Package: GHRmodel (via r-universe)

June 5, 2026

Title Bayesian Hierarchical Modelling of Spatio-Temporal Health Data

Version 0.1.1

Description Supports modeling health outcomes using Bayesian hierarchical spatio-temporal models with complex covariate effects (e.g., linear, non-linear, interactions, distributed lag linear and non-linear models) in the 'INLA' framework. It is designed to help users identify key drivers and predictors of disease risk by enabling streamlined model exploration, comparison, and visualization of complex covariate effects. See an application of the modelling framework in Lowe, Lee, O'Reilly et al. (2021) <doi:10.1016/S2542-5196(20)30292-8>.

License GPL (>= 2)

URL https://gitlab.earth.bsc.es/ghr/ghrmodel, https://bsc-es.github.io/GHRtools/docs/GHRmodel/GHRmodel

BugReports https://gitlab.earth.bsc.es/ghr/ghrmodel/-/issues

Depends R (>= 4.1.0)

Imports cowplot, dlnm, dplyr, ggplot2 (>= 3.5.0), GHRexplore, rlang, scales, tidyr, tidyselect

Suggests INLA, sf, sn, RColorBrewer, colorspace, testthat (>= 3.0.0), spdep, knitr, rmarkdown

Additional_repositories https://inla.r-inla-download.org/R/stable

Config/testthat/edition 3

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Author Carles Milà [aut, cre] (ORCID: <https://orcid.org/0000-0003-0470-0760>), Giovenale Moirano [aut] (ORCID: <https://orcid.org/0000-0001-8748-3321>), Anna B.

Kawiecki [aut] (ORCID:
<https://orcid.org/0000-0002-0499-2612>), Rachel Lowe [aut]
(ORCID: <https://orcid.org/0000-0003-3939-7343>)

Maintainer Carles Milà <carles.milagarcia@bsc.es>

Repository https://cran.r-universe.dev

Date/Publication 2025-11-07 16:41:19 UTC

RemoteUrl https://github.com/cran/GHRmodel

RemoteRef HEAD

RemoteSha d994990ac7c1d05cb9626e666f6fcb75605392de

Contents

as_GHRformulas *Convert R-INLA Model Formulas into a GHRformulas Object*

Description

This function converts a character vector of suitable R-INLA formulas into a structured GHRformulas object. The GHRformulas object contains the standardized information about the fixed effects, the random effects, and the outcome variable, ensuring consistency across multiple models to be fitted using the fit_models function.

Usage

```
as_GHRformulas(formulas)
```

Arguments

formulas	A character vector of model formulas formatted for R-INLA. Each formula must contain a single ~ separating the outcome variable from the predictors. Formulas generated with write_inla_formulas are compatible with this function.

Details

The as_GHRformulas() function parses each input formula to extract the outcome variable, fixed effects (covariates), and random effects. The resulting GHRformulas object is designed to be used with the fit_models function for model fitting with R-INLA.

Value

A structured list of class GHRformulas with the following components:

formulas A character vector of the original INLA-compatible model formulas.

vars A data frame where each row corresponds to a formula and each column to a covariate. Entries indicate whether a covariate is included in the formula.

re A character vector listing the random effects specified across all formulas.

outcome A character string indicating the outcome variable (must be consistent across formulas).

See Also

write_inla_formulas to generate R-INLA compatible input formulas

Examples

```
# Define formulas
formulas <- c(
"dengue_cases ~ 1 + f(month_id, model = 'rw1')",
"dengue_cases ~ 1 + f(month_id, model = 'rw1') + tmin.l1")

# Convert the formulas into a GHRformulas object
```

```
formulas <- as_GHRformulas(formulas)

# Inspect the structured GHRformulas object
print(formulas)
# Visualize output: GHRformulas object
class(formulas)
```

---

cov_add *Add Covariates to All Combinations*

Description

This function appends one or more covariate names to all elements (i.e., covariate sets) in a list of character vectors. This is useful when a covariate (like a confounder or control variable) needs to be included in every model. It also works with a single character vector input. The resulting list can be input into the `covariates` argument in `write_inla_formulas`.

Usage

```
cov_add(covariates, name, add = FALSE)
```

Arguments

`covariates`	A character vector or a list of character vectors, where each vector represents a set of covariates (e.g., from `cov_multi`).
`name`	A character vector of covariate names to be added to each set.
`add`	Boolean that indicates if the original combinations in the `covariates` argument must be kept. Defaults to FALSE.

Value

A list of character vectors, with each vector containing the original covariates plus the additional ones specified in the `name` argument.

Examples

```
# Multiple combinations
cov_sets <- list(
  c("tmin", "pdsi"),
  c("tmin.l1", "pdsi"),
  c("tmin.l2", "pdsi")
)
cov_add(cov_sets, name = "urban")
```

cov_interact *Generate Interaction Terms Between Covariates*

Description

This function generates interaction terms between covariates specified in the `pattern` or `name` arguments. It requires a list of character vectors and appends interaction terms to each vector based on pairwise or three-way interactions. The resulting list can be input into the `covariates` argument in `write_inla_formulas`.

Usage

```
cov_interact(covariates = NULL, pattern = NULL, name = NULL, add = FALSE)
```

Arguments

`covariates`	A list of character vectors, each vector containing variable names. Typically an output of `cov_multi` or `cov_uni`.
`pattern`	A character vector of length 2 or 3 specifying prefixes of variables to interact (e.g., "tmin" matches "tmin", "tmin.l1", etc.).
`name`	A character vector specifying the exact variable names to be included in the interactions.
`add`	Logical; if `TRUE`, appends the newly created formulas to the original list. Default is `FALSE`.

Details

- If two variables are matched, their pairwise interaction is added (`var1:var2`).
- If three variables are matched, two-way and three-way interactions are generated.
- Only variables that are expressed as linear terms can be used in interactions.
- Use either `pattern`, `name`, or both to identify variables for interaction.

Value

A list of character vectors, where each vector includes covariates and their corresponding interaction terms. This object can be passed to the `covariates` argument in `write_inla_formulas`.

Examples

```
# Example dataset
data <- data.frame(tmin.l1 = rnorm(10), pdsi.l1 = rnorm(10), urban = rnorm(10))

# Extract names
covs <- extract_names(data, pattern = c("tmin", "pdsi", "urban"))

# Create combinations
```

```
combos <- cov_multi(covariates = covs, pattern = c("tmin", "pdsi"))

# Add interaction terms
cov_interact(covariates = combos, pattern = c("tmin", "pdsi"))

# Output can be passed to write_inla_formulas()
new_covs <- cov_interact(combos, pattern = c("tmin", "pdsi"))
formulas <- write_inla_formulas(outcome = "cases", covariates = new_covs)
```

`cov_multi` *Create Covariate Combinations Across Groups*

Description

This function generates all possible combinations of covariates by selecting one variable from each user-defined group. Groups can be defined either by a regular expression pattern (`pattern`) or by exact variable names (`name`). The resulting list can be input into the `covariates` argument in `write_inla_formulas` to generate multivariable model formulas where all combinations of covariates are needed.

Usage

```
cov_multi(covariates, pattern = NULL, name = NULL, add = FALSE)
```

Arguments

`covariates`	A character vector or a list of single-element character vectors. Typically obtained from `extract_names` or `cov_uni` or `cov_nl`.
`pattern`	A character vector of regular expression patterns (e.g., "tmin" matches "tmin", "tmin.l1", etc.). Each pattern defines a group to draw covariates from.
`name`	A character vector of exact variable names to include as an additional group.
`add`	Logical; if `TRUE`, appends the generated combinations to the original `covariates` object. Default is `FALSE`.

Value

A list of character vectors. Each element is a unique combination of covariates, where one variable is drawn from each specified group. The resulting list is suitable as input in the `covariates` argument in `write_inla_formulas`.

Examples

```
data <- data.frame(tmin = rnorm(10), tmin.l1 = rnorm(10),
                   pdsi = rnorm(10), urban = rnorm(10))

# Extract covariate names
covs <- extract_names(data, pattern = c("tmin", "pdsi", "urban"))
```

```
# Combine "tmin" and "pdsi" into all possible pairings
cov_multi(covariates = covs, pattern = c("tmin", "pdsi"))

# Combine "tmin" and "urban", treating "urban" as an exact match
cov_multi(covariates = covs, pattern = "tmin", name = "urban")

# Use output as input to write_inla_formulas()
combined_covs <- cov_multi(covariates = covs, pattern = c("tmin", "pdsi"))
formulas <- write_inla_formulas(outcome = "cases", covariates = combined_covs)
```

`cov_nl` *Create Non-Linear Effects for INLA*

Description

This function transforms selected covariates identified by `pattern` or `name` into non-linear terms using INLA's `f()` syntax. It supports random walk models (`rw1`, `rw2`) and allows discretization by quantiles or equal intervals. Transformed covariates are returned as character vectors inside a list ready to be passed to the `write_inla_formulas` function.

