Package 'mvgam'

September 4, 2024

Title Multivariate (Dynamic) Generalized Additive Models

Version 1.1.3

Date 2024-09-03

Description
Fit Bayesian Dynamic Generalized Additive Models to sets of time series. Users can build dynamic nonlinear State-Space models that can incorporate semiparametric effects in observation and process components, using a wide range of observation families. Estimation is performed using Markov Chain Monte Carlo with Hamiltonian Monte Carlo in the software 'Stan'. References: Clark & Wells (2022) <doi:10.1111/2041-210X.13974>.

URL https://github.com/nicholasjclark/mvgam, https://nicholasjclark.github.io/mvgam/

BugReports https://github.com/nicholasjclark/mvgam/issues

License MIT + file LICENSE

Depends R (>= 3.6.0)

Imports brms (>= 2.21.0), methods, mgcv (>= 1.8-13), insight (>= 0.19.1), marginaleffects (>= 0.16.0), Rcpp (>= 0.12.0), rstan (>= 2.29.0), posterior (>= 1.0.0), loo (>= 2.3.1), rstantools (>= 2.1.1), bayesplot (>= 1.5.0), ggplot2 (>= 2.0.0), parallel, pbapply, mvnfast, purrr, zoo, smooth, dplyr, magrittr, Matrix, rlang

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests scoringRules, matrixStats, cmdstanr (>= 0.5.0), tweedie, splines2, extraDistr, wrswoR, xts, lubridate, knitr, collapse, rmarkdown, rjags, coda, runjags, usethis, testthat

Enhances gratia (>= 0.9.0), tibble (>= 3.0.0), tidyr

Additional_repositories https://mc-stan.org/r-packages/

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

NeedsCompilation yes

Author Nicholas J Clark [aut, cre] (<https://orcid.org/0000-0001-7131-3301>)

Maintainer Nicholas J Clark <nicholas.j.clark1214@gmail.com>

Repository CRAN

Date/Publication 2024-09-04 03:40:02 UTC

Contents

`add_residuals.mvgam` *Calculate randomized quantile residuals for mvgam objects*

Description

Calculate randomized quantile residuals for mvgam objects

Usage

```
add_residuals(object, ...)

## S3 method for class 'mvgam'
add_residuals(object, ...)
```

Arguments

`object`	`list` object returned from `mvgam`. See `mvgam()`
`...`	unused

Details

For each series, randomized quantile (i.e. Dunn-Smyth) residuals are calculated for inspecting model diagnostics If the fitted model is appropriate then Dunn-Smyth residuals will be standard normal in distribution and no autocorrelation will be evident. When a particular observation is missing, the residual is calculated by comparing independent draws from the model's posterior distribution

Value

A list object of class `mvgam` with residuals included in the `'resids'` slot

---

`all_neon_tick_data` *NEON Amblyomma and Ixodes tick abundance survey data*

Description

A dataset containing timeseries of Amblyomma americanum and Ixodes scapularis nymph abundances at NEON sites

Usage

```
all_neon_tick_data
```

Format

A tibble/dataframe containing covariate information alongside the main fields of:

Year Year of sampling

epiWeek Epidemiological week of sampling

plot_ID NEON plot ID for survey location

siteID NEON site ID for survey location

amblyomma_americanum Counts of A. americanum nymphs

ixodes_scapularis Counts of I. scapularis nymphs

Source

https://www.neonscience.org/data

code *Stan code and data objects for mvgam models*

Description

Generate Stan code and data objects for **mvgam** models

Usage

```
code(object)

## S3 method for class 'mvgam_prefit'
stancode(object, ...)

## S3 method for class 'mvgam'
stancode(object, ...)

## S3 method for class 'mvgam_prefit'
standata(object, ...)
```

Arguments

`object`	An object of class `mvgam` or `mvgam_prefit`, returned from a call to `mvgam`
`...`	ignored

Value

Either a character string containing the fully commented **Stan** code to fit a **mvgam** model or a named list containing the data objects needed to fit the model in Stan.

Examples

```
simdat <- sim_mvgam()
mod <- mvgam(y ~ s(season) +
               s(time, by = series),
             family = poisson(),
             data = simdat$data_train,
             run_model = FALSE)

# View Stan model code
stancode(mod)

# View Stan model data
sdata <- standata(mod)
str(sdata)
```

conditional_effects.mvgam

Display Conditional Effects of Predictors

Description

Display conditional effects of one or more numeric and/or categorical predictors in mvgam models, including two-way interaction effects.

Usage

```
## S3 method for class 'mvgam'
conditional_effects(
  x,
  effects = NULL,
  type = "response",
  points = TRUE,
  rug = TRUE,
  ...
)

## S3 method for class 'mvgam_conditional_effects'
plot(x, plot = TRUE, ask = FALSE, ...)

## S3 method for class 'mvgam_conditional_effects'
print(x, ...)
```

Arguments

x	Object of class mvgam or mvgam_conditional_effects
effects	An optional character vector naming effects (main effects or interactions) for which to compute conditional plots. Interactions are specified by a : between variable names. If NULL (the default), plots are generated for all main effects and two-way interactions estimated in the model. When specifying effects manually, *all* two-way interactions (including grouping variables) may be plotted even if not originally modeled.
type	character specifying the scale of predictions. When this has the value link (default) the linear predictor is calculated on the link scale. If expected is used, predictions reflect the expectation of the response (the mean) but ignore uncertainty in the observation process. When response is used, the predictions take uncertainty in the observation process into account to return predictions on the outcome scale. Two special cases are also allowed: type latent_N will return the estimated latent abundances from an N-mixture distribution, while type detection will return the estimated detection probability from an N-mixture distribution

points	Logical. Indicates if the original data points should be added, but only if type == 'response'. Default is TRUE.
rug	Logical. Indicates if displays tick marks should be plotted on the axes to mark the distribution of raw data, but only if type == 'response'. Default is TRUE.
...	other arguments to pass to plot_predictions
plot	Logical; indicates if plots should be plotted directly in the active graphic device. Defaults to TRUE.
ask	Logical. Indicates if the user is prompted before a new page is plotted. Only used if plot is TRUE. Default is FALSE.

Details

This function acts as a wrapper to the more flexible plot_predictions. When creating conditional_effects for a particular predictor (or interaction of two predictors), one has to choose the values of all other predictors to condition on. By default, the mean is used for continuous variables and the reference category is used for factors. Use plot_predictions to change these and create more bespoke conditional effects plots.

Value

conditional_effects returns an object of class mvgam_conditional_effects which is a named list with one slot per effect containing a ggplot object, which can be further customized using the **ggplot2** package. The corresponding plot method will draw these plots in the active graphic device

Author(s)

Nicholas J Clark

See Also

plot_predictions, plot_slopes

Examples

```
# Simulate some data
simdat <- sim_mvgam(family = poisson(),
                    seasonality = 'hierarchical')

# Fit a model
mod <- mvgam(y ~ s(season, by = series, k = 5) + year:series,
             family = poisson(),
             data = simdat$data_train,
             chains = 2)

# Plot all main effects on the response scale
conditional_effects(mod)

# Change the prediction interval to 70% using plot_predictions() argument
# 'conf_level'
conditional_effects(mod, conf_level = 0.7)
```

```
# Plot all main effects on the link scale
conditional_effects(mod, type = 'link')

# Works the same for smooth terms, including smooth interactions
set.seed(0)
dat <- mgcv::gamSim(1, n = 200, scale = 2)
mod <- mvgam(y ~ te(x0, x1, k = 5) + s(x2, k = 6) + s(x3, k = 6),
            data = dat,
            family = gaussian(),
            chains = 2)
conditional_effects(mod)
conditional_effects(mod, conf_level = 0.5, type = 'link')

## Not run:
# ggplot objects can be modified and combined with the help of many
# additional packages. Here is an example using the patchwork package

# Simulate some nonlinear data
dat <- mgcv::gamSim(1, n = 200, scale = 2)
mod <- mvgam(y ~ s(x1, bs = 'moi') +
               te(x0, x2),
             data = dat,
             family = gaussian())

# Extract the list of ggplot conditional_effect plots
m <- plot(conditional_effects(mod), plot = FALSE)

# Add custom labels and arrange plots together using patchwork::wrap_plots()
library(patchwork)
library(ggplot2)
wrap_plots(m[[1]] + labs(title = 's(x1, bs = "moi")'),
           m[[2]] + labs(title = 'te(x0, x2)'))

## End(Not run)
```

---

dynamic *Defining dynamic coefficients in mvgam formulae*

Description

Set up time-varying (dynamic) coefficients for use in mvgam models. Currently, only low-rank Gaussian Process smooths are available for estimating the dynamics of the time-varying coefficient.

Usage

```
dynamic(variable, k, rho = 5, stationary = TRUE, scale = TRUE)
```

Arguments

variable	The variable that the dynamic smooth will be a function of
k	Optional number of basis functions for computing approximate GPs. If missing, k will be set as large as possible to accurately estimate the nonlinear function
rho	Either a positive numeric stating the length scale to be used for approximating the squared exponential Gaussian Process smooth (see gp.smooth for details) or missing, in which case the length scale will be estimated by setting up a Hilbert space approximate GP
stationary	Logical. If TRUE (the default) and rho is supplied, the latent Gaussian Process smooth will not have a linear trend component. If FALSE, a linear trend in the covariate is added to the Gaussian Process smooth. Leave at TRUE if you do not believe the coefficient is evolving with much trend, as the linear component of the basis functions can be hard to penalize to zero. This sometimes causes divergence issues in Stan. See gp.smooth for details. Ignored if rho is missing (in which case a Hilbert space approximate GP is used)
scale	Logical; If TRUE (the default) and rho is missing, predictors are scaled so that the maximum Euclidean distance between two points is 1. This often improves sampling speed and convergence. Scaling also affects the estimated length-scale parameters in that they resemble those of scaled predictors (not of the original predictors) if scale is TRUE.

Details

mvgam currently sets up dynamic coefficients as low-rank squared exponential Gaussian Process smooths via the call `s(time, by = variable, bs = "gp", m = c(2, rho, 2))`. These smooths, if specified with reasonable values for the length scale parameter, will give more realistic out of sample forecasts than standard splines such as thin plate or cubic. But the user must set the value for rho, as there is currently no support for estimating this value in mgcv. This may not be too big of a problem, as estimating latent length scales is often difficult anyway. The rho parameter should be thought of as a prior on the smoothness of the latent dynamic coefficient function (where higher values of rho lead to smoother functions with more temporal covariance structure. Values of k are set automatically to ensure enough basis functions are used to approximate the expected wiggliness of the underlying dynamic function (k will increase as rho decreases)

Value

a list object for internal usage in 'mvgam'

Author(s)

Nicholas J Clark

Examples

```
# Simulate a time-varying coefficient
# (as a Gaussian Process with length scale = 10)
set.seed(1111)
N <- 200
```

```
# A function to simulate from a squared exponential Gaussian Process
sim_gp = function(N, c, alpha, rho){
 Sigma <- alpha ^ 2 *
          exp(-0.5 * ((outer(1:N, 1:N, "-") / rho) ^ 2)) +
          diag(1e-9, N)
c + mgcv::rmvn(1,
               mu = rep(0, N),
               V = Sigma)
}

beta <- sim_gp(alpha = 0.75,
               rho = 10,
               c = 0.5,
               N = N)
plot(beta, type = 'l', lwd = 3,
    bty = 'l', xlab = 'Time',
    ylab = 'Coefficient',
    col = 'darkred')

# Simulate the predictor as a standard normal
predictor <- rnorm(N, sd = 1)

# Simulate a Gaussian outcome variable
out <- rnorm(N, mean = 4 + beta * predictor,
             sd = 0.25)
time <- seq_along(predictor)
plot(out,  type = 'l', lwd = 3,
    bty = 'l', xlab = 'Time', ylab = 'Outcome',
    col = 'darkred')

# Gather into a data.frame and fit a dynamic coefficient model
data <- data.frame(out, predictor, time)

# Split into training and testing
data_train <- data[1:190,]
data_test <- data[191:200,]

# Fit a model using the dynamic function
mod <- mvgam(out ~
              # mis-specify the length scale slightly as this
              # won't be known in practice
              dynamic(predictor, rho = 8, stationary = TRUE),
             family = gaussian(),
             data = data_train,
             chains = 2)

# Inspect the summary
summary(mod)

# Plot the time-varying coefficient estimates
plot(mod, type = 'smooths')
```

```
# Extrapolate the coefficient forward in time
plot_mvgam_smooth(mod, smooth = 1, newdata = data)
abline(v = 190, lty = 'dashed', lwd = 2)

# Overlay the true simulated time-varying coefficient
lines(beta, lwd = 2.5, col = 'white')
lines(beta, lwd = 2)
```

`ensemble.mvgam_forecast` *Combine mvgam forecasts into evenly weighted ensembles*

Description

Generate evenly weighted ensemble forecast distributions from `mvgam_forecast` objects

Usage

```
ensemble(object, ...)

## S3 method for class 'mvgam_forecast'
ensemble(object, ..., ndraws = 5000)
```

Arguments

`object`	`list` object of class `mvgam_forecast`. See `forecast.mvgam()`
`...`	More `mvgam_forecast` objects.
`ndraws`	Positive integer specifying the number of draws to use from each forecast distribution for creating the ensemble. If some of the ensemble members have fewer draws than `ndraws`, their forecast distributions will be resampled with replacement to achieve the correct number of draws

Details

It is widely recognised in the forecasting literature that combining forecasts from different models often results in improved forecast accuracy. The simplest way to create an ensemble is to use evenly weighted combinations of forecasts from the different models. This is straightforward to do in a Bayesian setting with `mvgam` as the posterior MCMC draws contained in each `mvgam_forecast` object will already implicitly capture correlations among the temporal posterior predictions.

Value

An object of class `mvgam_forecast` containing the ensemble predictions. This object can be readily used with the supplied S3 functions `plot` and `score`

Author(s)

Nicholas J Clark

See Also

plot.mvgam_forecast, score.mvgam_forecast

Examples

```
# Simulate some series and fit a few competing dynamic models
set.seed(1)
simdat <- sim_mvgam(n_series = 1,
                    prop_trend = 0.6,
                    mu = 1)

plot_mvgam_series(data = simdat$data_train,
                  newdata = simdat$data_test)

m1 <- mvgam(y ~ 1,
            trend_formula = ~ time +
              s(season, bs = 'cc', k = 9),
            trend_model = AR(p = 1),
            noncentred = TRUE,
            data = simdat$data_train,
            newdata = simdat$data_test)

m2 <- mvgam(y ~ time,
            trend_model = RW(),
            noncentred = TRUE,
            data = simdat$data_train,
            newdata = simdat$data_test)

# Calculate forecast distributions for each model
fc1 <- forecast(m1)
fc2 <- forecast(m2)

# Generate the ensemble forecast
ensemble_fc <- ensemble(fc1, fc2)

# Plot forecasts
plot(fc1)
plot(fc2)
plot(ensemble_fc)

# Score forecasts
score(fc1)
score(fc2)
score(ensemble_fc)
```

`evaluate_mvgams` *Evaluate forecasts from fitted mvgam objects*

Description

Evaluate forecasts from fitted mvgam objects

Usage

```
eval_mvgam(
  object,
  n_samples = 5000,
  eval_timepoint = 3,
  fc_horizon = 3,
  n_cores = 2,
  score = "drps",
  log = FALSE,
  weights
)

roll_eval_mvgam(
  object,
  n_evaluations = 5,
  evaluation_seq,
  n_samples = 5000,
  fc_horizon = 3,
  n_cores = 2,
  score = "drps",
  log = FALSE,
  weights
)

compare_mvgams(
  model1,
  model2,
  n_samples = 1000,
  fc_horizon = 3,
  n_evaluations = 10,
  n_cores = 2,
  score = "drps",
  log = FALSE,
  weights
)
```

Arguments

`object`	`list` object returned from `mvgam`

n_samples	integer specifying the number of samples to generate from the model's posterior distribution
eval_timepoint	integer indexing the timepoint that represents our last 'observed' set of outcome data
fc_horizon	integer specifying the length of the forecast horizon for evaluating forecasts
n_cores	integer specifying number of cores for generating particle forecasts in parallel
score	character specifying the type of ranked probability score to use for evaluation. Options are: variogram, drps or crps
log	logical. Should the forecasts and truths be logged prior to scoring? This is often appropriate for comparing performance of models when series vary in their observation ranges
weights	optional vector of weights (where length(weights) == n_series) for weighting pairwise correlations when evaluating the variogram score for multivariate forecasts. Useful for down-weighting series that have larger magnitude observations or that are of less interest when forecasting. Ignored if score != 'variogram'
n_evaluations	integer specifying the total number of evaluations to perform
evaluation_seq	Optional integer sequence specifying the exact set of timepoints for evaluating the model's forecasts. This sequence cannot have values <3 or > max(training timepoints) - fc_horizon
model1	list object returned from mvgam representing the first model to be evaluated
model2	list object returned from mvgam representing the second model to be evaluated

Details

eval_mvgam may be useful when both repeated fitting of a model using update.mvgam for exact leave-future-out cross-validation and approximate leave-future-out cross-validation using lfo_cv are impractical. The function generates a set of samples representing fixed parameters estimated from the full mvgam model and latent trend states at a given point in time. The trends are rolled forward a total of fc_horizon timesteps according to their estimated state space dynamics to generate an 'out-of-sample' forecast that is evaluated against the true observations in the horizon window. This function therefore simulates a situation where the model's parameters had already been estimated but we have only observed data up to the evaluation timepoint and would like to generate forecasts from the latent trends that have been observed up to that timepoint. Evaluation involves calculating an appropriate Rank Probability Score and a binary indicator for whether or not the true value lies within the forecast's 90% prediction interval

roll_eval_mvgam sets up a sequence of evaluation timepoints along a rolling window and iteratively calls eval_mvgam to evaluate 'out-of-sample' forecasts. Evaluation involves calculating the Rank Probability Scores and a binary indicator for whether or not the true value lies within the forecast's 90% prediction interval

compare_mvgams automates the evaluation to compare two fitted models using rolling window forecast evaluation and provides a series of summary plots to facilitate model selection. It is essentially a wrapper for roll_eval_mvgam

Value

For `eval_mvgam`, a `list` object containing information on specific evaluations for each series (if using `drps` or `crps` as the score) or a vector of scores when using `variogram`.

For `roll_eval_mvgam`, a `list` object containing information on specific evaluations for each series as well as a total evaluation summary (taken by summing the forecast score for each series at each evaluation and averaging the coverages at each evaluation)

For `compare_mvgams`, a series of plots comparing forecast Rank Probability Scores for each competing model. A lower score is preferred. Note however that it is possible to select a model that ultimately would perform poorly in true out-of-sample forecasting. For example if a wiggly smooth function of 'year' is included in the model then this function will be learned prior to evaluating rolling window forecasts, and the model could generate very tight predictions as a result. But when forecasting ahead to timepoints that the model has not seen (i.e. next year), the smooth function will end up extrapolating, sometimes in very strange and unexpected ways. It is therefore recommended to only use smooth functions for covariates that are adequately measured in the data (i.e. 'seasonality', for example) to reduce possible extrapolation of smooths and let the latent trends in the `mvgam` model capture any temporal dependencies in the data. These trends are time series models and so will provide much more stable forecasts

See Also

`forecast`, `score`, `lfo_cv`

Examples

```
## Not run:
# Simulate from a Poisson-AR2 model with a seasonal smooth
set.seed(100)
dat <- sim_mvgam(T = 75,
                 n_series = 1,
                 prop_trend = 0.75,
                 trend_model = 'AR2',
                 family = poisson())


# Fit an appropriate model
mod_ar2 <- mvgam(y ~ s(season, bs = 'cc'),
                 trend_model = AR(p = 2),
                 family = poisson(),
                 data = dat$data_train,
                 newdata = dat$data_test,
                 chains = 2)

# Fit a less appropriate model
mod_rw <- mvgam(y ~ s(season, bs = 'cc'),
                trend_model = RW(),
                family = poisson(),
                data = dat$data_train,
                newdata = dat$data_test,
                chains = 2)
```

```
# Compare Discrete Ranked Probability Scores for the testing period
fc_ar2 <- forecast(mod_ar2)
fc_rw <- forecast(mod_rw)
score_ar2 <- score(fc_ar2, score = 'drps')
score_rw <- score(fc_rw, score = 'drps')
sum(score_ar2$series_1$score)
sum(score_rw$series_1$score)

# Use rolling evaluation for approximate comparisons of 3-step ahead
# forecasts across the training period
compare_mvgams(mod_ar2,
               mod_rw,
               fc_horizon = 3,
               n_samples = 1000,
               n_evaluations = 5)

# Now use approximate leave-future-out CV to compare
# rolling forecasts; start at time point 40 to reduce
# computational time and to ensure enough data is available
# for estimating model parameters
lfo_ar2 <- lfo_cv(mod_ar2,
                  min_t = 40,
                  fc_horizon = 3)
lfo_rw <- lfo_cv(mod_rw,
                 min_t = 40,
                 fc_horizon = 3)

# Plot Pareto-K values and ELPD estimates
plot(lfo_ar2)
plot(lfo_rw)

# Proportion of timepoints in which AR2 model gives
# better forecasts
length(which((lfo_ar2$elpds - lfo_rw$elpds) > 0)) /
      length(lfo_ar2$elpds)

# A higher total ELPD is preferred
lfo_ar2$sum_ELPD
lfo_rw$sum_ELPD

## End(Not run)
```

`fitted.mvgam` *Expected Values of the Posterior Predictive Distribution*

Description

This method extracts posterior estimates of the fitted values (i.e. the actual predictions, included estimates for any trend states, that were obtained when fitting the model). It also includes an option for obtaining summaries of the computed draws.

Usage

```
## S3 method for class 'mvgam'
fitted(
  object,
  process_error = TRUE,
  scale = c("response", "linear"),
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  ...
)
```

Arguments

object	An object of class mvgam
process_error	Logical. If TRUE and a dynamic trend model was fit, expected uncertainty in the process model is accounted for by using draws from the latent trend SD parameters. If FALSE, uncertainty in the latent trend component is ignored when calculating predictions
scale	Either "response" or "linear". If "response", results are returned on the scale of the response variable. If "linear", results are returned on the scale of the linear predictor term, that is without applying the inverse link function or other transformations.
summary	Should summary statistics be returned instead of the raw values? Default is TRUE..
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Details

This method gives the actual fitted values from the model (i.e. what you will see if you generate hindcasts from the fitted model using hindcast.mvgam with type = 'expected'). These predictions can be overly precise if a flexible dynamic trend component was included in the model. This is in contrast to the set of predict functions (i.e. posterior_epred.mvgam or predict.mvgam), which will assume any dynamic trend component has reached stationarity when returning hypothetical predictions

Value

An array of predicted *mean* response values. If summary = FALSE the output resembles those of posterior_epred.mvgam and predict.mvgam.

If summary = TRUE the output is an n_observations x E matrix. The number of summary statistics E is equal to 2 + length(probs): The Estimate column contains point estimates (either mean or median depending on argument robust), while the Est.Error column contains uncertainty estimates (either standard deviation or median absolute deviation depending on argument robust). The remaining columns starting with Q contain quantile estimates as specified via argument probs.

