect estimates (TRUE by default).
`add`	Logical: whether or not to add the random effect estimates to an existing plot (FALSE by default).
`main`	Changing the main title for the plot, see also title.
`xlab`	Changing the label for the x axis, defaults to a description of x.
`ylab`	Changing the label for the y axis, defaults to a description of y.
`ylim`	Changing the y limits of the plot.
`h0`	A vector indicating where to add solid horizontal lines for reference. By default 0.
`v0`	A vector indicating where to add dotted vertical lines for reference. By default no values provided.
`eegAxis`	Whether or not to reverse the y-axis (plotting negative upwards).
`...`	other options to pass on to `lines`, see `par`

Value

A data frame with estimates for random effects is optionally returned.

Author(s)

Jacolien van Rij

See Also

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `gamtabs()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `plot_topo()`, `pvisgam()`

Examples

```
# load data:
data(simdat)

## Not run:
# Condition as factor, to have a random intercept
# for illustration purposes:
simdat$Condition <- as.factor(simdat$Condition)

# Model with random effect and interactions:
m2 <- bam(Y ~ s(Time) + s(Trial)
+ ti(Time, Trial)
+ s(Condition, bs='re')
+ s(Time, Subject, bs='fs', m=1),
data=simdat)

# extract with wrong select value:
newd <- inspect_random(m2, select=4)
# results in warning, automatically takes select=5
head(newd)
inspect_random(m2, select=5, cond=list(Subject=c('a01','a02','a03')))

# Alternatively, fix random effect of Condition, and plot
# random effects for subjects with lattice:
newd <- inspect_random(m2, select=5,
    cond=list(Subject=unique(simdat[simdat$Condition==0,'Subject'])),
    plot=FALSE)

# Make lattice plot:
require(lattice)
lattice::xyplot(fit~Time | Subject,
    data=newd, type='l',
    xlab='Time', ylab='Partial effect')

# Using argument 'fun':
inspect_random(m2, select=5, fun=mean,
    cond=list(Subject=unique(simdat[simdat$Condition==0,'Subject'])))
inspect_random(m2, select=5, fun=mean,
    cond=list(Subject=unique(simdat[simdat$Condition==2,'Subject'])),
        col='red', add=TRUE)

## End(Not run)

# see the vignette for examples:
vignette('overview', package='itsadug')
```

---

itsadug *Interpreting Time Series, Autocorrelated Data Using GAMMs (itsadug)*

Description

Itsadug provides a set of functions that facilitate the evaluation, interpretation, and visualization of GAMM models that are implemented in the package mgcv.

Tutorials

- `vignette('inspect', package='itsadug')` - summarizes different functions for visualizing the model.
- `vignette('test', package='itsadug')` - summarizes different functions for significance testing.
- `vignette('acf', package='itsadug')` - summarizes how to check and account for autocorrelation in the residuals.

Also available online on https://www.jacolienvanrij.com.

Interpretation and visualization

Main functions that are provided in `itsadug` for interpretation and visualization of GAMM models:

- `pvisgam` plots partial interaction surfaces; it also allows for visualizing 3-way or higher interactions.
- `fvisgam` plots summed interaction surfaces, with the possibility to exclude random effects.
- `plot_smooth` plots 1D model estimates, and has the possibility to exclude random effects.
- `plot_parametric` plot group estimates.
- `inspect_random` plots and optionally averages random smooths
- `plot_data` plots the data
- `plot_topo` plots EEG topographies

Testing for significance

- `compareML` Performs Chisquare test on two models
- `plot_diff` Calculates and visualizes the difference between two conditions within a model
- `plot_diff2` Calculates and visualizes the 2 dimensional difference between two conditions within a model

Evaluation of the model

- `check_resid` plots four different plots to inspect the distribution of and structure in the residuals
- `plot_modelfit` plots an overlay of the data and the modelfit for randomly selected trials
- `diagnostics` produces plots of the distributions of residuals and predictors in the model

Checking and handling autocorrelation

- `acf_resid` different ways to inspect autocorrelation in the residuals
- `start_event` creates an AR.start column
- `resid_gam` returns residuals corrected for the AR1 model

Predictions

Further, there are some wrappers around the `predict.gam` function to facilitate the extraction of model predictions. These can be used for customized plots. See for an example in the vignette 'plotfunctions' (`vignette('plotfunctions', package='itsadug')`).

- `get_predictions` for getting the estimates for given settings of some or all of the model predictors;
- `get_difference` for extracting the difference between two conditions or two smooths or two surfaces.
- `get_modelterm` for extracting the smooth term (partial) estimates.
- `inspect_random` and `get_random` for extracting random effects only.

Notes

- Use `infoMessages(FALSE)` to suppress all information messages for the current session. This may be helpful when creating knitr or R markdown reports.
- The vignettes are available via `browseVignettes()`. When working on a server via the command line, using `ssh -X` instead of `ssh` may make the HTML files available.
- A list of all available functions is provided in `help(package='itsadug')`.

Author(s)

Jacolien van Rij, Martijn Wieling, R.Harald Baayen, Hedderik van Rijn

Maintainer: Jacolien van Rij (<`vanrij.jacolien@gmail.com`>)

University of Groningen, The Netherlands

`missing_est` *Return indices of data that were not fitted by the model.*

Description

Return indices of data that were not fitted by the model.

Usage

```
missing_est(model)
```

Arguments

`model`	A fitted regression model (using lm, glm, gam, or bam).

Value

The indices of the data that were not fitted by the model.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

# Add missing values:
set.seed(123)
simdat[sample(nrow(simdat), size=20),]$Y <- NA
# Fit simple linear model:
lm1 <- lm(Y ~ Time, data=simdat)
na.el <- missing_est(lm1)
length(na.el)
```

modeledf *Retrieve the degrees of freedom specified in the model.*

Description

Retrieve the degrees of freedom specified in the model.

Usage

```
modeledf(model)
```

Arguments

`model`	A fitted regression model (using gam, or bam).

Value

Numeric value: degrees of freedom specified in the model.

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

## Not run:
# models take somewhat longer time to run:

# Fit simple GAM model:
gam1 <- bam(Y ~ s(Time), data=simdat, discrete=TRUE)
modeledf(gam1)
gam2 <- bam(Y ~ s(Time)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
modeledf(gam2)
gam3 <- bam(Y ~ Subject+s(Time, by=Subject),
    data=simdat, discrete=TRUE)
modeledf(gam3)
gam4 <- bam(Y ~ Group+s(Time)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
modeledf(gam4)
gam5 <- bam(Y ~ Group+s(Time, by=Group)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
modeledf(gam5)

# Fit a gamm:
```

```
gam6 <- gamm(Y ~ Group+s(Time), random=list(Subject=~1) data=simdat, discrete=TRUE)
# this produces an error...
modeledf(gam6)
# ... but this works:
modeledf(gam6$gam)

## End(Not run)
```

---

observations *Number of observations in the model.*

Description

Number of observations in the model.

Usage

```
observations(model)
```

Arguments

`model`	A fitted regression model (using gam, bam, (g)lm, (g)lmer).

Value

Numeric value: number of observations that are considered by the model.