Usage

```
cov_nl(
  covariates,
  pattern = NULL,
  name = NULL,
  model = "rw2",
  method = "quantile",
  n = 10,
  replicate = NULL,
  add = FALSE
)
```

Arguments

`covariates`	A character vector or list of character vectors. Usually from `cov_multi` or `cov_uni`.
`pattern`	Character vector of patterns to match covariates for transformation (e.g., "tmin" matches "tmin", "tmin.l1", etc.).
`name`	Character vector of exact covariate names to transform.
`model`	Character; either `"rw1"` or `"rw2"` to specify the non-linear INLA model.
`method`	Character; either `"cut"` or `"quantile"` for discretization. Default is `"quantile"`.
`n`	Integer; number of intervals or quantile bins. Must be >= 2. Default is 10.
`replicate`	Optional character string indicating a replicate structure for non-linear effects.
`add`	Logical; if `TRUE`, adds the transformed covariates to the original ones. Default is `FALSE`.

Details

- Use `pattern` or `name` (or both) to specify which variables to transform.
- The method and n arguments discretize the covariate into evenly populated bins.
- The function supports discretization with either equal-width (`cut`) or quantile-based (`quantile`) bins.
- The model argument imposes smoothness on the grouped effect, capturing non-linear trends.
- Non-linear effects are created using `.single_non_linear_eff_inla()` (internal helper).

Value

A list of character vectors. This object can be passed to the `covariates` argument in `write_inla_formulas`.

See Also

See Bayesian inference with INLA: Smoothing for more information on smoothing and non-linear effects in R-INLA models.

Examples

```
data <- data.frame(tmin.l1 = rnorm(10), pdsi.l1 = rnorm(10))

covs <- extract_names(data, pattern = c("tmin", "pdsi"))
covlist <- cov_multi(covs, pattern = c("tmin", "pdsi"))

# Apply non-linear transformation to tmin variables
cov_nl(covlist, pattern = "tmin", model = "rw2")

# Include original variables along with transformed ones
cov_nl(covlist, pattern = "tmin", model = "rw2", add = TRUE)
```

`cov_uni` *Build Univariable Covariate Sets*

Description

This function returns a list where each element contains a single covariate, based on covariates specified in the `pattern` or `name` arguments. This structure is suitable for generating separate univariable model formulas using `write_inla_formulas`.

Usage

```
cov_uni(covariates = NULL, pattern = NULL, name = NULL)
```

Arguments

covariates	A character vector of covariate names. Typically the output from extract_names.
pattern	A character vector specifying the prefix pattern(s) to match (e.g., "tmin" matches "tmin", "tmin.l1", etc.).
name	A character vector specifying exact variable name(s) to extract.

Value

A list of character vectors, each of length 1, containing the matched covariate name. The resulting list is suitable for use as the covariates argument in write_inla_formulas.

Examples

```
data <- data.frame(tmin = rnorm(10), tmin.l1 = rnorm(10), urban = rnorm(10))
covs <- extract_names(data, pattern = "tmin", name = "urban")
cov_uni(covs, pattern = "tmin")
cov_uni(covs, name = "urban")
```

---

cov_varying *Create Spatially or Temporally Varying Effects for INLA*

Description

This function transforms covariates identified by pattern or name into varying effect terms of the form:f(unit, covariate, model = 'iid'), which allows covariates to have varying slopes across spatial or temporal units. The output can be used directly in the covariates argument of write_inla_formulas.

Usage

```
cov_varying(
  covariates,
  unit,
  pattern = NULL,
  name = NULL,
  model = "iid",
  constr = FALSE,
  add = FALSE
)
```

Arguments

covariates	A character vector or a list of character vectors of covariate names. Typically output from cov_multi, cov_uni, or extract_names.
unit	Character string specifying the unit of variation (e.g., "spat_id", "year").

pattern	A character vector specifying the prefix pattern(s) to match (e.g., "tmin" matches "tmin", "tmin.l1", etc.) for transformation.
name	Character vector of exact variable names to be transformed.
model	Character string specifying the INLA model for the varying effect. Currently, only "iid" is supported.
constr	Logical. If TRUE it will impose a sum-to-zero constraint to the random effect. Default is FALSE.
add	Logical; if TRUE, appends the transformed covariates to the original ones. Default is FALSE.

Details

- Use pattern or name (or both) to specify which covariates to transform.
- The resulting terms use INLA's f() syntax: f(unit, covariate, model = "iid").
- Currently only supports "iid" models for varying effects.

Value

A list of character vectors, each including covariates with varying effects. The output is suitable as input for write_inla_formulas.

Examples

```
data <- data.frame(tmin.l1 = rnorm(10), pdsi.l1 = rnorm(10))

covs <- extract_names(data, pattern = c("tmin", "pdsi"))
covlist <- cov_multi(covs, pattern = c("tmin", "pdsi"))

# Apply varying effect to tmin
cov_varying(covlist, pattern = "tmin", unit = "spat_id")

# Keep original and add varying effect terms
cov_varying(covlist, pattern = "tmin", unit = "spat_id", add = TRUE)
```

---

crossbasis_inla *Create a Two-Dimensional INLA-compatible Cross-basis Matrix*

Description

This function is a wrapper around dlnm::crossbasis to generate cross-basis matrices that capture nonlinear effects of a predictor across both exposure and lag dimensions. The input covariate is passed as a numeric matrix of lagged values, and the resulting columns can be renamed via basis_name for easier reference in model formulas.

Usage

```
crossbasis_inla(
  covariate,
  basis_name,
  lag,
  argvar = list(),
  arglag = list(),
  ...
)
```

Arguments

covariate	A numeric matrix of covariate values. Typically this will be a matrix of lagged covariate values (which can be generated using lag_cov).
basis_name	A character string specifying the prefix for the spline columns in the resulting basis matrix (replacing the default "v").
lag	A numeric vector with min and max lag of the matrix (as in crossbasis).
argvar	A list specifying the shape of the exposure-response function (as in crossbasis).
arglag	A list specifying the shape of the lag-response function (as in crossbasis).
...	Additional arguments passed to dlnm::crossbasis, such as df, degree, knots, etc.

Value

An object of class "crossbasis_inla" (also inheriting class "crossbasis"), as returned by dlnm::crossbasis() but with customized column names.

Examples

```
# Build cross-basis with a custom prefix for columns

# Import example data set
data("dengue_MS")

lag_mat <- lag_cov(data = dengue_MS,
  name = c("tmin"),
  time = "date",
  lag = c(1:6),
  group = "micro_code",
  add = FALSE) # add = FALSE return only the lagged matrix

cb_inla <- crossbasis_inla(
  covariate  = lag_mat,
  basis_name = "tempLag",
  lag = c(1,6),
  argvar = list(fun = "bs", df = 3),
  arglag = list(fun = "poly", degree = 2)
)
```

```
# Check class of the cross-basis object
class(cb_inla)

# View resulting cross-basis matrix
head(colnames(cb_inla))
```

`crosspred_inla` *Generate DLNM Predictions from* `GHRmodels` *Objects*

Description

This function takes an object of class `GHRmodels`, extracts the relevant coefficients and variance-covariance matrix, and then calls dlnm::crosspred to compute predictions over a range of covariate values (or at specified points).

Usage

```
crosspred_inla(
  models,
  basis,
  mod_id,
  at = NULL,
  from = NULL,
  to = NULL,
  by = NULL,
  lag,
  bylag = 1,
  cen = NULL,
  ci.level = 0.95,
  cumul = FALSE,
  ...
)
```

Arguments

`models`	An object of class `GHRmodels`, containing fitted model output (e.g., `$fixed` and `$vcov` lists).
`basis`	A cross-basis or one-basis object, typically created by `crossbasis_inla` or `onebasis_inla`.
`mod_id`	An integer or character string specifying which model within the input `GHRmodels` object to use (e.g., if `model$fixed` and `model$vcov` both have multiple entries).
`at`	A numeric vector of values at which to compute predictions (e.g.,`seq(10,25, by=0.2)`)
`from, to`	Numeric values specifying the range of the prediction sequence if `at` is not specified (e.g., `from = 10` and `to = 25`).

by	Numeric increment for the sequence if at is not specified (e.g., by = 0.2).
lag	A vector of two elements with min and max lag as declared in the crossbasis_inla function.
bylag	Numeric increment for lag steps (default is 1).
cen	A centering value (e.g., a reference exposure level).
ci.level	The credible interval level (default 0.95).
cumul	Logical; if TRUE, cumulative predictions are computed (default FALSE).
...	Additional arguments passed on to crosspred, such as bound, ci.arg, etc.