See Also

hindcast.mvgam

Examples

```
## Not run:
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = 'AR1',
             data = simdat$data_train,
             chains = 2,
             burnin = 300,
             samples = 300)

# Extract fitted values (posterior expectations)
expectations <- fitted(mod)
str(expectations)

## End(Not run)
```

---

forecast.mvgam *Extract or compute hindcasts and forecasts for a fitted* mvgam *object*

Description

Extract or compute hindcasts and forecasts for a fitted mvgam object

Usage

```
forecast(object, ...)

## S3 method for class 'mvgam'
forecast(object, newdata, data_test, n_cores = 1, type = "response", ...)
```

Arguments

object	list object returned from mvgam. See mvgam()
...	Ignored

newdata	Optional dataframe or list of test data containing at least 'series' and 'time' in addition to any other variables included in the linear predictor of the original formula. If included, the covariate information in newdata will be used to generate forecasts from the fitted model equations. If this same newdata was originally included in the call to mvgam, then forecasts have already been produced by the generative model and these will simply be extracted and plotted. However if no newdata was supplied to the original model call, an assumption is made that the newdata supplied here comes sequentially after the data supplied in the original model (i.e. we assume there is no time gap between the last observation of series 1 in the original data and the first observation for series 1 in newdata)
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
n_cores	integer specifying number of cores for generating forecasts in parallel
type	When this has the value link (default) the linear predictor is calculated on the link scale. If expected is used, predictions reflect the expectation of the response (the mean) but ignore uncertainty in the observation process. When response is used, the predictions take uncertainty in the observation process into account to return predictions on the outcome scale. When variance is used, the variance of the response with respect to the mean (mean-variance relationship) is returned. When type = "terms", each component of the linear predictor is returned separately in the form of a list (possibly with standard errors, if summary = TRUE): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. Two special cases are also allowed: type latent_N will return the estimated latent abundances from an N-mixture distribution, while type detection will return the estimated detection probability from an N-mixture distribution

Details

Posterior predictions are drawn from the fitted mvgam and used to simulate a forecast distribution

Value

An object of class mvgam_forecast containing hindcast and forecast distributions. See mvgam_forecast-class for details.

See Also

hindcast, score, ensemble

Examples

```
simdat <- sim_mvgam(n_series = 3, trend_model = AR())
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)
```

```
# Hindcasts on response scale
hc <- hindcast(mod)
str(hc)
plot(hc, series = 1)
plot(hc, series = 2)
plot(hc, series = 3)

# Forecasts on response scale
fc <- forecast(mod, newdata = simdat$data_test)
str(fc)
plot(fc, series = 1)
plot(fc, series = 2)
plot(fc, series = 3)

# Forecasts as expectations
fc <- forecast(mod, newdata = simdat$data_test, type = 'expected')
plot(fc, series = 1)
plot(fc, series = 2)
plot(fc, series = 3)

# Dynamic trend extrapolations
fc <- forecast(mod, newdata = simdat$data_test, type = 'trend')
plot(fc, series = 1)
plot(fc, series = 2)
plot(fc, series = 3)
```

`formula.mvgam` *Extract formulae from mvgam objects*

Description

Extract formulae from mvgam objects

Usage

```
## S3 method for class 'mvgam'
formula(x, trend_effects = FALSE, ...)

## S3 method for class 'mvgam_prefit'
formula(x, trend_effects = FALSE, ...)
```

Arguments

`x`	`mvgam` or `mvgam_prefit` object
`trend_effects`	`logical`, return the formula from the observation model (if `FALSE`) or from the underlying process model (if`TRUE`)
`...`	Ignored

Value

A formula object

Author(s)

Nicholas J Clark

---

get_mvgam_priors *Extract information on default prior distributions for an mvgam model*

Description

This function lists the parameters that can have their prior distributions changed for a given mvgam model, as well listing their default distributions

Usage

```
get_mvgam_priors(
  formula,
  trend_formula,
  data,
  data_train,
  family = "poisson",
  knots,
  trend_knots,
  use_lv = FALSE,
  n_lv,
  use_stan = TRUE,
  trend_model = "None",
  trend_map,
  drift = FALSE
)
```

Arguments

formula	A character string specifying the GAM observation model formula. These are exactly like the formula for a GLM except that smooth terms, s(), te(), ti(), t2(), as well as time-varying dynamic() terms, can be added to the right hand side to specify that the linear predictor depends on smooth functions of predictors (or linear functionals of these). In nmix() family models, the formula is used to set up a linear predictor for the detection probability. Details of the formula syntax used by **mvgam** can be found in mvgam_formulae
trend_formula	An optional character string specifying the GAM process model formula. If supplied, a linear predictor will be modelled for the latent trends to capture process model evolution separately from the observation model. Should not have a response variable specified on the left-hand side of the formula (i.e. a valid

option would be ~ season + s(year)). Also note that you should not use the identifier series in this formula to specify effects that vary across time series. Instead you should use trend. This will ensure that models in which a trend_map is supplied will still work consistently (i.e. by allowing effects to vary across process models, even when some time series share the same underlying process model). This feature is only currently available for RW(), AR() and VAR() trend models. In nmix() family models, the trend_formula is used to set up a linear predictor for the underlying latent abundance. Be aware that it can be very challenging to simultaneously estimate intercept parameters for both the observation mode (captured by formula) and the process model (captured by trend_formula). Users are recommended to drop one of these using the - 1 convention in the formula right hand side.

data — A dataframe or list containing the model response variable and covariates required by the GAM formula and optional trend_formula. Should include columns: #'

- series (a factor index of the series IDs; the number of levels should be identical to the number of unique series labels (i.e. n_series = length(levels(data$series))))
- time (numeric or integer index of the time point for each observation). For most dynamic trend types available in mvgam (see argument trend_model), time should be measured in discrete, regularly spaced intervals (i.e. c(1, 2, 3, ...)). However you can use irregularly spaced intervals if using trend_model = CAR(1), though note that any temporal intervals that are exactly 0 will be adjusted to a very small number (1e-12) to prevent sampling errors. See an example of CAR() trends in CAR

Should also include any other variables to be included in the linear predictor of formula

data_train — Deprecated. Still works in place of data but users are recommended to use data instead for more seamless integration into R workflows

family — family specifying the exponential observation family for the series. Currently supported families are:

- gaussian() for real-valued data
- betar() for proportional data on (0,1)
- lognormal() for non-negative real-valued data
- student_t() for real-valued data
- Gamma() for non-negative real-valued data
- bernoulli() for binary data
- poisson() for count data
- nb() for overdispersed count data
- binomial() for count data with imperfect detection when the number of trials is known; note that the cbind() function must be used to bind the discrete observations and the discrete number of trials
- beta_binomial() as for binomial() but allows for overdispersion
- nmix() for count data with imperfect detection when the number of trials is unknown and should be modeled via a State-Space N-Mixture model. The latent states are Poisson, capturing the 'true' latent abundance, while

the observation process is Binomial to account for imperfect detection. See mvgam_families for an example of how to use this family

Note that only nb() and poisson() are available if using JAGS as the backend. Default is poisson(). See mvgam_families for more details

knots An optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the k value supplied (note that the number of knots is not always just k). Different terms can use different numbers of knots, unless they share a covariate

trend_knots As for knots above, this is an optional list of knot values for smooth functions within the trend_formula

use_lv logical. If TRUE, use dynamic factors to estimate series' latent trends in a reduced dimension format. Only available for RW(), AR() and GP() trend models. Defaults to FALSE

n_lv integer the number of latent dynamic factors to use if use_lv == TRUE. Cannot be > n_series. Defaults arbitrarily to min(2, floor(n_series / 2))

use_stan Logical. If TRUE, the model will be compiled and sampled using Hamiltonian Monte Carlo with a call to cmdstan_model or a call to stan. Note that there are many more options when using Stan vs JAGS

trend_model character or function specifying the time series dynamics for the latent trend. Options are:

- None (no latent trend component; i.e. the GAM component is all that contributes to the linear predictor, and the observation process is the only source of error; similarly to what is estimated by gam)
- 'RW' or RW()
- 'AR1' or AR(p = 1)
- 'AR2' or AR(p = 2)
- 'AR3' or AR(p = 3)
- 'CAR1' or CAR(p = 1)
- 'VAR1' or VAR()(only available in Stan)
- 'PWlogistic', 'PWlinear' or PW() (only available in Stan)
- 'GP' or GP() (Gaussian Process with squared exponential kernel; only available in Stan)

For all trend types apart from GP(), CAR() and PW(), moving average and/or correlated process error terms can also be estimated (for example, RW(cor = TRUE) will set up a multivariate Random Walk if n_series > 1). See mvgam_trends for more details

trend_map Optional data.frame specifying which series should depend on which latent trends. Useful for allowing multiple series to depend on the same latent trend process, but with different observation processes. If supplied, a latent factor model is set up by setting use_lv = TRUE and using the mapping to set up the shared trends. Needs to have column names series and trend, with integer values in the trend column to state which trend each series should depend on. The series column should have a single unique entry for each series in the data

(names should perfectly match factor levels of the `series` variable in `data`). Note that if this is supplied, the intercept parameter in the process model will NOT be automatically suppressed. See examples for details

`drift` Deprecated. If you wish to estimate drift parameters, include parametric fixed effects of 'time' in your formulae instead.

Details

Users can supply a model formula, prior to fitting the model, so that default priors can be inspected and altered. To make alterations, change the contents of the `prior` column and supplying this `data.frame` to the `mvgam` function using the argument `priors`. If using `Stan` as the backend, users can also modify the parameter bounds by modifying the `new_lowerbound` and/or `new_upperbound` columns. This will be necessary if using restrictive distributions on some parameters, such as a Beta distribution for the trend sd parameters for example (Beta only has support on `(0,1)`), so the upperbound cannot be above `1`. Another option is to make use of the prior modification functions in `brms` (i.e. `prior`) to change prior distributions and bounds (just use the name of the parameter that you'd like to change as the `class` argument; see examples below)

Value

either a `data.frame` containing the prior definitions (if any suitable priors can be altered by the user) or `NULL`, indicating that no priors in the model can be modified through the `mvgam` interface

Note

Only the `prior`, `new_lowerbound` and/or `new_upperbound` columns of the output should be altered when defining the user-defined priors for the `mvgam` model. Use only if you are familiar with the underlying probabilistic programming language. There are no sanity checks done to ensure that the code is legal (i.e. to check that lower bounds are smaller than upper bounds, for example)

Author(s)

Nicholas J Clark

See Also

`mvgam`, `mvgam_formulae`, `prior`

Examples

```
# Simulate three integer-valued time series
library(mvgam)
dat <- sim_mvgam(trend_rel = 0.5)

# Get a model file that uses default mvgam priors for inspection (not always necessary,
# but this can be useful for testing whether your updated priors are written correctly)
mod_default <- mvgam(y ~ s(series, bs = 're') +
              s(season, bs = 'cc') - 1,
              family = nb(),
              data = dat$data_train,
```

```
               trend_model = AR(p = 2),
               run_model = FALSE)

# Inspect the model file with default mvgam priors
code(mod_default)

# Look at which priors can be updated in mvgam
test_priors <- get_mvgam_priors(y ~ s(series, bs = 're') +
                                s(season, bs = 'cc') - 1,
                                family = nb(),
                                data = dat$data_train,
                                trend_model = AR(p = 2))
test_priors

# Make a few changes; first, change the population mean for the series-level
# random intercepts
test_priors$prior[2] <- 'mu_raw ~ normal(0.2, 0.5);'

# Now use stronger regularisation for the series-level AR2 coefficients
test_priors$prior[5] <- 'ar2 ~ normal(0, 0.25);'

# Check that the changes are made to the model file without any warnings by
# setting 'run_model = FALSE'
mod <- mvgam(y ~ s(series, bs = 're') +
             s(season, bs = 'cc') - 1,
             family = nb(),
             data = dat$data_train,
             trend_model = AR(p = 2),
             priors = test_priors,
             run_model = FALSE)
code(mod)

# No warnings, the model is ready for fitting now in the usual way with the addition
# of the 'priors' argument

# The same can be done using 'brms' functions; here we will also change the ar1 prior
# and put some bounds on the ar coefficients to enforce stationarity; we set the
# prior using the 'class' argument in all brms prior functions
brmsprior <- c(prior(normal(0.2, 0.5), class = mu_raw),
               prior(normal(0, 0.25), class = ar1, lb = -1, ub = 1),
               prior(normal(0, 0.25), class = ar2, lb = -1, ub = 1))
brmsprior

mod <- mvgam(y ~ s(series, bs = 're') +
             s(season, bs = 'cc') - 1,
           family = nb(),
           data = dat$data_train,
           trend_model = AR(p = 2),
           priors = brmsprior,
           run_model = FALSE)
code(mod)

# Look at what is returned when an incorrect spelling is used
```

```
test_priors$prior[5] <- 'ar2_bananas ~ normal(0, 0.25);'
mod <- mvgam(y ~ s(series, bs = 're') +
             s(season, bs = 'cc') - 1,
             family = nb(),
             data = dat$data_train,
             trend_model = AR(p = 2),
             priors = test_priors,
             run_model = FALSE)
code(mod)

# Example of changing parametric (fixed effect) priors
simdat <- sim_mvgam()

# Add a fake covariate
simdat$data_train$cov <- rnorm(NROW(simdat$data_train))

priors <- get_mvgam_priors(y ~ cov + s(season),
                           data = simdat$data_train,
                           family = poisson(),
                           trend_model = AR())

# Change priors for the intercept and fake covariate effects
priors$prior[1] <- '(Intercept) ~ normal(0, 1);'
priors$prior[2] <- 'cov ~ normal(0, 0.1);'

mod2 <- mvgam(y ~ cov + s(season),
              data = simdat$data_train,
              trend_model = AR(),
              family = poisson(),
              priors = priors,
              run_model = FALSE)
code(mod2)

# Likewise using 'brms' utilities (note that you can use
# Intercept rather than `(Intercept)`) to change priors on the intercept
brmsprior <- c(prior(normal(0.2, 0.5), class = cov),
               prior(normal(0, 0.25), class = Intercept))
brmsprior

mod2 <- mvgam(y ~ cov + s(season),
              data = simdat$data_train,
              trend_model = AR(),
              family = poisson(),
              priors = brmsprior,
              run_model = FALSE)
code(mod2)

# The "class = 'b'" shortcut can be used to put the same prior on all
# 'fixed' effect coefficients (apart from any intercepts)
set.seed(0)
dat <- mgcv::gamSim(1, n = 200, scale = 2)
dat$time <- 1:NROW(dat)
mod <- mvgam(y ~ x0 + x1 + s(x2) + s(x3),
```

```
            priors = prior(normal(0, 0.75), class = 'b'),
            data = dat,
            family = gaussian(),
            run_model = FALSE)
code(mod)
```

GP *Specify dynamic Gaussian processes*

Description

Set up low-rank approximate Gaussian Process trend models using Hilbert basis expansions in `mvgam`. This function does not evaluate its arguments – it exists purely to help set up a model with particular GP trend models.

Usage

```
GP(...)
```

Arguments

`...`	unused

Details

A GP trend is estimated for each series using Hilbert space approximate Gaussian Processes. In `mvgam`, latent squared exponential GP trends are approximated using by default `20` basis functions and using a multiplicative factor of `c = 5/4`, which saves computational costs compared to fitting full GPs while adequately estimating GP `alpha` and `rho` parameters.

Value

An object of class `mvgam_trend`, which contains a list of arguments to be interpreted by the parsing functions in `mvgam`

See Also

gp

`gratia_mvgam_enhancements` *Enhance mvgam post-processing using gratia functionality*

Description

These evaluation and plotting functions exist to allow some popular `gratia` methods to work with `mvgam` models

Usage

```
drawDotmvgam(
  object,
  trend_effects = FALSE,
  data = NULL,
  select = NULL,
  parametric = FALSE,
  terms = NULL,
  residuals = FALSE,
  scales = c("free", "fixed"),
  ci_level = 0.95,
  n = 100,
  n_3d = 16,
  n_4d = 4,
  unconditional = FALSE,
  overall_uncertainty = TRUE,
  constant = NULL,
  fun = NULL,
  dist = 0.1,
  rug = TRUE,
  contour = TRUE,
  grouped_by = FALSE,
  ci_alpha = 0.2,
  ci_col = "black",
  smooth_col = "black",
  resid_col = "steelblue3",
  contour_col = "black",
  n_contour = NULL,
  partial_match = FALSE,
  discrete_colour = NULL,
  discrete_fill = NULL,
  continuous_colour = NULL,
  continuous_fill = NULL,
  position = "identity",
  angle = NULL,
  ncol = NULL,
  nrow = NULL,
```

```
  guides = "keep",
  widths = NULL,
  heights = NULL,
  crs = NULL,
  default_crs = NULL,
  lims_method = "cross",
  wrap = TRUE,
  envir = environment(formula(object)),
  ...
)

eval_smoothDothilbertDotsmooth(
  smooth,
  model,
  n = 100,
  n_3d = NULL,
  n_4d = NULL,
  data = NULL,
  unconditional = FALSE,
  overall_uncertainty = TRUE,
  dist = NULL,
  ...
)

eval_smoothDotmodDotsmooth(
  smooth,
  model,
  n = 100,
  n_3d = NULL,
  n_4d = NULL,
  data = NULL,
  unconditional = FALSE,
  overall_uncertainty = TRUE,
  dist = NULL,
  ...
)

eval_smoothDotmoiDotsmooth(
  smooth,
  model,
  n = 100,
  n_3d = NULL,
  n_4d = NULL,
  data = NULL,
  unconditional = FALSE,
  overall_uncertainty = TRUE,
  dist = NULL,
  ...
```

)

Arguments

object	a fitted mvgam, the result of a call to mvgam().
trend_effects	logical specifying whether smooth terms from the trend_formula should be drawn. If FALSE, only terms from the observation formula are drawn. If TRUE, only terms from the trend_formula are drawn.
data	a data frame of covariate values at which to evaluate the model's smooth functions.
select	character, logical, or numeric; which smooths to plot. If NULL, the default, then all model smooths are drawn. Character select matches the labels for smooths as shown for example in the output from summary(object). Logical select operates as per numeric select in the order that smooths are stored.
parametric	logical; plot parametric terms also? Note that select is used for selecting which smooths to plot. The terms argument is used to select which parametric effects are plotted. The default, as with mgcv::plot.gam(), is to not draw parametric effects.
terms	character; which model parametric terms should be drawn? The Default of NULL will plot all parametric terms that can be drawn.
residuals	currently ignored for mvgam models.
scales	character; should all univariate smooths be plotted with the same y-axis scale? If scales = "free", the default, each univariate smooth has its own y-axis scale. If scales = "fixed", a common y axis scale is used for all univariate smooths. Currently does not affect the y-axis scale of plots of the parametric terms.
ci_level	numeric between 0 and 1; the coverage of credible interval.
n	numeric; the number of points over the range of the covariate at which to evaluate the smooth.
n_3d, n_4d	numeric; the number of points over the range of last covariate in a 3D or 4D smooth. The default is NULL which achieves the standard behaviour of using n points over the range of all covariate, resulting in n^d evaluation points, where d is the dimension of the smooth. For d > 2 this can result in very many evaluation points and slow performance. For smooths of d > 4, the value of n_4d will be used for all dimensions > 4, unless this is NULL, in which case the default behaviour (using n for all dimensions) will be observed.
unconditional	ignored for mvgam models as all appropriate uncertainties are already included in the posterior estimates.
overall_uncertainty	ignored for mvgam models as all appropriate uncertainties are already included in the posterior estimates.
constant	numeric; a constant to add to the estimated values of the smooth. constant, if supplied, will be added to the estimated value before the confidence band is computed.

fun	function; a function that will be applied to the estimated values and confidence interval before plotting. Can be a function or the name of a function. Function fun will be applied after adding any constant, if provided.
dist	numeric; if greater than 0, this is used to determine when a location is too far from data to be plotted when plotting 2-D smooths. The data are scaled into the unit square before deciding what to exclude, and dist is a distance within the unit square. See mgcv::exclude.too.far() for further details.
rug	logical; draw a rug plot at the bottom of each plot for 1-D smooths or plot locations of data for higher dimensions.
contour	logical; should contours be draw on the plot using ggplot2::geom_contour().
grouped_by	logical; should factor by smooths be drawn as one panel per level of the factor (FALSE, the default), or should the individual smooths be combined into a single panel containing all levels (TRUE)?
ci_alpha	numeric; alpha transparency for confidence or simultaneous interval.
ci_col	colour specification for the confidence/credible intervals band. Affects the fill of the interval.
smooth_col	colour specification for the smooth line.
resid_col	colour specification for residual points. Ignored.
contour_col	colour specification for contour lines.
n_contour	numeric; the number of contour bins. Will result in n_contour - 1 contour lines being drawn. See ggplot2::geom_contour().
partial_match	logical; should smooths be selected by partial matches with select? If TRUE, select can only be a single string to match against.
discrete_colour	a suitable colour scale to be used when plotting discrete variables.
discrete_fill	a suitable fill scale to be used when plotting discrete variables.
continuous_colour	a suitable colour scale to be used when plotting continuous variables.
continuous_fill	a suitable fill scale to be used when plotting continuous variables.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
angle	numeric; the angle at which the x axis tick labels are to be drawn passed to the angle argument of ggplot2::guide_axis().
ncol, nrow	numeric; the numbers of rows and columns over which to spread the plots
guides	character; one of "keep" (the default), "collect", or "auto". Passed to patchwork::plot_layout()
widths, heights	The relative widths and heights of each column and row in the grid. Will get repeated to match the dimensions of the grid. If there is more than 1 plot and widths = NULL, the value of widths will be set internally to widths = 1 to accommodate plots of smooths that use a fixed aspect ratio.
crs	the coordinate reference system (CRS) to use for the plot. All data will be projected into this CRS. See ggplot2::coord_sf() for details.

default_crs	the coordinate reference system (CRS) to use for the non-sf layers in the plot. If left at the default NULL, the CRS used is 4326 (WGS84), which is appropriate for spline-on-the-sphere smooths, which are parameterized in terms of latitude and longitude as coordinates. See ggplot2::coord_sf() for more details.
lims_method	character; affects how the axis limits are determined. See ggplot2::coord_sf(). Be careful; in testing of some examples, changing this to "orthogonal" for example with the chlorophyll-a example from Simon Wood's GAM book quickly used up all the RAM in my test system and the OS killed R. This could be incorrect usage on my part; right now the grid of points at which SOS smooths are evaluated (if not supplied by the user) can produce invalid coordinates for the corners of tiles as the grid is generated for tile centres without respect to the spacing of those tiles.
wrap	logical; wrap plots as a patchwork? If FALSE, a list of ggplot objects is returned, 1 per term plotted.
envir	an environment to look up the data within.
...	additional arguments passed to other methods.
smooth	a smooth object of class "gp.smooth" (returned from a model using either the dynamic() function or the gp() function) or of class "moi.smooth" or "mod.smooth" (returned from a model using the 'moi' or 'mod' basis).
model	a fitted mgcv model of clas gam or bam.

Details

These methods allow mvgam models to be *Enhanced* if users have the gratia package installed, making available the popular draw() function to plot partial effects of mvgam smooth functions using ggplot2::ggplot() utilities

Author(s)

Nicholas J Clark

Examples

```
## Not run:
# Fit a simple GAM and draw partial effects of smooths using gratia
set.seed(0)
library(ggplot2); theme_set(theme_bw())
library(gratia)
dat <- mgcv::gamSim(1, n = 200, scale = 2)
mod <- mvgam(y ~ s(x1, bs = 'moi') +
              te(x0, x2), data = dat,
             family = gaussian())

draw(mod)

## End(Not run)
```

hindcast.mvgam *Extract hindcasts for a fitted* mvgam *object*

Description

Extract hindcasts for a fitted mvgam object

Usage

```
hindcast(object, ...)

## S3 method for class 'mvgam'
hindcast(object, type = "response", ...)
```

Arguments

object	list object returned from mvgam. See mvgam()
...	Ignored
type	When this has the value link (default) the linear predictor is calculated on the link scale. If expected is used, predictions reflect the expectation of the response (the mean) but ignore uncertainty in the observation process. When response is used, the predictions take uncertainty in the observation process into account to return predictions on the outcome scale. When variance is used, the variance of the response with respect to the mean (mean-variance relationship) is returned. When type = "terms", each component of the linear predictor is returned separately in the form of a list (possibly with standard errors, if summary = TRUE): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. Two special cases are also allowed: type latent_N will return the estimated latent abundances from an N-mixture distribution, while type detection will return the estimated detection probability from an N-mixture distribution

Details

Posterior retrodictions are drawn from the fitted mvgam and organized into a convenient format

Value

An object of class mvgam_forecast containing hindcast distributions. See mvgam_forecast-class for details.

