# Package 'bivarhr'

May 7, 2026

**Title** Bivariate Hurdle Regression with Bayesian Model Averaging

**Version** 0.1.5

**Description** Provides tools for fitting bivariate hurdle negative binomial models with horseshoe priors, Bayesian Model Averaging (BMA) via stacking, and comprehensive causal inference methods including G-computation, transfer entropy, Threshold Vector Autoregressive (TVAR) and Smooth Transition Autoregressive (STAR) models, Dynamic Bayesian Networks (DBN), Hidden Markov Models (HMM), and sensitivity analysis.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1.0)

**Imports** stats, utils, grDevices, dplyr (>= 1.1.0), rlang, data.table (>= 1.14.0), tidyr, tibble, readr, cli, furrr, future, future.apply, posterior, loo (>= 2.5.0), progressr

**Suggests** cmdstanr, testthat (>= 3.0.0), MASS, RTransferEntropy, bnlearn, depmixS4, sensemakr, CausalImpact, bsts, vars, tsDyn, openxlsx, ggplot2, bayesplot, Rgraphviz

**Additional_repositories** https://stan-dev.r-universe.dev

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** José Mauricio Gómez Julián [aut, cre] (ORCID: <https://orcid.org/0009-0000-2412-3150>)

**Maintainer** José Mauricio Gómez Julián <isadore.nabi@pm.me>

**Repository** CRAN

**Date/Publication** 2025-12-19 20:20:16 UTC

## Contents

---

`bivarhr-package` *bivarhr: Bivariate Hurdle Regression*

---

## Description

Implements bivariate hurdle regression models using Stan/CmdStan with horseshoe priors, Bayesian Model Averaging via stacking, and comprehensive causal inference methods.

## Author(s)

**Maintainer**: José Mauricio Gómez Julián <isadore.nabi@pm.me> (ORCID)

## add_qsig *Add BH-adjusted q-values and significance stars*

### Description

Adds Benjamini-Hochberg adjusted q-values and a simple significance code column based on p-values contained in a data frame.

### Usage

```
add_qsig(df)
```

### Arguments

| | |
|---|---|
| df | A data frame containing at least a numeric column p_value. If df is NULL or has zero rows, it is returned unchanged. |

### Details

The function:

- Computes q_value using p.adjust(method = "BH").
- Creates a sig column with significance codes:
  - "***" for q_value <= 0.001
  - "**" for 0.001 < q_value <= 0.01
  - "*" for 0.01 < q_value <= 0.05
  - "" otherwise

### Value

The input data frame with added columns q_value and sig. If df is NULL or empty, it is returned as is.

## build_design *Build Design Matrices for Bivariate Hurdle Model*

### Description

Constructs design matrices for the zero and count components of both outcome variables with cross-lags, trends, regimes, transition dummies, and control variables.

## Usage

```
build_design(
  DT,
  k,
  include_C_to_I = TRUE,
  include_I_to_C = TRUE,
  include_trend = TRUE,
  controls = character(0),
  include_regimes = TRUE,
  include_transitions = TRUE
)
```

## Arguments

| | |
|---|---|
| DT | A data.table with required columns. |
| k | Integer; lag order. |
| include_C_to_I | Logical; include C lags in I equations. |
| include_I_to_C | Logical; include I lags in C equations. |
| include_trend | Logical; include polynomial time trend. |
| controls | Character vector of control variable names. |
| include_regimes | Logical; include regime dummies. |
| include_transitions | Logical; include transition dummies. |

## Value

A list containing design matrices and outcome vectors.

---

# contrafactual_ATE *Contrafactual Average Treatment Effects (ATE) for the Bivariate Hurdle Model*

---

## Description

Computes time-varying contrafactual Average Treatment Effects (ATE) for both series (I and C) from a fitted bivariate hurdle negative binomial model. For each time point and posterior draw, the function compares the expected outcome under the observed design matrix with a contrafactual scenario where cross-lag terms and transition covariates are set to zero.

## Usage

```
contrafactual_ATE(fit_obj, compute_intervals = TRUE, ndraws = 1200, seed = 42)
```

## Arguments

| | |
|---|---|
| `fit_obj` | A list returned by `fit_one()` (or an equivalent fitting function), containing at least:<br>• `$fit`: a CmdStanR fit object.<br>• `$des`: a list with design matrices `X_pi_I`, `X_mu_I`, `X_pi_C`, `X_mu_C`, a vector `log_exposure50`, and an index vector `idx`. |
| `compute_intervals` | Logical; if `TRUE`, returns posterior means and 95\ `FALSE`, only posterior means are returned. |
| `ndraws` | Integer; maximum number of posterior draws to use. If `ndraws` exceeds the number of available draws, it is truncated. |
| `seed` | Integer; random seed used to subsample posterior draws. |

## Details

The function identifies in the design matrices:

- Cross-lag terms via column names containing `"zC_L"` / `"C_L"` (for `I`) and `"zI_L"` / `"I_L"` (for `C`).
- Transition covariates via column names starting with `"trans_"`.

For each time point `t` and posterior draw `s`, the expected value under the observed design ($E[Y \mid X]$) is contrasted with a contrafactual design where these cross-lag and transition columns are set to zero ($E[Y \mid X_{cf}]$). The ATE at time `t` is defined as the posterior distribution of $E[Y \mid X] - E[Y \mid X_{cf}]$, computed separately for `I` and `C`.

## Value

A tibble with one row per effective time index (length `des$idx`). If `compute_intervals = TRUE`, the columns are:

- `t`: time index (from `des$idx`).
- `ATE_I_mean`, `ATE_I_low`, `ATE_I_high`: posterior mean and 95\
- `ATE_C_mean`, `ATE_C_low`, `ATE_C_high`: posterior mean and 95\

If `compute_intervals = FALSE`, only `ATE_I_mean` and `ATE_C_mean` are returned (plus `t`).

## Examples

```
if (interactive() && requireNamespace("cmdstanr", quietly = TRUE)) {
  n <- 120
  DT <- data.table::data.table(
    I = rpois(n, 5), C = rpois(n, 3),
    Regime = factor(sample(c("A","B","C"), n, TRUE)),
    trans_PS = c(rep(1,5), rep(0,n-5)),
    trans_SF = c(rep(0,60), rep(1,5), rep(0,n-65)),
    trans_FC = rep(0, n),
    log_exposure50 = log(runif(n, 40, 60))
  )
```

```
  fit_obj <- fit_one(DT, k = 1, spec = "C")
  ate_tab <- contrafactual_ATE(fit_obj, compute_intervals = TRUE)
  head(ate_tab)
}
```

---

`disc_terciles` *Discretize Numeric Vector into Terciles*

---

## Description

Converts a numeric vector into an ordered factor with three levels (low, medium, high) using deterministic percent ranks to break ties.

## Usage

```
disc_terciles(x)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector to discretize. |

## Value

An ordered factor with levels "low", "medium", "high".

## Examples

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
disc_terciles(x)
```

---

`export_results` *Export Analysis Results*

---

## Description

Exports analysis results to Excel and/or CSV format.

## Usage

```
export_results(results, output_dir, format = "xlsx", verbose = TRUE)
```

### Arguments

| | |
|---|---|
| results | Named list containing analysis results. Expected components include: hurdle, te, te_by_type (list with counts/rates/binary), placebo, tvarstar, varx, eba, dbn_arcs, hmm, sensemakr_I, sensemakr_C, oos, ate. |
| output_dir | Directory path for output files. Created if it does not exist. |
| format | Character; output format. One of "xlsx", "csv", or "both". |
| verbose | Logical; if TRUE, print progress messages. |

### Value

Invisible path to output directory.