Details

The function identifies which coefficients in model$fixed[mod_id] and which rows/columns in model$vcov[mod_id] correspond to the one-basis or cross-basis terms (i.e., matching the column names in basis). Then it passes these slices to dlnm::crosspred to generate predictions. The centering value (cen), if specified, indicates the reference exposure (e.g., a mean temperature) at which to center the effect estimates (e.g., the effect a given temperature value on the outcome will be compared to the effect of the centering value on the outcome, in this case the mean temperature).

Value

An object of class "GHRcrosspred", inheriting from "crosspred", with fields for the predicted values, credible intervals, and optionally cumulative predictions, as determined by crosspred.

See Also

dlnm::crosspred for details on how predictions are computed.

Examples

```
# Load example GHRmodels object from the package
model_dlnm_file <- system.file("examples", "model_dlnm.rds", package = "GHRmodel")
model_dlnm <- readRDS(model_dlnm_file)

# Load example cross-basis matrix from the package: 2-dimensional cross-basis matrix of the
# non-linear effect of dengue risk across tmin values and lags:
cb_tmin_file <- system.file("examples","cb_tmin.rds", package = "GHRmodel")
cb_tmin <- readRDS(cb_tmin_file) # loads cross-basis matrix into the environment

# Generate predictions
pred_result <- crosspred_inla(
  models    = model_dlnm,
  basis    = cb_tmin,
  mod_id = "mod3",
  at       = seq(17, 24, by = 1),  # e.g., temperature sequence
  lag      = 2,
  cen      = 20,
  ci.level = 0.95
)
```

```
# Inspect predictions
pred_result$predvar  # the sequence of 'at' values
pred_result$allfit   # fitted values
pred_result$alllow   # lower CI
pred_result$allhigh  # upper CI
```

---

dengue_MS *Dengue cases from the "Mato Grosso do Sul" state of Brazil*

Description

The dengue_MS example data set contains monthly counts of notified dengue cases by microregion, along with a range of spatial and spatiotemporal covariates (e.g., environmental, socio-economic and meteo-climatic factors). This data set represents a subset of a larger national data set that covers the entire territory of Brazil. The subset focuses on a specific region, *Mato Grosso do Sul*, for the purposes of illustration and computational efficiency. See @source for access to the complete data set.

Usage

```
dengue_MS
```

Format

A data frame with 2,600 rows and 27 columns:

micro_code Unique ID number to each micro region (11 units)

micro_name Name of each micro region

micro_name_ibge Name of each micro region following IBGE

meso_code Unique ID number to each meso region (4 units)

meso_name Name of each meso region

state_code Unique ID number to each state (1 unit)

state_name Name of each state

region_code Unique ID number given to each Brazilian Region, In this data frame all observations come from the "Southeast Region"

region_name Name of each Brazilian Region, In this data frame all observations come from the "Southeast Region"

biome_code Biome code

biome_name Biome name

ecozone_code Ecozone code

ecozone_name Ecozone name

main_climate Most prevalent climate regime in the microregion. Based on Koppen Geiger climate regimes

month Calendar month index, 1 = January, 12 = December

`year` Year 2000 - 2019

`time` Time index starting at 1 for January 2000

`dengue_cases` Number of notified dengue cases registered in the notifiable diseases system in Brazil (SINAN) in the microregion of reference, at the month of first symptoms

`population` Estimated population, based on projections calculated using the 2000 and 2010 censuses, and counts taken in 2007 and 2017

`pop_density` Population density (number of people per km2)

`tmax` Monthly average daily maximum temperature; gridded values (at a 0.5 deg resolution) averaged across each microregion

`tmin` Monthly average daily minimum temperature; gridded values (at a 0.5 deg resolution) averaged across each microregion

`pdsi` Self-calibrated Palmer drought severity index for each microregion. It measures how wet or dry a region is relative to usual conditions. Negative values represent periods of drought, positive values represent wetter periods. Calculated by taking the mean value within each microregion

`urban` Percentage of inhabitants living in urban areas (2010 census)

`water_network` Percentage of inhabitants with access to the piped water network according to the 2010 census

`water_shortage` Frequency of reported water shortages per microregion between 2000 - 2016

`date` First day of the Month, in date format ("%d-%m-%Y")

Source

source code on GitHub; source code on Zenodo;

`dengue_SP` *Dengue cases from the "São Paulo" state of Brazil*

Description

The `dengue_SP` example data set reports the weekly number of notified dengue cases in the municipality of São Paulo together with climatic covariates. Data was sourced from Infodengue (see `@source`).

Usage

```
dengue_SP
```

Format

A data frame with 678 rows and 8 columns:

`date` First day of the week, in date format ("%d-%m-%Y")

`geocode` Unique ID code for São Paulo microregion

`cases` Number of notified dengue cases

`year` Year 2000 - 2022

`temp_med` Weekly average daily mean temperature

`precip_tot` Weekly cumulative precipitation

`enso` El Niño-Southern Oscillation Index

`pop` Number of inhabitants

Source

Infodengue API

---

extract_names *Extract Covariate Names*

Description

This function allows the user to select variables from a data set by prefix (using the `pattern` argument) or by exact name matching. The return object is a character vector with the selected covariate names that can be used as input for `cov_add`, `cov_uni`, `cov_multi`, `cov_interact`, `cov_nl`, and `cov_varying` functions.

Usage

```
extract_names(data = NULL, pattern = NULL, name = NULL)
```

Arguments

`data`	A `data.frame` containing the variables.
`pattern`	A character vector specifying prefix(es) to match (e.g., "tmin" matches "tmin", "tmin.l1", etc.).
`name`	A character vector of exact variable name(s) to extract.

Value

A character vector of matched covariate names.

Examples

```
data <- data.frame(tmin = 1:10, tmin.l1 = 1:10, urban = 1:10)
extract_names(data, pattern = "tmin")
extract_names(data, name = "urban")
extract_names(data, pattern = "tmin", name = "urban")
```

`fit_models` *Fit Multiple INLA Models*

Description

This function fits a set of `INLA` model formulas, provided in a `GHRformulas` object, to a specified dataset. For each fitted model, it extracts a range of outputs, including goodness-of-fit (GoF) metrics and other model outputs (fitted values, fixed effects, random effects). Results are extracted and stored in a `GHRmodels` object.

Usage

```
fit_models(
  formulas,
  data,
  family,
  name,
  offset = NULL,
  control_compute = list(config = FALSE, vcov = FALSE),
  nthreads = 8,
  pb = FALSE
)
```

Arguments

`formulas`	A `GHRformulas` object containing multiple INLA model formulas.
`data`	A data frame containing the variables used in the model formulas.
`family`	A character string specifying the likelihood family (e.g., `"poisson"`, `"nbinomial"`, etc.).
`name`	A character string to label each fitted model (e.g., `"mod"`).
`offset`	A character string specifying the name of the offset variable in `data`. If `NULL`, no offset is applied. Default is `NULL`. Internally, `log(offset_values)` is applied.
`control_compute`	A named list controlling additional computation options: `config` Logical ; if `TRUE`, stores the Gaussian Markov Random Field (GMRF) and enables the computation of posterior predictive distribution (1,000 draws). Defaults to `FALSE`. `vcov` Logical if `TRUE`, returns the variance-covariance (correlation) matrix of fixed effects. Defaults to `FALSE`.
`nthreads`	An integer specifying the number of threads for parallel computation. Default is `8`.
`pb`	Logical; if `TRUE`, displays a progress bar while fitting models. Default is `FALSE`.

Details

This function iterates over each formula in the `GHRformulas` object and fits the corresponding `INLA` model using the internal function `.fit_single_model()`. For each fitted model, it extracts the fitted values, fixed effects, and random effects summaries. Then, it calculates a series of model evaluation metrics using the `.gof_single_model()` internal function.

The goodness-of-fit (GoF) metrics are organized into two categories:

A) Model-Specific Goodness-of-Fit Metrics

These are computed separately for each model:

1. **Deviance Information Criterion (DIC)**

$$DIC = \bar{D} + p_D$$

where $\bar{D}$ is the posterior mean deviance and p_D is the effective number of parameters. Lower DIC values indicate a better model fit, balancing goodness-of-fit and model complexity.

2. **Watanabe-Akaike Information Criterion (WAIC)**

$$WAIC = -2\left(\text{lppd} - p_{\text{WAIC}}\right)$$

WAIC evaluates predictive accuracy and penalizes model complexity through the log pointwise predictive density (lppd). Lower values imply better generalization.

3. **Log Mean Score (LMS)**

$$LMS = \frac{1}{n}\sum_{i=1}^{n}\left(-\log(\text{CPO}_i)\right)$$

LMS assesses the average negative log-predictive density using Conditional Predictive Ordinates (CPO). Lower LMS values indicate stronger predictive performance by penalizing models that assign low probability to observed outcomes.

4. **Mean Absolute Error (MAE)**

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Measures the average absolute deviation between observed values y_i and predicted values $\hat{y}_i$. Lower MAE values indicate improved fit. If `config = TRUE`, MAE is computed using the full posterior predictive distribution (PPD); otherwise, it uses point estimates from INLA's `summary.fitted.values`.