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)
# simulate some missing data:
simdat[sample(1:nrow(simdat), size=15),]$Y <- NA
simdat[sample(1:nrow(simdat), size=7),]$Group <- NA

# Fit simple GAM models:
gam1 <- bam(Y ~ s(Time), data=simdat, discrete=TRUE)
gam2 <- bam(Y ~ Group + s(Time, by=Group), data=simdat, discrete=TRUE)

# number of data points in data frame:
nrow(simdat)

# observations model gam1:
observations(gam1)
# observations model gam2:
```

```
observations(gam2)
```

`plot_data` *Visualization of the model fit for time series data.*

Description

Plots the data, fitted values, or residuals.

Usage

```
plot_data(
  model,
  view,
  split_by = NULL,
  cond = NULL,
  input = "data",
  rm.ranef = NULL,
  alpha = NULL,
  col = NULL,
  add = FALSE,
  eegAxis = FALSE,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ylim = NULL,
  h0 = 0,
  v0 = NULL,
  hide.label = FALSE,
  transform = NULL,
  transform.view = NULL,
  print.summary = getOption("itsadug_print"),
  ...
)
```

Arguments

`model`	A lm or gam object, produced by `gam` or `bam`, `lm`, `glm`.
`view`	Text string containing the predictor or column in the data to be displayed on the x-axis. Note that variables coerced to factors in the model formula won't work as view variables.
`split_by`	Vector with names of model predictors that determine the time series in the data, or should be used to split the ACF plot by. Alternatively, `split_pred` can be a named list (each with equal length as the data) that group the data, fitted values or residuals values of x into trials or timeseries events. Generally other columns from the same data frame as the model was fitted on.

`cond`	A named list of the values to use for the other predictor terms (not in view) or to select specific trials or time series to plot.
`input`	Text string: 'data' (default) plots the data, 'resid' plots the model residuals, and 'fitted' plots the fitted values.
`rm.ranef`	Logical: whether or not to include the random effects in the model predictions. Default is TRUE. Relevant for `input='fitted'` and `input='resid'` (i.e., whether or not the residuals contain the random effects, TRUE and FALSE respectively).
`alpha`	Value between 0 and 1 indicating the transparency. A value of 0 is completely transparant, whereas a value of 1 is completely untransparant.
`col`	Vector with one color value (i.e., all data points will have the same color), color values for each grouping condition specified in `split_by` or a vector with color values for each data point.
`add`	Logical: whether or not to add the lines/points to an existing plot, or start a new plot (default).
`eegAxis`	Logical: whether or not to reverse the y-axis, plotting the negative amplitudes upwards as traditionally is done in EEG research. If eeg.axes is TRUE, labels for x- and y-axis are provided, when not provided by the user. Default value is FALSE.
`main`	Changing the main title for the plot, see also title.
`xlab`	Changing the label for the x axis, defaults to a description of x.
`ylab`	Changing the label for the y axis, defaults to a description of y.
`ylim`	the y limits of the plot.
`h0`	A vector indicating where to add solid horizontal lines for reference. By default no values provided.
`v0`	A vector indicating where to add dotted vertical lines for reference. By default no values provided.
`hide.label`	Logical: whether or not to hide the label (i.e., 'fitted values'). Default is FALSE.
`transform`	Function for transforming the fitted values. Default is NULL.
`transform.view`	Function for transforming the view values. Default is NULL.
`print.summary`	Logical: whether or not to print a summary. Default set to the print info messages option (see `infoMessages`).
`...`	other options to pass on to lines and plot, see `par`

Notes

This function plots the fitted effects, including intercept and other predictors.

Author(s)

Jacolien van Rij, idea of Tino Sering

See Also

Other Functions for model inspection: dispersion(), fvisgam(), gamtabs(), inspect_random(), plot_parametric(), plot_smooth(), plot_topo(), pvisgam()

Examples

```
data(simdat)

## Not run:
# Create grouping predictor for time series:
simdat$Event <- interaction(simdat$Subject, simdat$Trial)

# model without random effects:
m1 <- bam(Y ~ te(Time, Trial) + s(Subject, bs='re'),
    data=simdat)

# All data points, without clustering:
plot_data(m1, view='Time')

# All data, clustered by Trial (very small dots):
plot_data(m1, view='Time', split_by='Trial',
    cex=.25)
# Add a smooth for each trial:
plot_smooth(m1, view='Time', plot_all='Trial',
    add=TRUE, rm.ranef=TRUE)
# Add the model predictions in same color:
plot_smooth(m1, view='Time', plot_all='Trial', add=TRUE, rm.ranef=TRUE)

# Alternatively, use data to select events:
plot_data(m1, view='Time', split_by=list(Event=simdat$Event),
    type='l')
# which is the same as:
plot_data(m1, view='Time', split_by=list(Subject=simdat$Subject, Trial=simdat$Trial),
    type='l')
# Only for Trial=0
plot_data(m1, view='Time', split_by=list(Event=simdat$Event),
   cond=list(Trial=0), type='l')
# This is the same:
plot_data(m1, view='Time', split_by='Subject',
   cond=list(Trial=0), type='l')
# Add subject smooths:
plot_smooth(m1, view='Time', plot_all='Subject',
    cond=list(Trial=0), add=TRUE)

# Change the colors:
plot_data(m1, view='Time', split_by='Subject',
   cond=list(Trial=0), type='l', col='gray', alpha=1)

## End(Not run)
```

plot_diff *Plot difference curve based on model predictions.*

Description

Plot difference curve based on model predictions.

Usage

```
plot_diff(
  model,
  view,
  comp,
  cond = NULL,
  se = 1.96,
  sim.ci = FALSE,
  n.grid = 100,
  add = FALSE,
  rm.ranef = TRUE,
  mark.diff = TRUE,
  col.diff = "red",
  col = "black",
  eegAxis = FALSE,
  transform.view = NULL,
  print.summary = getOption("itsadug_print"),
  plot = TRUE,
  main = NULL,
  ylab = NULL,
  xlab = NULL,
  xlim = NULL,
  ylim = NULL,
  hide.label = FALSE,
  ...
)
```

Arguments

`model`	A GAMM model, resulting from the functions `gam` or `bam`.
`view`	Name of continuous predictor that should be plotted on the x- axis.
`comp`	Named list with the grouping predictor (categorical variable) and the 2 levels to calculate the difference for.
`cond`	A named list of the values to use for the predictor terms. Variables omitted from this list will have the closest observed value to the median for continuous variables, or the reference level for factors.

`se`	If less than or equal to zero then only the predicted smooth is plotted, but if greater than zero, then the predicted values plus confidence intervals are plotted. The value of `se` will be multiplied with the standard error (i.e., 1.96 results in 95%CI and 2.58). Default is set to 1.96 (95%CI).
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`n.grid`	Number of data points sampled as predictions. Defaults to 100.
`add`	Logical: whether or not to add the line to an existing plot. Default is FALSE. When no plot window is available and `add=TRUE`, the function will generate an error.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE. Alternatively a string (or vector of strings) with the name of the random effect(s) to remove.
`mark.diff`	Logical: whether or not marking where the difference is significantly different from 0.
`col.diff`	Color to mark differences (red by default).
`col`	Line color. Shading color is derived from line color.
`eegAxis`	Logical: whether or not to reverse the y-axis, plotting negative values upwards. Default is FALSE.
`transform.view`	Function for transforming the values on the x-axis. Defaults to NULL (no transformation). (See `plot_smooth` for more info.)
`print.summary`	Logical: whether or not to print the summary. Default set to the print info messages option (see `infoMessages`).
`plot`	Logical: whether or not to plot the difference. If FALSE, then the output is returned as a list, with the estimated difference (`est`) and the standard error over the estimate (`se.est`) and the x-values (`x`). Default is TRUE.
`main`	Text string, alternative title for plot.
`ylab`	Text string, alternative label for y-axis.
`xlab`	Text string, alternative label for x-axis.
`xlim`	Range of x-axis. If not specified, the function automatically generates an appropriate x-axis.
`ylim`	Range of y-axis. If not specified, the function automatically generates an appropriate y-axis.
`hide.label`	Logical: whether or not to hide the label (i.e., 'difference'). Default is FALSE.
`...`	Optional arguments for `emptyPlot`, or `plot_error`.

Value

If the result is not being plotted, a list is returned with the estimated difference (`est`) and the standard error over the estimate (`se`) and the x-values (`x`) is returned.

Author(s)

Martijn Wieling, Jacolien van Rij

See Also

Other Testing for significance: `compareML()`, `plot_diff2()`, `report_stats()`, `wald_gam()`

Examples

```
data(simdat)
## Not run:
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group)
    + s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')))
# in this model, excluding random effects does not change the difference:
plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')),
    rm.ranef=TRUE)
# simultaneous CI:
plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')),
    rm.ranef=TRUE, sim.ci=TRUE)
# Reversed y-axis (for EEG data) and no shading:
plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')),
    eegAxis=TRUE, shade=FALSE)
plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')),
density=15, angle=90, ci.lwd=3)
# Retrieving plot values...
out <- plot_diff(m1, view='Time', comp=list(Group=c('Children', 'Adults')),
   plot=FALSE)
#... which might be used for indicating differences:
x <- find_difference(out$est, out$se, f=1.96, xVals=out$xVals)
# add lines:
arrows(x0=x$start, x1=x$end, y0=0, y1=0,code=3, length=.1, col='red')