### Examples

```
results <- list(
  hurdle = data.frame(model = "test", elpd = -100),
  te = data.frame(dir = "I->C", stat = 0.5, p_value = 0.01)
)
export_results(results, tempdir(), format = "both")
```

---

## fit_one *Fit Single Bivariate Hurdle Model*

---

### Description

Fits a bivariate hurdle negative binomial model with horseshoe priors using Stan/CmdStan.

### Usage

```
fit_one(
  DT,
  k,
  spec = c("A", "B", "C", "D"),
  controls = character(0),
  model = NULL,
  output_dir = NULL,
  iter_warmup = 1000,
  iter_sampling = 1200,
  chains = 4,
  seed = NULL,
  adapt_delta = 0.95,
  max_treedepth = 12,
  threads_per_chain = 1L,
  hs_tau0 = 0.5,
  hs_slab_scale = 5,
```

```
  hs_slab_df = 4,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| DT | A data.table with the data. |
| k | Integer; lag order. |
| spec | Character; model specification ("A", "B", "C", "D"). |
| controls | Character vector of control variable names. |
| model | A compiled CmdStan model object. If NULL, the package default model is loaded. |
| output_dir | Directory for CmdStan output files. If NULL, uses a temporary directory. |
| iter_warmup | Integer; warmup iterations. |
| iter_sampling | Integer; sampling iterations. |
| chains | Integer; number of chains. |
| seed | Integer; random seed. |
| adapt_delta | Numeric; adaptation target acceptance rate. |
| max_treedepth | Integer; maximum tree depth. |
| threads_per_chain | Integer; threads per chain. |
| hs_tau0 | Numeric; horseshoe tau0 parameter. |
| hs_slab_scale | Numeric; horseshoe slab scale. |
| hs_slab_df | Numeric; horseshoe slab degrees of freedom. |
| verbose | Logical; print progress messages. |

## Value

A list with components:

| | |
|---|---|
| fit | The CmdStanMCMC fit object. |
| des | The design matrices used. |
| spec | The model specification. |
| k | The lag order. |
| hs_tau0, hs_slab_scale, hs_slab_df | Horseshoe hyperparameters. |
| controls | Control variables used. |
| output_dir | Directory with output files. |

## get_hurdle_model *Get Default Hurdle Model*

### Description

Loads and compiles the package's default Stan model.

### Usage

```
get_hurdle_model()
```

### Value

A compiled CmdStanModel object.

## load_saved_results *Load Saved Results from Directory*

### Description

Loads previously saved .rds result files from a specified directory.

### Usage

```
load_saved_results(
  dir_out,
  which = c("varx", "tsdyn", "bma", "dbn", "hmm", "sensemakr", "synth"),
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| dir_out | Directory containing saved .rds files. |
| which | Character vector specifying which results to load. Valid options: "varx", "tsdyn", "bma", "dbn", "hmm", "sensemakr", "synth". Default loads all available. |
| verbose | Logical; if TRUE, print messages about loaded files. |

### Value

Named list of loaded objects. Components not found are NULL.

## Examples

```
# 1. Create a temporary directory (CRAN safe)
tmp_dir <- file.path(tempdir(), "test_results")
dir.create(tmp_dir, showWarnings = FALSE)

# 2. Create dummy data files matching the names expected by the function
saveRDS(list(aic = 100), file.path(tmp_dir, "varx_fit.rds"))
saveRDS(list(model = "BMA"), file.path(tmp_dir, "best_fit_bma.rds"))

# 3. Load the results (this will now work correctly)
results <- load_saved_results(tmp_dir, which = c("varx", "bma"))

# 4. Clean up
unlink(tmp_dir, recursive = TRUE)
```

---

# make_lags *Create Lag Matrix*

---

## Description

Creates a matrix of lagged values for a numeric vector.

## Usage

```
make_lags(x, k)
```

## Arguments

| | |
|---|---|
| x | Numeric vector. |
| k | Integer; maximum lag order. |

## Value

A matrix with k columns containing lags 1 through k.

---

# placebo_temporal *Temporal Placebo Test via Time-Index Permutations*

---

## Description

Implements a temporal placebo test for the bivariate hurdle model by randomly permuting the time ordering of DT, re-estimating the model on each permuted dataset, and comparing the PSIS-LOO ELPD of the original fit against the permuted fits.

## Usage

```
placebo_temporal(
  DT,
  spec = "C",
  k = 2,
  controls = character(0),
  n_perm = 10,
  seed = 999,
  dir_csv = NULL
)
```

## Arguments

| | |
|---|---|
| DT | A data.table (or data.frame) containing the data used by fit_one(). |
| spec | Character scalar; model specification (e.g. "A", "B", "C", "D") passed to fit_one(). |
| k | Integer; lag order passed to fit_one(). |
| controls | Character vector of control variable names passed to fit_one(). |
| n_perm | Integer; number of temporal permutations (placebo datasets) to run. |
| seed | Integer; base random seed used for reproducibility of the original fit and the permutations. |
| dir_csv | Character scalar; directory path to save the summary CSV. If NULL (default), the CSV is not saved to disk. |

## Details

The function:

- Fits the model on the original DT via fit_one(), extracts "log_lik_joint" and computes PSIS-LOO (with moment_match = TRUE).
- For each of n_perm iterations, permutes the row order of DT, refits the model on the permuted data, recomputes PSIS-LOO, and stores the permuted ELPD.
- Reports, for each permutation, the original ELPD, the permuted ELPD, and their difference (elpd_orig - elpd_perm).

This procedure evaluates whether the temporal structure captured by the model is informative: if the model is exploiting genuine time dependence, the original ELPD should typically be higher than that of the permuted (time-scrambled) datasets.

The function assumes that fit_one() is available in the search path.

## Value

A data.frame with one row per permutation and columns:

- perm: permutation index (1, . . . , n_perm).
- elpd_orig: ELPD of the original (non-permuted) fit.
- elpd_perm: ELPD of the model fit on the permuted data.
- diff: difference elpd_orig - elpd_perm.

## Examples

```
# 1. Create a temporary directory for output
tmp_dir <- file.path(tempdir(), "placebo_out")
dir.create(tmp_dir, showWarnings = FALSE)

# 2. Create dummy data (DT)
# Needed because R CMD check runs in a clean environment
N <- 50
DT <- data.frame(
  time = 1:N,
  y = rpois(N, lambda = 4),
  X1 = rnorm(N),
  X2 = rnorm(N)
)
# Ensure it's a data.table if fit_one expects it, or leave as DF
# (The function internally ensures data.table behavior)

# 3. Define auxiliary parameters
k_grid  <- 0:1

# 4. Run the function
# We use a small n_perm for the example to run faster
try({
  out_placebo <- placebo_temporal(DT, spec = "C", k = 1,
                                  controls = c("X1", "X2"),
                                  n_perm = 2, seed = 999,
                                  dir_csv = tmp_dir)

  head(out_placebo)
})

# 5. Cleanup
unlink(tmp_dir, recursive = TRUE)
```

---

`prewhiten_bin_glm` *Pre-whiten binary series with logistic GLM*

---

## Description

Fits a logistic regression (binomial GLM with logit link) to a binary 0/1 response and returns Pearson residuals as a pre-whitened series.