See Also

forecast.mvgam

Examples

```
simdat <- sim_mvgam(n_series = 3, trend_model = AR())
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)

# Hindcasts on response scale
hc <- hindcast(mod)
str(hc)
plot(hc, series = 1)
plot(hc, series = 2)
plot(hc, series = 3)

# Hindcasts as expectations
hc <- hindcast(mod, type = 'expected')
str(hc)
plot(hc, series = 1)
plot(hc, series = 2)
plot(hc, series = 3)

# Estimated latent trends
hc <- hindcast(mod, type = 'trend')
str(hc)
plot(hc, series = 1)
plot(hc, series = 2)
plot(hc, series = 3)
```

---

index-mvgam *Index* mvgam *objects*

Description

Index mvgam objects

Usage

```
## S3 method for class 'mvgam'
variables(x, ...)
```

Arguments

x	list object returned from mvgam. See mvgam()
...	Arguments passed to individual methods (if applicable).

Value

a list object of the variables that can be extracted, along with their aliases

Examples

```
## Not run:
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             data = simdat$data_train,
            burnin = 300,
            samples = 300,
            chains = 2)
variables(mod)

## End(Not run)
```

`irf.mvgam` *Calculate latent VAR impulse response functions*

Description

Compute Generalized or Orthogonalized Impulse Response Functions (IRFs) from `mvgam` models with Vector Autoregressive dynamics

Usage

```
irf(object, ...)

## S3 method for class 'mvgam'
irf(object, h = 1, cumulative = FALSE, orthogonal = FALSE, ...)
```

Arguments

`object`	`list` object of class `mvgam` resulting from a call to `mvgam()` that used a Vector Autoregressive latent process model (either as `VAR(cor = FALSE)` or `VAR(cor = TRUE)`)
`...`	ignored
`h`	Positive `integer` specifying the forecast horizon over which to calculate the IRF
`cumulative`	`Logical` flag indicating whether the IRF should be cumulative
`orthogonal`	`Logical` flag indicating whether orthogonalized IRFs should be calculated. Note that the order of the variables matters when calculating these

Details

Generalized or Orthogonalized Impulse Response Functions can be computed using the posterior estimates of Vector Autoregressive parameters. This function generates a positive "shock" for a target process at time `t = 0` and then calculates how each of the remaining processes in the latent VAR are expected to respond over the forecast horizon `h`. The function computes IRFs for all processes in the object and returns them in an array that can be plotted using the S3 `plot` function

Value

An object of class mvgam_irf containing the posterior IRFs. This object can be used with the supplied S3 functions plot

Author(s)

Nicholas J Clark

See Also

VAR, plot.mvgam_irf

Examples

```
# Simulate some time series that follow a latent VAR(1) process
simdat <- sim_mvgam(family = gaussian(),
                    n_series = 4,
                    trend_model = VAR(cor = TRUE),
                    prop_trend = 1)
plot_mvgam_series(data = simdat$data_train, series = 'all')

# Fit a model that uses a latent VAR(1)
mod <- mvgam(y ~ -1,
             trend_formula = ~ 1,
             trend_model = VAR(cor = TRUE),
             family = gaussian(),
             data = simdat$data_train,
             silent = 2)

# Calulate Generalized IRFs for each series
irfs <- irf(mod, h = 12, cumulative = FALSE)

# Plot them
plot(irfs, series = 1)
plot(irfs, series = 2)
plot(irfs, series = 3)
```

---

lfo_cv.mvgam *Approximate leave-future-out cross-validation of fitted* mvgam *objects*

Description

Approximate leave-future-out cross-validation of fitted mvgam objects

Usage

```
lfo_cv(object, ...)

## S3 method for class 'mvgam'
lfo_cv(
  object,
  data,
  min_t,
  fc_horizon = 1,
  pareto_k_threshold = 0.7,
  silent = 1,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
...	Ignored
data	A dataframe or list containing the model response variable and covariates required by the GAM formula. Should include columns: 'series' (character or factor index of the series IDs) 'time' (numeric index of the time point for each observation). Any other variables to be included in the linear predictor of formula must also be present
min_t	Integer specifying the minimum training time required before making predictions from the data. Default is either 30, or whatever training time allows for at least 10 lfo-cv calculations (i.e. pmin(max(data$time) - 10, 30)). This value is essentially arbitrary so it is highly recommended to change it to something that is more suitable to the data and models being evaluated
fc_horizon	Integer specifying the number of time steps ahead for evaluating forecasts
pareto_k_threshold	Proportion specifying the threshold over which the Pareto shape parameter is considered unstable, triggering a model refit. Default is 0.7
silent	Verbosity level between 0 and 2. If 1 (the default), most of the informational messages of compiler and sampler are suppressed. If 2, even more messages are suppressed. The actual sampling progress is still printed. Set refresh = 0 to turn this off as well. If using backend = "rstan" you can also set open_progress = FALSE to prevent opening additional progress bars.

Details

Approximate leave-future-out cross-validation uses an expanding training window scheme to evaluate a model on its forecasting ability. The steps used in this function mirror those laid out in the lfo vignette from the loo package, written by Paul Bürkner, Jonah Gabry, Aki Vehtari. First, we refit the model using the first min_t observations to perform a single exact fc_horizon-ahead forecast step. This forecast is evaluated against the min_t + fc_horizon out of sample observations using the Expected Log Predictive Density (ELPD). Next, we approximate each successive round of expanding window forecasts by moving forward one step at a time for i in 1:N_evaluations and

re-weighting draws from the model's posterior predictive distribution using Pareto Smoothed Importance Sampling (PSIS). In each iteration `i`, PSIS weights are obtained for the next observation that would have been included in the model if we had re-fit (i.e. the last observation that would have been in the training data, or `min_t + i`). If these importance ratios are stable, we consider the approximation adequate and use the re-weighted posterior's forecast for evaluating the next holdout set of testing observations (`(min_t + i + 1):(min_t + i + fc_horizon)`). At some point the importance ratio variability will become too large and importance sampling will fail. This is indicated by the estimated shape parameter `k` of the generalized Pareto distribution crossing a certain threshold `pareto_k_threshold`. Only then do we refit the model using all of the observations up to the time of the failure. We then restart the process and iterate forward until the next refit is triggered (Bürkner et al. 2020).

Value

A `list` of class `mvgam_lfo` containing the approximate ELPD scores, the Pareto-k shape values and 'the specified `pareto_k_threshold`

Author(s)

Nicholas J Clark

References

Paul-Christian Bürkner, Jonah Gabry & Aki Vehtari (2020). Approximate leave-future-out cross-validation for Bayesian time series models Journal of Statistical Computation and Simulation. 90:14, 2499-2523.

See Also

`forecast`, `score`, `compare_mvgams`

Examples

```
## Not run:
# Simulate from a Poisson-AR2 model with a seasonal smooth
set.seed(100)
dat <- sim_mvgam(T = 75,
                 n_series = 1,
                 prop_trend = 0.75,
                 trend_model = 'AR2',
                 family = poisson())

# Plot the time series
plot_mvgam_series(data = dat$data_train,
                  newdata = dat$data_test,
                  series = 1)

# Fit an appropriate model
mod_ar2 <- mvgam(y ~ s(season, bs = 'cc', k = 6),
                 trend_model = AR(p = 2),
                 family = poisson(),
```

```
                data = dat$data_train,
                newdata = dat$data_test,
                burnin = 300,
                samples = 300,
                chains = 2)

# Fit a less appropriate model
mod_rw <- mvgam(y ~ s(season, bs = 'cc', k = 6),
              trend_model = RW(),
              family = poisson(),
              data = dat$data_train,
              newdata = dat$data_test,
              burnin = 300,
              samples = 300,
              chains = 2)

# Compare Discrete Ranked Probability Scores for the testing period
fc_ar2 <- forecast(mod_ar2)
fc_rw <- forecast(mod_rw)
score_ar2 <- score(fc_ar2, score = 'drps')
score_rw <- score(fc_rw, score = 'drps')
sum(score_ar2$series_1$score)
sum(score_rw$series_1$score)

# Now use approximate leave-future-out CV to compare
# rolling forecasts; start at time point 40 to reduce
# computational time and to ensure enough data is available
# for estimating model parameters
lfo_ar2 <- lfo_cv(mod_ar2,
                 min_t = 40,
                 fc_horizon = 3)
lfo_rw <- lfo_cv(mod_rw,
                min_t = 40,
                fc_horizon = 3)

# Plot Pareto-K values and ELPD estimates
plot(lfo_ar2)
plot(lfo_rw)

# Proportion of timepoints in which AR2 model gives better forecasts
length(which((lfo_ar2$elpds - lfo_rw$elpds) > 0)) /
      length(lfo_ar2$elpds)

# A higher total ELPD is preferred
lfo_ar2$sum_ELPD
lfo_rw$sum_ELPD

## End(Not run)
```

`logLik.mvgam` *Compute pointwise Log-Likelihoods from fitted* `mvgam` *objects*

Description

Compute pointwise Log-Likelihoods from fitted mvgam objects

Usage

```
## S3 method for class 'mvgam'
logLik(object, linpreds, newdata, family_pars, include_forecast = TRUE, ...)
```

Arguments

object	list object returned from mvgam
linpreds	Optional matrix of linear predictor draws to use for calculating pointwise log-likelihoods
newdata	Optional data.frame or list object specifying which series each column in linpreds belongs to. If linpreds is supplied, then newdata must also be supplied
family_pars	Optional list containing posterior draws of family-specific parameters (i.e. shape, scale or overdispersion parameters). Required if linpreds and newdata are supplied
include_forecast	Logical. If newdata were fed to the model to compute forecasts, should the log-likelihood draws for these observations also be returned. Defaults to TRUE
...	Ignored

Value

A matrix of dimension n_samples x n_observations containing the pointwise log-likelihood draws for all observations in newdata. If no newdata is supplied, log-likelihood draws are returned for all observations that were originally fed to the model (training observations and, if supplied to the original model via the newdata argument in mvgam, testing observations)

Examples

```
## Not run:
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             data = simdat$data_train,
             burnin = 300,
             samples = 300,
             chains = 2)

# Extract logLikelihood values
lls <- logLik(mod)
str(lls)

## End(Not run)
```

loo.mvgam *LOO information criteria for* mvgam *models*

Description

Extract the LOOIC (leave-one-out information criterion) using loo::loo()

Usage

```
## S3 method for class 'mvgam'
loo(x, incl_dynamics = TRUE, ...)

## S3 method for class 'mvgam'
loo_compare(x, ..., model_names = NULL, incl_dynamics = TRUE)
```

Arguments

x	Object of class mvgam
incl_dynamics	Logical; indicates if any latent dynamic structures that were included in the model should be considered when calculating in-sample log-likelihoods. Defaults to TRUE
...	More mvgam objects.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

When comparing two (or more) fitted mvgam models, we can estimate the difference in their in-sample predictive accuracies using the Expcted Log Predictive Density (ELPD). This metric can be approximated using Pareto Smoothed Importance Sampling, which is a method to re-weight posterior draws to approximate what predictions the models might have made for a given datapoint had that datapoint not been included in the original model fit (i.e. if we were to run a leave-one-out cross-validation and then made a prediction for the held-out datapoint). See details from loo::loo() and loo::loo_compare() for further information on how this importance sampling works.

There are two fundamentally different ways to calculate ELPD from mvgam models that included dynamic latent processes (i.e. "trend_models"). The first is to use the predictions that were generated when estimating these latent processes by setting incl_dynamics = TRUE. This works in the same way that setting incl_autocor = TRUE in brms::prepare_predictions(). But it may also be desirable to compare predictions by considering that the dynamic processes are nuisance parameters that we'd wish to account for when making inferences about other processes in the model (i.e. the linear predictor effects). Setting incl_dynamics = FALSE will accomplish this by ignoring the dynamic processes when making predictions. This option matches up with what mvgam's prediction functions return (i.e. predict.mvgam, ppc, pp_check.mvgam, posterior_epred.mvgam) and will be far less forgiving of models that may be overfitting the training data due to highly flexible dynamic processes (such as Random Walks, for example). However setting incl_dynamics =

FALSE will often result in less stable Pareto k diagnostics for models with dynamic trends, making ELPD comparisons difficult and unstable. It is therefore recommended to generally stick with incl_dynamics = TRUE when comparing models based on in-sample fits, and then to perhaps use forecast evaluations for further scrutiny of models (see for example forecast.mvgam, score.mvgam_forecast and lfo_cv)

Value

for loo.mvgam, an object of class psis_loo (see loo::loo() for details). For loo_compare.mvgam, an object of class compare.loo (loo::loo_compare() for details)

Examples

```
# Simulate 4 time series with hierarchical seasonality
# and independent AR1 dynamic processes
set.seed(111)
simdat <- sim_mvgam(seasonality = 'hierarchical',
                    trend_model = AR(),
                    family = gaussian())

# Fit a model with shared seasonality
mod1 <- mvgam(y ~ s(season, bs = 'cc', k = 6),
              data = rbind(simdat$data_train,
              simdat$data_test),
              family = gaussian(),
              chains = 2)

# Inspect the model and calculate LOO
conditional_effects(mod1)
mc.cores.def <- getOption('mc.cores')
options(mc.cores = 1)
loo(mod1)

# Now fit a model with hierarchical seasonality
mod2 <- update(mod1,
               formula = y ~ s(season, bs = 'cc', k = 6) +
               s(season, series, bs = 'fs',
               xt = list(bs = 'cc'), k = 4),
               chains = 2)
conditional_effects(mod2)
loo(mod2)

# Now add AR1 dynamic errors to mod2
mod3 <- update(mod2,
               trend_model = AR(),
               chains = 2)
conditional_effects(mod3)
plot(mod3, type = 'trend')
loo(mod3)

# Compare models using LOO
loo_compare(mod1, mod2, mod3)
```

```
options(mc.cores = mc.cores.def)

# Compare forecast abilities using an expanding training window and
# forecasting ahead 1 timepoint from each window; the first window by includes
# the first 92 timepoints (of the 100 that were simulated)
max(mod2$obs_data$time)
lfo_mod2 <- lfo_cv(mod2, min_t = 92)
lfo_mod3 <- lfo_cv(mod3, min_t = 92)

# Take the difference in forecast ELPDs; a model with higher ELPD is preferred,
# so negative values here indicate that mod3 gave better forecasts for a particular
# out of sample timepoint
plot(y = lfo_mod2$elpds - lfo_mod3$elpds,
    x = lfo_mod2$eval_timepoints, pch = 16,
    ylab = 'ELPD_mod2 - ELPD_mod3',
    xlab = 'Evaluation timepoint')
abline(h = 0, lty = 'dashed')
```

`lv_correlations` *Calculate trend correlations based on mvgam latent factor loadings*

Description

This function uses samples of latent trends for each series from a fitted mvgam model to calculates correlations among series' trends

Usage

```
lv_correlations(object)
```

Arguments

`object`	`list` object returned from `mvgam`

Value

A `list` object containing the mean posterior correlations and the full array of posterior correlations

Examples

```
## Not run:
simdat <- sim_mvgam()
mod <- mvgam(y ~ s(season, bs = 'cc',
                   k = 6),
             trend_model = AR(),
             use_lv = TRUE,
             n_lv = 2,
             data = simdat$data_train,
             burnin = 300,
```

```
            samples = 300,
            chains = 2)
lvcors <- lv_correlations(mod)
names(lvcors)
lapply(lvcors, class)

## End(Not run)
```

`mcmc_plot.mvgam` *MCMC plots as implemented in* **bayesplot**

Description

Convenient way to call MCMC plotting functions implemented in the **bayesplot** package

Usage

```
## S3 method for class 'mvgam'
mcmc_plot(
  object,
  type = "intervals",
  variable = NULL,
  regex = FALSE,
  use_alias = TRUE,
  ...
)
```

Arguments

`object`	An R object typically of class `brmsfit`
`type`	The type of the plot. Supported types are (as names) `hist`, `dens`, `hist_by_chain`, `dens_overlay`, `violin`, `intervals`, `areas`, `areas_ridges`, `combo`, `acf`, `acf_bar`, `trace`, `trace_highlight`, `scatter`, `hex`, `pairs`, `violin`, `rhat`, `rhat_hist`, `neff`, `neff_hist` and `nuts_energy`. For an overview on the various plot types see `MCMC-overview`.
`variable`	Names of the variables (parameters) to plot, as given by a character vector or a regular expression (if `regex = TRUE`). By default, a hopefully not too large selection of variables is plotted.
`regex`	Logical; Indicates whether `variable` should be treated as regular expressions. Defaults to `FALSE`.
`use_alias`	Logical. If more informative names for parameters are available (i.e. for beta coefficients `b` or for smoothing parameters `rho`), replace the uninformative names with the more informative alias. Defaults to `TRUE`
`...`	Additional arguments passed to the plotting functions. See `MCMC-overview` for more details.

Value

A ggplot object that can be further customized using the **ggplot2** package.

See Also

mvgam_draws for an overview of some of the shortcut strings that can be used for argument variable

Examples

```
## Not run:
simdat <- sim_mvgam(n_series = 1, trend_model = AR())
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)
mcmc_plot(mod)
mcmc_plot(mod, type = 'neff_hist')
mcmc_plot(mod, variable = 'betas', type = 'areas')
mcmc_plot(mod, variable = 'trend_params', type = 'combo')

## End(Not run)
```

---

model.frame.mvgam *Extract model.frame from a fitted mvgam object*

Description

Extract model.frame from a fitted mvgam object

Usage

```
## S3 method for class 'mvgam'
model.frame(formula, trend_effects = FALSE, ...)

## S3 method for class 'mvgam_prefit'
model.frame(formula, trend_effects = FALSE, ...)
```

Arguments

formula	a model formula or terms object or an R object.
trend_effects	logical, return the model.frame from the observation model (if FALSE) or from the underlying process model (ifTRUE)
...	Ignored

Value

A matrix containing the fitted model frame

Author(s)

Nicholas J Clark

`monotonic` *Monotonic splines in mvgam*

Description

Uses constructors from package **splines2** to build monotonically increasing or decreasing splines. Details also in Wang & Yan (2021).

Usage

```
## S3 method for class 'moi.smooth.spec'
smooth.construct(object, data, knots)

## S3 method for class 'mod.smooth.spec'
smooth.construct(object, data, knots)

## S3 method for class 'moi.smooth'
Predict.matrix(object, data)

## S3 method for class 'mod.smooth'
Predict.matrix(object, data)
```

Arguments

`object`	A smooth specification object, usually generated by a term `s(x, bs = "moi", ...)` or `s(x, bs = "mod", ...)`
`data`	a list containing just the data (including any `by` variable) required by this term, with names corresponding to `object$term` (and `object$by`). The `by` variable is the last element.
`knots`	a list containing any knots supplied for basis setup — in same order and with same names as `data`. Can be `NULL`. See details for further information.

Details

The constructor is not normally called directly, but is rather used internally by mvgam. If they are not supplied then the knots of the spline are placed evenly throughout the covariate values to which the term refers: For example, if fitting 101 data with an 11 knot spline of x then there would be a knot at every 10th (ordered) x value. The spline is an implementation of the closed-form I-spline basis based on the recursion formula given by Ramsay (1988), in which the basis coefficients must be constrained to either be non-negative (for monotonically increasing functions) or non-positive (monotonically decreasing)

Take note that when using either monotonic basis, the number of basis functions `k` must be supplied as an even integer due to the manner in which monotonic basis functions are constructed

Value

An object of class "moi.smooth" or "mod.smooth". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain a slot called boundary that defines the endpoints beyond which the spline will begin extrapolating (extrapolation is flat due to the first order penalty placed on the smooth function)

Note

This constructor will result in a valid smooth if using a call to gam or bam, however the resulting functions will not be guaranteed to be monotonic because constraints on basis coefficients will not be enforced

Author(s)

Nicholas J Clark

References

Wang, Wenjie, and Jun Yan. "Shape-Restricted Regression Splines with R Package splines2." Journal of Data Science 19.3 (2021).

Ramsay, J. O. (1988). Monotone regression splines in action. Statistical Science, 3(4), 425–441.

Examples

```
# Simulate data from a monotonically increasing function
set.seed(123123)
x <- runif(80) * 4 - 1
x <- sort(x)
f <- exp(4 * x) / (1 + exp(4 * x))
y <- f + rnorm(80) * 0.1
plot(x, y)

# A standard TRPS smooth doesn't capture monotonicity
library(mgcv)
mod_data <- data.frame(y = y, x = x)
mod <- gam(y ~ s(x, k = 16),
           data = mod_data,
           family = gaussian())

library(marginaleffects)
plot_predictions(mod,
                 by = 'x',
                 newdata = data.frame(x = seq(min(x) - 0.5,
                                              max(x) + 0.5,
                                              length.out = 100)),
                 points = 0.5)

# Using the 'moi' basis in mvgam rectifies this
mod_data$time <- 1:NROW(mod_data)
mod2 <- mvgam(y ~ s(x, bs = 'moi', k = 18),
```

```
             data = mod_data,
             family = gaussian(),
             chains = 2)

plot_predictions(mod2,
                 by = 'x',
                 newdata = data.frame(x = seq(min(x) - 0.5,
                                              max(x) + 0.5,
                                              length.out = 100)),
                 points = 0.5)

plot(mod2, type = 'smooth', realisations = TRUE)

# 'by' terms that produce a different smooth for each level of the 'by'
# factor are also allowed
set.seed(123123)
x <- runif(80) * 4 - 1
x <- sort(x)

# Two different monotonic smooths, one for each factor level
f <- exp(4 * x) / (1 + exp(4 * x))
f2 <- exp(3.5 * x) / (1 + exp(3 * x))
fac <- c(rep('a', 80), rep('b', 80))
y <- c(f + rnorm(80) * 0.1,
       f2 + rnorm(80) * 0.2)
plot(x, y[1:80])
plot(x, y[81:160])

# Gather all data into a data.frame, including the factor 'by' variable
mod_data <- data.frame(y, x, fac = as.factor(fac))
mod_data$time <- 1:NROW(mod_data)

# Fit a model with different smooths per factor level
mod <- mvgam(y ~ s(x, bs = 'moi', by = fac, k = 8),
             data = mod_data,
             family = gaussian(),
             chains = 2)

# Visualise the different monotonic functions
plot_predictions(mod, condition = c('x', 'fac', 'fac'),
                 points = 0.5)
plot(mod, type = 'smooth', realisations = TRUE)

# First derivatives (on the link scale) should never be
# negative for either factor level
(derivs <- slopes(mod, variables = 'x',
                  by = c('x', 'fac'),
                  type = 'link'))
all(derivs$estimate > 0)
```

`mvgam` *Fit a Bayesian dynamic GAM to a univariate or multivariate set of time series*

Description

This function estimates the posterior distribution for Generalised Additive Models (GAMs) that can include smooth spline functions, specified in the GAM formula, as well as latent temporal processes, specified by `trend_model`. Further modelling options include State-Space representations to allow covariates and dynamic processes to occur on the latent 'State' level while also capturing observation-level effects. Prior specifications are flexible and explicitly encourage users to apply prior distributions that actually reflect their beliefs. In addition, model fits can easily be assessed and compared with posterior predictive checks, forecast comparisons and leave-one-out / leave-future-out cross-validation.

Usage

```
mvgam(
  formula,
  trend_formula,
  knots,
  trend_knots,
  data,
  data_train,
  newdata,
  data_test,
  run_model = TRUE,
  prior_simulation = FALSE,
  return_model_data = FALSE,
  family = "poisson",
  share_obs_params = FALSE,
  use_lv = FALSE,
  n_lv,
  trend_map,
  trend_model = "None",
  drift = FALSE,
  noncentred = FALSE,
  chains = 4,
  burnin = 500,
  samples = 500,
  thin = 1,
  parallel = TRUE,
  threads = 1,
  priors,
  refit = FALSE,
  lfo = FALSE,
  residuals = TRUE,
```