## End(Not run)
```

`plot_diff2` *Plot difference surface based on model predictions.*

Description

Plot difference surface based on model predictions.

Usage

```
plot_diff2(
  model,
  view,
  comp,
  cond = NULL,
  color = "terrain",
  nCol = 100,
  col = NULL,
  add.color.legend = TRUE,
  se = 1.96,
  sim.ci = FALSE,
  show.diff = FALSE,
  col.diff = 1,
  alpha.diff = 0.5,
  n.grid = 30,
  nlevels = 10,
  zlim = NULL,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  rm.ranef = TRUE,
  transform.view = NULL,
  hide.label = FALSE,
  dec = NULL,
  f = 1.96,
  print.summary = getOption("itsadug_print"),
  ...
)
```

Arguments

`model`	A GAMM model, resulting from the functions `gam` or `bam`.
`view`	Name of continuous predictors that should be plotted on the x- and y-axes. Vector of two values.
`comp`	Named list with the grouping predictor (categorical variable) and the 2 levels to calculate the difference for.
`cond`	Named list of the values to use for the other predictor terms (not in view).
`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray' or 'bw'. Alternatively a vector with some colors can be provided for a custom color palette.
`nCol`	Range of colors of background of contour plot.
`col`	Line color.

`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then the predicted values plus confidence intervals are plotted. The value of `se` will be multiplied with the standard error (i.e., 1.96 results in 95%CI and 2.58). Default is set to 1.96 (95%CI).
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`show.diff`	Logical: whether or not to indicate the regions that are significantly different from zero. Note that these regions are just an indication and dependent on the value of `n.grid`. Defaults to FALSE.
`col.diff`	Color to shade the nonsignificant areas.
`alpha.diff`	Level of transparency to mark the nonsignificant areas.
`n.grid`	Resolution.
`nlevels`	Levels of contour lines.
`zlim`	A two item array giving the lower and upper limits for the z- axis scale. NULL to choose automatically.
`xlim`	A two item array giving the lower and upper limits for the x- axis scale. NULL to choose automatically.
`ylim`	A two item array giving the lower and upper limits for the y- axis scale. NULL to choose automatically.
`main`	Title of plot.
`xlab`	Label x-axis.
`ylab`	Label y-axis.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE. Alternatively a string (or vector of strings) with the name of the random effect(s) to remove.
`transform.view`	List with two functions for transforming the values on the x- and y-axis respectively. If one of the axes need to be transformed, set the other to NULL (no transformation). (See `fvisgam` for more info.)
`hide.label`	Logical: whether or not to hide the label (i.e., 'difference'). Default is FALSE.
`dec`	Numeric: number of decimals for rounding the color legend. When NULL (default), no rounding. If -1 (default), automatically determined. Note: if value = -1 (default), rounding will be applied also when `zlim` is provided.

f	Scaling factor to determine the CI from the se, for marking the difference with 0. Only applies when se is smaller or equal to zero and show.diff is set to TRUE.
print.summary	Logical: whether or not to print a summary. Default set to the print info messages option (see infoMessages).
...	Optional arguments for plotsurface.

Value

If the result is not being plotted, a list is returned with the estimated difference (est) and the standard error over the estimate (se.est) and the x-values (x) is returned.

Warning

When the argument show.diff is set to TRUE a shading area indicates where the confidence intervals include zero. Or, in other words, the areas that are not significantly different from zero. Be careful with the interpretation, however, as the precise shape of the surface is dependent on model constraints such as the value of choose.k and the smooth function used, and the size of the confidence intervals are dependent on the model fit and model characteristics (see vignette('acf', package='itsadug')). In addition, the value of n.grid determines the precision of the plot.

Author(s)

Martijn Wieling, reimplemented by Jacolien van Rij

See Also

Other Testing for significance: compareML(), plot_diff(), report_stats(), wald_gam()

Examples

```
data(simdat)
## Not run:
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group),
    data=simdat)
plot_diff2(m1, view=c('Time', 'Trial'),
    comp=list(Group=c('Children', 'Adults')))

## End(Not run)
```

---

plot_modelfit *Visualization of the model fit for time series data.*

Description

Plots the fitted values and the data for n trials of time series data. For example, plots n trials of the same participant.

Usage

```
plot_modelfit(
  x,
  view,
  event = NULL,
  n = 3,
  random = TRUE,
  cond = NULL,
  col = c(alpha(1), "red"),
  add = FALSE,
  eegAxis = FALSE,
  fill = FALSE,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ylim = NULL,
  h0 = 0,
  v0 = NULL,
  transform = NULL,
  hide.label = FALSE,
  hide.legend = FALSE,
  print.summary = getOption("itsadug_print"),
  ...
)
```

Arguments

`x`	A lm or gam object, produced by `gam` or `bam`, `lm`, `glm`.
`view`	Text string containing the predictor or column in the data to be displayed on the x-axis. Note that variables coerced to factors in the model formula won't work as view variables.
`event`	column name from the data that specifies the time series from which `n` are being plotted.
`n`	Number of time series to plot. Default is 3. Set to -1 for plotting all time series (which may take a considerable time).
`random`	Numeric: if set to TRUE (default), `n` random events are selected to plot. If set to FALSE, the first `n` events are selected to plot. The events could be precisely controlled with the argument `cond`.
`cond`	A named list of the values to use for the other predictor terms (not in view) or to select specific trials or time series to plot.
`col`	Two value vector specifiying the colors for the data and the modelfit respectively.
`add`	Logical: whether or not to add the lines to an existing plot, or start a new plot (default).
`eegAxis`	Logical: whether or not to reverse the y-axis, plotting the negative amplitudes upwards as traditionally is done in EEG research. If eeg.axes is TRUE, labels for x- and y-axis are provided, when not provided by the user. Default value is FALSE.

`fill`	Logical: whether or not to fill the area between the data and the fitted values with shading. Default is FALSE.
`main`	Changing the main title for the plot, see also title.
`xlab`	Changing the label for the x axis, defaults to a description of x.
`ylab`	Changing the label for the y axis, defaults to a description of y.
`ylim`	the y limits of the plot.
`h0`	A vector indicating where to add solid horizontal lines for reference. By default no values provided.
`v0`	A vector indicating where to add dotted vertical lines for reference. By default no values provided.
`transform`	Function for transforming the fitted values. Default is NULL.
`hide.label`	Logical: whether or not to hide the label (i.e., 'fitted values'). Default is FALSE.
`hide.legend`	Logical: whether or not to hide the legend. Default is FALSE.
`print.summary`	Logical: whether or not to print a summary. Default set to the print info messages option (see `infoMessages`).
`...`	other options to pass on to lines and plot, see `par`

Notes

This function plots the fitted effects, including intercept and other predictors.

Author(s)

Jacolien van Rij

See Also

Other Model evaluation: `check_resid()`, `diagnostics()`

Examples

```
data(simdat)

# Create grouping predictor for time series:
simdat$Event <- interaction(simdat$Subject, simdat$Trial)

# model without random effects:
m1 <- bam(Y ~ te(Time, Trial),
    data=simdat)
plot_modelfit(m1, view='Time', event=simdat$Event)

# colorizing residuals:
plot_modelfit(m1, view='Time', event=simdat$Event, fill=TRUE)

# All trials of one subject:
## Not run:
# this produces error:
```

```
plot_modelfit(m1, view='Time', event=simdat$Event,
    cond=list(Subject='a01'), n=-1)

## End(Not run)
# instead try this:
simdat$Subj <- ifelse(simdat$Subject=='a01', TRUE, FALSE)
plot_modelfit(m1, view='Time', event=simdat$Event,
    cond=list(Subject=simdat$Subj), n=-1)

## Not run:
# Model with random intercepts for subjects:
m2 <- bam(Y ~ te(Time, Trial)+s(Subject, bs='re'),
    data=simdat)
# now selecting a subject works, because it is in the model:
plot_modelfit(m2, view='Time', event=simdat$Event,
    cond=list(Subject='a01'), n=-1, ylim=c(-13,13))

# Model with random effect and interactions:
m3 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat)
plot_modelfit(m3, view='Time', event=simdat$Event,
    cond=list(Subject='a01'), n=-1, ylim=c(-13,13))