## Usage

```
prewhiten_bin_glm(DT, yname)
```

### Arguments

| | |
|---|---|
| DT | A data.frame or data.table containing the binary response and covariates. It must include at least:<br>• The binary variable named by yname (values 0/1).<br>• t_norm: normalized time index.<br>• Regime, EconCycle, PopDensity, Epidemics, Climate, War. |
| yname | Character scalar; name of the binary response column in DT. The function checks that all values are in c(0, 1) and stops otherwise. |

### Value

A numeric vector of Pearson residuals (one per row in DT used in the fit).

### Examples

```
if (interactive()) {
  n <- 100
  DT <- data.frame(
    t_norm = seq_len(n) / n,
    I_zero = rbinom(n, 1, 0.3),
    Regime = factor(sample(c("A","B"), n, TRUE)),
    EconCycle = rnorm(n), PopDensity = runif(n),
    Epidemics = rbinom(n, 1, 0.1), Climate = rnorm(n), War = rbinom(n, 1, 0.05)
  )
  r_I_zero <- prewhiten_bin_glm(DT, "I_zero")
  head(r_I_zero)
}
```

---

## prewhiten_count_glm *Pre-whiten count series with GLM / NegBin model*

---

### Description

Fits a generalized linear model for count data using either a negative binomial model with log link and offset, or a Poisson fallback, and returns Pearson residuals to be used as a pre-whitened series.

### Usage

```
prewhiten_count_glm(DT, yname)
```

### Arguments

| | |
|---|---|
| DT | A data.frame or data.table containing the response and covariates. It must include at least:<br>• The count variable named by yname.<br>• t_norm: normalized time index. |

- Regime, EconCycle, PopDensity, Epidemics, Climate, War.
- log_exposure50: log exposure (offset).

yname Character scalar; name of the count response column in DT.

## Details

The function first attempts to fit a negative binomial GLM via MASS::glm.nb() with a log link and log_exposure50 as an offset. If the fit fails (e.g., due to convergence issues), it falls back to a Poisson GLM via glm(family = poisson()) with the same formula and offset.

## Value

A numeric vector of Pearson residuals (one per row in DT used in the fit).

## Examples

```
if (interactive()) {
  n <- 100
  DT <- data.frame(
    t_norm = seq_len(n) / n,
    I = rpois(n, 5),
    Regime = factor(sample(c("A","B"), n, TRUE)),
    EconCycle = rnorm(n), PopDensity = runif(n),
    Epidemics = rbinom(n, 1, 0.1), Climate = rnorm(n), War = rbinom(n, 1, 0.05),
    log_exposure50 = log(runif(n, 40, 60))
  )
  r_I <- prewhiten_count_glm(DT, "I")
  head(r_I)
}
```

---

# prewhiten_rate_glm *Pre-whiten rate series with log-link Gaussian GLM*

---

## Description

Fits a Gaussian GLM with log link to a rate variable (count/exposure) without offset, applying a small lower bound to avoid zeros, and returns Pearson residuals as a pre-whitened series.

## Usage

```
prewhiten_rate_glm(DT, yname)
```

### Arguments

| | |
|---|---|
| DT | A data.frame or data.table containing the rate variable and covariates. It must include at least:<br>• The rate variable named by yname.<br>• t_norm: normalized time index.<br>• Regime, EconCycle, PopDensity, Epidemics, Climate, War. |
| yname | Character scalar; name of the rate response column in DT. |

### Details

The response y is first sanitized via y_safe <- pmax(y, 1e-8) to avoid taking logs of zero. The model is then fit with glm(family = gaussian(link = "log")).

### Value

A numeric vector of Pearson residuals (one per row in DT used in the fit).

### Examples

```
if (interactive()) {
  n <- 100
  DT <- data.frame(
    t_norm = seq_len(n) / n,
    I_rate = rgamma(n, 2, 1),
    Regime = factor(sample(c("A","B"), n, TRUE)),
    EconCycle = rnorm(n), PopDensity = runif(n),
    Epidemics = rbinom(n, 1, 0.1), Climate = rnorm(n), War = rbinom(n, 1, 0.05)
  )
  r_I_rate <- prewhiten_rate_glm(DT, "I_rate")
  head(r_I_rate)
}
```

---

## print_floor_smoketest *Print summary of FLOOR smoke test (ELPD ranking invariance)*

---

### Description

Nicely prints a summary of the FLOOR smoke test produced by smoketest_floor_elpd_invariance, indicating whether the ELPD-based ranking of models is invariant across different FLOOR constants and listing the combined results.

### Usage

```
print_floor_smoketest(st)
```

### Arguments

| | |
|---|---|
| st | A list returned by smoketest_floor_elpd_invariance, containing at least:<br>• same_order: logical flag indicating whether the ELPD ranking is identical for all FLOOR values.<br>• combined: data frame or tibble with columns FLOOR, fit_id, elpd, elpd_se, and rank_elpd, among others. |

### Details

The function uses **cli** to print a section header and an info message stating whether the ELPD ranking is invariant across values of FLOOR. It then arranges the combined table by FLOOR and decreasing elpd, selects a subset of columns, and prints it to the console.

This is a convenience/reporting helper and does not modify st.

### Value

Invisibly returns the input object st, so it can be used in pipes if desired.

### Examples

```
# 1. Define dummy data inside the example so it runs on CRAN checks
st_dummy <- list(
  same_order = TRUE,
  combined = data.frame(
    FLOOR     = rep(c(-1e6, -1e4), each = 2),
    fit_id    = rep(c("model_1", "model_2"), 2),
    elpd      = c(-100.1, -101.3, -100.1, -101.3),
    elpd_se   = c(1.2, 1.3, 1.2, 1.3),
    rank_elpd = c(1L, 2L, 1L, 2L)
  )
)

# 2. Run the function
print_floor_smoketest(st_dummy)
```

---

## read_bma_all *Read and consolidate BMA weight tables*

---

### Description

Searches for BMA weight CSV files produced by the Hurdle-NB model, reads them using automatic delimiter detection, and returns a single stacked data frame with normalized column names and a combo identifier.

### Usage

```
read_bma_all(dir_csv, dir_out, stop_if_empty = TRUE, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| dir_csv | Character scalar; directory where BMA CSV files are expected (for example "bma_weights_specC_ctrl*.csv"). |
| dir_out | Character scalar; output directory used during the experiment, which may contain BMA files or a fallback RDS object. |
| stop_if_empty | Logical; if TRUE, an informative error is thrown when no valid BMA tables are found. If FALSE, a warning is issued and an empty tibble is returned. |
| verbose | Logical; if TRUE, prints diagnostic messages about the search paths, files found, and detected ELPD column. |

### Details

The function:

- Looks for CSV files matching the pattern "bma_weights_specC_ctrl*.csv" in dir_csv, and if none are found, searches recursively in dir_out.
- Reads each candidate file via rc_auto() and keeps only non-empty data frames.
- If no CSV files are usable, optionally falls back to an RDS file "experimento_mejorado_all.rds" under dir_out and tries to extract BMA tables from allobj$bma.
- Normalizes column names with normalize_names(), ensures a combo column exists, detects the ELPD column, and sorts rows by decreasing ELPD.

### Value

A data frame with all BMA tables stacked and an added combo_id column (source identifier) and a combo column (control combo). If nothing is found and stop_if_empty = FALSE, an empty tibble is returned.

---

## rolling_oos *Rolling Out-of-Sample Forecast Evaluation*

---

### Description

Computes rolling out-of-sample (OOS) forecast accuracy for the selected bivariate hurdle model by repeatedly truncating the sample at different cut points Tcut, generating multi-step-ahead predictive distributions, and summarizing them via RMSE for I and C.

## Usage