```
  use_stan = TRUE,
  backend = getOption("brms.backend", "cmdstanr"),
  algorithm = getOption("brms.algorithm", "sampling"),
  autoformat = TRUE,
  save_all_pars = FALSE,
  max_treedepth = 12,
  adapt_delta = 0.85,
  silent = 1,
  jags_path,
  ...
)
```

Arguments

formula	A character string specifying the GAM observation model formula. These are exactly like the formula for a GLM except that smooth terms, s(), te(), ti(), t2(), as well as time-varying dynamic() terms, can be added to the right hand side to specify that the linear predictor depends on smooth functions of predictors (or linear functionals of these). In nmix() family models, the formula is used to set up a linear predictor for the detection probability. Details of the formula syntax used by **mvgam** can be found in mvgam_formulae
trend_formula	An optional character string specifying the GAM process model formula. If supplied, a linear predictor will be modelled for the latent trends to capture process model evolution separately from the observation model. Should not have a response variable specified on the left-hand side of the formula (i.e. a valid option would be ~ season + s(year)). Also note that you should not use the identifier series in this formula to specify effects that vary across time series. Instead you should use trend. This will ensure that models in which a trend_map is supplied will still work consistently (i.e. by allowing effects to vary across process models, even when some time series share the same underlying process model). This feature is only currently available for RW(), AR() and VAR() trend models. In nmix() family models, the trend_formula is used to set up a linear predictor for the underlying latent abundance. Be aware that it can be very challenging to simultaneously estimate intercept parameters for both the observation mode (captured by formula) and the process model (captured by trend_formula). Users are recommended to drop one of these using the - 1 convention in the formula right hand side.
knots	An optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the k value supplied (note that the number of knots is not always just k). Different terms can use different numbers of knots, unless they share a covariate
trend_knots	As for knots above, this is an optional list of knot values for smooth functions within the trend_formula
data	A dataframe or list containing the model response variable and covariates required by the GAM formula and optional trend_formula. Should include columns: #'

- series (a factor index of the series IDs; the number of levels should be identical to the number of unique series labels (i.e. n_series = length(levels(data$series))))
- time (numeric or integer index of the time point for each observation). For most dynamic trend types available in mvgam (see argument trend_model), time should be measured in discrete, regularly spaced intervals (i.e. c(1, 2, 3, ...)). However you can use irregularly spaced intervals if using trend_model = CAR(1), though note that any temporal intervals that are exactly 0 will be adjusted to a very small number (1e-12) to prevent sampling errors. See an example of CAR() trends in CAR

Should also include any other variables to be included in the linear predictor of formula

data_train Deprecated. Still works in place of data but users are recommended to use data instead for more seamless integration into R workflows

newdata Optional dataframe or list of test data containing at least series and time in addition to any other variables included in the linear predictor of formula. If included, the observations in variable y will be set to NA when fitting the model so that posterior simulations can be obtained

data_test Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows

run_model logical. If FALSE, the model is not fitted but instead the function will return the model file and the data / initial values that are needed to fit the model outside of mvgam

prior_simulation
logical. If TRUE, no observations are fed to the model, and instead simulations from prior distributions are returned

return_model_data
logical. If TRUE, the list of data that is needed to fit the model is returned, along with the initial values for smooth and AR parameters, once the model is fitted. This will be helpful if users wish to modify the model file to add other stochastic elements that are not currently available in mvgam. Default is FALSE to reduce the size of the returned object, unless run_model == FALSE

family family specifying the exponential observation family for the series. Currently supported families are:

- gaussian() for real-valued data
- betar() for proportional data on (0,1)
- lognormal() for non-negative real-valued data
- student_t() for real-valued data
- Gamma() for non-negative real-valued data
- bernoulli() for binary data
- poisson() for count data
- nb() for overdispersed count data
- binomial() for count data with imperfect detection when the number of trials is known; note that the cbind() function must be used to bind the discrete observations and the discrete number of trials
- beta_binomial() as for binomial() but allows for overdispersion

- nmix() for count data with imperfect detection when the number of trials is unknown and should be modeled via a State-Space N-Mixture model. The latent states are Poisson, capturing the 'true' latent abundance, while the observation process is Binomial to account for imperfect detection. See mvgam_families for an example of how to use this family

Note that only nb() and poisson() are available if using JAGS as the backend. Default is poisson(). See mvgam_families for more details

share_obs_params

logical. If TRUE and the family has additional family-specific observation parameters (e.g. variance components in student_t() or gaussian(), or dispersion parameters in nb() or betar()), these parameters will be shared across all series. This is handy if you have multiple time series that you believe share some properties, such as being from the same species over different spatial units. Default is FALSE.

use_lv

logical. If TRUE, use dynamic factors to estimate series' latent trends in a reduced dimension format. Only available for RW(), AR() and GP() trend models. Defaults to FALSE

n_lv

integer the number of latent dynamic factors to use if use_lv == TRUE. Cannot be > n_series. Defaults arbitrarily to min(2, floor(n_series / 2))

trend_map

Optional data.frame specifying which series should depend on which latent trends. Useful for allowing multiple series to depend on the same latent trend process, but with different observation processes. If supplied, a latent factor model is set up by setting use_lv = TRUE and using the mapping to set up the shared trends. Needs to have column names series and trend, with integer values in the trend column to state which trend each series should depend on. The series column should have a single unique entry for each series in the data (names should perfectly match factor levels of the series variable in data). Note that if this is supplied, the intercept parameter in the process model will NOT be automatically suppressed. See examples for details

trend_model

character or function specifying the time series dynamics for the latent trend. Options are:

- None (no latent trend component; i.e. the GAM component is all that contributes to the linear predictor, and the observation process is the only source of error; similarly to what is estimated by gam)
- 'RW' or RW()
- 'AR1' or AR(p = 1)
- 'AR2' or AR(p = 2)
- 'AR3' or AR(p = 3)
- 'CAR1' or CAR(p = 1)
- 'VAR1' or VAR()(only available in Stan)
- 'PWlogistic', 'PWlinear' or PW() (only available in Stan)
- 'GP' or GP() (Gaussian Process with squared exponential kernel; only available in Stan)

For all trend types apart from GP(), CAR() and PW(), moving average and/or correlated process error terms can also be estimated (for example, RW(cor = TRUE)

will set up a multivariate Random Walk if n_series > 1). See mvgam_trends for more details

drift — Deprecated. If you wish to estimate drift parameters, include parametric fixed effects of 'time' in your formulae instead.

noncentred — logical Use the non-centred parameterisation for autoregressive trend models? Setting to TRUE will reparameterise the model to avoid possible degeneracies that can show up when estimating the latent dynamic random effects. For some models, this can produce big gains in efficiency, meaning that fewer burnin and sampling iterations are required for posterior exploration. But for other models, where the data are highly informative about the latent dynamic processes, this can actually lead to worse performance. Only available for certain trend models (i.e. RW(), AR(), or CAR(), or for trend = 'None' when using a trend_formula). Not yet available for moving average or correlated error models

chains — integer specifying the number of parallel chains for the model. Ignored if algorithm %in% c('meanfield', 'fullrank', 'pathfinder', 'laplace')

burnin — integer specifying the number of warmup iterations of the Markov chain to run to tune sampling algorithms. Ignored if algorithm %in% c('meanfield', 'fullrank', 'pathfinder', 'laplace')

samples — integer specifying the number of post-warmup iterations of the Markov chain to run for sampling the posterior distribution

thin — Thinning interval for monitors. Ignored if algorithm %in% c('meanfield', 'fullrank', 'pathfinder', 'laplace')

parallel — logical specifying whether multiple cores should be used for generating MCMC simulations in parallel. If TRUE, the number of cores to use will be min(c(chains, parallel::detectCores() - 1))

threads — integer Experimental option to use multithreading for within-chain parallelisation in Stan. We recommend its use only if you are experienced with Stan's reduce_sum function and have a slow running model that cannot be sped up by any other means. Only available for some families(poisson(), nb(), gaussian()) and when using Cmdstan as the backend

priors — An optional data.frame with prior definitions (in JAGS or Stan syntax). if using Stan, this can also be an object of class brmsprior (see. prior for details). See get_mvgam_priors and 'Details' for more information on changing default prior distributions

refit — Logical indicating whether this is a refit, called using update.mvgam. Users should leave as FALSE

lfo — Logical indicating whether this is part of a call to lfo_cv.mvgam. Returns a lighter version of the model with no residuals and fewer monitored parameters to speed up post-processing. But other downstream functions will not work properly, so users should always leave this set as FALSE

residuals — Logical indicating whether to compute series-level randomized quantile residuals and include them as part of the returned object. Defaults to TRUE, but you can set to FALSE to save computational time and reduce the size of the returned object (users can always add residuals to an object of class mvgam using add_residuals)

use_stan — Logical. If TRUE, the model will be compiled and sampled using Hamiltonian Monte Carlo with a call to cmdstan_model or a call to stan. Note that there are many more options when using Stan vs JAGS

backend — Character string naming the package to use as the backend for fitting the Stan model (if use_stan = TRUE). Options are "cmdstanr" (the default) or "rstan". Can be set globally for the current R session via the "brms.backend" option (see options). Details on the rstan and cmdstanr packages are available at https://mc-stan.org/rstan/ and https://mc-stan.org/cmdstanr/, respectively

algorithm — Character string naming the estimation approach to use. Options are "sampling" for MCMC (the default), "meanfield" for variational inference with factorized normal distributions, "fullrank" for variational inference with a multivariate normal distribution, "laplace" for a Laplace approximation (only available when using cmdstanr as the backend) or "pathfinder" for the pathfinder algorithm (only currently available when using cmdstanr as the backend). Can be set globally for the current R session via the "brms.algorithm" option (see options). Limited testing suggests that "meanfield" performs best out of the non-MCMC approximations for dynamic GAMs, possibly because of the difficulties estimating covariances among the many spline parameters and latent trend parameters. But rigorous testing has not been carried out

autoformat — Logical. Use the stanc parser to automatically format the Stan code and check for deprecations. Only for development purposes, so leave to TRUE

save_all_pars — Logical flag to indicate if draws from all variables defined in Stan's parameters block should be saved (default is FALSE).

max_treedepth — positive integer placing a cap on the number of simulation steps evaluated during each iteration when use_stan == TRUE. Default is 12. Increasing this value can sometimes help with exploration of complex posterior geometries, but it is rarely fruitful to go above a max_treedepth of 14

adapt_delta — positive numeric between 0 and 1 defining the target average proposal acceptance probability during Stan's adaptation period, if use_stan == TRUE. Default is 0.8. In general you should not need to change adapt_delta unless you see a warning message about divergent transitions, in which case you can increase adapt_delta from the default to a value closer to 1 (e.g. from 0.95 to 0.99, or from 0.99 to 0.999, etc). The step size used by the numerical integrator is a function of adapt_delta in that increasing adapt_delta will result in a smaller step size and fewer divergences. Increasing adapt_delta will typically result in a slower sampler, but it will always lead to a more robust sampler

silent — Verbosity level between 0 and 2. If 1 (the default), most of the informational messages of compiler and sampler are suppressed. If 2, even more messages are suppressed. The actual sampling progress is still printed. Set refresh = 0 to turn this off as well. If using backend = "rstan" you can also set open_progress = FALSE to prevent opening additional progress bars.

jags_path — Optional character vector specifying the path to the location of the JAGS executable (.exe) to use for modelling if use_stan == FALSE. If missing, the path will be recovered from a call to findjags

... — Further arguments passed to Stan. For backend = "rstan" the arguments are passed to sampling or vb. For backend = "cmdstanr" the arguments are passed

to the cmdstanr::sample, cmdstanr::variational, cmdstanr::laplace or cmdstanr::pathfinder method

Details

Dynamic GAMs are useful when we wish to predict future values from time series that show temporal dependence but we do not want to rely on extrapolating from a smooth term (which can sometimes lead to unpredictable and unrealistic behaviours). In addition, smooths can often try to wiggle excessively to capture any autocorrelation that is present in a time series, which exacerbates the problem of forecasting ahead. As GAMs are very naturally viewed through a Bayesian lens, and we often must model time series that show complex distributional features and missing data, parameters for mvgam models are estimated in a Bayesian framework using Markov Chain Monte Carlo by default. A general overview is provided in the primary vignettes: vignette("mvgam_overview") and vignette("data_in_mvgam"). For a full list of available vignettes see vignette(package = "mvgam")

Formula syntax: Details of the formula syntax used by **mvgam** can be found in mvgam_formulae. Note that it is possible to supply an empty formula where there are no predictors or intercepts in the observation model (i.e. y ~ 0 or y ~ -1). In this case, an intercept-only observation model will be set up but the intercept coefficient will be fixed at zero. This can be handy if you wish to fit pure State-Space models where the variation in the dynamic trend controls the average expectation, and/or where intercepts are non-identifiable (as in piecewise trends, see examples below)

Families and link functions: Details of families supported by **mvgam** can be found in mvgam_families.

Trend models: Details of latent trend dynamic models supported by **mvgam** can be found in mvgam_trends.

Priors: Default priors for intercepts and any scale parameters are generated using the same practice as **brms**. Prior distributions for most important model parameters can be altered by the user to inspect model sensitivities to given priors (see get_mvgam_priors for details). Note that latent trends are estimated on the link scale so choose priors accordingly. However more control over the model specification can be accomplished by first using mvgam as a baseline, then editing the returned model accordingly. The model file can be edited and run outside of mvgam by setting run_model = FALSE and this is encouraged for complex modelling tasks. Note, no priors are formally checked to ensure they are in the right syntax for the respective probabilistic modelling framework, so it is up to the user to ensure these are correct (i.e. use dnorm for normal densities in JAGS, with the mean and precision parameterisation; but use normal for normal densities in Stan, with the mean and standard deviation parameterisation)

Random effects: For any smooth terms using the random effect basis (smooth.construct.re.smooth.spec), a non-centred parameterisation is automatically employed to avoid degeneracies that are common in hierarchical models. Note however that centred versions may perform better for series that are particularly informative, so as with any foray into Bayesian modelling, it is worth building an understanding of the model's assumptions and limitations by following a principled workflow. Also note that models are parameterised using drop.unused.levels = FALSE in jagam to ensure predictions can be made for all levels of the supplied factor variable

Observation level parameters: When more than one series is included in data and an observa-

tion family that contains more than one parameter is used, additional observation family parameters (i.e. phi for nb() or sigma for gaussian()) are by default estimated independently for each series. But if you wish for the series to share the same observation parameters, set share_obs_params = TRUE

Factor regularisation: When using a dynamic factor model for the trends with JAGS factor precisions are given regularized penalty priors to theoretically allow some factors to be dropped from the model by squeezing increasing factors' variances to zero. This is done to help protect against selecting too many latent factors than are needed to capture dependencies in the data, so it can often be advantageous to set n_lv to a slightly larger number. However larger numbers of factors do come with additional computational costs so these should be balanced as well. When using Stan, all factors are parameterised with fixed variance parameters

Residuals: For each series, randomized quantile (i.e. Dunn-Smyth) residuals are calculated for inspecting model diagnostics If the fitted model is appropriate then Dunn-Smyth residuals will be standard normal in distribution and no autocorrelation will be evident. When a particular observation is missing, the residual is calculated by comparing independent draws from the model's posterior distribution

Using Stan: mvgam is primarily designed to use Hamiltonian Monte Carlo for parameter estimation via the software Stan (using either the cmdstanr or rstan interface). There are great advantages when using Stan over Gibbs / Metropolis Hastings samplers, which includes the option to estimate nonlinear effects via Hilbert space approximate Gaussian Processes, the availability of a variety of inference algorithms (i.e. variational inference, laplacian inference etc...) and capabilities to enforce stationarity for complex Vector Autoregressions. Because of the many advantages of Stan over JAGS, *further development of the package will only be applied to* Stan. This includes the planned addition of more response distributions, plans to handle zero-inflation, and plans to incorporate a greater variety of trend models. Users are strongly encouraged to opt for Stan over JAGS in any proceeding workflows

How to start?: The mvgam cheatsheet is a good starting place if you are just learning to use the package. It gives an overview of the package's key functions and objects, as well as providing a reasonable workflow that new users can follow. In general it is recommended to

- 1. Check that your time series data are in a suitable long format for mvgam modeling (see the data formatting vignette for guidance)
- 2. Inspect features of the data using plot_mvgam_series. Now is also a good time to familiarise yourself with the package's example workflows that are detailed in the vignettes. In particular, the getting started vignette, the shared latent states vignette, the time-varying effects vignette and the State-Space models vignette all provide detailed information about how to structure, fit and interrogate Dynamic Generalized Additive Models in mvgam. Some more specialized how-to articles include "Incorporating time-varying seasonality in forecast models" and "Temporal autocorrelation in GAMs and the mvgam package"
- 3. Carefully think about how to structure linear predictor effects (i.e. smooth terms using s, te or ti, GPs using gp, dynamic time-varying effects using dynamic, and parametric terms), latent temporal trend components (see mvgam_trends) and the appropriate observation family (see mvgam_families). Use get_mvgam_priors to see default prior distributions for stochastic parameters

- 4. Change default priors using appropriate prior knowledge (see `prior`)
- 5. Fit the model using either Hamiltonian Monte Carlo or an approximation algorithm (i.e. change the backend argument) and use `summary.mvgam`, `conditional_effects.mvgam`, `mcmc_plot.mvgam`, `pp_check.mvgam` and `plot.mvgam` to inspect / interrogate the model
- 6. Update the model as needed and use `loo_compare.mvgam` for in-sample model comparisons, or alternatively use `forecast.mvgam` and `score.mvgam_forecast` to compare models based on out-of-sample forecasts (see the forecast evaluation vignette for guidance)
- 7. When satisfied with the model structure, use `predict.mvgam`, `plot_predictions` and/or `plot_slopes` for more targeted inferences (see "How to interpret and report nonlinear effects from Generalized Additive Models" for some guidance on interpreting GAMs)

Value

A `list` object of class `mvgam` containing model output, the text representation of the model file, the mgcv model output (for easily generating simulations at unsampled covariate values), Dunn-Smyth residuals for each series and key information needed for other functions in the package. See `mvgam-class` for details. Use `methods(class = "mvgam")` for an overview on available methods.

Author(s)

Nicholas J Clark

References

Nicholas J Clark & Konstans Wells (2020). Dynamic generalised additive models (DGAMs) for forecasting discrete ecological time series. Methods in Ecology and Evolution. 14:3, 771-784.

See Also

`jagam`, `gam`, `gam.models`, `get_mvgam_priors`

Examples

```
# Simulate a collection of three time series that have shared seasonal dynamics
# and independent AR1 trends, with a Poisson observation process
set.seed(0)
dat <- sim_mvgam(T = 80,
                 n_series = 3,
                 mu = 2,
                 trend_model = AR(p = 1),
                 prop_missing = 0.1,
                 prop_trend = 0.6)

# Plot key summary statistics for a single series
plot_mvgam_series(data = dat$data_train, series = 1)

# Plot all series together
plot_mvgam_series(data = dat$data_train, series = 'all')

# Formulate a model using Stan where series share a cyclic smooth for
```

```
# seasonality and each series has an independent AR1 temporal process.
# Note that 'noncentred = TRUE' will likely give performance gains.
# Set run_model = FALSE to inspect the returned objects
mod1 <- mvgam(formula = y ~ s(season, bs = 'cc', k = 6),
              data = dat$data_train,
              trend_model = AR(),
              family = poisson(),
              noncentred = TRUE,
              use_stan = TRUE,
              run_model = FALSE)

# View the model code in Stan language
stancode(mod1)

# View the data objects needed to fit the model in Stan
sdata1 <- standata(mod1)
str(sdata1)

# Now fit the model
mod1 <- mvgam(formula = y ~ s(season, bs = 'cc', k = 6),
              data = dat$data_train,
              trend_model = AR(),
              family = poisson(),
              noncentred = TRUE,
              chains = 2)

# Extract the model summary
summary(mod1)

# Plot the estimated historical trend and forecast for one series
plot(mod1, type = 'trend', series = 1)
plot(mod1, type = 'forecast', series = 1)

# Residual diagnostics
plot(mod1, type = 'residuals', series = 1)
resids <- residuals(mod1)
str(resids)

# Compute the forecast using covariate information in data_test
fc <- forecast(mod1, newdata = dat$data_test)
str(fc)
plot(fc)

# Plot the estimated seasonal smooth function
plot(mod1, type = 'smooths')

# Plot estimated first derivatives of the smooth
plot(mod1, type = 'smooths', derivatives = TRUE)

# Plot partial residuals of the smooth
plot(mod1, type = 'smooths', residuals = TRUE)

# Plot posterior realisations for the smooth
```

```
plot(mod1, type = 'smooths', realisations = TRUE)

# Plot conditional response predictions using marginaleffects
library(marginaleffects)
conditional_effects(mod1)
plot_predictions(mod1, condition = 'season', points = 0.5)

# Generate posterior predictive checks using bayesplot
pp_check(mod1)

# Extract observation model beta coefficient draws as a data.frame
beta_draws_df <- as.data.frame(mod1, variable = 'betas')
head(beta_draws_df)
str(beta_draws_df)

# Investigate model fit
mc.cores.def <- getOption('mc.cores')
options(mc.cores = 1)
loo(mod1)
options(mc.cores = mc.cores.def)


# Example of supplying a trend_map so that some series can share
# latent trend processes
sim <- sim_mvgam(n_series = 3)
mod_data <- sim$data_train

# Here, we specify only two latent trends; series 1 and 2 share a trend,
# while series 3 has it's own unique latent trend
trend_map <- data.frame(series = unique(mod_data$series),
                        trend = c(1, 1, 2))

# Fit the model using AR1 trends
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_map = trend_map,
             trend_model = AR(),
             data = mod_data,
             return_model_data = TRUE,
             chains = 2)

# The mapping matrix is now supplied as data to the model in the 'Z' element
mod$model_data$Z
code(mod)

# The first two series share an identical latent trend; the third is different
plot(mod, type = 'trend', series = 1)
plot(mod, type = 'trend', series = 2)
plot(mod, type = 'trend', series = 3)


# Example of how to use dynamic coefficients
# Simulate a time-varying coefficient for the effect of temperature
set.seed(123)
```

```
N <- 200
beta_temp <- vector(length = N)
beta_temp[1] <- 0.4
for(i in 2:N){
 beta_temp[i] <- rnorm(1, mean = beta_temp[i - 1] - 0.0025, sd = 0.05)
}
plot(beta_temp)

# Simulate a covariate called 'temp'
temp <- rnorm(N, sd = 1)

# Simulate the Gaussian observation process
out <- rnorm(N, mean = 4 + beta_temp * temp,
             sd = 0.5)

# Gather necessary data into a data.frame; split into training / testing
data = data.frame(out, temp, time = seq_along(temp))
data_train <- data[1:180,]
data_test <- data[181:200,]

# Fit the model using the dynamic() formula helper
mod <- mvgam(out ~
               dynamic(temp,
                       scale = FALSE,
                       k = 40),
             family = gaussian(),
             data = data_train,
             newdata = data_test,
             chains = 2)

# Inspect the model summary, forecast and time-varying coefficient distribution
summary(mod)
plot(mod, type = 'smooths')
fc <- forecast(mod, newdata = data_test)
plot(fc)

# Propagating the smooth term shows how the coefficient is expected to evolve
plot_mvgam_smooth(mod, smooth = 1, newdata = data)
abline(v = 180, lty = 'dashed', lwd = 2)
points(beta_temp, pch = 16)


# Example showing how to incorporate an offset; simulate some count data
# with different means per series
set.seed(100)
dat <- sim_mvgam(prop_trend = 0, mu = c(0, 2, 2),
                 seasonality = 'hierarchical')

# Add offset terms to the training and testing data
dat$data_train$offset <- 0.5 * as.numeric(dat$data_train$series)
dat$data_test$offset <- 0.5 * as.numeric(dat$data_test$series)

# Fit a model that includes the offset in the linear predictor as well as
```

```
# hierarchical seasonal smooths
mod <- mvgam(formula = y ~ offset(offset) +
               s(series, bs = 're') +
               s(season, bs = 'cc') +
               s(season, by = series, m = 1, k = 5),
             data = dat$data_train,
             chains = 2)