## End(Not run)
```

plot_parametric *Visualization of group estimates.*

Description

Plots a smooth from a gam or bam model based on predictions. In contrast with the default plot.gam, this function plots the summed effects and optionally removes the random effects.

Usage

```
plot_parametric(
  x,
  pred,
  cond = list(),
  parametricOnly = FALSE,
  rm.ranef = TRUE,
  col = "black",
  se = 1.96,
  print.summary = getOption("itsadug_print"),
  main = NULL,
  xlab = NULL,
  ...
)
```

Arguments

`x`	A gam object, produced by `gam` or `bam`.
`pred`	A named list of the values to use for the predictor terms to plot.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`parametricOnly`	Logical: whether or not to cancel out all smooth terms and only use the predictors in the parametric summary.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.
`col`	The colors for the lines and the error bars of the plot.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then the predicted values plus confidence intervals are plotted. The value of se will be multiplied with the standard error (i.e., 1.96 results in 95%CI and 2.58).
`print.summary`	Logical: whether or not to print summary. Default set to the print info messages option (see `infoMessages`).
`main`	Changing the main title for the plot, see also title.
`xlab`	Changing the label for the x axis, defaults to a description of x.
`...`	other options to pass on to `dotplot_error`, see `par`

Warning

Use `parametricOnly` with care! When set to TRUE, all smooth predictors are set to 0. Note that this might result in strange predictions, because a value of 0 does not always represents a realistic situation (e.g., body temperature of 0 is highly unlikely). Note that linear slopes are not set to zero, because they are considered as parametric terms. If `cond` does not specify a value for these continuous predictors, the closes value to the mean is automatically selected.

Author(s)

Jacolien van Rij, based on a function of Fabian Tomaschek

See Also

`plot.gam`

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_smooth()`, `plot_topo()`, `pvisgam()`

Examples

```
data(simdat)
## Not run:
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group)
    + s(Time, Subject, bs='fs', m=1), data=simdat)
plot_parametric(m1, pred=list(Group=c('Adults', 'Children')))
# Note the summary that is printed.
# use rm.ranef to cancel random effects:
plot_parametric(m1, pred=list(Group=c('Adults', 'Children')),
    rm.ranef = TRUE)

# It is possible to get estimates that do not make sense:
out <- plot_parametric(m1,
    pred=list(Group=c('Adults', 'Children'), Subject=c('a01', 'a02', 'c01')))
print(out)

## End(Not run)

# see the vignette for examples:
vignette('overview', package='itsadug')
```

`plot_pca_surface` *Visualization of the effect predictors in nonlinear interactions with principled components.*

Description

Produces perspective or contour plot views of gam model predictions of the additive effects interactions. The code is based on the script for `vis.gam`, but allows to cancel random effects.

Usage

```
plot_pca_surface(
  x,
  pca.term = NULL,
  weights = NULL,
  view = NULL,
  cond = list(),
  partial = FALSE,
  select = NULL,
  se = -1,
  n.grid = 30,
  too.far = 0,
  rm.ranef = NULL,
  col = NA,
  color = "topo",
  contour.col = NULL,
  nCol = 50,
  plotCI = FALSE,
  add.color.legend = TRUE,
  plot.type = "contour",
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  print.summary = getOption("itsadug_print"),
  transform = NULL,
  transform.view = NULL,
  hide.label = FALSE,
  dec = NULL,
  ...
)
```

Arguments

`x`	A gam object, produced by `gam` or `bam`.
`pca.term`	Text string, name of model predictor that represents a principle component.
`weights`	Named list with the predictors that are combined in the PC and their weights. See examples.
`view`	A two-value vector containing the names of the two terms to plot. The two terms should be part of the PC. Note that variables coerced to factors in the model formula won't work as view variables.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`partial`	Logical value: whether or not to plot the partial effect (TRUE) or the summed effect (FALSE, default).
`select`	Numeric value, model term. In case `partial=TRUE` a model term needs to be selected.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`too.far`	Plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.
`col`	The colors for the facets of the plot.
`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray' or 'bw'.
`contour.col`	sets the color of contours when using plot.
`nCol`	The number of colors to use in color schemes.
`plotCI`	Logical: whether or not to plot the confidence intervals. The value of `se` determines the size of the CI.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.

`plot.type`	one of 'contour' or 'persp' (default is 'contour').
`xlim`	A two item array giving the lower and upper limits for the x- axis scale. NULL to choose automatically.
`ylim`	A two item array giving the lower and upper limits for the y- axis scale. NULL to choose automatically.
`zlim`	A two item array giving the lower and upper limits for the z- axis scale. NULL to choose automatically.
`print.summary`	Logical: whether or not to print a summary. Default set to the print info messages option (see `infoMessages`).
`transform`	Function for transforming the fitted values. Default is NULL.
`transform.view`	List with two functions for transforming the values on the x- and y-axis respectively. If one of the axes need to be transformed, set the other to NULL (no transformation). See examples below.
`hide.label`	Logical: whether or not to hide the label (i.e., 'fitted values'). Default is FALSE.
`dec`	Numeric: number of decimals for rounding the color legend. When NULL, no rounding (default). If -1, automatically determined. Note: if value = -1, rounding will be applied also when `zlim` is provided.
`...`	other options to pass on to persp, image or contour. In particular ticktype='detailed' will add proper axes labeling to the plots.

Author(s)

Jacolien van Rij data(simdat) # add hypothetical correlated term: simdat$predictor <- (simdat$Trial+10)^.75 + rnorm(nrow(simdat)) # principal components analysis: pca <- prcomp(simdat[, c('Trial', 'predictor')]) # only first PC term contributes: summary(pca) # get rotation (weights of predictors in PC): pcar <- pca$rotation # add PC1 to data: simdat$PC1 <- pca$x[,1]

# model: m1 <- bam(Y ~ Group + te(Time, PC1, by=Group) + s(Time, Subject, bs='fs', m=1, k=5), data=simdat) # inspect surface: fvisgam(m1, view=c('Time', 'PC1'), cond=list(Group='Children'), rm.ranef=TRUE) # how does Trial contribute? plot_pca_surface(m1, pca.term='PC1', weights=pcar[,'PC1'], view=c('Time', 'Trial'), cond=list(Group='Children'),rm.ranef=TRUE) # Note that the range of Trial is estimated based on the values of PC1. # A better solution is to specify the range: plot_pca_surface(m1, pca.term='PC1', weights=pcar[,'PC1'], view=list(Time=range(simdat$Time), Trial=range(simdat$Trial)), cond=list(Group='Children'),rm.ranef=TRUE)

# Partial effects: pvisgam(m1, view=c('Time', 'PC1'), cond=list(Group='Children'), select=1, rm.ranef=TRUE)
# PCA: plot_pca_surface(m1, pca.term='PC1', weights=pcar[,'PC1'], partial=TRUE, select=1, view=list(Time=range(simda
Trial=range(simdat$Trial)), cond=list(Group='Children'))

See Also

`fvisgam`, `pvisgam`

Other Functions for PCA interpretation: `get_pca_predictions`()

plot_smooth *Visualization of smooths.*

Description

Plots a smooth from a gam or bam model based on predictions. In contrast with the default plot.gam, this function plots the summed effects and optionally removes the random effects.

Usage