```
rolling_oos(
  best_fit,
  DT,
  h = 5,
  cuts = seq(round(0.6 * nrow(DT)), round(0.9 * nrow(DT)), length.out = 5)
)
```

## Arguments

| | |
|---|---|
| best_fit | A fitted model object as returned by `fit_one()`, containing at least:<br>• `$fit`: CmdStanR fit object with posterior draws.<br>• `$des`: design matrices used by the model.<br>• `$k`: lag order used in the fit.<br>This object is passed directly to `predict_multistep()`. |
| DT | A `data.frame` or `data.table` containing the original time series and covariates used to fit the model, including at least columns `I` and `C`. |
| h | Integer; maximum forecast horizon (number of steps ahead) requested at each cut. For a given `Tcut`, the effective horizon is `min(h, nrow(DT) - Tcut)`. |
| cuts | Numeric vector of time indices (training end points) at which to perform the rolling evaluation. By default, a grid of five equally spaced cut points between 60\ used: `seq(round(0.6 * nrow(DT)), round(0.9 * nrow(DT)), length.out = 5)`. |

## Details

For each `Tcut` in `cuts`, the function:

1. Calls `predict_multistep()` with `fit_obj = best_fit`, the full `DT`, lag `k = best_fit$k`, and horizon `h_eff = min(h, nrow(DT) - Tcut)` to obtain posterior predictive paths `pred_I` and `pred_C`.
2. Computes the posterior-mean forecast for each step (`mI`, `mC`) as the column means of `pred_I` and `pred_C`.
3. Extracts the realized outcomes `yI = I[(Tcut + 1):(Tcut + h_eff)]` and analogously for `yC`.
4. Computes RMSE for each series: `RMSE_I = sqrt(mean((yI - mI)^2))`, `RMSE_C = sqrt(mean((yC - mC)^2))`.

Progress is reported via **progressr**. The resulting table is written as `"rolling_oos.csv"` in the directory specified by a global character scalar `dir_csv`.

## Value

A `data.frame` with one row per `Tcut` and columns:

- `Tcut`: training end index.
- `RMSE_I`: rolling OOS RMSE for series `I`.
- `RMSE_C`: rolling OOS RMSE for series `C`.

## Examples

```
# Minimal synthetic example illustrating the expected data structure:
set.seed(123)
DT <- data.frame(
  id = rep(1:10, each = 2),
  t  = rep(1:2, times = 10),
  I  = rpois(20, lambda = 0.5),
  C  = rpois(20, lambda = 1.0)
)

# Directory for CSV output (in practice, use a persistent path chosen
# by the user):
dir_csv <- file.path(tempdir(), "bivarhr_oos_csv")

# Typical workflow (commented out to avoid heavy computation and
# external dependencies such as CmdStan during R CMD check):
#
# best_fit <- fit_one(
#   data = DT,
#   k    = 2,
#   spec = "C"
# )
#
# oos_res <- rolling_oos(
#   fit     = best_fit,
#   data    = DT,
#   h       = 6,
#   dir_csv = dir_csv
# )
# print(oos_res)
```

---

`run_dbn` *Fit a Two-Slice Dynamic Bayesian Network (DBN) for I, C, and Regime*

---

## Description

Constructs and estimates a simple two-slice Dynamic Bayesian Network (DBN) over discretized versions of `I`, `C`, and `Regime` using **bnlearn**. The network includes current and lag-1 nodes for each variable, with structural constraints enforcing the DBN topology.

## Usage

```
run_dbn(DT)
```

## Arguments

| | |
|---|---|
| DT | A data.frame or data.table containing at least:<br>• I_cat, C_cat: discretized (e.g., tercile) versions of I and C.<br>• Regime: categorical regime indicator.<br>The function internally renames these to Ic, Cc, and R, constructs their lag-1 counterparts, and drops rows with missing lags. |

## Details

The DBN is defined on the nodes Ic, Cc, R, Ic_l1, Cc_l1, R_l1. A blacklist is used to forbid arrows from current to lagged nodes, while a whitelist ensures arrows from lagged to current nodes:

- Blacklist: Ic → Ic_l1, Cc → Cc_l1, R → R_l1.
- Whitelist: Ic_l1 → Ic, Cc_l1 → Cc, R_l1 → R.

The structure is learned via hill-climbing (bnlearn::hc()) with BDe score (score = "bde") and imaginary sample size iss = 10. Parameters are then estimated via bnlearn::bn.fit() using Bayesian estimation with the same iss.

If **Rgraphviz** is available, a graph of the learned DAG is produced and saved as "dbn_graph.png" in the directory specified by a global object dir_figs (character scalar). The preprocessed data used to fit the DBN are written to "dbn_data.csv" in dir_csv, and the fitted objects are saved as "dbn_fit.rds" in dir_out.

The function assumes that dir_csv, dir_out, and (optionally) dir_figs exist as global character scalars specifying output directories.

## Value

A list with components:

- dag: the learned Bayesian network structure (bnlearn "bn" object).
- fit: the fitted DBN ("bn.fit" object).
- data: the processed data frame (Ic, Cc, R, and their lag-1 versions) used to learn/fit the DBN.

## Examples

```
library(data.table)

# 1. Create dummy data (Fixed: wrapped in factor() for bnlearn)
DT <- data.table(
  I_cat  = factor(sample(c("Low", "Medium", "High"), 100, replace = TRUE)),
  C_cat  = factor(sample(c("Low", "Medium", "High"), 100, replace = TRUE)),
  Regime = factor(sample(c("Growth", "Crisis"), 100, replace = TRUE))
)

# 2. Define global paths using tempdir()
tmp_dir <- tempdir()
dir_csv  <- file.path(tmp_dir, "csv")
dir_out  <- file.path(tmp_dir, "dbn")
```

```
dir_figs <- file.path(tmp_dir, "figs")

dir.create(dir_csv,  showWarnings = FALSE, recursive = TRUE)
dir.create(dir_out,  showWarnings = FALSE, recursive = TRUE)
dir.create(dir_figs, showWarnings = FALSE, recursive = TRUE)

# 3. Run the function
dbn_res <- run_dbn(DT)

# Inspect the result
print(dbn_res$dag)
```

---

`run_eba` *Extreme-Bounds Analysis (EBA) over Control-Variable Combinations*

---

## Description

Runs an Extreme-Bounds Analysis (EBA) over a predefined set of control variable combinations, fitting (or re-fitting) the bivariate hurdle model for each combination and extracting posterior mean coefficients for all regression blocks (`mu_I`, `pi_I`, `mu_C`, `pi_C`).

## Usage