# Inspect the model file to see the modification to the linear predictor
# (eta)
code(mod)

# Forecasts for the first two series will differ in magnitude
fc <- forecast(mod, newdata = dat$data_test)
layout(matrix(1:2, ncol = 2))
plot(fc, series = 1, ylim = c(0, 75))
plot(fc, series = 2, ylim = c(0, 75))
layout(1)

# Changing the offset for the testing data should lead to changes in
# the forecast
dat$data_test$offset <- dat$data_test$offset - 2
fc <- forecast(mod, newdata = dat$data_test)
plot(fc)

# Relative Risks can be computed by fixing the offset to the same value
# for each series
dat$data_test$offset <- rep(1, NROW(dat$data_test))
preds_rr <- predict(mod, type = 'link', newdata = dat$data_test,
                    summary = FALSE)
series1_inds <- which(dat$data_test$series == 'series_1')
series2_inds <- which(dat$data_test$series == 'series_2')

# Relative Risks are now more comparable among series
layout(matrix(1:2, ncol = 2))
plot(preds_rr[1, series1_inds], type = 'l', col = 'grey75',
     ylim = range(preds_rr),
     ylab = 'Series1 Relative Risk', xlab = 'Time')
for(i in 2:50){
 lines(preds_rr[i, series1_inds], col = 'grey75')
}

plot(preds_rr[1, series2_inds], type = 'l', col = 'darkred',
     ylim = range(preds_rr),
     ylab = 'Series2 Relative Risk', xlab = 'Time')
for(i in 2:50){
 lines(preds_rr[i, series2_inds], col = 'darkred')
 }
layout(1)


# Example showcasing how cbind() is needed for Binomial observations
# Simulate two time series of Binomial trials
```

```
trials <- sample(c(20:25), 50, replace = TRUE)
x <- rnorm(50)
detprob1 <- plogis(-0.5 + 0.9*x)
detprob2 <- plogis(-0.1 -0.7*x)
dat <- rbind(data.frame(y = rbinom(n = 50, size = trials, prob = detprob1),
                        time = 1:50,
                        series = 'series1',
                        x = x,
                        ntrials = trials),
             data.frame(y = rbinom(n = 50, size = trials, prob = detprob2),
                        time = 1:50,
                        series = 'series2',
                        x = x,
                        ntrials = trials))
dat <- dplyr::mutate(dat, series = as.factor(series))
dat <- dplyr::arrange(dat, time, series)
plot_mvgam_series(data = dat, series = 'all')

# Fit a model using the binomial() family; must specify observations
# and number of trials in the cbind() wrapper
mod <- mvgam(cbind(y, ntrials) ~ series + s(x, by = series),
             family = binomial(),
             data = dat,
             chains = 2)
summary(mod)
pp_check(mod, type = "bars_grouped",
         group = "series", ndraws = 50)
pp_check(mod, type = "ecdf_overlay_grouped",
         group = "series", ndraws = 50)
conditional_effects(mod, type = 'link')
```

`mvgam-class` *Fitted* `mvgam` *object description*

Description

A fitted `mvgam` object returned by function `mvgam`. Run `methods(class = "mvgam")` to see an overview of available methods.

Details

A `mvgam` object contains the following elements:

- `call` the original observation model formula
- `trend_call` If a `trend_formula` was supplied, the original trend model formula is returned. Otherwise `NULL`
- `family` `character` description of the observation distribution
- `trend_model` `character` description of the latent trend model

- trend_map data.frame describing the mapping of trend states to observations, if supplied in the original model. Otherwise NULL
- drift Logical specifying whether a drift term was used in the trend model
- priors If the model priors were updated from their defaults, the prior dataframe will be returned. Otherwise NULL
- model_output The MCMC object returned by the fitting engine. If the model was fitted using Stan, this will be an object of class stanfit (see stanfit-class for details). If JAGS was used as the backend, this will be an object of class runjags (see runjags-class for details)
- model_file The character string model file used to describe the model in either Stan or JAGS syntax
- model_data If return_model_data was set to TRUE when fitting the model, the list object containing all data objects needed to condition the model is returned. Each item in the list is described in detail at the top of the model_file. Otherwise NULL
- inits If return_model_data was set to TRUE when fitting the model, the initial value functions used to initialise the MCMC chains will be returned. Otherwise NULL
- monitor_pars The parameters that were monitored during MCMC sampling are returned as a character vector
- sp_names A character vector specifying the names for each smoothing parameter
- mgcv_model An object of class gam containing the mgcv version of the observation model. This object is used for generating the linear predictor matrix when making predictions for new data. The coefficients in this model object will contain the posterior median coefficients from the GAM linear predictor, but these are only used if generating plots of smooth functions that mvgam currently cannot handle (such as plots for three-dimensional smooths). This model therefore should not be used for inference. See gamObject for details
- trend_mgcv_model If a trend_formula was supplied, an object of class gam containing the mgcv version of the trend model. Otherwise NULL
- ytimes The matrix object used in model fitting for indexing which series and timepoints were observed in each row of the supplied data. Used internally by some downstream plotting and prediction functions
- resids A named list object containing posterior draws of Dunn-Smyth randomized quantile residuals
- use_lv Logical flag indicating whether latent dynamic factors were used in the model
- n_lv If use_lv == TRUE, the number of latent dynamic factors used in the model
- upper_bounds If bounds were supplied in the original model fit, they will be returned. Otherwise NULL
- obs_data The original data object (either a list or dataframe) supplied in model fitting.
- test_data If test data were supplied (as argument newdata in the original model), it will be returned. Othwerise NULL
- fit_engine Character describing the fit engine, either as stan or jags
- backend Character describing the backend used for modelling, either as rstan, cmdstanr or rjags
- algorithm Character describing the algorithm used for finding the posterior, either as sampling, laplace, pathfinder, meanfield or fullrank

- max_treedepth If the model was fitted using Stan, the value supplied for the maximum treedepth tuning parameter is returned (see stan for details). Otherwise NULL
- adapt_delta If the model was fitted using Stan, the value supplied for the adapt_delta tuning parameter is returned (see stan for details). Otherwise NULL

Author(s)

Nicholas J Clark

See Also

mvgam

---

mvgam_diagnostics *Extract diagnostic quantities of* **mvgam** *models*

Description

Extract quantities that can be used to diagnose sampling behavior of the algorithms applied by **Stan** at the back-end of **mvgam**.

Usage

```
## S3 method for class 'mvgam'
nuts_params(object, pars = NULL, ...)

## S3 method for class 'mvgam'
log_posterior(object, ...)

## S3 method for class 'mvgam'
rhat(x, pars = NULL, ...)

## S3 method for class 'mvgam'
neff_ratio(object, pars = NULL, ...)
```

Arguments

object, x	A mvgam object.
pars	An optional character vector of parameter names. For nuts_params these will be NUTS sampler parameter names rather than model parameters. If pars is omitted all parameters are included.
...	Arguments passed to individual methods.

Details

For more details see bayesplot-extractors.

Value

The exact form of the output depends on the method.

Examples

```
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)
np <- nuts_params(mod)
head(np)

# extract the number of divergence transitions
sum(subset(np, Parameter == "divergent__")$Value)

head(neff_ratio(mod))
```

---

mvgam_draws *Extract posterior draws from fitted* mvgam *objects*

Description

Extract posterior draws in conventional formats as data.frames, matrices, or arrays.

Usage

```
## S3 method for class 'mvgam'
as.data.frame(
  x,
  row.names = NULL,
  optional = TRUE,
  variable = "betas",
  use_alias = TRUE,
  regex = FALSE,
  ...
)

## S3 method for class 'mvgam'
as.matrix(x, variable = "betas", regex = FALSE, use_alias = TRUE, ...)

## S3 method for class 'mvgam'
as.array(x, variable = "betas", regex = FALSE, use_alias = TRUE, ...)

## S3 method for class 'mvgam'
as_draws(
```

```
  x,
  variable = NULL,
  regex = FALSE,
  inc_warmup = FALSE,
  use_alias = TRUE,
  ...
)

## S3 method for class 'mvgam'
as_draws_matrix(
  x,
  variable = NULL,
  regex = FALSE,
  inc_warmup = FALSE,
  use_alias = TRUE,
  ...
)

## S3 method for class 'mvgam'
as_draws_df(
  x,
  variable = NULL,
  regex = FALSE,
  inc_warmup = FALSE,
  use_alias = TRUE,
  ...
)

## S3 method for class 'mvgam'
as_draws_array(
  x,
  variable = NULL,
  regex = FALSE,
  inc_warmup = FALSE,
  use_alias = TRUE,
  ...
)

## S3 method for class 'mvgam'
as_draws_list(
  x,
  variable = NULL,
  regex = FALSE,
  inc_warmup = FALSE,
  use_alias = TRUE,
  ...
)
```

```
## S3 method for class 'mvgam'
as_draws_rvars(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)
```

Arguments

x	list object of class mvgam
row.names	Ignored
optional	Ignored
variable	A character specifying which parameters to extract. Can either be one of the following options: • obs_params (other parameters specific to the observation model, such as overdispsersions for negative binomial models or observation error SD for gaussian / student-t models) • betas (beta coefficients from the GAM observation model linear predictor; default) • smooth_params (smoothing parameters from the GAM observation model) • linpreds (estimated linear predictors on whatever link scale was used in the model) • trend_params (parameters governing the trend dynamics, such as AR parameters, trend SD parameters or Gaussian Process parameters) • trend_betas (beta coefficients from the GAM latent process model linear predictor; only available if a trend_formula was supplied in the original model) • trend_smooth_params (process model GAM smoothing parameters; only available if a trend_formula was supplied in the original model) • trend_linpreds (process model linear predictors on the identity scale; only available if a trend_formula was supplied in the original model) OR can be a character vector providing the variables to extract
use_alias	Logical. If more informative names for parameters are available (i.e. for beta coefficients b or for smoothing parameters rho), replace the uninformative names with the more informative alias. Defaults to TRUE
regex	Logical. If not using one of the prespecified options for extractions, should variable be treated as a (vector of) regular expressions? Any variable in x matching at least one of the regular expressions will be selected. Defaults to FALSE.
...	Ignored
inc_warmup	Should warmup draws be included? Defaults to FALSE.

Value

A data.frame, matrix, or array containing the posterior draws.

Examples

```
## Not run:
sim <- sim_mvgam(family = Gamma())
mod1 <- mvgam(y ~ s(season, bs = 'cc'),
              trend_model = 'AR1',
              data = sim$data_train,
              family = Gamma(),
              chains = 2,
              samples = 300)
beta_draws_df <- as.data.frame(mod1, variable = 'betas')
head(beta_draws_df)
str(beta_draws_df)

beta_draws_mat <- as.matrix(mod1, variable = 'betas')
head(beta_draws_mat)
str(beta_draws_mat)

shape_pars <- as.matrix(mod1, variable = 'shape', regex = TRUE)
head(shape_pars)
## End(Not run)
```

---

mvgam_families *Supported mvgam families*

Description

Supported mvgam families

Usage

```
tweedie(link = "log")

student_t(link = "identity")

betar(...)

nb(...)

lognormal(...)

student(...)

bernoulli(...)

beta_binomial(...)

nmix(link = "log")
```

Arguments

link	a specification for the family link function. At present these cannot be changed
...	Arguments to be passed to the **mgcv** version of the associated functions

Details

mvgam currently supports the following standard observation families:

- gaussian with identity link, for real-valued data
- poisson with log-link, for count data
- Gamma with log-link, for non-negative real-valued data
- binomial with logit-link, for count data when the number of trials is known (and must be supplied)

In addition, the following extended families from the mgcv and brms packages are supported:

- betar with logit-link, for proportional data on (0,1)
- nb with log-link, for count data
- lognormal with identity-link, for non-negative real-valued data
- bernoulli with logit-link, for binary data
- beta_binomial with logit-link, as for binomial() but allows for overdispersion

Finally, mvgam supports the three extended families described here:

- tweedie with log-link, for count data (power parameter p fixed at 1.5)
- student_t() (or student) with identity-link, for real-valued data
- nmix for count data with imperfect detection modeled via a State-Space N-Mixture model. The latent states are Poisson (with log link), capturing the 'true' latent abundance, while the observation process is Binomial to account for imperfect detection. The observation formula in these models is used to set up a linear predictor for the detection probability (with logit link). See the example below for a more detailed worked explanation of the nmix() family

Only poisson(), nb(), and tweedie() are available if using JAGS. All families, apart from tweedie(), are supported if using Stan.

Note that currently it is not possible to change the default link functions in mvgam, so any call to change these will be silently ignored

Value

Objects of class family

Author(s)

Nicholas J Clark

Examples

```
# Example showing how to set up N-mixture models
set.seed(999)
# Simulate observations for species 1, which shows a declining trend and 0.7 detection probability
data.frame(site = 1,
          # five replicates per year; six years
          replicate = rep(1:5, 6),
          time = sort(rep(1:6, 5)),
          species = 'sp_1',
          # true abundance declines nonlinearly
          truth = c(rep(28, 5),
                    rep(26, 5),
                    rep(23, 5),
                    rep(16, 5),
                    rep(14, 5),
                    rep(14, 5)),
          # observations are taken with detection prob = 0.7
          obs = c(rbinom(5, 28, 0.7),
                  rbinom(5, 26, 0.7),
                  rbinom(5, 23, 0.7),
                  rbinom(5, 15, 0.7),
                  rbinom(5, 14, 0.7),
                  rbinom(5, 14, 0.7))) %>%
 # add 'series' information, which is an identifier of site, replicate and species
 dplyr::mutate(series = paste0('site_', site,
                               '_', species,
                               '_rep_', replicate),
               time = as.numeric(time),
               # add a 'cap' variable that defines the maximum latent N to
               # marginalize over when estimating latent abundance; in other words
               # how large do we realistically think the true abundance could be?
               cap = 80) %>%
 dplyr::select(- replicate) -> testdat

# Now add another species that has a different temporal trend and a smaller
# detection probability (0.45 for this species)
testdat = testdat %>%
 dplyr::bind_rows(data.frame(site = 1,
                             replicate = rep(1:5, 6),
                             time = sort(rep(1:6, 5)),
                             species = 'sp_2',
                             truth = c(rep(4, 5),
                                       rep(7, 5),
                                       rep(15, 5),
                                       rep(16, 5),
                                       rep(19, 5),
                                       rep(18, 5)),
                             obs = c(rbinom(5, 4, 0.45),
                                     rbinom(5, 7, 0.45),
                                     rbinom(5, 15, 0.45),
                                     rbinom(5, 16, 0.45),
                                     rbinom(5, 19, 0.45),
```

```
                                  rbinom(5, 18, 0.45))) %>%
                    dplyr::mutate(series = paste0('site_', site,
                                                  '_', species,
                                                  '_rep_', replicate),
                                  time = as.numeric(time),
                                  cap = 50) %>%
                    dplyr::select(-replicate))

# series identifiers
testdat$species <- factor(testdat$species,
                          levels = unique(testdat$species))
testdat$series <- factor(testdat$series,
                         levels = unique(testdat$series))

# The trend_map to state how replicates are structured
testdat %>%
# each unique combination of site*species is a separate process
dplyr::mutate(trend = as.numeric(factor(paste0(site, species)))) %>%
 dplyr::select(trend, series) %>%
 dplyr::distinct() -> trend_map
trend_map

# Fit a model
mod <- mvgam(
            # the observation formula sets up linear predictors for
            # detection probability on the logit scale
            formula = obs ~ species - 1,

            # the trend_formula sets up the linear predictors for
            # the latent abundance processes on the log scale
            trend_formula = ~ s(time, by = trend, k = 4) + species,

            # the trend_map takes care of the mapping
            trend_map = trend_map,

            # nmix() family and data
            family = nmix(),
            data = testdat,

            # priors can be set in the usual way
            priors = c(prior(std_normal(), class = b),
                       prior(normal(1, 1.5), class = Intercept_trend)),
            chains = 2)

# The usual diagnostics
summary(mod)

# Plotting conditional effects
library(ggplot2); library(marginaleffects)
plot_predictions(mod, condition = 'species',
                 type = 'detection') +
     ylab('Pr(detection)') +
     ylim(c(0, 1)) +
```

```
    theme_classic() +
    theme(legend.position = 'none')

# Example showcasing how cbind() is needed for Binomial observations
# Simulate two time series of Binomial trials
trials <- sample(c(20:25), 50, replace = TRUE)
x <- rnorm(50)
detprob1 <- plogis(-0.5 + 0.9*x)
detprob2 <- plogis(-0.1 -0.7*x)
dat <- rbind(data.frame(y = rbinom(n = 50, size = trials, prob = detprob1),
                        time = 1:50,
                        series = 'series1',
                        x = x,
                        ntrials = trials),
             data.frame(y = rbinom(n = 50, size = trials, prob = detprob2),
                        time = 1:50,
                        series = 'series2',
                        x = x,
                        ntrials = trials))
dat <- dplyr::mutate(dat, series = as.factor(series))
dat <- dplyr::arrange(dat, time, series)

# Fit a model using the binomial() family; must specify observations
# and number of trials in the cbind() wrapper
mod <- mvgam(cbind(y, ntrials) ~ series + s(x, by = series),
             family = binomial(),
             data = dat)
summary(mod)
```

`mvgam_forecast-class` `mvgam_forecast` *object description*

Description

A `mvgam_forecast` object returned by function `hindcast` or `forecast`. Run `methods(class = "mvgam_forecast")` to see an overview of available methods.

Details

A `mvgam_forecast` object contains the following elements:

- `call` the original observation model formula
- `trend_call` If a `trend_formula` was supplied, the original trend model formula is returned. Otherwise `NULL`
- `family` `character` description of the observation distribution
- `family_pars` `list` containing draws of family-specific parameters (i.e. shape, scale or overdispersion parameters). Only returned if `type = link`. Otherwise `NULL`
- `trend_model` `character` description of the latent trend model

- drift Logical specifying whether a drift term was used in the trend model
- use_lv Logical flag indicating whether latent dynamic factors were used in the model
- fit_engine Character describing the fit engine, either as stan or jags
- type The type of predictions included (either link, response or trend)
- series_names Names of the time series, taken from levels(data$series) in the original model fit
- train_observations A list of training observation vectors of length n_series
- train_times A vector of the unique training times
- test_observations If the forecast function was used, a list of test observation vectors of length n_series. Otherwise NULL
- test_times If the forecast function was used, a vector of the unique validation (testing) times. Otherwise NULL
- hindcasts A list of posterior hindcast distributions of length n_series.
- forecasts If the forecast function was used, a list of posterior forecast distributions of length n_series. Otherwise NULL

Author(s)

Nicholas J Clark

See Also

mvgam, hindcast.mvgam, forecast.mvgam

---

mvgam_formulae *Details of formula specifications in* mvgam

Description

Details of formula specifications in mvgam

Details

mvgam will accept an observation model formula and an optional process model formula (via the argument trend_formula). Neither of these formulae can be specified as lists, contrary to the accepted behaviour in some mgcv or brms models.

Note that it is possible to supply an empty formula where there are no predictors or intercepts in the observation model (i.e. y ~ 0 or y ~ -1). In this case, an intercept-only observation model will be set up but the intercept coefficient will be fixed at zero. This can be handy if you wish to fit pure State-Space models where the variation in the dynamic trend controls the average expectation, and/or where intercepts are non-identifiable.

The formulae supplied to mvgam are exactly like those supplied to glm except that smooth terms,

s, te, ti and t2, time-varying effects using dynamic, monotonically increasing (using s(x, bs = 'moi')) or decreasing splines (using s(x, bs = 'mod'); see smooth.construct.moi.smooth.spec for details), as well as Gaussian Process functions using gp, can be added to the right hand side (and . is not supported in mvgam formulae).

Further details on specifying different kinds of smooth functions, and how to control their behaviours by modifying their potential complexities and / or how the penalties behave, can be found in the extensive documentation for the mgcv package.

Author(s)

Nicholas J Clark

See Also

mvgam, formula.gam, gam.models, jagam, gam, s, gp, formula

---

mvgam_irf-class mvgam_irf *object description*

Description

A mvgam_irf object returned by function irf. Run methods(class = "mvgam_irf") to see an overview of available methods.

Details

A mvgam_irf object contains a list of posterior IRFs, each stored as its own list

Author(s)

Nicholas J Clark

See Also

mvgam, VAR

mvgam_marginaleffects *Helper functions for mvgam marginaleffects calculations*

Description

Helper functions for mvgam marginaleffects calculations

Functions needed for working with marginaleffects

Functions needed for getting data / objects with insight

Usage

```
## S3 method for class 'mvgam'
get_coef(model, trend_effects = FALSE, ...)

## S3 method for class 'mvgam'
set_coef(model, coefs, trend_effects = FALSE, ...)

## S3 method for class 'mvgam'
get_vcov(model, vcov = NULL, ...)

## S3 method for class 'mvgam'
get_predict(model, newdata, type = "response", process_error = FALSE, ...)

## S3 method for class 'mvgam'
get_data(x, source = "environment", verbose = TRUE, ...)

## S3 method for class 'mvgam_prefit'
get_data(x, source = "environment", verbose = TRUE, ...)

## S3 method for class 'mvgam'
find_predictors(
  x,
  effects = c("fixed", "random", "all"),
 component = c("all", "conditional", "zi", "zero_inflated", "dispersion", "instruments",
    "correlation", "smooth_terms"),
  flatten = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'mvgam_prefit'
find_predictors(
  x,
  effects = c("fixed", "random", "all"),
 component = c("all", "conditional", "zi", "zero_inflated", "dispersion", "instruments",
    "correlation", "smooth_terms"),
```

```
  flatten = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

model	Model object
trend_effects	logical, extract from the process model component (only applicable if a trend_formula was specified in the model)
...	Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?slopes documentation for a non-exhaustive list of available arguments.
coefs	vector of coefficients to insert in the model object
vcov	Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values: • FALSE: Do not compute standard errors. This can speed up computation considerably. • TRUE: Unit-level standard errors using the default vcov(model) variance-covariance matrix. • String which indicates the kind of uncertainty estimates to return. – Heteroskedasticity-consistent: "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC – Heteroskedasticity and autocorrelation consistent: "HAC" – Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger" – Other: "NeweyWest", "KernHAC", "OPG". See the sandwich package documentation. • One-sided formula which indicates the name of cluster variables (e.g., ~unit_id). This formula is passed to the cluster argument of the sandwich::vcovCL function. • Square covariance matrix • Function which returns a covariance matrix (e.g., stats::vcov(model))
newdata	Grid of predictor values at which we evaluate the slopes. • Warning: Please avoid modifying your dataset between fitting the model and calling a marginaleffects function. This can sometimes lead to unexpected results. • NULL (default): Unit-level slopes for each observed value in the dataset (empirical distribution). The dataset is retrieved using insight::get_data(), which tries to extract data from the environment. This may produce unexpected results if the original data frame has been altered since fitting the model.

- `datagrid()` call to specify a custom grid of regressors. For example:
  - `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes.
  - See the Examples section and the `datagrid()` documentation.
- `subset()` call with a single argument to select a subset of the dataset used to fit the model, ex: `newdata = subset(treatment == 1)`
- `dplyr::filter()` call with a single argument to select a subset of the dataset used to fit the model, ex: `newdata = filter(treatment == 1)`
- string:
  - "mean": Marginal Effects at the Mean. Slopes when each predictor is held at its mean or mode.
  - "median": Marginal Effects at the Median. Slopes when each predictor is held at its median or mode.
  - "marginalmeans": Marginal Effects at Marginal Means. See Details section below.
  - "tukey": Marginal Effects at Tukey's 5 numbers.
  - "grid": Marginal Effects on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the first entry in the error message is used by default.

`process_error` `logical`. If `TRUE`, uncertainty in the latent process (or trend) model is incorporated in predictions

`x` A fitted model.