```
plot_smooth(
  x,
  view = NULL,
  cond = list(),
  plot_all = NULL,
  rm.ranef = TRUE,
  n.grid = 30,
  rug = NULL,
  add = FALSE,
  se = 1.96,
  sim.ci = FALSE,
  shade = TRUE,
  eegAxis = FALSE,
  col = NULL,
  lwd = NULL,
  lty = NULL,
  print.summary = getOption("itsadug_print"),
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  h0 = 0,
  v0 = NULL,
  transform = NULL,
  transform.view = NULL,
  legend_plot_all = NULL,
  hide.label = FALSE,
  ...
)
```

Arguments

x	A gam object, produced by gam or bam.
view	Text string containing the name of the smooth to be displayed. Note that variables coerced to factors in the model formula won't work as view variables.

`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`plot_all`	A vector with a name / names of model predictors, for which all levels should be plotted.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`rug`	Logical: when TRUE then the covariate to which the plot applies is displayed as a rug plot at the foot of each plot. By default set to NULL, which sets rug to TRUE when the dataset size is <= 10000 and FALSE otherwise. Setting to FALSE will speed up plotting for large datasets.
`add`	Logical: whether or not to add the lines to an existing plot, or start a new plot (default).
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then the predicted values plus confidence intervals are plotted. The value of `se` will be multiplied with the standard error (i.e., 1.96 results in 95%CI and 2.58). Default is set to 1.96 (95%CI).
`sim.ci`	Logical: Using simultaneous confidence intervals or not (default set to FALSE). The implementation of simultaneous CIs follows Gavin Simpson's blog of December 15, 2016: `https://fromthebottomoftheheap.net/2016/12/15/simultaneous-interval-r` This interval is calculated from simulations based. Please specify a seed (e.g., `set.seed(123)`) for reproducable results. Note: in contrast with Gavin Simpson's code, here the Bayesian posterior covariance matrix of the parameters is uncertainty corrected (`unconditional=TRUE`) to reflect the uncertainty on the estimation of smoothness parameters.
`shade`	Logical: Set to TRUE to produce shaded regions as confidence bands for smooths (not avaliable for parametric terms, which are plotted using termplot).
`eegAxis`	Logical: whether or not to reverse the y-axis, plotting the negative amplitudes upwards as traditionally is done in EEG research. If eeg.axes is TRUE, labels for x- and y-axis are provided, when not provided by the user. Default value is FALSE.
`col`	The colors for the lines and the error bars of the plot.
`lwd`	The line width for the lines of the plot.
`lty`	The line type for the lines of the plot.
`print.summary`	Logical: whether or not to print summary. Default set to the print info messages option (see `infoMessages`).
`main`	Changing the main title for the plot, see also title.
`xlab`	Changing the label for the x axis, defaults to a description of x.
`ylab`	Changing the label for the y axis, defaults to a description of y.
`xlim`	the x limits of the plot.
`ylim`	the y limits of the plot.
`h0`	A vector indicating where to add solid horizontal lines for reference. By default no values provided.

`v0`	A vector indicating where to add dotted vertical lines for reference. By default no values provided.
`transform`	Function for transforming the fitted values. Default is NULL.
`transform.view`	Function for transforming the values on the x-axis. Defaults to NULL (no transformation).
`legend_plot_all`	Legend location. This could be a keyword from the list 'bottomright', 'bottom', 'bottomleft', 'left', 'topleft', 'top', 'topright', 'right' and 'center', or a list with `x` and `y` coordinate (e.g., `list(x=0,y=0)`).
`hide.label`	Logical: whether or not to hide the label (i.e., 'fitted values'). Default is FALSE.
`...`	other options to pass on to lines and plot, see `par`

Notes

This function plots the summed effects, including intercept and other predictors. For plotting partial effects, see the function `plot.gam`, or see the examples with `get_modelterm` for more flexibility (e.g., plotting using the `lattice` package or `ggplots`).

Author(s)

Jacolien van Rij and Martijn Wieling.

See Also

`plot.gam`, `plot_diff`

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_topo()`, `pvisgam()`

Examples

```
data(simdat)

## Not run:
# Model with random effect and interactions:
m1 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)

# Default plot produces only surface of Time x Trial:
plot(m1, select=1)
# Only the Time component:
plot_smooth(m1, view='Time')
# Note the summary that is printed.

# without random effects:
plot_smooth(m1, view='Time', rm.ranef=TRUE)

# Plot summed effects:
dev.new(width=8, height=4) # use x11(,8,4) on Linux
par(mfrow=c(1,2))
```

```
fvisgam(m1, view=c('Time', 'Trial'),
    plot.type='contour', color='topo', main='interaction',
    rm.ranef=TRUE)
arrows(x0=0, x1=2200, y0=-5, y1=-5, col='red',
    code=2, length=.1, lwd=2, xpd=TRUE)
plot_smooth(m1, view='Time', cond=list(Trial=-5),
    main='Trial=-5', rm.ranef=TRUE)


# Model with random effect and interactions:
m2 <- bam(Y ~ Group + s(Time, by=Group)
    +s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)

# Plot all levels of a predictor:
plot_smooth(m2, view='Time', plot_all='Group',
    rm.ranef=TRUE)
# It also possible to combine predictors in plot_all.
# Note: this is not a meaningfull plot, because Subjects
# fall in only one group. Just for illustration purposes!
plot_smooth(m2, view='Time', plot_all=c('Group', 'Subject'))
# Clearly visible difference in confidence interval, because
# a01 does not occur in Group 'Children':
# (Note that this plot generates warning)
plot_smooth(m2, view='Time', plot_all=c('Group', 'Subject'), cond=list(Subject='a01'))

# Using sim.ci: simultaneous CI instead of pointwise CI
dev.new(width=8, height=4) # use x11(,8,4) on Linux
par(mfrow=c(1,2))
plot_smooth(m2, view='Time', plot_all='Group', rm.ranef=TRUE)
plot_smooth(m2, view='Time', rm.ranef=TRUE, plot_all='Group', sim.ci=TRUE,
    add=TRUE, shade=FALSE, xpd=TRUE)
plot_smooth(m2, view='Time', rm.ranef=TRUE, sim.ci=TRUE, col='red')



# Using transform
# Plot log-transformed dependent predictor on original scale:
plot_smooth(m1, view='Time', rm.ranef=TRUE, transform=exp)

# Notes on transform.view:
# This will generate an error, because x-values <= 0 will result in NaN:
plot_smooth(m1, view='Time', rm.ranef=TRUE, transform.view=log)
# adjusting the x-axis helps:
plot_smooth(m1, view='Time', rm.ranef=TRUE, transform.view=log,
   xlim=c(1,2000))


## End(Not run)

# and for a quick overview of plotfunctions:
vignette('overview', package='itsadug')
```

plot_topo *Visualization of EEG topo maps.*

Description

Visualization of EEG topo maps.

Usage

```
plot_topo(
  model,
  view,
  el.pos = NULL,
  fun = "fvisgam",
  add.color.legend = TRUE,
  size = 5,
  n.grid = 100,
  col = 1,
  pch = 21,
  bg = alpha(1),
  color = "bwr",
  xlab = "",
  ylab = "",
  setmargins = TRUE,
  ...
)
```

Arguments

`model`	A gam object, produced by `gam` or `bam`.
`view`	A two-value vector containing the names of the two main effect terms to be displayed on the x and y dimensions of the plot. Note that variables coerced to factors in the model formula won't work as view variables.
`el.pos`	A list with X and Y positions and Electrodes, which are used for fitting the model.
`fun`	Text string, 'fvisgam', 'pvisgam', or 'plot_diff2' signalling which function to use for plotting.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`size`	Size in inch of plot window.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`col`	The colors for the background of the plot.

`pch`	The type of points as indications for the electrode positions. The value NA will suppress the plotting of electrode positions.
`bg`	The background color of the points.
`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray', 'bw', or 'rwb' (red-white-blue; default).
`xlab`	Label x-axis. Default excluded.
`ylab`	Label y-axis. Default excluded.
`setmargins`	Logical: whether or not to automatically set the margins. By default set to TRUE. If set to false, the size can
`...`	other options to pass on to `fvisgam`, `pvisgam`, or `plot_diff2`.

Notes

X-positions of electrodes should have lower values for electrodes on the left hemisphere (e.g. T7) than for electrodes on the right hemisphere. Y-positions of electrodes should have lower values for electrodes at the back of the head than for the frontal electrodes.

Author(s)

Jacolien van Rij

See Also

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `pvisgam()`

Examples

```
data(eeg)

## Not run:
# simple GAMM model:
m1 <- gam(Ampl ~ te(Time, X, Y, k=c(10,5,5),
    d=c(1,2)), data=eeg)

# topo plot, by default uses fvisgam
# and automatically selects a timestamp (270ms):
plot_topo(m1, view=c('X', 'Y'))
# or:
plot_topo(m1, view=c('X', 'Y'), setmargins=FALSE, size=1)

# add electrodes:
electrodes <- eeg[,c('X','Y','Electrode')]
electrodes <- as.list( electrodes[!duplicated(electrodes),] )
plot_topo(m1, view=c('X', 'Y'), el.pos=electrodes)