5. **Root Mean Squared Error (RMSE)**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Captures average squared deviation between observed and predicted values. RMSE penalizes larger errors more heavily. Lower values reflect better model fit. If `config = TRUE`, RMSE uses the PPD; otherwise, it uses point estimates.

6. **Continuous Ranked Probability Score (CRPS)**

$$\mathrm{CRPS}(F, y) = \int_{-\infty}^{\infty} \left[F(t) - \mathbf{1}\{y \leq t\}\right]^2 dt$$

CRPS assesses how well the predictive cumulative distribution aligns with the observed outcome. Lower scores suggest better calibrated predictive distributions. Only available when `config = TRUE`.

B) Model Comparison Metrics (relative to the first model)

The first model in the list is treated as the baseline for model comparisons. All other models are evaluated against it using the following metrics:

1. **Difference in DIC and WAIC** Stored as `dic_vs_first` and `waic_vs_first`. These represent how much higher (or lower) each model's DIC/WAIC is compared to the first model. Additionally, 95% credible intervals for these differences are stored as `*_vs_first_lci` and `*_vs_first_uci`.
2. **Difference in MAE and RMSE** Stored as `mae_vs_first` and `rmse_vs_first`. These reflect the absolute difference in prediction error compared to the first model. No credible intervals are computed for these metrics.
3. **Continuous Ranked Probability Score Skill Score (CRPSS)**

$$\mathrm{CRPSS} = 1 - \frac{\mathrm{CRPS}_{\text{model}}}{\mathrm{CRPS}_{\text{baseline}}}$$

Indicates how much better the predictive distribution of the current model is relative to the baseline model. Values closer to 1 indicate improvement; negative values imply worse performance. Available only when `config = TRUE`.

4. **Pseudo R-squared based on deviance**

$$R^2 = 1 - \exp\left(\frac{-2}{n}\left(\frac{dev_{\text{model}}}{-2} - \frac{dev_{\text{base}}}{-2}\right)\right)$$

Captures relative deviance reduction compared to the baseline model. Values range from 0 (no improvement) to 1 (strong improvement).

5. **Random Effect Variance**

$$\mathrm{Var}_{re} = \frac{1}{\text{precision}}$$

Quantifies residual variance due to group- or cluster-level effects. Computed only when random effects are defined in the model formula.

6. **Proportional Change in Random Effect Variance**

$$\frac{\mathrm{Var}_{re}}{\mathrm{Var}_{re}^{(1)}} - 1$$

Represents the relative change in group-level variance compared to the baseline model. Helps assess how much variance is explained by added covariates.

Value

An object of class `GHRmodels` containing:

`$mod_gof` A data frame of model-specific goodness-of-fit metrics.

`$fitted` A list of fitted values (one element per model). If `config = TRUE`, these are derived from the posterior predictive distribution (PPD); otherwise, they are extracted from INLA's `summary.fitted.values`.

`$fixed` A list of summary tables for fixed effects (one element per model).

`$random` A list of summary tables for random effects (one element per model).

`$formulas` A character vector of the original model formulas used.

`$re` A character vector specifying any random effects defined in `formulas`.

`$outcome` A character string indicating the outcome variable used.

`$data` The original data frame passed to the function.

See Also

`as_GHRformulas` converts a set of R-INLA-compatible formulas into a `GHRformulas` object.

Examples

```
# Load example dataset
data(dengueMS)

# Declare formulas
formulas <- c(
  "dengue_cases ~ tmin +  f(year, model='rw1')",
  "dengue_cases ~ pdsi +  f(year, model='rw1')"
)

# Tranform formulas into a 'GHRformulas' object
ghr_formulas <- as_GHRformulas(formulas)

# Fit multiple models
results <- fit_models(
  formulas = ghr_formulas,
  data     = dengue_MS,
  family   = "nbinomial",
  name     = "TestModel",
  offset   = "population",
  nthreads = 2,
  control_compute = list(config = FALSE),
  pb       = TRUE
)

# Inspect goodness-of-fit metrics
results$mod_gof
```

get_covariates *Retrieve Covariates from a* GHRmodels *Object as a List of Character Vectors*

Description

Extracts covariates from a GHRmodels object and returns them as a list of character vectors. If unique = TRUE, the output contains unique covariates across models. If unique = FALSE, the output preserves the original combinations of covariates as specified in the GHRmodels object.

Usage

```
get_covariates(model, unique = TRUE)
```

Arguments

model	A GHRmodels object containing fitted models.
unique	Logical; if TRUE, returns unique covariates across models. If FALSE, returns vectors of covariate combinations as declared in the GHRmodels object.

Value

A list of character vectors.

Examples

```
# Load example dataset
data(dengueMS)

# Declare formulas
formulas <- c(
  "dengue_cases ~ tmin +  f(year, model='rw1')",
  "dengue_cases ~ pdsi +  f(year, model='rw1')"
)

# Tranform formulas into a 'GHRformulas' object
ghr_formulas <- as_GHRformulas(formulas)

# Fit multiple models
results <- fit_models(
  formulas = ghr_formulas,
  data     = dengue_MS,
  family   = "nbinomial",
  name     = "TestModel",
  offset   = "population",
  nthreads = 2,
  control_compute = list(config = FALSE),
  pb       = TRUE
)
```

```
# Extract the list of covariates from the models
get_covariates(results)
```

---

lag_cov *Generate lagged variables for one or more lags*

Description

This function creates lagged versions of one or more numeric or categorical variables in an equally spaced time-series data set. A single call can create multiple lags for each selected variable and, optionally, for each spatial/grouping unit.

Usage

```
lag_cov(data, name, time, lag, group = NULL, add = TRUE)
```

Arguments

data	A data.frame containing equally spaced observations.
name	A character vector: name of the variable (or variables) to lag.
time	A single character string: name of the time-index variable (e.g., "date").
lag	A numeric vector of one or more positive integers. Each value is interpreted as a 'lag' (i.e. shift the series backward by k observations).
group	Optional character vector naming column(s) that define independent time-series (e.g. regions). If NULL, the whole data set is treated as one series.
add	Logical. If TRUE (default) the lagged columns are appended to data; if FALSE the function returns only the lagged columns as a matrix.

Value

Either a data frame (when add = TRUE) containing the original data plus the new lagged columns, or a numeric matrix of lagged values (when add = FALSE).

Examples

```
## Daily series for two micro-regions
d <- data.frame(
  date       = as.Date("2023-01-01") + 0:9,
  micro_code = rep(c("A", "B"), each = 5),
  tmin       = rnorm(10, 10, 2),
  pdsi       = rnorm(10)
)

## Create lags 1 to 3 for tmin and pdsi
lagged <- lag_cov(
```

```
  data  = d,
  name   = c("tmin", "pdsi"),
  time  = "date",
  group = "micro_code",
  lag   = c(1:3)
)

## Only lagged columns (matrix),
lag_only <- lag_cov(
  data = d, name = "tmin", time = "date",
  lag  = c(1:3), add = FALSE
)
```

`map_MS` *Administrative Map for Municipalities in the Mato Grosso do Sul*

Description

A simple feature (`sf`) multipolygon object representing a map of *Mato Grosso do Sul*, Brazil, including 11 municipalities. See `@source` for access to the complete data set.

Usage

```
map_MS
```

Format

A simple feature (sf) object including 11 rows and 2 columns:

`$code` Unique ID number for each micro region (11 units)

`$geometry` geometries of the sf multipolygon

Source

source code on GitHub; source code on Zenodo;

`onebasis_inla` *Create a One-Dimensional Basis for INLA*

Description

This function is a wrapper around onebasis to create a one-dimensional basis for spline modeling. This wrapper enhances the original function by allowing users to specify a custom prefix for the column names using the `basis_name` argument, such that each set of basis variables can be easily identified in the model formula by the **INLA** framework.

Usage

```
onebasis_inla(covariate, fun, basis_name, ...)
```

Arguments

covariate	A numeric vector representing the covariate
fun	A character string specifying the shape function to be used by onebasis.
basis_name	A character string giving a base name for the columns in the resulting basis matrix. The default prefix (usually "b") is replaced by this string.
...	Additional arguments passed to onebasis, such as degree, df, knots, etc.

Value

An object of class "onebasis", as returned by onebasis, with column names modified according to basis_name.