```
run_eba(DT, spec = "C", k_bma_table = NULL, seed = 123)
```

## Arguments

| | |
|---|---|
| `DT` | A `data.table` or `data.frame` with the data passed to `fit_one()`. |
| `spec` | Character scalar; model specification (e.g.\ `"A"`, `"B"`, `"C"`, `"D"`) passed to `fit_one()`. |
| `k_bma_table` | Optional object (typically a named list or `list`-like structure) indexed by control-combination tags that indicates for which combinations a BMA selection table already exists. If `k_bma_table[[tag]]` is `NULL` or `bma_weights_*` CSV is missing, the function falls back to a default fit with `k = 2` and default horseshoe hyperparameters. |
| `seed` | Integer; base random seed for the fits. For different control combinations, the seed is jittered to avoid identical pseudo-random sequences. |

## Details

The function assumes the existence of:

- `control_combos`: a named object whose names are control tags (e.g.\ `"None"`, `"X1+X2"`, `"X1+X3+X4"`), defining which control sets to explore.
- `dir_csv`: a character scalar with the directory where CSV files will be read/written.
- `fit_one()`: a function that fits a single bivariate hurdle model and returns at least `$fit` (CmdStanR fit) and `$des` (design matrices).

For each control-combination tag tag:

- If a BMA weights file "bma_weights_spec<spec>_ctrl<tag>.csv" exists in dir_csv and k_bma_table[[tag]] is not NULL, the top-weighted row (highest weight) is used to select k and horseshoe hyperparameters (hs_tau0, hs_slab_scale, hs_slab_df) for the fit.
- Otherwise, the model is fit with k = 2 and default horseshoe hyperparameters.
- Posterior means of the regression coefficients with prefixes "b_mu_I", "b_pi_I", "b_mu_C", "b_pi_C" are extracted and mapped back to the corresponding column names of the design matrices.

All coefficient summaries are stacked into a single table and written to "eba_coefficients.csv" in dir_csv.

## Value

A data.frame with the columns:

- name: name of the covariate (design-matrix column).
- mean: posterior mean of the corresponding coefficient.
- block: block identifier ("mu_I", "pi_I", "mu_C", "pi_C").
- combo: control-combination tag used for that fit.

## Examples

```
library(data.table)

# 1. Create a COMPLETE dummy dataset
# This satisfies ALL requirements of build_design() and fit_one()
DT <- data.table(
  year = 2000:2020,
  # Dependent variables (Raw)
  I = rpois(21, lambda = 4),
  C = rpois(21, lambda = 3),
  # Dependent variables (Standardized/Transformed - required by build_design)
  zI = rnorm(21),
  zC = rnorm(21),
  # Trend variables (required if include_trend=TRUE)
  t_norm = seq(-1, 1, length.out = 21),
  t_poly2 = seq(-1, 1, length.out = 21)^2,
  # Regime (required if include_regimes=TRUE)
  Regime = factor(sample(c("A", "B"), 21, replace = TRUE)),
  # Transition dummies (required if include_transitions=TRUE)
  # Specifically: trans_PS, trans_SF, trans_FC
  trans_PS = sample(0:1, 21, replace = TRUE),
  trans_SF = sample(0:1, 21, replace = TRUE),
  trans_FC = sample(0:1, 21, replace = TRUE),
  # Exposure offset (required by fit_one)
  log_exposure50 = rep(0, 21),
  # Control variables (used in this specific example)
  X1 = rnorm(21),
  X2 = rnorm(21),
```

```
  X3 = rnorm(21)
)

# 2. Define global objects required by run_eba
control_combos <- list(
  None      = character(0),
  "X1+X2"   = c("X1", "X2"),
  "X1+X2+X3"= c("X1", "X2", "X3")
)

# 3. Define global paths using tempdir()
tmp_dir <- tempdir()
dir_csv <- file.path(tmp_dir, "csv")
if (!dir.exists(dir_csv)) dir.create(dir_csv, recursive = TRUE)

# 4. Run the function
# Note: This will attempt to run Stan. If CmdStan is not configured,
# it might fail later, but the DATA error is fixed.
try({
  eba_tab <- run_eba(DT, spec = "C", k_bma_table = list(), seed = 123)
  print(head(eba_tab))
})
```

---

`run_hmm` *Hidden Markov Model (HMM) for Path Dependence (Counts I and C)*

---

## Description

Fits a univariate time-series Hidden Markov Model (HMM) with Poisson emissions for the count variables `I` and `C` using **depmixS4**. The estimated state sequence is exported and the fit object is saved to disk.

## Usage

```
run_hmm(DT, nstates = 3, seed = NULL)
```

## Arguments

| | |
|---|---|
| `DT` | A `data.frame` or `data.table` containing at least the columns `I` and `C`, interpreted as non-negative count series observed over time. |
| `nstates` | Integer; number of latent Markov states to fit in the HMM (default is 3). |
| `seed` | Integer or NULL; optional seed for reproducibility. If NULL (default), no seed is set and results may vary between runs. |

## Details

The model is specified via `depmixS4::depmix()` as a multivariate Poisson HMM with two observed series:

- `I ~ 1`
- `C ~ 1`

and `nstates` hidden regimes. The function:

1. Builds a data frame with columns `I` and `C`.
2. Constructs the HMM with Poisson emission distributions for both series.
3. Optionally sets a random seed if the `seed` argument is provided.
4. Fits the model with `fit(mod, verbose = FALSE)` wrapped in `try()` to avoid stopping on optimization failures.
5. If fitting succeeds, extracts the posterior state sequence via `depmixS4::posterior()`.

The function assumes that two global character scalars are defined:

- `dir_csv`: directory where the state sequence CSV will be written.
- `dir_out`: directory where the fitted HMM object RDS will be saved.

A CSV file named `"hmm_states.csv"` is written to `dir_csv` with columns `t` (time index) and `state` (most probable state). The fitted HMM object is saved as `"hmm_fit.rds"` in `dir_out`.

## Value

If the optimization succeeds, a list with components:

- `fit`: the fitted `"depmix"` model object.
- `states`: integer vector of inferred latent states (one per time point).

If fitting fails (e.g., non-convergence), the function returns `NULL`.

## Examples

```
library(data.table)

# 1. Create dummy data (Only 'I' and 'C' counts are required by this function)
DT <- data.table(
  I = rpois(50, lambda = 4),
  C = rpois(50, lambda = 3)
)

# 2. Define global paths using tempdir() (Fixes CRAN policy)
# run_hmm expects these variables to exist in the global environment
tmp_dir <- tempdir()
dir_csv <- file.path(tmp_dir, "csv")
dir_out <- file.path(tmp_dir, "hmm")

dir.create(dir_csv, showWarnings = FALSE, recursive = TRUE)
```

```
dir.create(dir_out, showWarnings = FALSE, recursive = TRUE)

# 3. Run the function
# Using nstates=2 for a faster example check
res_hmm <- run_hmm(DT, nstates = 2)

# Inspect result if successful
if (!is.null(res_hmm)) {
  print(table(res_hmm$states))
}
```

---

`run_sensemakr` *Sensitivity Analysis to Unobserved Confounding (sensemakr)*

---

## Description

Performs the Cinelli & Hazlett style sensitivity analysis using **sensemakr** for two linear models:

- `I ~ trans_FC + t_norm + PopDensity + War`
- `C ~ trans_FC + t_norm + PopDensity + War`

treating `trans_FC` as the exposure of interest and using `PopDensity` and `War` as benchmark covariates.

## Usage

```
run_sensemakr(DT)
```

## Arguments

| | |
|---|---|
| `DT` | A `data.frame` or `data.table` containing at least the columns `I`, `C`, `trans_FC`, `t_norm`, `PopDensity`, and `War`. |

## Details

For each outcome (`I` and `C`), an OLS model is estimated and passed to `sensemakr::sensemakr()` with:

- `treatment = "trans_FC"`
- `benchmark_covariates = c("PopDensity", "War")`

The resulting `sensemakr` objects are summarized via `summary()`, converted to data frames, and written to CSV files:

- `"sensemakr_I_FC.csv"` for outcome `I`.
- `"sensemakr_C_FC.csv"` for outcome `C`.

The function assumes that a global character scalar `dir_csv` is defined and points to the directory where CSV outputs should be saved.

## Value

A list with components:

- `I`: the `sensemakr` object for the model with outcome `I`.
- `C`: the `sensemakr` object for the model with outcome `C`.

## Examples