# some formatting options:
plot_topo(m1, view=c('X', 'Y'), el.pos=electrodes,
    main='Topo plot', zlim=c(-.5,.5),
    pch=15, col='red', color='terrain')

# plotting more than one panel only works if
# each figure region is a square:
dev.new(width=12, height=4)
par(mfrow=c(1,3))

for(i in c(100, 200, 300)){
    # make sure to keep zlim constant:
\t   plot_topo(m1, view=c('X', 'Y'), zlim=c(-.5, .5),
    cond=list(Time=i), el.pos=electrodes,
    main=i)
}

dev.new(width=12, height=4)
par(mfrow=c(1,3), cex=1.1)
# The three different functions for plotting:
plot_topo(m1, view=c('X', 'Y'), zlim=c(-.5, .5),
    el.pos=electrodes,
    fun='fvisgam', main='fvisgam',
    cond=list(Time=200), rm.ranef=TRUE)
plot_topo(m1, view=c('X', 'Y'), zlim=c(-.5, .5),
    el.pos=electrodes, select=1,
    fun='pvisgam', main='pvisgam',
    cond=list(Time=200))
plot_topo(m1, view=c('X', 'Y'), zlim=c(-.5, .5),
    el.pos=electrodes, comp=list(Time=c(300,100)),
    fun='plot_diff2', main='plot_diff2',
    plotCI=TRUE)

# Add labels:
plot_topo(m1, view=c('X', 'Y'), zlim=c(-.5, .5),
    fun='fvisgam', main='',
    cond=list(Time=200), add.color.legend=FALSE)
text(electrodes[['X']], electrodes[['Y']],
    labels=electrodes[['Electrode']], cex=.75,
    xpd=TRUE)

## End(Not run)
```

print_summary *Print a named list of strings, output from summary_data.*

Description

Print a named list of strings, output from summary_data.

Usage

```
print_summary(sumlist, title = NULL)
```

Arguments

`sumlist`	Named list, output of `summary_data`.
`title`	Optional, text string that will be printed as title.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `refLevels()`, `res_df()`, `summary_data()`, `timeBins()`

`pvisgam` *Visualization of partial nonlinear interactions.*

Description

Produces perspective or contour plot views of gam model predictions of the partial effects interactions. Combines the function `plot.gam` for interaction surfaces with the function `vis.gam`. Similar to `plot.gam`, `pvisgam` plots the partial interaction surface, without including values for other predictors that are not being shown. Similar to `vis.gam` the user can set the two predictors to be viewed, and colors are added behind the contours to facilitate interpretation. In contrast to `plot.gam`, this function allows to plotting of interactions with three of more continuous predictors by breaking it down in two-dimensional surfaces. The code is derivated from the script for `vis.gam`.

Usage

```
pvisgam(
  x,
  view = NULL,
  select = NULL,
  cond = list(),
  n.grid = 30,
  too.far = 0,
  col = NA,
  color = "terrain",
  contour.col = NULL,
  add.color.legend = TRUE,
  se = 0,
  plot.type = "contour",
  zlim = NULL,
  xlim = NULL,
  ylim = NULL,
  nCol = 50,
  labcex = 0.6,
  hide.label = FALSE,
  print.summary = getOption("itsadug_print"),
  show.diff = FALSE,
  col.diff = 1,
  alpha.diff = 0.5,
  dec = NULL,
  f = 1.96,
  ...
)
```

Arguments

`x`	A gam object, produced by `gam` or `bam`.
`view`	A two-value vector containing the names of the two main effect terms to be displayed on the x and y dimensions of the plot. Note that variables coerced to factors in the model formula won't work as view variables.
`select`	A number, selecting a single model term for printing. e.g. if you want the plot for the second smooth term set select=2.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`n.grid`	The number of grid nodes in each direction used for calculating the plotted surface.
`too.far`	Plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded.
`col`	The colors for the facets of the plot.
`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray' or 'bw'. Alternatively a vector with some colors can be provided for a custom color palette (see examples).
`contour.col`	sets the color of contours when using plot.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`se`	If less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
`plot.type`	one of 'contour' or 'persp' (default is 'contour').
`zlim`	A two item array giving the lower and upper limits for the z- axis scale. NULL to choose automatically.
`xlim`	A two item array giving the lower and upper limits for the x- axis scale. NULL to choose automatically.

`ylim`	A two item array giving the lower and upper limits for the y- axis scale. NULL to choose automatically.
`nCol`	The number of colors to use in color schemes.
`labcex`	Size of the contour labels.
`hide.label`	Logical: whether or not to hide the label (i.e., 'partial effect'). Default is FALSE.
`print.summary`	Logical: whether or not to print summary. Default set to the print info messages option (see `infoMessages`).
`show.diff`	Logical: whether or not to indicate the regions that are significantly different from zero. Note that these regions are just an indication and dependent on the value of `n.grid`. Defaults to FALSE.
`col.diff`	Color to shade the nonsignificant areas.
`alpha.diff`	Level of transparency to mark the nonsignificant areas.
`dec`	Numeric: number of decimals for rounding the color legend. When NULL (default), no rounding. If -1 the values are automatically determined. Note: if value = -1 (default), rounding will be applied also when `zlim` is provided.
`f`	Scaling factor to determine the CI from the se, for marking the difference with 0. Only applies when `se` is smaller or equal to zero and `show.diff` is set to TRUE.
`...`	other options to pass on to persp, image or contour. In particular ticktype='detailed' will add proper axes labeling to the plots.

Warnings

- In contrast to vis.gam, do not specify other predictors in `cond` that are not to be plotted.
- When the argument `show.diff` is set to TRUE a shading area indicates where the confidence intervals include zero. Or, in other words, the areas that are not significantly different from zero. Be careful with the interpretation, however, as the precise shape of the surface is dependent on model constraints such as the value of `choose.k` and the smooth function used, and the size of the confidence intervals are dependent on the model fit and model characteristics (see `vignette('acf', package='itsadug')`). In addition, the value of `n.grid` determines the precision of the plot.

Author(s)

Jacolien van Rij. Modification of `vis.gam` from package `mgcv` of Simon N. Wood.

See Also

`vis.gam`, `plot.gam`

Other Functions for model inspection: `dispersion()`, `fvisgam()`, `gamtabs()`, `inspect_random()`, `plot_data()`, `plot_parametric()`, `plot_smooth()`, `plot_topo()`

Examples

```
data(simdat)

## Not run:
# Model with random effect and interactions:
m1 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)

# Plot summed effects:
vis.gam(m1, view=c('Time', 'Trial'), plot.type='contour', color='topo')
# Partial effect of interaction:
pvisgam(m1, view=c('Time', 'Trial'), select=1)
# Same:
plot(m1, select=1, scheme=2)
plot(m1, select=1)
# Alternatives:
pvisgam(m1, view=c('Trial', 'Time'), select=1)
pvisgam(m1, view=c('Trial', 'Time'), select=1, zlim=c(-20,20))
pvisgam(m1, view=c('Trial', 'Time'), select=1, zlim=c(-20,20),
    color='terrain')
pvisgam(m1, view=c('Trial', 'Time'), select=1, zlim=c(-20,20),
    color=c('blue', 'white', 'red'))

# Notes on the color legend:
# Labels can easily fall off the plot, therefore the numbers are
# automatically rounded.
# To undo the rounding, set dec=NULL:
pvisgam(m1, view=c('Time', 'Trial'), dec=NULL)
# For custom rounding, set dec to a value:
pvisgam(m1, view=c('Time', 'Trial'), dec=1)
# To increase the left marging of the plot (so that the numbers fit):
oldmar <- par()$mar
par(mar=oldmar + c(0,0,0,1) ) # add one line to the right
pvisgam(m1, view=c('Time', 'Trial'), dec=3)
par(mar=oldmar) # restore to default settings

# too.far:
n <- which(simdat$Time > 1500 & simdat$Trial > 5)
simdat[n,]$Y <- NA
simdat[simdat$Trial == -3,]$Y <- NA
m1 <- bam(Y ~ te(Time, Trial)+s(Time, Subject, bs='fs', m=1),
    data=simdat, discrete=TRUE)
pvisgam(m1, view=c('Time', 'Trial'), select=1, too.far=0.03)