`source` String, indicating from where data should be recovered. If `source = "environment"` (default), data is recovered from the environment (e.g. if the data is in the workspace). This option is usually the fastest way of getting data and ensures that the original variables used for model fitting are returned. Note that always the *current* data is recovered from the environment. Hence, if the data was modified *after* model fitting (e.g., variables were recoded or rows filtered), the returned data may no longer equal the model data. If `source = "frame"` (or `"mf"`), the data is taken from the model frame. Any transformed variables are back-transformed, if possible. This option returns the data even if it is not available in the environment, however, in certain edge cases back-transforming to the original data may fail. If `source = "environment"` fails to recover the data, it tries to extract the data from the model frame; if `source = "frame"` and data cannot be extracted from the model frame, data will be recovered from the environment. Both ways only returns observations that have no missing data in the variables used for model fitting.

`verbose` Toggle messages and warnings.

`effects` Should model data for fixed effects (`"fixed"`), random effects (`"random"`) or both (`"all"`) be returned? Only applies to mixed or gee models.

component	Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the *conditional* component is also called *count* or *mean* component, depending on the model.
flatten	Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

Value

Objects suitable for internal 'marginaleffects' functions to proceed. See `marginaleffects::get_coef()`, `marginaleffects::set_coef()`, `marginaleffects::get_vcov()`, `marginaleffects::get_predict()`, `insight::get_data()` and `insight::find_predictors()` for details

Author(s)

Nicholas J Clark

`mvgam_trends` *Supported mvgam trend models*

Description

Supported mvgam trend models

Details

`mvgam` currently supports the following dynamic trend models:

- `None` (no latent trend component; i.e. the GAM component is all that contributes to the linear predictor, and the observation process is the only source of error; similarly to what is estimated by `gam`)
- `RW()`
- `AR(p = 1, 2, or 3)`
- `CAR(p = 1)`(continuous time autoregressive trends; only available in `Stan`)
- `VAR()`(only available in `Stan`)
- `PW()` (piecewise linear or logistic trends; only available in `Stan`)
- `GP()` (Gaussian Process with squared exponential kernel; only available in `Stan`)

For most dynamic trend types available in `mvgam` (see argument `trend_model`), time should be measured in discrete, regularly spaced intervals (i.e. `c(1, 2, 3, ...)`). However you can use irregularly spaced intervals if using `trend_model = CAR(1)`, though note that any temporal intervals that are exactly `0` will be adjusted to a very small number (`1e-12`) to prevent sampling errors. For all trend types apart from `GP()`, `PW()`, and `CAR()`, moving average and/or correlated process error terms can also be estimated (for example, `RW(cor = TRUE)` will set up a multivariate Random Walk if `data` contains `>1` series). Character strings can also be supplied instead of the various trend functions. The full list of possible models that are currently supported is:

- 'RW'
- 'RWMA'
- 'RWcor'
- 'RWMAcor'
- 'AR1'
- 'AR1MA'
- 'AR1cor'
- 'AR1MAcor'
- 'AR2'
- 'AR2MA'
- 'AR2cor'
- 'AR2MAcor'
- 'AR3'
- 'AR3MA'
- 'AR3cor'
- 'AR3MAcor'
- 'CAR1'
- 'VAR'
- 'VARcor'
- 'VAR1' (same as 'VAR')
- 'VAR1cor' (same as 'VARcor')
- 'VARMA'
- 'VARMAcor'
- 'VARMA1,1cor'
- 'PWlinear'
- 'PWlogistic'
- 'GP'
- 'None'

Note that only `RW`, `AR1`, `AR2` and `AR3` are available if using `JAGS`. All trend models are supported if using `Stan`. Dynamic factor models can be used in which the latent factors evolve as either `RW`, `AR1-3`, `VAR` or `GP`. For `VAR` models (i.e. `VAR` and `VARcor` models), users can either fix the trend error covariances to be `0` (using `VAR`) or estimate them and potentially allow for contemporaneously correlated errors using `VARcor`. For all `VAR` models, stationarity of the latent process is enforced through the prior using the parameterisation given by Heaps (2022). Stationarity is not enforced when using `AR1`, `AR2` or `AR3` models, though this can be changed by the user by specifying lower and upper bounds on autoregressive parameters using functionality in get_mvgam_priors and the `priors` argument in mvgam. Piecewise trends follow the formulation in the popular `prophet` package produced by `Facebook`, where users can allow for changepoints to control the potential flexibility of the trend. See Taylor and Letham (2018) for details

References

Sarah E. Heaps (2022) Enforcing stationarity through the prior in Vector Autoregressions. Journal of Computational and Graphical Statistics. 32:1, 1-10.

Sean J. Taylor and Benjamin Letham (2018) Forecasting at scale. The American Statistician 72.1, 37-45.

See Also

RW, AR, CAR, VAR, PW, GP

`pairs.mvgam` *Create a matrix of output plots from a* `mvgam` *object*

Description

A `pairs` method that is customized for MCMC output.

Usage

```
## S3 method for class 'mvgam'
pairs(x, variable = NULL, regex = FALSE, use_alias = TRUE, ...)
```

Arguments

`x`	An object of class `mvgam`
`variable`	Names of the variables (parameters) to plot, as given by a character vector or a regular expression (if `regex = TRUE`). By default, a hopefully not too large selection of variables is plotted.
`regex`	Logical; Indicates whether `variable` should be treated as regular expressions. Defaults to `FALSE`.
`use_alias`	Logical. If more informative names for parameters are available (i.e. for beta coefficients `b` or for smoothing parameters `rho`), replace the uninformative names with the more informative alias. Defaults to `TRUE`
`...`	Further arguments to be passed to `mcmc_pairs`.

Details

For a detailed description see `mcmc_pairs`.

Value

Plottable objects whose classes depend on the arguments supplied. See `mcmc_pairs` for details.

Examples

```
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)
pairs(mod)
pairs(mod, variable = c('ar1', 'sigma'), regex = TRUE)
```

`plot.mvgam` *Default mvgam plots*

Description

This function takes a fitted `mvgam` object and produces plots of smooth functions, forecasts, trends and uncertainty components

Usage

```
## S3 method for class 'mvgam'
plot(
  x,
  type = "residuals",
  series = 1,
  residuals = FALSE,
  newdata,
  data_test,
  trend_effects = FALSE,
  ...
)
```

Arguments

`x`	`list` object returned from `mvgam`. See `mvgam()`
`type`	`character` specifying which type of plot to return. Options are: series, residuals, smooths, re (random effect smooths), pterms (parametric effects), forecast, trend, uncertainty, factors
`series`	`integer` specifying which series in the set is to be plotted. This is ignored if `type == 're'`
`residuals`	`logical`. If `TRUE` and `type = 'smooths'`, posterior quantiles of partial residuals are added to plots of 1-D smooths as a series of ribbon rectangles. Partial residuals for a smooth term are the median Dunn-Smyth residuals that would be obtained by dropping the term concerned from the model, while leaving all other

	estimates fixed (i.e. the estimates for the term plus the original median Dunn-Smyth residuals). Note that because mvgam works with Dunn-Smyth residuals and not working residuals, which are used by mgcv, the magnitudes of partial residuals will be different to what you would expect from plot.gam. Interpretation is similar though, as these partial residuals should be evenly scattered around the smooth function if the function is well estimated
newdata	Optional dataframe or list of test data containing at least 'series' and 'time' in addition to any other variables included in the linear predictor of the original formula. This argument is optional when plotting out of sample forecast period observations (when type = forecast) and required when plotting uncertainty components (type = uncertainty).
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
trend_effects	logical. If TRUE and a trend_formula was used in model fitting, terms from the trend (i.e. process) model will be plotted
...	Additional arguments for each individual plotting function.

Details

These plots are useful for getting an overview of the fitted model and its estimated random effects or smooth functions, but the individual plotting functions and the functions from the marginaleffects and gratia packages offer far more more customisation.

Value

A base R plot or set of plots

Author(s)

Nicholas J Clark

See Also

plot_mvgam_resids, plot_mvgam_smooth, plot_mvgam_fc, plot_mvgam_trend, plot_mvgam_uncertainty, plot_mvgam_factors, plot_mvgam_randomeffects, conditional_effects.mvgam, plot_predictions, plot_slopes, gratia_mvgam_enhancements

Examples

```
# Simulate some time series
dat <- sim_mvgam(T = 80, n_series = 3)

# Fit a basic model
mod <- mvgam(y ~ s(season, bs = 'cc') + s(series, bs = 're'),
             data = dat$data_train,
             trend_model = RW(),
             chains = 2)

# Plot predictions and residuals for each series
```

```
plot(mod, type = 'forecast', series = 1)
plot(mod, type = 'forecast', series = 2)
plot(mod, type = 'forecast', series = 3)
plot(mod, type = 'residuals', series = 1)
plot(mod, type = 'residuals', series = 2)
plot(mod, type = 'residuals', series = 3)

# Plot model effects
plot(mod, type = 'smooths')
plot(mod, type = 're')

# More flexible plots with 'marginaleffects' utilities
library(marginaleffects)
plot_predictions(mod, condition = 'season', type = 'link')
plot_predictions(mod,
                 condition = c('season', 'series', 'series'),
                 type = 'link')
plot_predictions(mod, condition = 'series', type = 'link')

# When using a State-Space model with predictors on the process
# model, set trend_effects = TRUE to visualise process effects
mod <- mvgam(y ~ -1,
             trend_formula = ~ s(season, bs = 'cc'),
             data = dat$data_train,
             trend_model = RW(),
             chains = 2)
plot(mod, type = 'smooths', trend_effects = TRUE)

# But marginaleffects functions work without any modification
plot_predictions(mod, condition = 'season', type = 'link')
```

`plot.mvgam_irf` *Plot impulse responses from an mvgam_irf object This function takes an* `mvgam_irf` *object and produces plots of Impulse Response Functions*

Description

Plot impulse responses from an mvgam_irf object This function takes an `mvgam_irf` object and produces plots of Impulse Response Functions

Usage

```
## S3 method for class 'mvgam_irf'
plot(x, series = 1, ...)
```

Arguments

x	list object of class mvgam_irf. See irf()
series	integer specifying which process series should be given the shock
...	ignored

Value

A base R plot or set of plots

Author(s)

Nicholas J Clark

---

plot.mvgam_lfo *Plot Pareto-k and ELPD values from a leave-future-out object*

Description

This function takes an object of class mvgam_lfo and creates several informative diagnostic plots

Usage

```
## S3 method for class 'mvgam_lfo'
plot(x, ...)
```

Arguments

x	An object of class mvgam_lfo
...	Ignored

Value

A base R plot of Pareto-k and ELPD values over the evaluation timepoints. For the Pareto-k plot, a dashed red line indicates the specified threshold chosen for triggering model refits. For the ELPD plot, a dashed red line indicates the bottom 10% quantile of ELPD values. Points below this threshold may represent outliers that were more difficult to forecast

plot_mvgam_factors *Latent factor summaries for a fitted mvgam object*

Description

This function takes a fitted mvgam object and returns plots and summary statistics for the latent dynamic factors

Usage

```
plot_mvgam_factors(object, plot = TRUE)
```

Arguments

object	list object returned from mvgam. See mvgam()
plot	logical specifying whether factors should be plotted

Details

If the model in object was estimated using dynamic factors, it is possible that not all factors contributed to the estimated trends. This is due to the regularisation penalty that acts independently on each factor's Gaussian precision, which will squeeze un-needed factors to a white noise process (effectively dropping that factor from the model). In this function, each factor is tested against a null hypothesis of white noise by calculating the sum of the factor's 2nd derivatives. A factor that has a larger contribution will have a larger sum due to the weaker penalty on the factor's precision. If plot == TRUE, the factors are also plotted.

Value

A dataframe of factor contributions and, optionally, a series of base R plots

Author(s)

Nicholas J Clark

Examples

```
simdat <- sim_mvgam()
mod <- mvgam(y ~ s(season, bs = 'cc',
                   k = 6),
             trend_model = AR(),
             use_lv = TRUE,
             n_lv = 2,
             data = simdat$data_train,
             chains = 2)
plot_mvgam_factors(mod)
```

`plot_mvgam_forecasts` *Plot mvgam posterior predictions for a specified series*

Description

Plot mvgam posterior predictions for a specified series

Usage

```
plot_mvgam_fc(
  object,
  series = 1,
  newdata,
  data_test,
  realisations = FALSE,
  n_realisations = 15,
  hide_xlabels = FALSE,
  xlab,
  ylab,
  ylim,
  n_cores = 1,
  return_forecasts = FALSE,
  return_score = FALSE,
  ...
)

## S3 method for class 'mvgam_forecast'
plot(
  x,
  series = 1,
  realisations = FALSE,
  n_realisations = 15,
  hide_xlabels = FALSE,
  xlab,
  ylab,
  ylim,
  return_score = FALSE,
  ...
)
```

Arguments

`object`	`list` object returned from `mvgam`. See `mvgam()`
`series`	`integer` specifying which series in the set is to be plotted
`newdata`	Optional `dataframe` or `list` of test data containing at least 'series' and 'time' in addition to any other variables included in the linear predictor of the original `formula`. If included, the covariate information in `newdata` will be used to

	generate forecasts from the fitted model equations. If this same newdata was originally included in the call to mvgam, then forecasts have already been produced by the generative model and these will simply be extracted and plotted. However if no newdata was supplied to the original model call, an assumption is made that the newdata supplied here comes sequentially after the data supplied as data in the original model (i.e. we assume there is no time gap between the last observation of series 1 in data and the first observation for series 1 in newdata). If newdata contains observations in column y, these observations will be used to compute a Discrete Rank Probability Score for the forecast distribution
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
realisations	logical. If TRUE, forecast realisations are shown as a spaghetti plot, making it easier to visualise the diversity of possible forecasts. If FALSE, the default, empirical quantiles of the forecast distribution are shown
n_realisations	integer specifying the number of posterior realisations to plot, if realisations = TRUE. Ignored otherwise
hide_xlabels	logical. If TRUE, no xlabels are printed to allow the user to add custom labels using axis from base R
xlab	label for x axis.
ylab	label for y axis.
ylim	Optional vector of y-axis limits (min, max)
n_cores	integer specifying number of cores for generating forecasts in parallel
return_forecasts	logical. If TRUE, the function will plot the forecast as well as returning the forecast object (as a matrix of dimension n_samples x horizon)
return_score	logical. If TRUE and out of sample test data is provided as newdata, a probabilistic score will be calculated and returned. The score used will depend on the observation family from the fitted model. Discrete families (poisson, negative binomial, tweedie) use the Discrete Rank Probability Score. Other families use the Continuous Rank Probability Score. The value returned is the sum of all scores within the out of sample forecast horizon
...	further par graphical parameters.
x	Object of class mvgam_forecast

Details

plot_mvgam_fc generates posterior predictions from an object of class mvgam, calculates posterior empirical quantiles and plots them against the observed data. If realisations = FALSE, the returned plot shows 90, 60, 40 and 20 percent posterior quantiles (as ribbons of increasingly darker shades or red) as well as the posterior median (as a dark red line). If realisations = FALSE, a set of n_realisations posterior draws are shown.

plot.mvgam_forecast takes an object of class mvgam_forecast, in which forecasts have already been computed, and plots the resulting forecast distribution.

If realisations = FALSE, these posterior quantiles are plotted along with the true observed data that was used to train the model. Otherwise, a spaghetti plot is returned to show possible forecast paths.

Value

A base R graphics plot and an optional list containing the forecast distribution and the out of sample probabilistic forecast score

---

plot_mvgam_pterms *Plot mvgam parametric term partial effects*

Description

This function plots posterior empirical quantiles for partial effects of parametric terms

Usage

```
plot_mvgam_pterms(object, trend_effects = FALSE)
```

Arguments

object	list object returned from mvgam. See mvgam()
trend_effects	logical. If TRUE and a trend_formula was used in model fitting, terms from the trend (i.e. process) model will be plotted

Details

Posterior empirical quantiles of each parametric term's partial effect estimates (on the link scale) are calculated and visualised as ribbon plots. These effects can be interpreted as the partial effect that a parametric term contributes when all other terms in the model have been set to 0

Value

A base R graphics plot

plot_mvgam_randomeffects *Plot mvgam random effect terms*

Description

This function plots posterior empirical quantiles for random effect smooths (bs = re)

Usage

```
plot_mvgam_randomeffects(object, trend_effects = FALSE)
```

Arguments

object	list object returned from mvgam. See mvgam()
trend_effects	logical. If TRUE and a trend_formula was used in model fitting, terms from the trend (i.e. process) model will be plotted

Details

Posterior empirical quantiles of random effect coefficient estimates (on the link scale) are calculated and visualised as ribbon plots. Labels for coefficients are taken from the levels of the original factor variable that was used to specify the smooth in the model's formula

Value

A base R graphics plot

plot_mvgam_resids *Residual diagnostics for a fitted mvgam object*

Description

This function takes a fitted mvgam object and returns various residual diagnostic plots

Usage

```
plot_mvgam_resids(object, series = 1, newdata, data_test)
```

Arguments

object	list object returned from mvgam. See mvgam()
series	integer specifying which series in the set is to be plotted
newdata	Optional dataframe or list of test data containing at least 'series', 'y', and 'time' in addition to any other variables included in the linear predictor of formula. If included, the covariate information in newdata will be used to generate forecasts from the fitted model equations. If this same newdata was originally included in the call to mvgam, then forecasts have already been produced by the generative model and these will simply be extracted and used to calculate residuals. However if no newdata was supplied to the original model call, an assumption is made that the newdata supplied here comes sequentially after the data supplied as data in the original model (i.e. we assume there is no time gap between the last observation of series 1 in data_train and the first observation for series 1 in newdata).
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows

Details

A total of four base R plots are generated to examine Dunn-Smyth residuals for the specified series. Plots include a residuals vs fitted values plot, a Q-Q plot, and two plots to check for any remaining temporal autocorrelation in the residuals. Note, all plots use posterior medians of fitted values / residuals, so uncertainty is not represented.

Value

A series of base R plots

Author(s)

Nicholas J Clark

---

plot_mvgam_series *Plot observed time series used for mvgam modelling*

Description

This function takes either a fitted mvgam object or a data_train object and produces plots of observed time series, ACF, CDF and histograms for exploratory data analysis

Usage

```
plot_mvgam_series(
  object,
  data,
  data_train,
```

```
  newdata,
  data_test,
  y = "y",
  lines = TRUE,
  series = 1,
  n_bins,
  log_scale = FALSE
)
```

Arguments

object	Optional list object returned from mvgam. Either object or data_train must be supplied.
data	Optional dataframe or list of training data containing at least 'series' and 'time'. Use this argument if training data have been gathered in the correct format for mvgam modelling but no model has yet been fitted.
data_train	Deprecated. Still works in place of data but users are recommended to use data instead for more seamless integration into R workflows
newdata	Optional dataframe or list of test data containing at least 'series' and 'time' for the forecast horizon, in addition to any other variables included in the linear predictor of formula. If included, the observed values in the test data are compared to the model's forecast distribution for exploring biases in model predictions.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
y	Character. What is the name of the outcome variable in the supplied data? Defaults to 'y'
lines	Logical. If TRUE, line plots are used for visualising time series. If FALSE, points are used.
series	Either a integer specifying which series in the set is to be plotted or the string 'all', which plots all series available in the supplied data
n_bins	integer specifying the number of bins to use for binning observed values when plotting a the histogram. Default is to use the number of bins returned by a call to hist in base R
log_scale	logical. If series == 'all', this flag is used to control whether the time series plot is shown on the log scale (using log(Y + 1)). This can be useful when visualising many series that may have different observed ranges. Default is FALSE

Value

A set of base R graphics plots. If series is an integer, the plots will show observed time series, autocorrelation and cumulative distribution functions, and a histogram for the series. If series == 'all', a set of observed time series plots is returned in which all series are shown on each plot but only a single focal series is highlighted, with all remaining series shown as faint gray lines.

Author(s)

Nicholas J Clark

Examples

```
# Simulate and plot series with observations bounded at 0 and 1 (Beta responses)
sim_data <- sim_mvgam(family = betar(),
                      trend_model = RW(), prop_trend = 0.6)
plot_mvgam_series(data = sim_data$data_train, series = 'all')
plot_mvgam_series(data = sim_data$data_train,
                  newdata = sim_data$data_test, series = 1)

# Now simulate series with overdispersed discrete observations
sim_data <- sim_mvgam(family = nb(), trend_model = RW(),
                      prop_trend = 0.6, phi = 10)
plot_mvgam_series(data = sim_data$data_train, series = 'all')
```

---

plot_mvgam_smooth *Plot mvgam smooth terms*

Description

This function plots posterior empirical quantiles for a series-specific smooth term

Usage

```
plot_mvgam_smooth(
  object,
  trend_effects = FALSE,
  series = 1,
  smooth,
  residuals = FALSE,
  n_resid_bins = 25,
  realisations = FALSE,
  n_realisations = 15,
  derivatives = FALSE,
  newdata
)
```

Arguments

object	list object returned from mvgam. See mvgam()
trend_effects	logical. If TRUE and a trend_formula was used in model fitting, terms from the trend (i.e. process) model will be plotted
series	integer specifying which series in the set is to be plotted
smooth	either a character or integer specifying which smooth term to be plotted

residuals	logical. If TRUE then posterior quantiles of partial residuals are added to plots of 1-D smooths as a series of ribbon rectangles. Partial residuals for a smooth term are the median Dunn-Smyth residuals that would be obtained by dropping the term concerned from the model, while leaving all other estimates fixed (i.e. the estimates for the term plus the original median Dunn-Smyth residuals). Note that because mvgam works with Dunn-Smyth residuals and not working residuals, which are used by mgcv, the magnitudes of partial residuals will be different to what you would expect from plot.gam. Interpretation is similar though, as these partial residuals should be evenly scattered around the smooth function if the function is well estimated
n_resid_bins	integer specifying the number of bins group the covariate into when plotting partial residuals. Setting this argument too high can make for messy plots that are difficult to interpret, while setting it too low will likely mask some potentially useful patterns in the partial residuals. Default is 25
realisations	logical. If TRUE, posterior realisations are shown as a spaghetti plot, making it easier to visualise the diversity of possible functions. If FALSE, the default, empirical quantiles of the posterior distribution are shown
n_realisations	integer specifying the number of posterior realisations to plot, if realisations = TRUE. Ignored otherwise
derivatives	logical. If TRUE, an additional plot will be returned to show the estimated 1st derivative for the specified smooth (Note, this only works for univariate smooths)
newdata	Optional dataframe for predicting the smooth, containing at least 'series' in addition to any other variables included in the linear predictor of the original model's formula. Note that this currently is only supported for plotting univariate smooths

Details

Smooth functions are shown as empirical quantiles (or spaghetti plots) of posterior partial expectations across a sequence of values between the variable's min and max, while zeroing out effects of all other variables. At present, only univariate and bivariate smooth plots are allowed, though note that bivariate smooths rely on default behaviour from plot.gam. plot_mvgam_smooth generates posterior predictions from an object of class mvgam, calculates posterior empirical quantiles and plots them. If realisations = FALSE, the returned plot shows 90, 60, 40 and 20 percent posterior quantiles (as ribbons of increasingly darker shades or red) as well as the posterior median (as a dark red line). If realisations = FALSE, a set of n_realisations posterior draws are shown. For more nuanced visualisation, supply newdata just as you would when predicting from a gam model or use the more flexible conditional_effects.mvgam. Alternatively, if you prefer to use partial effect plots in the style of gratia, and if you have the gratia package installed, you can use draw.mvgam. See gratia_mvgam_enhancements for details.