```
library(data.table)

# 1. Create dummy data with ALL columns required by the lm() formulas
DT <- data.table(
  I = rpois(30, lambda = 5),
  C = rpois(30, lambda = 3),
  trans_FC = sample(0:1, 30, replace = TRUE),   # Treatment
  t_norm = rnorm(30),                           # Trend/Time
  PopDensity = rnorm(30),                       # Benchmark Covariate
  War = sample(0:1, 30, replace = TRUE)         # Benchmark Covariate
)

# 2. Define global path using tempdir() (Fixes CRAN policy)
# run_sensemakr writes output to 'dir_csv', so it must be defined.
tmp_dir <- tempdir()
dir_csv <- file.path(tmp_dir, "csv")
if (!dir.exists(dir_csv)) dir.create(dir_csv, recursive = TRUE)

# 3. Run the function
# This requires the 'sensemakr' package to be installed.
res_sense <- run_sensemakr(DT)

# Inspect results
if (!is.null(res_sense$I)) {
  print(summary(res_sense$I))
}
```

---

# run_synth_bsts *Synthetic Control via BSTS (CausalImpact)*

---

## Description

Builds a simple synthetic-control-style analysis using **CausalImpact**/BSTS for either `I` or `C` as the outcome, with treatment defined endogenously by a high level of a chosen control variable.

## Usage

```
run_synth_bsts(DT, outcome = c("I", "C"), control_var, seed = 123)
```

## Arguments

| | |
|---|---|
| DT | A data.frame or data.table containing at least:<br>• I, C: outcome candidates (counts or rates).<br>• EconCycle, PopDensity, Epidemics, Climate, War, t_norm: predictors used to build the synthetic control.<br>• The column named in control_var, used to define the treated period. |
| outcome | Character; which outcome series to use as the response, one of "I" or "C". |
| control_var | Character scalar; name of a column in DT whose high values define the treated period (e.g., intensity of some intervention or shock proxy). |
| seed | Integer; random seed for reproducibility of the BSTS fit. |

## Details

The function:

1. Selects the outcome series y <- DT[[outcome]].
2. Builds the predictor matrix from EconCycle, PopDensity, Epidemics, Climate, War, and t_norm.
3. Uses control_var to define a treated period as observations where control_var is in the top third (>= 2/3 quantile). If fewer than 5 treated observations are found, the function returns NULL.
4. Sets the intervention start time t0 as one period before the first treated index (with a minimum of 10 observations in the pre-period). The pre- and post-intervention windows are: pre.period = c(1, t0) and post.period = c(t0 + 1, length(y)).
5. Calls CausalImpact::CausalImpact() on the combined cbind(y, preds) matrix, with model.args = list(nseasons = 1).

From the resulting impact object, the function extracts the average absolute and relative effects from impact$summary and stores them in a small summary table with two rows: "abs_effect_mean" and "rel_effect_mean".

A CSV file named "causalimpact_<control_var>_on_<outcome>.csv" is written to the directory specified by a global character scalar dir_csv. If CausalImpact() fails, the function returns NULL.

## Value

On success, a list with components:

- impact: the full CausalImpact object.
- summary: a data.frame with the mean absolute and relative effects.

If the treated period is too short or the model fit fails, the function returns NULL.

## Examples

```
library(data.table)

# 1. Create dummy data with ALL required predictors
# The function explicitly selects: EconCycle, PopDensity, Epidemics, Climate, War, t_norm
DT <- data.table(
  year = 2000:2029,
  I = rpois(30, lambda = 10),
  C = rpois(30, lambda = 8),
  # Predictors required by run_synth_bsts internal selection
  EconCycle = rnorm(30),
  PopDensity = rnorm(30),
  Epidemics = rnorm(30),
  Climate = rnorm(30),
  War = rnorm(30),
  t_norm = seq(-1, 1, length.out = 30)
)

# 2. Define global paths using tempdir() (Fixes CRAN policy)
# run_synth_bsts writes output to 'dir_csv'
tmp_dir <- tempdir()
dir_csv <- file.path(tmp_dir, "csv")
if (!dir.exists(dir_csv)) dir.create(dir_csv, recursive = TRUE)

# 3. Run the function
# We use "War" as the control variable to define the treatment period
res_I <- run_synth_bsts(DT, outcome = "I", control_var = "War", seed = 123)

# Inspect results if successful (might return NULL if fit fails or not enough data)
if (!is.null(res_I)) {
  print(res_I$summary)
}
```

---

`run_transfer_entropy` *Transfer Entropy for Counts, Rates, and Binary Series*

---

## Description

Computes pairwise transfer entropy between `I` and `C` for three transformations of the data: raw counts, rates (count/exposure), and binary presence/absence. Each series is first pre-whitened via a GLM and transfer entropy is then estimated for a grid of lags using **RTransferEntropy**. Results are written to separate CSV files and to a combined summary.

## Usage

```
run_transfer_entropy(
  DT,
```

```
  lags = 1:3,
  shuffles = 1000,
  seed = 123,
  use_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| DT | A data.table or data.frame containing at least the following columns:<br>• I, C: count variables (non-negative integers).<br>• exposure50: exposure used to form rates (must be strictly positive).<br>• log_exposure50: log of the exposure (offset).<br>• t_norm, Regime, EconCycle, PopDensity, Epidemics, Climate, War: covariates used by the pre-whitening GLMs. |
| lags | Integer vector of lag orders L for which transfer entropy is computed (passed to lx and ly in RTransferEntropy::transfer_entropy()). |
| shuffles | Integer; number of shuffle replications for the surrogate-distribution-based significance test in transfer_entropy(). |
| seed | Integer; base random seed used for reproducibility of the pre-whitening and transfer entropy computations. |
| use_progress | Logical; reserved for future use to toggle progress reporting. Currently not used. |

## Details

The function proceeds in four steps:

1. **Counts**: I and C are pre-whitened via prewhiten_count_glm (Negative Binomial with offset and Poisson fallback). Transfer entropy is computed in both directions (I→C and C→I) for each lag in lags. Results are saved to "transfer_entropy_counts.csv".
2. **Rates**: I and C are divided by exposure50, pre-whitened via prewhiten_rate_glm, and transfer entropy is recomputed. Results are saved to "transfer_entropy_rates.csv". A check is performed to ensure exposure50 > 0 for all observations.
3. **Binary**: I and C are recoded as 0/1 presence/absence indicators and pre-whitened via prewhiten_bin_glm. Transfer entropy is computed again and results are saved to "transfer_entropy_binary.csv".
4. **Combined**: All tables are stacked into a single data frame with a type column ("counts", "rates", "binary") and written to "transfer_entropy.csv".

Internally, the helpers .get_stat and .get_pval are used to extract the transfer entropy statistic and p-value from the objects returned by RTransferEntropy::transfer_entropy(). The function assumes a global dir_csv object (character scalar) indicating the output directory for CSV files.

## Value

A data.frame with one row per lag and type, and columns:

- lag: lag order used in transfer_entropy().