Examples

```
# Import example data set
data("dengue_MS")

# Build a one-dimensional spline basis with a custom name
ob_inla <- onebasis_inla(
 covariate = dengue_MS$tmin,
 fun = "bs",
 basis_name = "tempBasis",
 degree = 2
)

# Check class of the one-basis object
class(ob_inla)

# View first rows of the one-basis matrix
head(ob_inla)
```

---

plot_coef_crosspred *Plot* crosspred *Objects: Overall, Slices, or Heatmap*

Description

Generate plots from a "crosspred" object. Three plot types are available:

- type = "overall": Shows the overall exposure–response relationship, aggregated across all lags.
- type = "slices": Produces line plots with credible interval ribbons, either across lags (for a fixed var) or across values of var (for a fixed lag).
- type = "heatmap": Displays a two-dimensional heatmap of effects across both var and lag. Not applicable for one-basis models.

Usage

```
plot_coef_crosspred(
  crosspred,
  type = c("heatmap", "slices", "overall"),
  var = NULL,
  lag = NULL,
  exp = FALSE,
  palette = "-RdBu",
  n_lag_smooth = 50,
  line_color = "black",
  line_size = 0.7,
  ribbon_color = NULL,
  ribbon_alpha = 0.2,
  title = "",
  ylab = NULL,
  xlab = NULL,
  ...
)
```

Arguments

crosspred	An object of class "crosspred" or "GHR_crosspred", produced by crosspred or crosspred_inla.
type	Character string. Options: "overall", "slices", or "heatmap".
var	Optional numeric vector of exposure values (used when type = "slices" to plot across lags).
lag	Optional numeric vector of lag values (used when type = "slices" to plot across variables).
exp	Logical. If TRUE, exponentiates the results (e.g., for log or logit links).
palette	Character string for heatmap palette when type = "heatmap". Options: GHR, RColorBrewer or colorspace palette (e.g. "Purp").
n_lag_smooth	Integer, number of interpolation points along lag for heatmap smoothing (default = 50).
line_color	Character string. Line color when type = "slices" or type = "overall". Default is "black".
line_size	Numeric. Line width (default = 0.7).
ribbon_color	Character string. Color for credible interval ribbons. Defaults to line_color.
ribbon_alpha	Numeric. Alpha transparency for ribbons (default = 0.2).
title	Character string. Plot title.
ylab	Character string. Label for y-axis.
xlab	Character string. Label for x-axis.
...	Additional arguments passed to ggplot2 functions.

Value

A ggplot object for the specified plot type.

See Also

crosspred

Examples

```
# Load example GHRmodels object from the package
model_dlnm_file <- system.file("examples", "model_dlnm.rds", package = "GHRmodel")
model_dlnm <- readRDS(model_dlnm_file)

# Load example cross-basis matrix from the package: 2-dimensional cross-basis matrix of the
# non-linear effect of dengue risk across tmin values and lags:
cb_tmin_file <- system.file("examples","cb_tmin.rds", package = "GHRmodel")
cb_tmin <- readRDS(cb_tmin_file) # loads cross-basis matrix into the environment

# Generate predictions
pred_result <- crosspred_inla(
  models    = model_dlnm,
  basis    = cb_tmin,
  mod_id = "mod3",
  at       = seq(17, 24, by = 1),  # e.g., temperature sequence
  lag      = 2,
  cen      = 20,
  ci.level = 0.95
)


# Plot DLNM predictions
plot_coef_crosspred(
crosspred = pred_result,    # Crosspred object with model predictions
type = "slices",           # Plot temperature-specific slices of exposure-response curves
exp = TRUE,                 # Exponentiate the coefficients (to relative risk scale)
var = c(22:24),             # Display results for temperature 22°C to 24°C
line_color = "red",         # Red color for the lines representing effect estimates
line_size = 0.8,            # Line thickness set to 0.8 for better visibility
ribbon_color = "red",       # Red shading for credible interval ribbons
ribbon_alpha = 0.3,         # Set ribbon transparency to 30%
title = "Effect of minimum temperatures 22°C to 23°C on dengue relative risk by lag",
xlab = "Lag",               # Label for the x-axis (exposure variable)
ylab = "Relative Risk (RR)" # Label for the y-axis (effect estimate scale)
)
```

---

plot_coef_lin *Produce a Forest Plot of Linear Covariates from a* GHRmodels *Object*

Description

This function extracts fixed-effect coefficients from a specified model in `models`, filters them by name or interaction pattern, and produces a forest plot (point estimates with error bars).

- If `name = NULL`, all fixed-effect terms (excluding the intercept) are shown.
- If `name` is a character vector, only the matching terms are included.

Usage

```
plot_coef_lin(
  models,
  mod_id = NULL,
  name = NULL,
  pattern = NULL,
  title = NULL,
  mod_label = NULL,
  var_label = NULL,
  palette = "IDE2",
  exp = FALSE,
  legend = "Model"
)
```

Arguments

`models`	An object of class `GHRmodels` containing fitted model output.
`mod_id`	Character vector of model identifiers (must match entries in `model$mod_gof$model_id`). If `NULL` (the default), all models are considered.
`name`	A character vector specifying exact linear covariates names to be plotted. If both `pattern` and `name` are `NULL` (the default), all terms (except `(Intercept)`) are plotted.
`pattern`	A character vector specifying prefix(es) to match (e.g., "tmin" matches "tmin", "tmin.l1", etc.) Covariates matching these patterns (case-insensitive search) will be plotted. If both `pattern` and `name` are `NULL` (the default), all terms (except `(Intercept)`) are plotted.
`title`	An optional string specifying an overall plot title.
`mod_label`	An optional named character vector mapping model names to custom labels, e.g. c("mod1" = "Model 1"). Any model not found in the vector names retains its original label.
`var_label`	An optional named character vector mapping variable (or interaction) names to custom labels. Interaction matching is order-insensitive: `"A:B"` matches `"B:A"`. Any term not found in the vector names retains its original label.
`palette`	GHR, RColorBrewer or colorspace palette (e.g. "Purp") colour palette to use for the different models. See all available options by running `GHR_palettes()`, `RColorBrewer::display.brewer.all()` and `colorspace::hcl_palettes(plot=TRUE)`. Single R colors in `colors()` or hex codes can also be used.
`exp`	Logical,if `TRUE` the coefficients are exponentiated, Default is if `FALSE`.
`legend`	Legend title for the replicate color scale. Default is `"Model"`.

Details

Intercept By default, `(Intercept)` is excluded unless explicitly included in `name`.

Individual terms e.g., `"temp"`.

Interaction Terms e.g. `"temp:precip"`. Split by `:`, sorted, and compared setwise; for example, `"temp:precip"` matches `"precip:temp"`.

Labels If `var_label` is supplied, any matched covariate or interaction string is replaced by its custom label on the y-axis.

Value

A **ggplot2** forest plot object (class `ggplot`).

See Also

`geom_pointrange` for the plotting environment.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Plot point estimates with confidence intervals for the linear covariates:
plot_coef_lin(
model = model_list,
mod_id = c("mod2","mod4"),
var_label = c("tmin.l1"= "Min. temp lag 1",
              "pdsi.l1" = "Drought index lag 1"),
title = "Effects of linear covariates"
)
```

---

plot_coef_nl *Plot Nonlinear Effects from a* `GHRmodels` *Object*

Description

Generates plots of nonlinear effects from one or more fitted models contained within a `GHRmodels` object. The function supports two main display modes:

- Grid (when `collapse = FALSE`): one plot per covariate and model, with effects by column and models by row.
  - If multiple models are specified, the user must provide either `name` or `pattern` to select which nonlinear effects to plot.
  - If only one model is selected and both `name` and `pattern` are `NULL`, all nonlinear effects in the model will be plotted.

- Collapsed (when `collapse = TRUE`): one non-linear effect combined across models into a single panel.
  - The user must explicitly specify the exact variable name using `name`. It only accepts one covariate name.
  - Collapse mode can only be used when the selected effect is not replicated (that is, does not have the format `f(covariate, model = ..., replicate = group)`) If replication is detected, an error will be thrown.

Usage