Value

A base R graphics plot

Author(s)

Nicholas J Clark

See Also

plot.gam, conditional_effects.mvgam, gratia_mvgam_enhancements

---

plot_mvgam_trend *Plot mvgam latent trend for a specified series*

Description

Plot mvgam latent trend for a specified series

Usage

```
plot_mvgam_trend(
  object,
  series = 1,
  newdata,
  data_test,
  realisations = FALSE,
  n_realisations = 15,
  n_cores = 1,
  derivatives = FALSE,
  hide_xlabels = FALSE,
  xlab,
  ylab,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
series	integer specifying which series in the set is to be plotted
newdata	Optional dataframe or list of test data containing at least 'series' and 'time' in addition to any other variables included in the linear predictor of the original formula.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
realisations	logical. If TRUE, posterior trend realisations are shown as a spaghetti plot, making it easier to visualise the diversity of possible trend paths. If FALSE, the default, empirical quantiles of the posterior distribution are shown
n_realisations	integer specifying the number of posterior realisations to plot, if realisations = TRUE. Ignored otherwise

n_cores	integer specifying number of cores for generating trend forecasts in parallel
derivatives	logical. If TRUE, an additional plot will be returned to show the estimated 1st derivative for the estimated trend
hide_xlabels	logical. If TRUE, no xlabels are printed to allow the user to add custom labels using axis from base R. Ignored if derivatives = TRUE
xlab	label for x axis.
ylab	label for y axis.
...	further par graphical parameters.

Value

A base R graphics plot

Examples

```
simdat <- sim_mvgam(n_series = 3, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc', k = 6),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)

# Plot estimated trends for some series
plot_mvgam_trend(mod)
plot_mvgam_trend(mod, series = 2)

# Extrapolate trends forward in time and plot on response scale
plot_mvgam_trend(mod, newdata = simdat$data_test)
plot_mvgam_trend(mod, newdata = simdat$data_test, series = 2)

# But it is recommended to compute extrapolations for all series
# first and then plot
trend_fc <- forecast(mod, newdata = simdat$data_test)
plot(trend_fc, series = 1)
plot(trend_fc, series = 2)
```

---

plot_mvgam_uncertainty

Plot mvgam forecast uncertainty contributions for a specified series

---

Description

Plot mvgam forecast uncertainty contributions for a specified series

Usage

```
plot_mvgam_uncertainty(
  object,
  series = 1,
  newdata,
  data_test,
  legend_position = "topleft",
  hide_xlabels = FALSE
)
```

Arguments

object	list object returned from mvgam. See mvgam()
series	integer specifying which series in the set is to be plotted
newdata	A dataframe or list containing at least 'series' and 'time' for the forecast horizon, in addition to any other variables included in the linear predictor of formula
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
legend_position	The location may also be specified by setting x to a single keyword from the list: "none", "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location (if it is not "none").
hide_xlabels	logical. If TRUE, no xlabels are printed to allow the user to add custom labels using axis from base R

Value

A base R graphics plot

---

portal_data *Portal Project rodent capture survey data*

Description

A dataset containing timeseries of total captures (across all control plots) for select rodent species from the Portal Project

Usage

```
portal_data
```

Format

A dataframe containing the following fields:

moon time of sampling in lunar cycles

DM Total captures of species Dipodomys merriami

DO Total captures of species Dipodomys ordii

PP Total captures of species Chaetodipus penicillatus

OT Total captures of species Onychomys torridus

year Sampling year

month Sampling month

mintemp Monthly mean minimum temperature

precipitation Monthly mean precipitation

ndvi Monthly mean Normalised Difference Vegetation Index

Source

https://github.com/weecology/PortalData/blob/main/SiteandMethods/Methods.md

---

posterior_epred.mvgam *Draws from the Expected Value of the Posterior Predictive Distribution*

Description

Compute posterior draws of the expected value of the posterior predictive distribution (i.e. the conditional expectation). Can be performed for the data used to fit the model (posterior predictive checks) or for new data. By definition, these predictions have smaller variance than the posterior predictions performed by the `posterior_predict.mvgam` method. This is because only the uncertainty in the expected value of the posterior predictive distribution is incorporated in the draws computed by `posterior_epred` while the residual error is ignored there. However, the estimated means of both methods averaged across draws should be very similar.

Usage

```
## S3 method for class 'mvgam'
posterior_epred(
  object,
  newdata,
  data_test,
  ndraws = NULL,
  process_error = TRUE,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
newdata	Optional dataframe or list of test data containing the variables included in the linear predictor of formula. If not supplied, predictions are generated for the original observations used for the model fit.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used.
process_error	Logical. If TRUE and newdata is supplied, expected uncertainty in the process model is accounted for by using draws from any latent trend SD parameters. If FALSE, uncertainty in the latent trend component is ignored when calculating predictions. If no newdata is supplied, draws from the fitted model's posterior predictive distribution will be used (which will always include uncertainty in any latent trend components)
...	Ignored

Details

Note that for all types of predictions for models that did not include a trend_formula, uncertainty in the dynamic trend component can be ignored by setting process_error = FALSE. However, if a trend_formula was supplied in the model, predictions for this component cannot be ignored. If process_error = TRUE, trend predictions will ignore autocorrelation coefficients or GP length scale coefficients, ultimately assuming the process is stationary. This method is similar to the types of posterior predictions returned from brms models when using autocorrelated error predictions for newdata. This function is therefore more suited to posterior simulation from the GAM components of a mvgam model, while the forecasting functions plot_mvgam_fc and forecast.mvgam are better suited to generate h-step ahead forecasts that respect the temporal dynamics of estimated latent trends.

Value

A matrix of dimension n_samples x new_obs, where n_samples is the number of posterior samples from the fitted object and n_obs is the number of observations in newdata

See Also

hindcast.mvgam posterior_linpred.mvgam posterior_predict.mvgam

Examples

```
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train)
```

```
# Compute posterior expectations
expectations <- posterior_epred(mod)
str(expectations)
```

`posterior_linpred.mvgam` *Posterior Draws of the Linear Predictor*

Description

Compute posterior draws of the linear predictor, that is draws before applying any link functions or other transformations. Can be performed for the data used to fit the model (posterior predictive checks) or for new data.

Usage

```
## S3 method for class 'mvgam'
posterior_linpred(
  object,
  transform = FALSE,
  newdata,
  ndraws = NULL,
  data_test,
  process_error = TRUE,
  ...
)
```

Arguments

`object`	`list` object returned from `mvgam`. See `mvgam()`
`transform`	Logical; if `FALSE` (the default), draws of the linear predictor are returned. If `TRUE`, draws of the transformed linear predictor, i.e. the conditional expectation, are returned.
`newdata`	Optional `dataframe` or `list` of test data containing the variables included in the linear predictor of `formula`. If not supplied, predictions are generated for the original observations used for the model fit.
`ndraws`	Positive `integer` indicating how many posterior draws should be used. If `NULL` (the default) all draws are used.
`data_test`	Deprecated. Still works in place of `newdata` but users are recommended to use `newdata` instead for more seamless integration into `R` workflows
`process_error`	Logical. If `TRUE` and `newdata` is supplied, expected uncertainty in the process model is accounted for by using draws from any latent trend SD parameters. If `FALSE`, uncertainty in the latent trend component is ignored when calculating predictions. If no `newdata` is supplied, draws from the fitted model's posterior predictive distribution will be used (which will always include uncertainty in any latent trend components)
`...`	Ignored

Details

Note that for all types of predictions for models that did not include a `trend_formula`, uncertainty in the dynamic trend component can be ignored by setting `process_error = FALSE`. However, if a `trend_formula` was supplied in the model, predictions for this component cannot be ignored. If `process_error = TRUE`, trend predictions will ignore autocorrelation coefficients or GP length scale coefficients, ultimately assuming the process is stationary. This method is similar to the types of posterior predictions returned from `brms` models when using autocorrelated error predictions for newdata. This function is therefore more suited to posterior simulation from the GAM components of a `mvgam` model, while the forecasting functions `plot_mvgam_fc` and `forecast.mvgam` are better suited to generate h-step ahead forecasts that respect the temporal dynamics of estimated latent trends.

Value

A `matrix` of dimension `n_samples` x `new_obs`, where `n_samples` is the number of posterior samples from the fitted object and `n_obs` is the number of observations in `newdata`

See Also

`posterior_epred.mvgam` `posterior_predict.mvgam`

`hindcast.mvgam` `posterior_epred.mvgam` `posterior_predict.mvgam`

Examples

```
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)

# Extract linear predictor values
linpreds <- posterior_linpred(mod)
str(linpreds)
```

`posterior_predict.mvgam` *Draws from the Posterior Predictive Distribution*

Description

Compute posterior draws of the posterior predictive distribution. Can be performed for the data used to fit the model (posterior predictive checks) or for new data. By definition, these draws have higher variance than draws of the expected value of the posterior predictive distribution computed by `posterior_epred.mvgam`. This is because the residual error is incorporated in `posterior_predict`. However, the estimated means of both methods averaged across draws should be very similar.

Usage

```
## S3 method for class 'mvgam'
posterior_predict(
  object,
  newdata,
  data_test,
  ndraws = NULL,
  process_error = TRUE,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
newdata	Optional dataframe or list of test data containing the variables included in the linear predictor of formula. If not supplied, predictions are generated for the original observations used for the model fit.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used.
process_error	Logical. If TRUE and newdata is supplied, expected uncertainty in the process model is accounted for by using draws from any latent trend SD parameters. If FALSE, uncertainty in the latent trend component is ignored when calculating predictions. If no newdata is supplied, draws from the fitted model's posterior predictive distribution will be used (which will always include uncertainty in any latent trend components)
...	Ignored

Details

Note that for all types of predictions for models that did not include a trend_formula, uncertainty in the dynamic trend component can be ignored by setting process_error = FALSE. However, if a trend_formula was supplied in the model, predictions for this component cannot be ignored. If process_error = TRUE, trend predictions will ignore autocorrelation coefficients or GP length scale coefficients, ultimately assuming the process is stationary. This method is similar to the types of posterior predictions returned from brms models when using autocorrelated error predictions for newdata. This function is therefore more suited to posterior simulation from the GAM components of a mvgam model, while the forecasting functions plot_mvgam_fc and forecast.mvgam are better suited to generate h-step ahead forecasts that respect the temporal dynamics of estimated latent trends.

Value

A matrix of dimension n_samples x new_obs, where n_samples is the number of posterior samples from the fitted object and n_obs is the number of observations in newdata

See Also

hindcast.mvgam posterior_linpred.mvgam posterior_epred.mvgam

Examples

```
## Not run:
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = 'AR1',
             data = simdat$data_train)

# Compute posterior predictions
predictions <- posterior_predict(mod)
str(predictions)

## End(Not run)
```

---

ppc.mvgam *Plot mvgam posterior predictive checks for a specified series*

Description

Plot mvgam posterior predictive checks for a specified series

Usage

```
ppc(object, ...)

## S3 method for class 'mvgam'
ppc(
  object,
  newdata,
  data_test,
  series = 1,
  type = "hist",
  n_bins,
  legend_position,
  xlab,
  ylab,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
...	further par graphical parameters.

newdata	Optional dataframe or list of test data containing at least 'series' and 'time' for the forecast horizon, in addition to any other variables included in the linear predictor of formula. If included, the observed values in the test data are compared to the model's forecast distribution for exploring biases in model predictions. Note this is only useful if the same newdata was also included when fitting the original model.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
series	integer specifying which series in the set is to be plotted
type	character specifying the type of posterior predictive check to calculate and plot. Valid options are: 'rootogram', 'mean', 'hist', 'density', 'prop_zero', 'pit' and 'cdf'
n_bins	integer specifying the number of bins to use for binning observed values when plotting a rootogram or histogram. Default is 50 bins for a rootogram, which means that if there are >50 unique observed values, bins will be used to prevent overplotting and facilitate interpretation. Default for a histogram is to use the number of bins returned by a call to hist in base R
legend_position	The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location. Or alternatively, use "none" to hide the legend.
xlab	label for x axis.
ylab	label for y axis.

Details

Posterior predictions are drawn from the fitted mvgam and compared against the empirical distribution of the observed data for a specified series to help evaluate the model's ability to generate unbiased predictions. For all plots apart from type = 'rootogram', posterior predictions can also be compared to out of sample observations as long as these observations were included as 'data_test' in the original model fit and supplied here. Rootograms are currently only plotted using the 'hanging' style.

Note that the predictions used for these plots are those that have been generated directly within the mvgam() model, so they can be misleading if the model included flexible dynamic trend components. For a broader range of posterior checks that are created using "new data" predictions, see pp_check.mvgam

Value

A base R graphics plot showing either a posterior rootogram (for type == 'rootogram'), the predicted vs observed mean for the series (for type == 'mean'), predicted vs observed proportion of zeroes for the series (for type == 'prop_zero'),predicted vs observed histogram for the series (for type == 'hist'), kernel density or empirical CDF estimates for posterior predictions (for type == 'density' or type == 'cdf') or a Probability Integral Transform histogram (for type == 'pit').

Author(s)

Nicholas J Clark

See Also

pp_check.mvgam, predict.mvgam

Examples

```
# Simulate some smooth effects and fit a model
set.seed(0)
dat <- mgcv::gamSim(1, n = 200, scale = 2)
mod <- mvgam(y ~ s(x0) + s(x1) + s(x2) + s(x3),
             data = dat,
             family = gaussian(),
             chains = 2)

# Posterior checks
ppc(mod, type = 'hist')
ppc(mod, type = 'density')
ppc(mod, type = 'cdf')

# Many more options are available with pp_check()
pp_check(mod)
pp_check(mod, type = "ecdf_overlay")
pp_check(mod, type = 'freqpoly')
```

---

pp_check.mvgam *Posterior Predictive Checks for* mvgam *Objects*

Description

Perform posterior predictive checks with the help of the **bayesplot** package.

Usage

```
## S3 method for class 'mvgam'
pp_check(
  object,
  type,
  ndraws = NULL,
  prefix = c("ppc", "ppd"),
  group = NULL,
  x = NULL,
  newdata = NULL,
  ...
)
```

Arguments

object	An object of class mvgam.
type	Type of the ppc plot as given by a character string. See PPC for an overview of currently supported types. You may also use an invalid type (e.g. type = "xyz") to get a list of supported types in the resulting error message.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL all draws are used. If not specified, the number of posterior draws is chosen automatically. Ignored if draw_ids is not NULL.
prefix	The prefix of the **bayesplot** function to be applied. Either ‘"ppc"‘ (posterior predictive check; the default) or ‘"ppd"‘ (posterior predictive distribution), the latter being the same as the former except that the observed data is not shown for ‘"ppd"‘.
group	Optional name of a factor variable in the model by which to stratify the ppc plot. This argument is required for ppc *_grouped types and ignored otherwise.
x	Optional name of a variable in the model. Only used for ppc types having an x argument and ignored otherwise.
newdata	Optional dataframe or list of test data containing the variables included in the linear predictor of formula. If not supplied, predictions are generated for the original observations used for the model fit.
...	Further arguments passed to predict.mvgam as well as to the PPC function specified in type.

Details

For a detailed explanation of each of the ppc functions, see the PPC documentation of the **bayesplot** package.

Value

A ggplot object that can be further customized using the **ggplot2** package.

Author(s)

Nicholas J Clark

See Also

ppc predict.mvgam

Examples

```
## Not run:
simdat <- sim_mvgam(seasonality = 'hierarchical')
mod <- mvgam(y ~ series +
               s(season, bs = 'cc', k = 6) +
               s(season, series, bs = 'fs', k = 4),
             data = simdat$data_train,
```

```
            burnin = 300,
            samples = 300)

# Use pp_check(mod, type = "xyz") for a list of available plot types

# Default is a density overlay for all observations
pp_check(mod)

# Rootograms particularly useful for count data
pp_check(mod, type = "rootogram")

# Grouping plots by series is useful
pp_check(mod, type = "bars_grouped",
        group = "series", ndraws = 50)
pp_check(mod, type = "ecdf_overlay_grouped",
        group = "series", ndraws = 50)
pp_check(mod, type = "stat_freqpoly_grouped",
        group = "series", ndraws = 50)

# Custom functions accepted
prop_zero <- function(x) mean(x == 0)
pp_check(mod, type = "stat", stat = "prop_zero")
pp_check(mod, type = "stat_grouped",
        stat = "prop_zero",
        group = "series")

# Some functions accept covariates to set the x-axes
pp_check(mod, x = "season",
        type = "ribbon_grouped",
        prob = 0.5,
        prob_outer = 0.8,
        group = "series")

# Many plots can be made without the observed data
pp_check(mod, prefix = "ppd")

## End(Not run)
```

`predict.mvgam` *Predict from the GAM component of an mvgam model*

Description

Predict from the GAM component of an mvgam model

Usage

```
## S3 method for class 'mvgam'
predict(
```

```
  object,
  newdata,
  data_test,
  type = "link",
  process_error = TRUE,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
newdata	Optional dataframe or list of test data containing the variables included in the linear predictor of formula. If not supplied, predictions are generated for the original observations used for the model fit.
data_test	Deprecated. Still works in place of newdata but users are recommended to use newdata instead for more seamless integration into R workflows
type	When this has the value link (default) the linear predictor is calculated on the link scale. If expected is used, predictions reflect the expectation of the response (the mean) but ignore uncertainty in the observation process. When response is used, the predictions take uncertainty in the observation process into account to return predictions on the outcome scale. When variance is used, the variance of the response with respect to the mean (mean-variance relationship) is returned. When type = "terms", each component of the linear predictor is returned separately in the form of a list (possibly with standard errors, if summary = TRUE): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. Two special cases are also allowed: type latent_N will return the estimated latent abundances from an N-mixture distribution, while type detection will return the estimated detection probability from an N-mixture distribution
process_error	Logical. If TRUE and a dynamic trend model was fit, expected uncertainty in the process model is accounted for by using draws from the latent trend SD parameters. If FALSE, uncertainty in the latent trend component is ignored when calculating predictions
summary	Should summary statistics be returned instead of the raw values? Default is TRUE..
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Ignored

Details

Note that for all types of predictions for models that did not include a `trend_formula`, uncertainty in the dynamic trend component can be ignored by setting `process_error = FALSE`. However, if a `trend_formula` was supplied in the model, predictions for this component cannot be ignored. If `process_error = TRUE`, trend predictions will ignore autocorrelation coefficients or GP length scale coefficients, ultimately assuming the process is stationary. This method is similar to the types of posterior predictions returned from `brms` models when using autocorrelated error predictions for newdata. This function is therefore more suited to posterior simulation from the GAM components of a `mvgam` model, while the forecasting functions `plot_mvgam_fc` and `forecast.mvgam` are better suited to generate h-step ahead forecasts that respect the temporal dynamics of estimated latent trends.

Value

Predicted values on the appropriate scale. If `summary = FALSE` and `type != "terms"`, the output is a matrix of dimension `n_draw x n_observations` containing predicted values for each posterior draw in `object`.

If `summary = TRUE` and `type != "terms"`, the output is an `n_observations` x `E` matrix. The number of summary statistics `E` is equal to `2 + length(probs)`: The `Estimate` column contains point estimates (either mean or median depending on argument `robust`), while the `Est.Error` column contains uncertainty estimates (either standard deviation or median absolute deviation depending on argument `robust`). The remaining columns starting with `Q` contain quantile estimates as specified via argument `probs`.

If `type = "terms"` and `summary = FALSE`, the output is a named `list` containing a separate slot for each effect, with the effects returned as matrices of dimension `n_draw x 1`. If `summary = TRUE`, the output resembles that from `predict.gam` when using the call `predict.gam(object, type = "terms", se.fit = TRUE)`, where mean contributions from each effect are returned in `matrix` form while standard errors (representing the interval: `(max(probs) - min(probs)) / 2`) are returned in a separate `matrix`

Examples

```
# Simulate 4 time series with hierarchical seasonality
# and independent AR1 dynamic processes
set.seed(111)
simdat <- sim_mvgam(seasonality = 'hierarchical',
                    trend_model = 'AR1',
                    family = gaussian())

# Fit a model with shared seasonality
mod1 <- mvgam(y ~ s(season, bs = 'cc', k = 6),
              data = simdat$data_train,
              family = gaussian(),
              trend_model = AR(),
              noncentred = TRUE,
              chains = 2)

# Generate predictions against observed data
preds <- predict(mod1, summary = TRUE)
```

```
head(preds)

# Generate predictions against test data
preds <- predict(mod1, newdata = simdat$data_test, summary = TRUE)
head(preds)
```

`print.mvgam` *Summary for a fitted mvgam object*

Description

This function takes a fitted `mvgam` object and prints a quick summary

Usage

```
## S3 method for class 'mvgam'
print(x, ...)
```

Arguments

`x`	`list` object returned from `mvgam`
`...`	Ignored

Details

A brief summary of the model's call is printed

Value

A `list` is printed on-screen

Author(s)

Nicholas J Clark

PW *Specify piecewise linear or logistic trends*

Description

Set up piecewise linear or logistic trend models in mvgam. These functions do not evaluate their arguments – they exist purely to help set up a model with particular piecewise trend models.

Usage

```
PW(
  n_changepoints = 10,
  changepoint_range = 0.8,
  changepoint_scale = 0.05,
  growth = "linear"
)
```

Arguments

n_changepoints	A non-negative integer specifying the number of potential changepoints. Potential changepoints are selected uniformly from the first changepoint_range proportion of timepoints in data. Default is 10
changepoint_range	Proportion of history in data in which trend changepoints will be estimated. Defaults to 0.8 for the first 80%.
changepoint_scale	Parameter modulating the flexibility of the automatic changepoint selection by altering the scale parameter of a Laplace distribution. The resulting prior will be double_exponential(0, changepoint_scale). Large values will allow many changepoints and a more flexible trend, while small values will allow few changepoints. Default is 0.05.
growth	Character string specifying either 'linear' or 'logistic' growth of the trend. If 'logistic', a variable labelled cap MUST be in data to specify the maximum saturation point for the trend (see details and examples in mvgam for more information). Default is 'linear'.

Details

Offsets and intercepts: For each of these trend models, an offset parameter is included in the trend estimation process. This parameter will be incredibly difficult to identify if you also include an intercept in the observation formula. For that reason, it is highly recommended that you drop the intercept from the formula (i.e. y ~ x + 0 or y ~ x - 1, where x are your optional predictor terms).

Logistic growth and the cap variable: When forecasting growth, there is often some maximum achievable point that a time series can reach. For example, total market size, total population size or carrying capacity in population dynamics. It can be advantageous for the forecast to saturate at or near this point so that predictions are more sensible. This function allows you to make forecasts

using a logistic growth trend model, with a specified carrying capacity. Note that this capacity does not need to be static over time, it can vary with each series x timepoint combination if necessary. But you must supply a cap value for each observation in the data when using growth = 'logistic'. For observation families that use a non-identity link function, the cap value will be internally transformed to the link scale (i.e. your specified cap will be log transformed if you are using a poisson() or nb() family). It is therefore important that you specify the cap values on the scale of your outcome. Note also that no missing values are allowed in cap.

Value

An object of class mvgam_trend, which contains a list of arguments to be interpreted by the parsing functions in mvgam

References

Taylor, Sean J., and Benjamin Letham. "Forecasting at scale." The American Statistician 72.1 (2018): 37-45.

Examples

```
# Example of logistic growth with possible changepoints
# Simple logistic growth model
dNt = function(r, N, k){
   r * N * (k - N)
}

# Iterate growth through time
Nt = function(r, N, t, k) {
for (i in 1:(t - 1)) {

 # population at next time step is current population + growth,
 # but we introduce several 'shocks' as changepoints
 if(i %in% c(5, 15, 25, 41, 45, 60, 80)){
   N[i + 1] <- max(1, N[i] + dNt(r + runif(1, -0.1, 0.1),
                                 N[i], k))
   } else {
   N[i + 1] <- max(1, N[i] + dNt(r, N[i], k))
   }
  }
 N
}

# Simulate expected values
set.seed(11)
expected <- Nt(0.004, 2, 100, 30)
plot(expected, xlab = 'Time')

# Take Poisson draws
y <- rpois(100, expected)
plot(y, xlab = 'Time')

# Assemble data into dataframe and model. We set a
```

```
# fixed carrying capacity of 35 for this example, but note that
# this value is not required to be fixed at each timepoint
mod_data <- data.frame(y = y,
                       time = 1:100,
                       cap = 35,
                       series = as.factor('series_1'))
plot_mvgam_series(data = mod_data)