## End(Not run)
# see the vignette for examples:
vignette('overview', package='itsadug')
```

`refLevels` *Return a list with reference levels for each factor.*

Description

Function for retrieving all reference levels.

Usage

```
refLevels(data)
```

Arguments

`data`	Data

Value

Named list with reference levels for each factor.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `res_df()`, `summary_data()`, `timeBins()`

`report_stats` *Returns a description of the statistics of the smooth terms for reporting.*

Description

Returns a description of the statistics of the smooth terms for reporting.

Usage

```
report_stats(model, summary = NULL, print.summary = getOption("itsadug_print"))
```

Arguments

`model`	A gam or bam object, produced by `gam` or `bam`.
`summary`	Optionally include the summary of the model when available, which may speed up the process for large models.
`print.summary`	Logical: whether or not to print the output. Default set to the print info messages option (see `infoMessages`).

Author(s)

Jacolien van Rij

See Also

Other Testing for significance: compareML(), plot_diff2(), plot_diff(), wald_gam()

Examples

```
data(simdat)

# model without random effects:
m1 <- bam(Y ~ te(Time, Trial), data=simdat)
report_stats(m1)
# save report for later use:
report <- report_stats(m1, print.summary=FALSE)
report[3,2]
```

---

resid_gam *Extract model residuals and remove the autocorrelation accounted for.*

Description

Extract model residuals and remove the autocorrelation accounted for.

Usage

```
resid_gam(model, AR_start = NULL, incl_na = FALSE, return_all = FALSE)
```

Arguments

model	A GAMM model build with gam or bam.
AR_start	Optional: vector with logicals, indicating the start of events. Default is NULL, because generally the function can retrieve all necessary information from the model.
incl_na	Whether or not to include missing values (NA)when returning the residuals. Defaults to FALSE.
return_all	Default is FALSE. Returns a list with normal residuals, corrected residuals, and the value of rho.

Value

Corrected residuals.

Author(s)

Jacolien van Rij

See Also

resid, missing_est

Other functions for model criticism: acf_n_plots(), acf_plot(), acf_resid(), derive_timeseries(), start_event(), start_value_rho()

## Examples

```
data(simdat)

## Not run:
# Add start event column:
simdat <- start_event(simdat, event=c('Subject', 'Trial'))
head(simdat)
# bam model with AR1 model (toy example, not serious model):
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group),
   data=simdat, rho=.5, AR.start=simdat$start.event)
# Standard residuals:
res1 <- resid(m1)
# Corrected residuals:
res2 <- resid_gam(m1)

# Result in different ACF's:
par(mfrow=c(1,2))
acf(res1)
acf(res2)

# Without AR.start included in the model:
m2 <- bam(Y ~ Group + te(Time, Trial, by=Group),
   data=simdat)
acf(resid_gam(m2), plot=F)
# Same as resid(m2)!
acf(resid(m2), plot=F)

### MISSING VALUES ###
# Note that corrected residuals cannot be calculated for the last
# point of each time series. These missing values are by default
# excluded.

# Therefore, this will result in an error...
simdat$res <- resid_gam(m1)
# ... and this will give an error too:
simdat$res <- NA
simdat[!is.na(simdat$Y),] <- resid_gam(m1)
# ... but this works:
simdat$res <- resid_gam(m1, incl_na=TRUE)

# The parameter incl_na will NOT add missing values
# for missing values in the *data*.
# Example:
simdat[sample(nrow(simdat), 15),]$Y <- NA
# Without AR.start included in the model:
m2 <- bam(Y ~ Group + te(Time, Trial, by=Group),
   data=simdat)
# This works:
acf(resid_gam(m2))
# ...but this results in error, although no AR1 model specified:
simdat$res <- resid_gam(m2)
# ... for this type of missing data, this does not solve the problem:
simdat$res <- resid_gam(m2, incl_na=TRUE)
# instead try this:
simdat$res <- NA
simdat[-missing_est(m2),]$res <- resid_gam(m2)

# With AR.start included in the model:
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group),
   data=simdat, rho=.5, AR.start=simdat$start.event)
# This works (incl_na=TRUE):
simdat$res <- NA
simdat[-missing_est(m2),]$res <- resid_gam(m2, incl_na=TRUE)


## End(Not run)
```

res_df   *Retrieve the residual degrees of freedom from the model.*

Description

Retrieve the residual degrees of freedom from the model.

Usage

```
res_df(model)
```

Arguments

`model`	A fitted regression model (using gam, or bam).

Value

Numeric value: residual degrees of freedom from the model.

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `summary_data()`, `timeBins()`

Examples

```
data(simdat)

# Fit simple GAM model:
gam1 <- bam(Y ~ s(Time), data=simdat, discrete=TRUE)
res_df(gam1)
# ... which is the same as:

modeledf(gam1)
```

`rug_model` *Add rug to plot, based on model.*

Description

Add rug based on model data.

Usage

```
rug_model(
  model,
  view,
  cond = NULL,
  data.rows = NULL,
  rm.ranef = NULL,
  print.summary = getOption("itsadug_print"),
  ...
)
```

Arguments

`model`	gam or bam object.
`view`	Text string containing the name of the smooth to be displayed. Note that variables coerced to factors in the model formula won't work as view variables.
`cond`	A named list of the values to use for the other predictor terms (not in view). Used for choosing between smooths that share the same view predictors.
`data.rows`	Vector of numbers (indices of rows in data) or vector of logical vales (same length as rows in data) for selecting specific data points.
`rm.ranef`	Logical: whether or not to remove random effects. Default is TRUE.
`print.summary`	Logical: whether or not to print information messages. Default set to the print info messages option (see `infoMessages`).
`...`	Optional graphical parameters (see `rug`).

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `fadeRug()`

Examples

```
plot(cars$speed, cars$dist, pch=16, col=alpha(1))
lm1 <- lm(dist ~ speed, dat=cars)
abline(lm1, col='red', lwd=2)
rug_model(lm1, view='speed')
rug_model(lm1, view='dist', side=2)

## Not run:
library(itsadug)
data(simdat)
m1 <- bam(Y ~ Group + te(Time, Trial, by=Group), data=simdat)
# plot:
fvisgam(m1, view=c('Time', 'Trial'), cond=list(Group='Adults'))
rug_model(m1, view='Time', cond=list(Group='Adults'))
rug_model(m1, view='Trial', cond=list(Group='Adults'), side=2)

## End(Not run)
```

`simdat` *Simulated time series data.*

Description

A dataset containing the sine wave data with random noise added.

Usage

```
simdat
```

Format

A data frame with 75600 rows and 6 variables:

`Group` Age group of participants: Adults or Children.

`Time` Time, time measure from start of each time series.

`Trial` Trial in the experiment, centered around zero.

`Condition` Continuous variable, ranging from -1 to 4. For example, stimulus onset asynchrony.

`Subject` Code for individual participants.

`Y` Time series measure. Similar to pupil size, sensor position, or voltage.

Author(s)

Jacolien van Rij

`start_event` *Determine the starting point for each time series.*

Description

Determine the starting point for each time series.

Usage

```
start_event(
  data,
  column = "Time",
  event = "Event",
  label = "start.event",
  label.event = NULL,
  order = TRUE,
  newcode = TRUE
)
```

Arguments

`data`	A data frame.
`column`	Test string, name of the column that describes the order withing the time series. Default is 'Time'.
`event`	A text string or vector indicating the columns that define the unique time series. Default is 'Event'.
`label`	The name of the new column with the start point of each time series. Default is 'start.event'.
`label.event`	In case `event` is not a single column, providing a text string will add a column with this name that defines unique time series. Default is NULL (no new column for time series is created).
`order`	Logical: whether or not to order each time series. Default is TRUE, maybe set to FALSE with large data frames that are already ordered.
`newcode`	Logical: whether or not to use the new (and hopefully faster) code.

Value

Data frame.

Note

When working with large data frames, it may be worth installing the package `data.table`. Although not required for the package, the function `start_event` will check if `data.table` is available and will use it's much faster function `rbindlist`. This speeds up the function `start_event`. Run the command `install.packages('data.table', repos='http://cran.us.r-project.org')` in the command window for installing the package `data.table`.

Author(s)

Jacolien van Rij

See Also

Other functions for model criticism: `acf_n_plots()`, `acf_plot()`, `acf_resid()`, `derive_timeseries()`, `resid_gam()`, `start_value_rho()`

Examples