```
plot_coef_nl(
  models,
  mod_id,
  mod_label = NULL,
  name = NULL,
  pattern = NULL,
  title = NULL,
  var_label = NULL,
  palette = "IDE2",
  xlim = NULL,
  ylab = NULL,
  xlab = NULL,
  histogram = FALSE,
  legend = NULL,
  hist_fill = "grey",
  rug = FALSE,
  collapse = FALSE,
  exp = FALSE
)
```

Arguments

`models`	A `GHRmodels` object containing fitted model outputs.
`mod_id`	Integer vector specifying which model(s) to plot (as indexed in `model$models`).
`mod_label`	An optional named character vector mapping model names to custom labels, e.g. c("mod1" = "Model 1"). Any model not found in the vector names retains its original label.
`name`	Optional character vector of variable names (as used in `inla.group(...)`) to select specific nonlinear effects. Required for collapse mode.
`pattern`	Optional regular expression pattern to match effect names. Used to select nonlinear effects when `name` is not provided.
`title`	Optional overall title for the plot.
`var_label`	Optional named character vector providing custom labels for each nonlinear variable. Names must match the variable names (e.g., used in `inla.group(x)`), not full effect names.
`palette`	Name of the color palette to use (passed to `GHR_palette`). Default is `"IDE2"`.

xlim	Optional named list specifying x-axis limits for each effect. Each element should be a numeric vector of length 2: `list(var1 = c(min, max), var2 = c(min, max))`. Variable names must match those used in `inla.group()`.
ylab	Optional y-axis label. If `NULL`, defaults to `"Effect size"`.
xlab	Optional x-axis label. If `NULL`, defaults to `"<variable> values"`. If explicitly set to `NULL`, no x-axis label will be shown.
histogram	Logical; if `TRUE` (default), includes a histogram below each partial-effect plot.
legend	Legend title for the replicate color scale (if multi-replicate effects are present). Default is `"Replicate"`.
hist_fill	Fill color for histogram bars. Default is `"grey"`.
rug	Include a rug plot in the x-axis. Default is FALSE.
collapse	Logical; if `TRUE`, attempts to collapse plots across models to show one plot per variable. This requires that selected nonlinear effect is not replicated (i.e. the covariate is not in the format f(covariate, model = ..., replicate = group))
exp	Logical,if `TRUE` the coefficients are exponentiated, Default is if `FALSE`.

Value

A `ggplot` or `cowplot` object, depending on the plotting mode.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Plot 2 models with non-linear PDSI at one month lag in collapsed mode:
plot_coef_nl(
  models = model_list,
  mod_id = c( "mod5", "mod6") ,
  mod_label = c("mod6" = "pdsi.l1_nl",
                "mod5" = "pdsi.l1_nl + tmin.l1_nl"),
  var_label = c("pdsi.l1" = "Drought index (PDSI)"),
  name = c("pdsi.l1"),
  title = "Change in PDSI with and without mean min. temp lag 1",
  xlab = "PDSI",
  palette = "IDE2",
  collapse = TRUE
)
```

plot_coef_varying *Produce a Forest Plot for a Spatially or Temporally Varying Effects from a* GHRmodels *object.*

Description

Generates a forest plot for a specified spatially or temporally varying coefficient (i.e. a random slope) from a fitted GHRmodels object. The plot displays the effect estimates (x-axis) for each spatial/temporal unit (y-axis).

Usage

```
plot_coef_varying(
  models,
  mod_id,
  name,
  unit_label = NULL,
  palette = "IDE2",
  title = NULL,
  xlab = "Effect size",
  ylab = NULL,
  exp = FALSE
)
```

Arguments

models	A GHRmodels object containing fitted model output.
mod_id	A character specifying which model to be plotted (as in modelsmod_gofmodel_id).
name	A character string naming the spatially or temporally varying coefficient to plot. This should match a random effect name in models$random[[mod_id]].
unit_label	Optional named character vector providing custom labels for each spatial/temporal unit.
palette	Character string for the GHR, RColorBrewer or colorspace palette (e.g. "Purp") colour palette to use for the different models. See all available options by running GHR_palettes(), RColorBrewer::display.brewer.all() and colorspace::hcl_palettes(pl Single R colors in colors() or hex codes can also be used.
title	Optional string for the plot title.
xlab	Optional character string for the x-axis label (default = "Effect size").
ylab	Optional character string for the y-axis label (default constructed from varying covariate name).
exp	Logical,if TRUE the coefficients are exponentiated, Default is if FALSE.

Value

A ggplot2 forest plot object representing the spatially or temporally varying effect, with each line corresponding to a different spatial or temporal unit.

Examples

```
# Load example GHRmodels object from the package:
model_cov_list_file <- system.file("examples", "model_cov_list.rds", package = "GHRmodel")
model_cov_list <- readRDS(model_cov_list_file)

plot_coef_varying(
  models = model_cov_list,               # A list of fitted INLA model objects
  mod_id = "mod8",                       # Select the model with varying slopes
  palette = "Blues",                     # Color palette for the plot
  name = "main_climate_f",               # The grouping variable
  title = "Effect of PDSI at one-month lag for each climate zone",  # Plot title
  ylab = "Main climate zones",           # Label for the y-axis
  unit_label = c(                        # Map factor levels to descriptive names
    "1" = "Tropical Rainforest Climate",
    "2" = "Tropical Monsoon Climate",
    "3" = "Tropical Savanna Climate with Dry Winter",
    "4" = "Humid Subtropical Climate")
)
```

---

plot_fit *Plot Observed vs. Fitted Cases*

Description

This function creates a time-series plot comparing observed cases with fitted values from one or more models in a `GHRmodels` object. The plot supports faceting by model and/or group.

Usage

```
plot_fit(
  models = NULL,
  mod_id = NULL,
  time = NULL,
  group = NULL,
  group_id = NULL,
  mod_label = NULL,
  mod_facet = FALSE,
  palette = "IDE2",
  ref_color = NULL,
  obs_color = NULL,
  obs_label = NULL,
  title = "",
  ci = FALSE,
  transform = "identity",
  xlab = "Time",
```

```
  ylab = "Cases",
  xlim = NULL,
  legend = "Model"
)
```

Arguments

models	A GHRmodels object containing fitted model output.
mod_id	Character vector of model identifiers (from modelsmod_gofmodel_id) to plot.
time	Character; name of the time-variable column in models$data.
group	Optional; character name of the column defining independent time series (e.g., spatial areas).
group_id	Optional vector of specific group values to subset if group is provided.
mod_label	Optional custom labels for each model. Can be a named vector (e.g., c("mod1" = "Base")) or an unnamed vector with the same length and order as mod_id.
mod_facet	Logical; if TRUE, faceting is applied by model. Can be combined with group.
palette	Character; name of the color palette for fitted lines. Default is "IDE2".
ref_color	Optional color to override the first model's line (reference model).
obs_color	Color for observed data line. Default is "black".
obs_label	Legend label for observed data. Default is "Observed".
title	Character; title of the plot.
ci	Logical; if TRUE, adds 95% credible interval ribbons for model fits.
transform	Character string for y-axis transformation. Defaults to "identity" (no transform). Other options include "log10p1", "log1p", "sqrt", etc.
xlab	Label for the x-axis. Default is "Time".
ylab	Label for the y-axis. Default is "Cases".
xlim	Character vector of length two in "yyyy-mm-dd" format (e.g., c("2010-01-01", "2020-12-31")). Use NA to leave one side open (e.g., c("2015-01-01", NA)).
legend	Legend title for model lines. Default is "Model".

Details

- Faceting is flexible: if mod_facet = TRUE and group is provided, both are used.
- If ci = TRUE, ribbons are plotted for fitted model uncertainty.
- mod_label, ref_color, and obs_color allow full customization of the legend.
- The function automatically sums values across replicates for grouped time series.

Value

A ggplot2 object:

- Time-series line plot of observed vs fitted cases
- Optionally includes credible intervals and facets by model or group
- X-axis can be limited by xlim; Y-axis can be transformed for readability

See Also

fit_models to generate GHRmodels.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Plot observed vs. fitted cases over time for three selected models
plot_fit(
  models = model_list,                    # A GHRmodels object containing the fitted models
  mod_id = c("mod1", "mod3", "mod5"),          # Vector of model IDs to plot
  mod_label = c("Baseline",                    # Custom display names
                "tmin.l1.nl",
                "pdsi.l1.nl_tmin.l1.nl"),
  ref_color = "grey",                          # Color for the reference model
  time = "date",                               # Name of the time variable
  palette = "Set2",                            # Color palette for fitted lines
  xlim = c("2010-01-01", "2020-01-01"),        # Limit x-axis to this date range
  title = "Fitted vs Observed"                 # Main plot title
)
```

---

plot_gof *Plot Models by Goodness-of-Fit*

Description

Provides visualization of model performance using selected goodness-of-fit (GoF) metrics for one or more models. It is typically used with the mod_gof component of a GHRmodels object (produced by fit_models), but it can also accept any custom data frame — provided it contains the same column names as the default mod_gof output (including model_id and the relevant metric column names). It supports visual grouping by aesthetics (color, shape, facet), arranging models by metric, and adding credible intervals for model differences.

Usage

```
plot_gof(
  mod_gof,
  metric = "dic",
  mod_id = NULL,
  mod_label = NULL,
  ci = FALSE,
  var_arrange = NULL,
  var_color = NULL,
  var_shape = NULL,
```