- TE_ItoC, p_ItoC: transfer entropy and p-value from I to C.
- TE_CtoI, p_CtoI: transfer entropy and p-value from C to I.
- type: transformation used ("counts", "rates", or "binary").

## Examples

```
library(data.table)

# 1. Create dummy data with ALL covariates required by prewhiten_*_glm()
# The internal GLM formulas likely include:
# I ~ t_norm + Regime + EconCycle + PopDensity + Epidemics + Climate + War
DT <- data.table(
  year = 2000:2029,
  I = rpois(30, lambda = 10),
  C = rpois(30, lambda = 8),
  exposure50 = runif(30, 100, 200),
  log_exposure50 = log(runif(30, 100, 200)),
  # Covariates
  t_norm = seq(-1, 1, length.out = 30),
  Regime = factor(sample(c("A", "B"), 30, replace = TRUE)),
  EconCycle = rnorm(30),
  PopDensity = rnorm(30),
  Epidemics = rnorm(30),
  Climate = rnorm(30),
  War = rnorm(30)
)

# 2. Define global paths using tempdir() (Fixes CRAN policy)
# run_transfer_entropy writes output to 'dir_csv'
tmp_dir <- tempdir()
dir_csv <- file.path(tmp_dir, "csv")
if (!dir.exists(dir_csv)) dir.create(dir_csv, recursive = TRUE)

# 3. Run the function
# Using fewer shuffles for a faster example check
te_tab <- run_transfer_entropy(DT, lags = 1, shuffles = 10, seed = 123)

# Inspect results
if (!is.null(te_tab)) {
  print(subset(te_tab, type == "counts"))
}
```

---

run_varx *Fit VARX model with diagnostics for I and C*

---

## Description

Estimates a bivariate VAR model for I and C with exogenous covariates (VARX), and computes a set of standard diagnostics (stability, serial correlation, normality, ARCH). The fitted model and diagnostics are saved to disk and also returned.

## Usage

```
run_varx(DT, p = 2)
```

## Arguments

| | |
|---|---|
| DT | A data.table (or data.frame) containing at least the following columns:<br>• I, C: endogenous variables for the VAR.<br>• EconCycle, PopDensity, Epidemics, Climate, War, t_norm: exogenous regressors included in the VARX. |
| p | Integer; lag order of the VAR part (number of lags for I and C). |

## Details

The endogenous vector is $y_t = (I_t, C_t)'$ and the exogenous regressors are: EconCycle, PopDensity, Epidemics, Climate, War, t_norm. The model is fit using vars::VAR() with type = "const" and the exogenous matrix passed via exogen.

After estimation, the following diagnostics from **vars** are (attempted to be) computed:

- vars::stability(fit, type = "OLS-CUSUM") for stability.
- vars::serial.test(fit, lags.pt = 10, type = "PT.asymptotic") for serial correlation.
- vars::normality.test(fit) for residual normality.
- vars::arch.test(fit, lags.multi = 5) for ARCH effects.

Each diagnostic call is wrapped in try(), so if a diagnostic fails, the corresponding element in the output will contain a "try-error" instead of stopping the function.

The result is saved as an RDS file named "varx_fit.rds" in the directory specified by a global object dir_out (character scalar).

## Value

A list with components:

- fit: the estimated VAR model (vars object).
- stability: result of vars::stability() (or "try-error" on failure).
- serial: result of vars::serial.test() (or "try-error" on failure).
- normal: result of vars::normality.test() (or "try-error" on failure).
- arch: result of vars::arch.test() (or "try-error" on failure).

## Examples

```
library(data.table)

# 1. Create dummy data with ALL required columns for VARX
# The function explicitly requires these specific exogenous variables
DT <- data.table(
  year = 2000:2049, # 50 obs to ensure diagnostics (lags.pt=10) don't fail
  I = rpois(50, lambda = 10),
  C = rpois(50, lambda = 8),
  # Exogenous regressors required by the function
  EconCycle = rnorm(50),
  PopDensity = rnorm(50),
  Epidemics = rnorm(50),
  Climate = rnorm(50),
  War = rnorm(50),
  t_norm = seq(-1, 1, length.out = 50)
)

# 2. Define global output directory using tempdir() (Fixes CRAN policy)
# run_varx looks for 'dir_out' in the global environment
tmp_dir <- tempdir()
dir_out <- file.path(tmp_dir, "varx")
if (!dir.exists(dir_out)) dir.create(dir_out, recursive = TRUE)

# 3. Run the function
# We use p=1 to keep it fast and stable for the example check
res_varx <- run_varx(DT, p = 1)

# Inspect the fitted VAR object if it didn't fail
if (!inherits(res_varx$fit, "try-error")) {
  print(res_varx$fit)
}
```

---

# select_by_bma *Select Best Model via Bayesian Model Averaging*

---

## Description

Fits multiple bivariate hurdle models across a grid of lag orders and horseshoe hyperparameters, then performs model selection using LOO-CV and stacking weights.

## Usage

```
select_by_bma(
  DT,
  spec = "C",
  controls = character(0),
```

```
  k_grid = 0:3,
 hs_grid = data.frame(hs_tau0 = c(0.1, 0.5, 1), hs_slab_scale = c(1, 5, 1, 5, 1, 5),
   hs_slab_df = 4, stringsAsFactors = FALSE),
 model = NULL,
 output_base_dir = NULL,
 iter_warmup = 900,
 iter_sampling = 1200,
 chains = 4,
 seed = 123,
 use_parallel = TRUE,
 verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| DT | A data.table with the data. |
| spec | Character; model specification ("A", "B", "C", "D"). |
| controls | Character vector of control variable names. |
| k_grid | Integer vector of lag orders to evaluate. |
| hs_grid | Data.frame with columns hs_tau0, hs_slab_scale, hs_slab_df defining the horseshoe hyperparameter grid. |
| model | A compiled CmdStan model. If NULL, loads the default. |
| output_base_dir | Base directory for output files. If NULL, uses tempdir(). |
| iter_warmup | Integer; warmup iterations. |
| iter_sampling | Integer; sampling iterations. |
| chains | Integer; number of chains. |
| seed | Integer; random seed. |
| use_parallel | Logical; if TRUE and furrr is available, fits models in parallel. |
| verbose | Logical; print progress messages. |

## Value

A list with components:

| | |
|---|---|
| fits | List of fitted model objects. |
| loos | List of LOO objects. |
| weights | Numeric vector of stacking weights. |
| table | Data.frame with results sorted by ELPD. |

## Examples

```
library(data.table)

# 1. Create a COMPLETE dummy dataset
# select_by_bma -> fit_one -> build_design requires ALL these columns:
DT <- data.table(
  year = 2000:2020,
  I = rpois(21, lambda = 4),
  C = rpois(21, lambda = 3),
  zI = rnorm(21),
  zC = rnorm(21),
  t_norm = seq(-1, 1, length.out = 21),
  t_poly2 = seq(-1, 1, length.out = 21)^2,
  Regime = factor(sample(c("A", "B"), 21, replace = TRUE)),
  trans_PS = sample(0:1, 21, replace = TRUE),
  trans_SF = sample(0:1, 21, replace = TRUE),
  trans_FC = sample(0:1, 21, replace = TRUE),
  log_exposure50 = rep(0, 21)
)