```
data(simdat)
head(simdat)
simdat <- simdat[sample(1:nrow(simdat)),]
simdat$Condition <- relevel(factor(simdat$Condition), ref="0")
contrasts(simdat$Condition) <- "contr.poly"
contrasts(simdat$Condition)
test <- start_event(simdat, event=c('Subject', 'Trial'), label.event='Event')
contrasts(test$Condition)
head(test)
test <- start_event(simdat, event="Subject")
test <- start_event(simdat, event=c('Subject', 'Trial'))
```

`start_value_rho` *Extract the Lag 1 value from the ACF of the residuals of a gam, bam, lm, lmer model, ...*

Description

Wrapper around `acf_plot` for regression models.

Usage

```
start_value_rho(model, plot = FALSE, lag = 2, main = NULL, ...)
```

Arguments

`model`	A regression model generated by `lm`, `glm`, `lmer`, `glmer`, `gam`, or `bam`. (See examples for how to avoid errors due to missing values.)
`plot`	Logical: whether or not to produce plot. Default is TRUE.
`lag`	Numeric value, indicating the lag. Default is 2.
`main`	Text string, title of ACF plot.
`...`	Other arguments for plotting the acf, see `acf`.

Value

The autocorrelation value of data points with the data points at lag `lag`.

Author(s)

Jacolien van Rij

See Also

Use acf for the original ACF function, and acf_plot, or acf_resid.

Other functions for model criticism: acf_n_plots(), acf_plot(), acf_resid(), derive_timeseries(), resid_gam(), start_event()

Examples

```
data(simdat)

# add missing values to simdat:
simdat[sample(nrow(simdat), 15),]$Y <- NA

## Not run:
# Run GAMM model:
m1 <- bam(Y ~ te(Time, Trial)+s(Subject, bs='re'), data=simdat)

# No plotting:
start_value_rho(m1)
# With plot:
rhom1 <- start_value_rho(m1, plot=TRUE)


## End(Not run)
# see the vignette for examples:
vignette('acf', package='itsadug')
```

---

summary_data   *Print a descriptive summary of a data frame.*

Description

The function prints a summary of the data. Similar to the function str, but easier readable.

Usage

```
summary_data(data, print = TRUE, n = 10)
```

Arguments

data	A data frame.
print	Logical: whether or not to print the summary.
n	Number: maximum number of values being mentioned in the summary. If NULL all values are being mentioned. Defaults to 10.

Value

Optionally returns a named list with info.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `timeBins()`

Examples

```
data(simdat)
summary_data(simdat)
```

`timeBins` *Label timestamps as timebins of a given binsize.*

Description

Function for calculating timebins.

Usage

```
timeBins(x, binsize, pos = 0.5)
```

Arguments

`x`	Numerical vector with timestamp information.
`binsize`	Size of the timebin, measured in the same units (often ms) as `x`.
`pos`	Numerical value that determines the label of the binsize as proportion of the binsize. A value of 0 will provide the minimum timestamp within the bin as label, a value of 1 will provide the maximum value within the bin as label. Defaults to 0.5, the center of the bin.

Value

Anumerical vector of the same size as `x` with timebin information.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `convertNonAlphanumeric()`, `corfit()`, `diff_terms()`, `find_difference()`, `missing_est()`, `modeledf()`, `observations()`, `print_summary()`, `refLevels()`, `res_df()`, `summary_data()`

Examples

```
data(simdat)
# grouping Time values in bins:
simdat$Timebin <- timeBins(simdat$Time, 200)
head(simdat)

# different labels:
simdat$Timebin2 <- timeBins(simdat$Time, 200, pos=0)
head(simdat)
```

`wald_gam` *Function for post-hoc comparison of the contrasts in a single GAMM model.*

Description

Function for post-hoc comparison of the intercept differences for different factors in a single GAMM model.

Usage

```
wald_gam(
  model,
  comp = NULL,
  select = NULL,
  t.test = FALSE,
  null.hypothesis = 0,
  summ = NULL,
  signif.stars = TRUE,
  print.output = getOption("itsadug_print")
)
```

Arguments

`model`	Model, currently only implemented for models generated with `bam` or `gam`.
`comp`	Named list with predictors (specified as names) and their levels to compare. Defaults to NULL, which returns all comparisons, unless `select` is specified.
`select`	Contrast matrix for manually specified contrasts. Alternatively, a vector or list could be provided as input. See examples below.
`t.test`	Logical default = FALSE), whether or not to return the t-test scores instead of the Wald test. Only implemented for Gaussian models. This option is not implemented for use with `select`.

null.hypothesis
: Numeric, value of null hypothesis. Defaults to 0 and is generally not changed.

summ
: Optional summary object. Defaults to NULL. For very large GAMM models it takes a long time to retrieve the summary. In these cases the summary could be provided to reduce processing time. However, it is generally recommended not to specifify a summary object, to reduce the chance of mismatch errors.

signif.stars
: Logical (default = TRUE). Whether or not to display stars indicating the level of significance on 95% confidence level.

print.output
: Logical: whether or not to print the output. By default controlled globally in the package options: If the function infoMessages is set to TRUE, the output will be automatically printed. Could be also set by explicitly providing TRUE or FALSE. See examples.

Value

Optionally returns a data frame with test statistics.

Warning

This function is intended for testing intercept differences only. This function compares purely the parametric components, without considering any interactions with smooth terms. So this could be considered as a partial effect comparison. For comparing the averages of conditions use get_difference, which outputs the difference in summed effects for different factor levels.

Author(s)

Petar Milin and Jacolien van Rij.

See Also

plot_parametric, plot_diff, plot_diff2

Other Testing for significance: compareML(), plot_diff2(), plot_diff(), report_stats()

Examples

```
data(simdat)
# Convert Condition to factorial predictor for illustration purposes:
simdat$Condition <- as.factor(simdat$Condition)

infoMessages('on')

## Not run:
# some arbitrary model:
m1 <- bam(Y ~ Condition*Group
\t+ s(Time, by=Condition)
\t+ s(Time, by=Group)
\t+ s(Subject, bs='re'),
\tdata=simdat)

# print summary to inspect parametric terms:
```

```
summary(m1)

# return all contrasts:
wald_gam(m1)

# USE OF COMP
# return only contrasts for Adults:
wald_gam(m1, comp=list(Condition=levels(simdat$Condition)))
# return specific contrasts:
wald_gam(m1, comp=list(Condition=c('-1', '0', '1'),
    Group=c('Adults', 'Children')))

# USE OF SELECT
# Specify contrast matrix.
# Note that intercept should be 0.
# Example: Compare Condition 0 with Conditions 2 and 3 for children.
# Method 1: matrix or vector:
R = matrix( c(0,-2,0,1,1,0,0,0,0,0,0,0), nrow=1)
wald_gam(m1, select=R)
wald_gam(m1, select=c(0,-2,0,1,1,0,0,0,0,0,0,0))
# Method 2: list
# first list element are reference coefficients,
# second list element are coefficients to compare
wald_gam(m1, select=list(2, c(4,5)))
# Replication of contrasts given in summary:
wald_gam(m1, select=c(0,1,0,0,0,0,0,0,0,0,0))

# USE OF T.TEST
# This option is not implemented for use with select
# Compare with second line of parametric summary:
wald_gam(m1, comp=list(Condition=c('-1', '0'),
    Group='Children'), t.test=TRUE)
# Compare with Wald test:
wald_gam(m1, comp=list(Condition=c('-1', '0'),
    Group='Children'))

# exclude significance stars:
wald_gam(m1, comp=list(Condition=c('-1', '0'),
    Group='Children'), signif.stars=FALSE)

# do not print output, but save table for later use:
test <- wald_gam(m1, comp=list(Condition=c('-1', '0'),
    Group='Children'), print.output=FALSE)
test
# alternative way:
infoMessages('off')
test2 <- wald_gam(m1, comp=list(Condition=c('-1', '0'),
    Group='Children'))
infoMessages('on')


## End(Not run)
```

Index