```
  var_facet = NULL,
  palette = "IDE2"
)
```

Arguments

mod_gof	A data frame containing goodness-of-fit statistics for each model. Typically this is the mod_gof component of a GHRmodels object. It must include at least a model_id column and the selected metric. Other columns can be used for aesthetics (e.g., color, shape).
metric	Character string specifying the GoF metric to plot. Common options include: • "dic", "waic", "lms", "mae", "rmse", "crps", "rsq" • Differences from baseline: "dic_vs_first", "waic_vs_first", "mae_vs_first", etc. • Random effect variances: "re_n_var", "re_n_var_change", where n is an index.
mod_id	Optional character vector of model IDs to include. If NULL, includes all in mod_gof.
mod_label	Optional named or unnamed vector to customize display names for models. If unnamed, must match the order of mod_id.
ci	Logical. If TRUE, adds credible intervals for "*_vs_first" metrics (if available).
var_arrange	Character string for a column name used to order models along the x-axis. Defaults to "model_id" order if NULL.
var_color	Optional; name of a column in mod_gof to use for color grouping.
var_shape	Optional; name of a column in mod_gof to use for point shape grouping.
var_facet	Optional; name of a column in mod_gof to use for faceting the plot.
palette	Character; name of a color palette to use if var_color is provided. Default is "IDE2".

Details

This function helps interpret and visualize comparative model performance:

- Relative metrics (e.g., "*_vs_first") assume the first model is a reference.
- If ci = TRUE, the function looks for columns like "dic_vs_first_lci" and "_uci".
- The user can customize model order with var_arrange and legend groupings using var_color, etc.

Value

A ggplot2 object showing the specified metric for each model, optionally grouped and faceted. The plot supports:

- Ranking or sorting models by a specified variable
- Highlighting credible intervals for relative metrics (e.g. "dic_vs_first")
- Group-level comparisons via color, shape, and facet aesthetics

See Also

fit_models for fitting multiple INLA models.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Plot models by difference in DIC

plot_gof(mod_gof = model_list$mod_gof,
        metric = "dic_vs_first",
        ci = TRUE,
        var_arrange = "dic",
        var_color = "covariate_1",
        var_shape = "covariate_2",
        palette= "IDE2")
```

---

plot_ppd *Plot Posterior Predictive Densities Versus Observed Data*

Description

This function draws kernel-density curves for posterior-predictive samples and observed data using ggplot2::geom_line(). Each predictive sample's density is plotted in light blue; the observed density is overlaid in black.

Usage

```
plot_ppd(
  ppd,
  xlab = "Outcome",
  ylab = "Density",
  title = "Posterior Predictive Distribution",
  xlim = NULL,
  obs_color = NULL,
  ppd_color = NULL
)
```

Arguments

ppd	A data.frame containing posterior-predictive samples (one column per sample) and the column with observed data.
xlab	Character: x-axis label. Default "Outcome".
ylab	Character: y-axis label. Default "Density".

title	Character: plot title. Default "Posterior Predictive Distribution".
xlim	Numeric vector of length 2 giving the minimum and maximum x-axis values, e.g. c(0, 25). If NULL (default) the limits are c(0, quantile(observed, 0.95)).
obs_color	Color for the observed line density
ppd_color	Color for the posterior predictive distribution lines density

Value

A **ggplot2** plot object.

Examples

```
# Load example dataset
data(dengueMS)

# Declare formulas
formulas <- c("dengue_cases ~ tmin +  f(year, model='rw1')")

# Tranform formulas into a 'GHRformulas' object
ghr_formula <- as_GHRformulas(formulas)

# Fit multiple models
results <- fit_models(
  formulas = ghr_formula,
  data     = dengue_MS[dengue_MS$year %in% 2005:2010,],
  family   = "nbinomial",
  name     = "model",
  offset   = "population",
  nthreads = 2,
  control_compute = list(config = FALSE),
  pb       = TRUE
)

# Generate 100 samples from the posterior predictive distribution of the model
ppd_df <- sample_ppd(
  results,
  mod_id = "model1",
  s = 100,
  nthreads = 2)

# Plot densities of the posterior predictive distribution and observed cases.
plot_ppd(ppd_df, obs_color = "blue", ppd_color = "red")
```

plot_re *Plot Random Effects*

Description

Generates plots of random effects from one or more fitted models contained within a GHRmodels object. The function supports two main display modes:

- Caterpillar plot of effect sizes with uncertainty intervals (the default).
- Choropleth map (when a spatial map (sf object) is provided in the map argument).

It also supports visualization of replicated or grouped effects via the rep_id argument.

Usage

```
plot_re(
  models,
  mod_id,
  re_id,
  rep_id = NULL,
  map = NULL,
  map_area = NULL,
  mod_label = NULL,
  re_label = NULL,
  rep_label = NULL,
  ref_color = NULL,
  palette = NULL,
  var_arrange = "ID",
  title = "",
  xlab = "Re ID",
  ylab = "Effect Size",
  legend = "Effect Size",
  centering = 0,
  exp = FALSE
)
```

Arguments

models	A GHRmodels object containing fitted models and random effects.
mod_id	Character vector of model IDs to plot (must match entries in modelsmod_gofmodel_id).
re_id	Character; name of the variable defining the random effect (from models$re).
rep_id	Optional character string; name of a grouping variable if random effects are replicated. Default is NULL.
map	Optional sf object providing spatial geometry. If NULL, returns a caterpillar plot.
map_area	Character; column name in map indicating spatial units (must match re_id order).

mod_label	Optional labels for models. Can be a named vector (e.g., c("mod1" = "Baseline", "mod2" = "Adjusted")) or an unnamed vector with the same order as mod_id.
re_label	Optional; variable in the data to label the random effect units (e.g., year names instead of numeric IDs).
rep_label	Optional; label for replicated grouping variable (e.g., for years or time periods).
ref_color	Color used for the reference model. If specified, this will apply to the first model in mod_id.
palette	Character; name of the color palette to use. Defaults to "IDE1" for maps and "IDE2" otherwise.
var_arrange	Character; how to arrange REs on the x-axis. Options: "median" or "ID". Default is "ID".
title	Title for the plot.
xlab	Label for the x-axis. Default is "Re ID".
ylab	Label for the y-axis. Default is "Effect Size".
legend	Label for the legend in map plots. Default is "Effect Size".
centering	Value at which to center the color scale for map plots. Default is 0.
exp	Logical; if TRUE, exponentiates the effects (useful for log-scale models). Default is FALSE.

Details

Plot Random Effects from GHRmodels

- If map is used, map_area must match a column in map and correspond in order to the RE unit.
- For BYM/BYM2 models, only the total random effect is plotted (structured/unstructured parts are merged).
- When no map is used, the plot compares models via colored points and intervals for each RE unit.
- Replicated REs (e.g., for years) can be plotted across facets using rep_label.
- Model comparison is visually aided using distinct colors; the first model in mod_id is the reference.

Value

A ggplot2 plot object:

- If map is NULL, returns a caterpillar plot showing median REs with 95% uncertainty intervals.
- If map is provided, returns a faceted choropleth map showing RE medians by area and (optionally) replicate.

See Also

fit_models for model fitting; as_GHRformulas for formula setup.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

#  Plot the estimated yearly random effects for three different models.
plot_re(
  model = model_list,                          # A GHRmodels object
  mod_id = c("mod1", "mod3", "mod5"),          # IDs of the models
  mod_label = c("Baseline",                    # Custom labels for the models
                "tmin.l1_nl",
                "pdsi.l1_nl + tmin.l1_nl"),
  re_id = "year_id",                           # Name of the random effect variable
  re_label = "year",                           # Label to map year_id to calendar years
  ref_color = "grey",                          # Color for the reference model's effects
  palette = "IDE2",                            # Color for other model effects
  title = "Yearly Random Effect",              # Title for the plot
  xlab = "Year"                                # Label for the x-axis
)
```

---

rank_models *Rank Models by Goodness-of-Fit*

Description

This function ranks fitted models in a `GHRmodels` object by a chosen metric (e.g., `dic`, `waic`, `crps`, etc.).

Usage

```
rank_models(models, metric = "dic", n = 10)
```

Arguments

`models`	A `GHRmodels` object containing fitted model output.
`metric`	A character string indicating which goodness-of-fit metric to use for ranking. One of: `"dic"`, `"waic"`,`"lms"`,`"mae"`, `"rmse"`,`"crps"`, `"rsq"`,`"dic_vs_first"`, `"waic_vs_first"`, `"mae_vs_first"`, `"rmse_vs_first"`, `"crps_vs_first"`, `"re_n_var"`, and `"re_n_var_change"` (where n is the number of random effect, for ex. `re_1_var`, `re_1_var_change`).
`n`	An integer specifying how many top-ranked models to return (default `10`).

Value

A character vector of the top model IDs (in ascending order of the specified `metric`).

See Also

fit_models for fitting multiple INLA models.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Get a list of the 5 best models by DIC
top_model_dic <- rank_models(
  models = model_list,
  metric = "dic",
  n = 5
)
top_model_dic
```

---

sample_ppd *Sample from the Posterior Predictive Distribution*

Description

This function refits a specified model from a GHRmodels object and generates samples from its posterior predictive distribution.

Usage

```
sample_ppd(models, mod_id, s = 1000, nthreads = 8)
```

Arguments

models	A GHRmodels object.
mod_id	Character; model identifier (from modelsmod_gofmodel_id).
s	An integer specifying the number of samples to draw from the posterior predictive distribution.
nthreads	An integer specifying the number of threads for parallel computation to refit the model. Default is 8.