# The intercept is nonidentifiable when using piecewise
# trends because the trend functions have their own offset
# parameters 'm'; it is recommended to always drop intercepts
# when using these trend models
mod <- mvgam(y ~ 0,
             trend_model = PW(growth = 'logistic'),
             family = poisson(),
             data = mod_data,
             chains = 2)
summary(mod)

# Plot the posterior hindcast
plot(mod, type = 'forecast')

# View the changepoints with ggplot2 utilities
library(ggplot2)
mcmc_plot(mod, variable = 'delta_trend',
          regex = TRUE) +
scale_y_discrete(labels = mod$trend_model$changepoints) +
labs(y = 'Potential changepoint',
     x = 'Rate change')
```

`residuals.mvgam` *Posterior draws of* `mvgam` *residuals*

Description

This method extracts posterior draws of Dunn-Smyth (randomized quantile) residuals in the order in which the data were supplied to the model. It included additional arguments for obtaining summaries of the computed residuals

Usage

```
## S3 method for class 'mvgam'
residuals(object, summary = TRUE, robust = FALSE, probs = c(0.025, 0.975), ...)
```

Arguments

`object`	An object of class `mvgam`
`summary`	Should summary statistics be returned instead of the raw values? Default is `TRUE`..

robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Details

This method gives residuals as Dunn-Smyth (randomized quantile) residuals. Any observations that were missing (i.e. NA) in the original data will have missing values in the residuals

Value

An array of randomized quantile residual values. If summary = FALSE the output resembles those of posterior_epred.mvgam and predict.mvgam.

If summary = TRUE the output is an n_observations x E matrix. The number of summary statistics E is equal to 2 + length(probs): The Estimate column contains point estimates (either mean or median depending on argument robust), while the Est.Error column contains uncertainty estimates (either standard deviation or median absolute deviation depending on argument robust). The remaining columns starting with Q contain quantile estimates as specified via argument probs.

Examples

```
# Simulate some data and fit a model
simdat <- sim_mvgam(n_series = 1, trend_model = 'AR1')
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)

# Extract posterior residuals
resids <- residuals(mod)
str(resids)
```

RW *Specify autoregressive dynamic processes*

Description

Set up autoregressive or autoregressive moving average trend models in mvgam. These functions do not evaluate their arguments – they exist purely to help set up a model with particular autoregressive trend models.

Usage

```
RW(ma = FALSE, cor = FALSE)

AR(p = 1, ma = FALSE, cor = FALSE)

CAR(p = 1)

VAR(ma = FALSE, cor = FALSE)
```

Arguments

ma	Logical Include moving average terms of order 1? Default is FALSE.
cor	Logical Include correlated process errors as part of a multivariate normal process model? If TRUE and if n_series > 1 in the supplied data, a fully structured covariance matrix will be estimated for the process errors. Default is FALSE.
p	A non-negative integer specifying the autoregressive (AR) order. Default is 1. Cannot currently be larger than 3 for AR terms, and cannot be anything other than 1 for continuous time AR (CAR) terms

Value

An object of class mvgam_trend, which contains a list of arguments to be interpreted by the parsing functions in mvgam

Examples

```
## Not run:
# A short example to illustrate CAR(1) models
# Function to simulate CAR1 data with seasonality
sim_corcar1 = function(n = 120,
                       phi = 0.5,
                       sigma = 1,
                       sigma_obs = 0.75){
# Sample irregularly spaced time intervals
time_dis <- c(0, runif(n - 1, -0.1, 1))
time_dis[time_dis < 0] <- 0; time_dis <- time_dis * 5

# Set up the latent dynamic process
x <- vector(length = n); x[1] <- -0.3
for(i in 2:n){
 # zero-distances will cause problems in sampling, so mvgam uses a
 # minimum threshold; this simulation function emulates that process
 if(time_dis[i] == 0){
   x[i] <- rnorm(1, mean = (phi ^ 1e-12) * x[i - 1], sd = sigma)
  } else {
    x[i] <- rnorm(1, mean = (phi ^ time_dis[i]) * x[i - 1], sd = sigma)
  }
}

# Add 12-month seasonality
```

```
cov1 <- sin(2 * pi * (1 : n) / 12); cov2 <- cos(2 * pi * (1 : n) / 12)
beta1 <- runif(1, 0.3, 0.7); beta2 <- runif(1, 0.2, 0.5)
seasonality <- beta1 * cov1 + beta2 * cov2

# Take Gaussian observations with error and return
data.frame(y = rnorm(n, mean = x + seasonality, sd = sigma_obs),
           season = rep(1:12, 20)[1:n],
           time = cumsum(time_dis))
}

# Sample two time series
dat <- rbind(dplyr::bind_cols(sim_corcar1(phi = 0.65,
                                          sigma_obs = 0.55),
                              data.frame(series = 'series1')),
             dplyr::bind_cols(sim_corcar1(phi = 0.8,
                              sigma_obs = 0.35),
                              data.frame(series = 'series2'))) %>%
      dplyr::mutate(series = as.factor(series))

# mvgam with CAR(1) trends and series-level seasonal smooths; the
# State-Space representation (using trend_formula) will be more efficient
mod <- mvgam(formula = y ~ 1,
             trend_formula = ~ s(season, bs = 'cc',
                                 k = 5, by = trend),
             trend_model = CAR(),
             data = dat,
             family = gaussian(),
             samples = 300,
             chains = 2)

# View usual summaries and plots
summary(mod)
conditional_effects(mod, type = 'expected')
plot(mod, type = 'trend', series = 1)
plot(mod, type = 'trend', series = 2)
plot(mod, type = 'residuals', series = 1)
plot(mod, type = 'residuals', series = 2)

## End(Not run)
```

`score.mvgam_forecast` *Compute probabilistic forecast scores for mvgam objects*

Description

Compute probabilistic forecast scores for mvgam objects

Usage

```
## S3 method for class 'mvgam_forecast'
```

```
score(
  object,
  score = "crps",
  log = FALSE,
  weights,
  interval_width = 0.9,
  n_cores = 1,
  ...
)

score(object, ...)
```

Arguments

object	mvgam_forecast object. See forecast.mvgam().
score	character specifying the type of proper scoring rule to use for evaluation. Options are: sis (i.e. the Scaled Interval Score), energy, variogram, elpd (i.e. the Expected log pointwise Predictive Density), drps (i.e. the Discrete Rank Probability Score) or crps (the Continuous Rank Probability Score). Note that when choosing elpd, the supplied object must have forecasts on the link scale so that expectations can be calculated prior to scoring. For all other scores, forecasts should be supplied on the response scale (i.e. posterior predictions)
log	logical. Should the forecasts and truths be logged prior to scoring? This is often appropriate for comparing performance of models when series vary in their observation ranges
weights	optional vector of weights (where length(weights) == n_series) for weighting pairwise correlations when evaluating the variogram score for multivariate forecasts. Useful for down-weighting series that have larger magnitude observations or that are of less interest when forecasting. Ignored if score != 'variogram'
interval_width	proportional value on [0.05,0.95] defining the forecast interval for calculating coverage and, if score = 'sis', for calculating the interval score
n_cores	integer specifying number of cores for calculating scores in parallel
...	Ignored

Value

a list containing scores and interval coverages per forecast horizon. If score %in% c('drps', 'crps', 'elpd'), the list will also contain return the sum of all series-level scores per horizon. If score %in% c('energy','variogram'), no series-level scores are computed and the only score returned will be for all series. For all scores apart from elpd, the in_interval column in each series-level slot is a binary indicator of whether or not the true value was within the forecast's corresponding posterior empirical quantiles. Intervals are not calculated when using elpd because forecasts will only contain the linear predictors

See Also

forecast.mvgam, ensemble

Examples

```
# Simulate observations for three count-valued time series
data <- sim_mvgam()
# Fit a dynamic model using 'newdata' to automatically produce forecasts
mod <- mvgam(y ~ 1,
             trend_model = RW(),
             data = data$data_train,
             newdata = data$data_test,
             chains = 2)

# Extract forecasts into a 'mvgam_forecast' object
fc <- forecast(mod)

# Compute Discrete Rank Probability Scores and 0.90 interval coverages
fc_scores <- score(fc, score = 'drps')
str(fc_scores)
```

`series_to_mvgam` *This function converts univariate or multivariate time series (*`xts` *or* `ts` *objects) to the format necessary for* `mvgam`

Description

This function converts univariate or multivariate time series (`xts` or `ts` objects) to the format necessary for `mvgam`

Usage

```
series_to_mvgam(series, freq, train_prop = 0.85)
```

Arguments

`series`	`xts` or `ts` object to be converted to `mvgam` format
`freq`	`integer`. The seasonal frequency of the series
`train_prop`	`numeric` stating the proportion of data to use for training. Should be between `0.25` and `0.95`

Value

A `list` object containing outputs needed for `mvgam`, including 'data_train' and 'data_test'

Examples

```
# A ts object example
data("sunspots")
series <- cbind(sunspots, sunspots)
colnames(series) <- c('blood', 'bone')
head(series)
series_to_mvgam(series, frequency(series), 0.85)

# An xts object example
library(xts)
dates <- seq(as.Date("2001-05-01"), length=30, by="quarter")
data  <- cbind(c(gas = rpois(30, cumprod(1+rnorm(30, mean = 0.01, sd = 0.001)))),
c(oil = rpois(30, cumprod(1+rnorm(30, mean = 0.01, sd = 0.001)))))
series <- xts(x = data, order.by = dates)
colnames(series) <- c('gas', 'oil')
head(series)
series_to_mvgam(series, freq = 4, train_prop = 0.85)
```

---

sim_mvgam *Simulate a set of time series for mvgam modelling*

Description

This function simulates sets of time series data for fitting a multivariate GAM that includes shared seasonality and dependence on state-space latent dynamic factors. Random dependencies among series, i.e. correlations in their long-term trends, are included in the form of correlated loadings on the latent dynamic factors

Usage

```
sim_mvgam(
  T = 100,
  n_series = 3,
  seasonality = "shared",
  use_lv = FALSE,
  n_lv = 0,
  trend_model = "RW",
  drift = FALSE,
  prop_trend = 0.2,
  trend_rel,
  freq = 12,
  family = poisson(),
  phi,
  shape,
  sigma,
  nu,
  mu,
```

```
  prop_missing = 0,
  prop_train = 0.85
)
```

Arguments

T	integer. Number of observations (timepoints)
n_series	integer. Number of discrete time series
seasonality	character. Either shared, meaning that all series share the exact same seasonal pattern, or hierarchical, meaning that there is a global seasonality but each series' pattern can deviate slightly
use_lv	logical. If TRUE, use dynamic factors to estimate series' latent trends in a reduced dimension format. If FALSE, estimate independent latent trends for each series
n_lv	integer. Number of latent dynamic factors for generating the series' trends. Defaults to 0, meaning that dynamics are estimated independently for each series
trend_model	character specifying the time series dynamics for the latent trend. Options are: • None (no latent trend component; i.e. the GAM component is all that contributes to the linear predictor, and the observation process is the only source of error; similarly to what is estimated by gam) • RW (random walk with possible drift) • AR1 (with possible drift) • AR2 (with possible drift) • AR3 (with possible drift) • VAR1 (contemporaneously uncorrelated VAR1) • VAR1cor (contemporaneously correlated VAR1) • GP (Gaussian Process with squared exponential kernel) See mvgam_trends for more details
drift	logical, simulate a drift term for each trend
prop_trend	numeric. Relative importance of the trend for each series. Should be between 0 and 1
trend_rel	Deprecated. Use prop_trend instead
freq	integer. The seasonal frequency of the series
family	family specifying the exponential observation family for the series. Currently supported families are: nb(), poisson(), bernoulli(), tweedie(), gaussian(), betar(), lognormal(), student() and Gamma()
phi	vector of dispersion parameters for the series (i.e. size for nb() or phi for betar()). If length(phi) < n_series, the first element of phi will be replicated n_series times. Defaults to 5 for nb() and tweedie(); 10 for betar()
shape	vector of shape parameters for the series (i.e. shape for gamma()) If length(shape) < n_series, the first element of shape will be replicated n_series times. Defaults to 10

sigma	vector of scale parameters for the series (i.e. sd for gaussian() or student(), log(sd) for lognormal()). If length(sigma) < n_series, the first element of sigma will be replicated n_series times. Defaults to 0.5 for gaussian() and student(); 0.2 for lognormal()
nu	vector of degrees of freedom parameters for the series (i.e. nu for student()) If length(nu) < n_series, the first element of nu will be replicated n_series times. Defaults to 3
mu	vector of location parameters for the series. If length(mu) < n_series, the first element of mu will be replicated n_series times. Defaults to small random values between -0.5 and 0.5 on the link scale
prop_missing	numeric stating proportion of observations that are missing. Should be between 0 and 0.8, inclusive
prop_train	numeric stating the proportion of data to use for training. Should be between 0.2 and 1

Value

A list object containing outputs needed for mvgam, including 'data_train' and 'data_test', as well as some additional information about the simulated seasonality and trend dependencies

Examples

```
# Simulate series with observations bounded at 0 and 1 (Beta responses)
sim_data <- sim_mvgam(family = betar(), trend_model = RW(), prop_trend = 0.6)
plot_mvgam_series(data = sim_data$data_train, series = 'all')

# Now simulate series with overdispersed discrete observations
sim_data <- sim_mvgam(family = nb(), trend_model = RW(), prop_trend = 0.6, phi = 10)
plot_mvgam_series(data = sim_data$data_train, series = 'all')
```

---

summary.mvgam *Summary for a fitted mvgam object*

Description

These functions take a fitted mvgam object and return various useful summaries

Usage

```
## S3 method for class 'mvgam'
summary(object, include_betas = TRUE, smooth_test = TRUE, digits = 2, ...)

## S3 method for class 'mvgam_prefit'
summary(object, ...)

## S3 method for class 'mvgam'
coef(object, summarise = TRUE, ...)
```

Arguments

object	list object returned from mvgam
include_betas	Logical. Print a summary that includes posterior summaries of all linear predictor beta coefficients (including spline coefficients)? Defaults to TRUE but use FALSE for a more concise summary
smooth_test	Logical. Compute estimated degrees of freedom and approximate p-values for smooth terms? Defaults to TRUE, but users may wish to set to FALSE for complex models with many smooth or random effect terms
digits	The number of significant digits for printing out the summary; defaults to 2.
...	Ignored
summarise	logical. Summaries of coefficients will be returned if TRUE. Otherwise the full posterior distribution will be returned

Details

summary.mvgam and summary.mvgam_prefit return brief summaries of the model's call, along with posterior intervals for some of the key parameters in the model. Note that some smooths have extra penalties on the null space, so summaries for the rho parameters may include more penalty terms than the number of smooths in the original model formula. Approximate p-values for smooth terms are also returned, with methods used for their calculation following those used for mgcv equivalents (see summary.gam for details). The Estimated Degrees of Freedom (edf) for smooth terms is computed using either edf.type = 1 for models with no trend component, or edf.type = 0 for models with trend components. These are described in the documentation for jagam. Experiments suggest these p-values tend to be more conservative than those that might be returned from an equivalent model fit with summary.gam using method = 'REML'

coef.mvgam returns either summaries or full posterior estimates for GAM component coefficients

Value

For summary.mvgam and summary.mvgam_prefit, a list is printed on-screen showing the summaries for the model

For coef.mvgam, either a matrix of posterior coefficient distributions (if summarise == FALSE or data.frame of coefficient summaries)

Author(s)

Nicholas J Clark

---

update.mvgam *Update an existing* mvgam *object*

Description

This function allows a previously fitted mvgam model to be updated

Usage

```
## S3 method for class 'mvgam'
update(
  object,
  formula,
  trend_formula,
  data,
  newdata,
  trend_model,
  trend_map,
  use_lv,
  n_lv,
  family,
  share_obs_params,
  priors,
  chains,
  burnin,
  samples,
  threads,
  algorithm,
  lfo = FALSE,
  ...
)
```

Arguments

object	list object returned from mvgam. See mvgam()
formula	Optional new formula object. Note, mvgam currently does not support dynamic formula updates such as removal of specific terms with - term. When updating, the entire formula needs to be supplied
trend_formula	An optional character string specifying the GAM process model formula. If supplied, a linear predictor will be modelled for the latent trends to capture process model evolution separately from the observation model. Should not have a response variable specified on the left-hand side of the formula (i.e. a valid option would be ~ season + s(year)). Also note that you should not use the identifier series in this formula to specify effects that vary across time series. Instead you should use trend. This will ensure that models in which a trend_map is supplied will still work consistently (i.e. by allowing effects to vary across process models, even when some time series share the same underlying process model). This feature is only currently available for RW(), AR() and VAR() trend models. In nmix() family models, the trend_formula is used to set up a linear predictor for the underlying latent abundance. Be aware that it can be very challenging to simultaneously estimate intercept parameters for both the observation mode (captured by formula) and the process model (captured by trend_formula). Users are recommended to drop one of these using the - 1 convention in the formula right hand side.
data	A dataframe or list containing the model response variable and covariates

required by the GAM `formula` and optional `trend_formula`. Should include columns: #'

- `series` (a `factor` index of the series IDs; the number of levels should be identical to the number of unique series labels (i.e. `n_series = length(levels(data$series))`))
- `time` (`numeric` or `integer` index of the time point for each observation). For most dynamic trend types available in `mvgam` (see argument `trend_model`), time should be measured in discrete, regularly spaced intervals (i.e. `c(1, 2, 3, ...)`). However you can use irregularly spaced intervals if using `trend_model = CAR(1)`, though note that any temporal intervals that are exactly `0` will be adjusted to a very small number (`1e-12`) to prevent sampling errors. See an example of `CAR()` trends in `CAR`

Should also include any other variables to be included in the linear predictor of `formula`

`newdata` Optional `dataframe` or `list` of test data containing at least `series` and `time` in addition to any other variables included in the linear predictor of `formula`. If included, the observations in variable `y` will be set to `NA` when fitting the model so that posterior simulations can be obtained

`trend_model` `character` or `function` specifying the time series dynamics for the latent trend. Options are:

- `None` (no latent trend component; i.e. the GAM component is all that contributes to the linear predictor, and the observation process is the only source of error; similarly to what is estimated by `gam`)
- `'RW'` or `RW()`
- `'AR1'` or `AR(p = 1)`
- `'AR2'` or `AR(p = 2)`
- `'AR3'` or `AR(p = 3)`
- `'CAR1'` or `CAR(p = 1)`
- `'VAR1'` or `VAR()`(only available in `Stan`)
- `'PWlogistic'`, `'PWlinear'` or `PW()` (only available in `Stan`)
- `'GP'` or `GP()` (Gaussian Process with squared exponential kernel; only available in `Stan`)

For all trend types apart from `GP()`, `CAR()` and `PW()`, moving average and/or correlated process error terms can also be estimated (for example, `RW(cor = TRUE)` will set up a multivariate Random Walk if `n_series > 1`). See mvgam_trends for more details

`trend_map` Optional `data.frame` specifying which series should depend on which latent trends. Useful for allowing multiple series to depend on the same latent trend process, but with different observation processes. If supplied, a latent factor model is set up by setting `use_lv = TRUE` and using the mapping to set up the shared trends. Needs to have column names `series` and `trend`, with integer values in the `trend` column to state which trend each series should depend on. The `series` column should have a single unique entry for each series in the data (names should perfectly match factor levels of the `series` variable in `data`). Note that if this is supplied, the intercept parameter in the process model will NOT be automatically suppressed. See examples for details

use_lv
: logical. If TRUE, use dynamic factors to estimate series' latent trends in a reduced dimension format. Only available for RW(), AR() and GP() trend models. Defaults to FALSE

n_lv
: integer the number of latent dynamic factors to use if use_lv == TRUE. Cannot be > n_series. Defaults arbitrarily to min(2, floor(n_series / 2))

family
: family specifying the exponential observation family for the series. Currently supported families are:

  - gaussian() for real-valued data
  - betar() for proportional data on (0,1)
  - lognormal() for non-negative real-valued data
  - student_t() for real-valued data
  - Gamma() for non-negative real-valued data
  - bernoulli() for binary data
  - poisson() for count data
  - nb() for overdispersed count data
  - binomial() for count data with imperfect detection when the number of trials is known; note that the cbind() function must be used to bind the discrete observations and the discrete number of trials
  - beta_binomial() as for binomial() but allows for overdispersion
  - nmix() for count data with imperfect detection when the number of trials is unknown and should be modeled via a State-Space N-Mixture model. The latent states are Poisson, capturing the 'true' latent abundance, while the observation process is Binomial to account for imperfect detection. See mvgam_families for an example of how to use this family

  Note that only nb() and poisson() are available if using JAGS as the backend. Default is poisson(). See mvgam_families for more details

share_obs_params
: logical. If TRUE and the family has additional family-specific observation parameters (e.g. variance components in student_t() or gaussian(), or dispersion parameters in nb() or betar()), these parameters will be shared across all series. This is handy if you have multiple time series that you believe share some properties, such as being from the same species over different spatial units. Default is FALSE.

priors
: An optional data.frame with prior definitions (in JAGS or Stan syntax). if using Stan, this can also be an object of class brmsprior (see. prior for details). See get_mvgam_priors and 'Details' for more information on changing default prior distributions

chains
: integer specifying the number of parallel chains for the model. Ignored if algorithm %in% c('meanfield', 'fullrank', 'pathfinder', 'laplace')

burnin
: integer specifying the number of warmup iterations of the Markov chain to run to tune sampling algorithms. Ignored if algorithm %in% c('meanfield', 'fullrank', 'pathfinder', 'laplace')

samples
: integer specifying the number of post-warmup iterations of the Markov chain to run for sampling the posterior distribution

threads — integer Experimental option to use multithreading for within-chain parallelisation in Stan. We recommend its use only if you are experienced with Stan's reduce_sum function and have a slow running model that cannot be sped up by any other means. Only available for some families(poisson(), nb(), gaussian()) and when using Cmdstan as the backend

algorithm — Character string naming the estimation approach to use. Options are "sampling" for MCMC (the default), "meanfield" for variational inference with factorized normal distributions, "fullrank" for variational inference with a multivariate normal distribution, "laplace" for a Laplace approximation (only available when using cmdstanr as the backend) or "pathfinder" for the pathfinder algorithm (only currently available when using cmdstanr as the backend). Can be set globally for the current R session via the "brms.algorithm" option (see options). Limited testing suggests that "meanfield" performs best out of the non-MCMC approximations for dynamic GAMs, possibly because of the difficulties estimating covariances among the many spline parameters and latent trend parameters. But rigorous testing has not been carried out

lfo — Logical indicating whether this is part of a call to lfo_cv.mvgam. Returns a lighter version of the model with no residuals and fewer monitored parameters to speed up post-processing. But other downstream functions will not work properly, so users should always leave this set as FALSE

... — Other arguments to be passed to mvgam

Value

A list object of class mvgam containing model output, the text representation of the model file, the mgcv model output (for easily generating simulations at unsampled covariate values), Dunn-Smyth residuals for each series and key information needed for other functions in the package. See mvgam-class for details. Use methods(class = "mvgam") for an overview on available methods.

Examples

```
# Simulate some data and fit a Poisson AR1 model
simdat <- sim_mvgam(n_series = 1, trend_model = AR())
mod <- mvgam(y ~ s(season, bs = 'cc'),
             trend_model = AR(),
             noncentred = TRUE,
             data = simdat$data_train,
             chains = 2)
summary(mod)
conditional_effects(mod, type = 'link')

# Update to an AR2 model
updated_mod <- update(mod, trend_model = AR(p = 2),
                      noncentred = TRUE)
summary(updated_mod)
conditional_effects(updated_mod, type = 'link')

# Now update to a Binomial AR1 by adding information on trials
# requires that we supply newdata that contains the 'trials' variable
simdat$data_train$trials <- max(simdat$data_train$y) + 15
```

```
updated_mod <- update(mod,
                      formula = cbind(y, trials) ~ s(season, bs = 'cc'),
                      noncentred = TRUE,
                      data = simdat$data_train,
                      family = binomial())
summary(updated_mod)
conditional_effects(updated_mod, type = 'link')
```

Index