Value

A data.frame containing columns for each of the posterior predictive samples and one column with observed data.

Examples

```
# Load example dataset
data(dengueMS)

# Declare formulas
formulas <- c("dengue_cases ~ tmin +  f(year, model='rw1')")

# Tranform formulas into a 'GHRformulas' object
ghr_formula <- as_GHRformulas(formulas)

# Fit multiple models
results <- fit_models(
  formulas = ghr_formula,
  data     = dengue_MS[dengue_MS$year %in% 2005:2010,],
  family   = "nbinomial",
  name     = "model",
  offset   = "population",
  nthreads = 2,
  control_compute = list(config = FALSE),
  pb       = TRUE
)

# Generate 100 samples from the posterior predictive distribution of the model
ppd_df <- sample_ppd(
  results,
  mod_id = "model1",
  s = 100,
  nthreads = 2)
```

`stack_models` *Merge GHRmodels*

Description

This function stack together two or more objects `GHRmodels` object, returning **one** `GHRmodels` object that contains *all* the input models.

If **any** `model_id` is duplicated across the inputs the `new_name` argument must be provided to ensure unique IDs.

Usage

```
stack_models(..., new_name = NULL, vs_first = FALSE)
```

Arguments

`...`	Two or more `GHRmodels` objects, or a single list of them.

new_name	NULL (default) **or** a character used to build the new model IDs.
vs_first	Logical. If TRUE columns comparing the model vs the first model are kept in the mod_gof, otherwise are discarded. Default is FALSE. Set to TRUE only when models contained in the GHRmodels object to be stacked are compared with the same first models.

Details

Combine (Stack) Multiple **GHRmodels** Objects

Value

A single GHRmodels object containing all models from the inputs.

See Also

subset_models for subsetting GHRmodels objects, fit_models for fitting INLA models.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Load example GHRmodels object with DLNM from the package:
model_dlnm_file <- system.file("examples", "model_dlnm.rds", package = "GHRmodel")
model_dlnm <- readRDS(model_dlnm_file)

# Merge models from the model_list and model_dlnm objects
model_stack <- stack_models(
  model_list,
  model_dlnm,
  new_name = "mod")

# The combined model_stack combines the models in the model_list and model_dlnm objects
model_stack$mod_gof$model_id
```

---

subset_models *Subset* GHRmodels *Objects*

Description

This function subsets selected models from a GHRmodels object into a new reduced GHRmodels object.

Usage

```
subset_models(models, mod_id, new_name = NULL)
```

Arguments

models	A GHRmodels object.
mod_id	A character vector of model IDs indicating which model(s) to keep. These must match modelsmod_gofmodel_id.
new_name	NULL (default) **or** a character used to build the new model IDs.

Value

A new GHRmodels object containing only the specified model(s).

See Also

stack_models for combining GHRmodels objects, fit_models for fitting INLA models.

Examples

```
# Load example GHRmodels object from the package:
model_list_file <- system.file("examples", "model_list.rds", package = "GHRmodel")
model_list <- readRDS(model_list_file)

# Extract a vector with the moded IDs of the 2 best fitting models by WAIC
best_waic <- rank_models(
  models = model_list,  # GHRmodels object containing model fit results
  metric = "waic",      # Metric used to rank models (lower WAIC is better)
  n = 2                 # Number of top-ranked models to return
)

# The output is a vector
best_waic

# Subset those specific models and assign new IDs
model_waic <- subset_models(
  model = model_list,
  mod_id = best_waic,
  new_name = "best_waic"
)

# Check output subset model names
model_waic$mod_gof$model_id
```

`write_inla_formulas` *Generate INLA-compatible Model Formulas*

Description

This function streamlines the creation of INLA-compatible model formulas by automatically structuring fixed effects, random effects, and interactions. It accepts a list of covariate sets and produces a corresponding set of model formulas that share a common random effect structure.

Usage

```
write_inla_formulas(
  outcome,
  covariates = NULL,
  baseline = TRUE,
  re1 = list(id = NULL, model = NULL, replicate = NULL, group = NULL, graph = NULL,
   cyclic = FALSE, scale.model = FALSE, constr = FALSE, adjust.for.con.comp = FALSE,
    hyper = NULL),
  re2 = NULL,
  re3 = NULL,
  re4 = NULL,
  re5 = NULL
)
```

Arguments

`outcome`	Character string specifying the name of the outcome variable.
`covariates`	A list of character vectors, where each vector contains covariate names to be included in the model. If a single vector is provided, a single model formula is generated.
`baseline`	Logical; If `TRUE`, a baseline formula without covariates is included. If no random effects are specified, this will be an intercept-only model. If random effects are specified, the baseline formula will include random effects but not covariates. This formula will be the first in the list. Default is `TRUE`.
`re1`	A list defining a random effect structure. Up to five such lists (`re1` through `re5`) can be passed.
`re2`	Additional random effect definitions, as described for `re1`.
`re3`	Additional random effect definitions, as described for `re1`.
`re4`	Additional random effect definitions, as described for `re1`.
`re5`	Additional random effect definitions, as described for `re1`.

Details

The `write_inla_formulas()` function simplifies the creation of multiple INLA models by automatically structuring fixed effects, random effects, and interactions. The function ensures that all models have a consistent structure, making them easier to analyze and modify.

If `baseline = TRUE`, a null formula (without covariates) is included as the first element of the list.

The number of formulas generated depends on the length of the `covariates` list.

Random effects can be added using `re1, ..., re5`, where each effect must be a named list (e.g. re1 = list(id = "year_id", model = "rw1")). In the list the following fields are strictly necessary:

- `id` *(character)*: the variable name that indexes the random effect (e.g., "year", "region").
- `model` *(character)*: the type of random effect. Supported values include: `"iid"`, `"rw1"`, `"rw2"`, `"bym"`, and `"bym2"`.
- The following optional fields can be provided in the random effect list:
  - `replicate` *(character)*: defines an additional variable used to replicate the random effect structure across groups (e.g., spatial units for repeated time-series).
  - `group` *(character)*: used to model group-specific effects or nested structures.
  - `graph` *(character)*: required for `"bym"` and `"bym2"` models; refers to the name of an object in the environment that holds the spatial adjacency matrix.
  - `cyclic` *(logical)*: indicates whether the random walk (`"rw1"` or `"rw2"`) is cyclic. Default is `FALSE`. Use for periodic structures (e.g., months).
  - `scale.model` *(logical)*: if `TRUE`, scales structured random effects (like `rw1`, `rw2`, `bym`) so the generalized variance is 1. For `bym2` INLA automatically applies `scale.model = TRUE` internally.
  - `constr` *(logical)*: If `TRUE`, a sum to zero constrain is introduced. This 'constr' option is applied only to 'iid' random effects. For `rw`, `ar`, `bym`, `bym2` INLA automatically applies `scale.model = TRUE` internally.
  - `adjust.for.con.comp` *(logical)*: if `TRUE`, accounts for disconnected components in spatial graphs. Recommended for `"bym"` and `"bym2"`. Default is `FALSE`.
  - `hyper` *(character)*: the name of an object in the environment that contains the hyperprior specification for the random effect's precision or other parameters.

For more information on random effects in R-INLA, see Bayesian inference with INLA: Mixed-effects Models.

Value

A character vector of INLA model formulas.

See Also

`as_GHRformulas` for transforming model formulas into structured objects.

Examples

```
# Define covariates of interest
covs <- c("tmin.l1", "tmin.l2", "pdsi.l1", "pdsi.l2", "urban_level")

# Combine covariate names using a pattern-matching functionality
combined_covariates <- cov_multi(
  covariates = covs,
  pattern    = c("tmin", "pdsi", "urban_level")
)

# Define hyperprior specifications for random effects
prior_re1 <- list(prec = list(prior = "loggamma", param = c(0.01, 0.01)))
prior_re2 <- list(prec = list(prior = "loggamma", param = c(0.01, 0.01)))
prior_re3 <- list(
  prec = list(prior = "pc.prec", param = c(0.5 / 0.31, 0.01)),
  phi  = list(prior = "pc",      param = c(0.5, 2 / 3))
)

# Write a set of INLA-compatible model formulas
inla_formulas <- write_inla_formulas(
  outcome    = "dengue_cases",
  covariates = combined_covariates,
  re1 = list(
    id        = "month_id",
    model        = "rw1",
    cyclic    = TRUE,
    hyper     = "prior_re1",
    replicate = "spat_meso_id"
  ),
  re2 = list(
    id    = "year_id",
    model    = "rw1",
    hyper = "prior_re2"
  ),
  re3 = list(
    id    = "spat_id",
    model    = "iid",
    hyper = "prior_re3"
  ),
  baseline = TRUE
)
```

Index