# 2. Run the function
# IMPORTANT: use_parallel = FALSE to avoid complexity/errors in CRAN checks
# We reduce the grid size (k_grid=0) for speed in this example
try({
  result <- select_by_bma(
    DT,
    spec = "C",
    k_grid = 0,
    hs_grid = data.frame(hs_tau0=0.5, hs_slab_scale=1, hs_slab_df=4),
    use_parallel = FALSE,
    iter_warmup = 100, iter_sampling = 100, chains = 1 # Minimal MCMC for speed
  )

  if (!is.null(result$table)) {
    print(result$table)
  }
})
```

---

# smoketest_floor_elpd_invariance

*Smoke Test for FLOOR ELPD Invariance*

---

## Description

Tests that the ELPD ranking is invariant to different FLOOR penalty values in the Stan model.

## Usage

```
smoketest_floor_elpd_invariance(
  DT,
  stan_code,
  floors = c(-1e+06, -1e+08, -10000),
  spec = "C",
  controls = character(0),
  k_grid = 0:1,
 hs_grid = data.frame(hs_tau0 = c(0.1, 0.5), hs_slab_scale = c(1, 5), hs_slab_df = 4),
  hs_rows = 1:2,
  iter_warmup = 200,
  iter_sampling = 200,
  chains = 2,
  seed = 123,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| DT | Data.table with the data. |
| stan_code | Character; Stan model code. |
| floors | Numeric vector of FLOOR values to test. |
| spec | Character; model specification. |
| controls | Character vector of control variables. |
| k_grid | Integer vector of lag values to test. |
| hs_grid | Data.frame with horseshoe hyperparameter grid. |
| hs_rows | Integer vector; which rows of hs_grid to use. |
| iter_warmup | Integer; warmup iterations. |
| iter_sampling | Integer; sampling iterations. |
| chains | Integer; number of chains. |
| seed | Integer; random seed. |
| verbose | Logical; print progress messages. |

## Value

A list with components:

| | |
|---|---|
| same_order | Logical; TRUE if ranking is identical across all FLOOR values. |
| floors | The tested FLOOR values. |
| tables | List of result tables for each FLOOR. |
| combined | Combined data.frame of all results. |
| rank_signatures | Character vector of ranking signatures. |

## standardize_continuous

*Standardize Continuous Columns*

### Description

Standardizes selected numeric columns using z-score or robust (median/MAD) methods. Binary columns (0/1) are left unchanged.

### Usage

```
standardize_continuous(
  DT,
  cols,
  method = c("zscore", "robust"),
  center = TRUE,
  scale = TRUE
)
```

### Arguments

| | |
|---|---|
| DT | A data.table or data.frame. |
| cols | Character vector of column names to standardize. |
| method | Character; either "zscore" or "robust". |
| center | Logical; whether to center the data. |
| scale | Logical; whether to scale the data. |

### Value

A list with components:

| | |
|---|---|
| DT | The standardized data.table. |
| scalers | A list of scaling parameters for each column. |

## standardize_continuous_in_place

*Standardize Continuous Columns In Place*

### Description

Standardizes selected numeric columns of a data.table in place using a z-score transformation. The function modifies DT by reference and stores the means and standard deviations used in an attribute called "standardization".

## Usage

```
standardize_continuous_in_place(DT, cols, center = TRUE, scale = TRUE)
```

## Arguments

| | |
|---|---|
| DT | A data.table. It is modified by reference. |
| cols | Character vector of column names to standardize. Columns that are not present in DT or are not numeric are silently skipped. |
| center | Logical; whether to subtract the column mean. |
| scale | Logical; whether to divide by the column standard deviation. |

## Value

The modified data.table DT (invisibly), with an attribute "standardization" containing the means, standard deviations, and names of the standardized columns.

## Examples

```
library(data.table)
DT <- data.table(x = rnorm(10), y = runif(10), z = 0:9)
standardize_continuous_in_place(DT, c("x", "y"))
attr(DT, "standardization")
```

---

# summarise_hurdle_top3_posthoc

*Summarise top-3 Hurdle-NB models across control combos*

---

## Description

Extracts and summarises the top three Hurdle-NB specifications (by estimated ELPD) from BMA selection tables, either taken from an in-memory list of results or read from CSV files on disk.

## Usage

```
summarise_hurdle_top3_posthoc(bma_per_combo, dir_csv)
```

## Arguments

| | |
|---|---|
| bma_per_combo | Optional named list of BMA results by control combination, where each element contains a component $table with columns such as elpd, elpd_se, weight, k, hs_tau0, hs_slab_scale, hs_slab_df, etc. |
| dir_csv | Character scalar; directory where BMA weight CSV files "bma_weights_specC_ctrl*.csv" are stored if bma_per_combo is NULL or empty. |

## Details

If bma_per_combo is provided and non-empty, the function uses its $table components. Otherwise, it scans dir_csv for BMA weight files matching the pattern "bma_weights_specC_ctrl*.csv" and reads them.

All valid rows are combined, ordered by decreasing elpd, and the top three models are retained. For each, a human-readable configuration string summarising k, the horseshoe hyperparameters and the control combo is constructed.

## Value

A data frame with up to three rows and columns:

- model: constant string "Hurdle-NB".
- config: textual description of the specification.
- elpd, elpd_se, weight: selection metrics from the BMA table.
- k, hs_tau0, hs_slab_scale, hs_slab_df, combo: numeric tuning parameters and control-combo tag.

If no valid tables are found, a single-row data frame with NA entries is returned.

---

# summarise_placebo_top3_posthoc

*Summarise top-3 temporal placebo results*

---

## Description

Summarises the three strongest temporal placebo results (based on the difference between original and permuted ELPD) from a temporal permutation test.

## Usage

```
summarise_placebo_top3_posthoc(placebo_tab, dir_csv)
```

## Arguments

| | |
|---|---|
| placebo_tab | Optional data frame with placebo results, typically containing columns perm, elpd_orig, elpd_perm, and diff. If NULL or empty, the function attempts to read "placebo_temporal.csv" from dir_csv. |
| dir_csv | Character scalar; directory where the placebo CSV file is stored. |

## Details

The table is ordered by decreasing diff (ELPD gain of the original fit over the permuted fit), and the top three permutations are retained.

### Value

A data frame with up to three rows and columns:

- `model`: constant string `"PlaceboTemporal"`.
- `config`: text of the form `"perm=<id>"`.
- `elpd_orig`, `elpd_perm`, `diff`: original ELPD, permuted ELPD, and their difference.

If no data are available, a single-row data frame with `NA` entries is returned.

---

## summarise_te_top3_by_type_posthoc

*Summarise top-3 transfer entropy results by type*

---

### Description

Produces a list of small tables with the three most significant transfer entropy estimates for each data type (counts, rates, binary) separately.

### Usage

```
summarise_te_top3_by_type_posthoc(te_tab, dir_csv)
```

### Arguments

| | |
|---|---|
| `te_tab` | Optional data frame with transfer entropy results, including a `type` column and at least `lag`, `TE_ItoC`, `TE_CtoI`, `p_ItoC`, `p_CtoI`. If `NULL` or empty, the function attempts to read the data from CSV files via `.read_te_all()`. |
| `dir_csv` | Character scalar; directory where the transfer entropy CSV files are stored (used when `te_tab` is missing). |

### Details

For each type in `c("counts", "rates", "binary")`, the function ranks all direction-lag combinations by p-value and retains the top three. Types with no valid rows remain `NULL` in the output list.

### Value

A named list with up to three elements:

- `$counts`, `$rates`, `$binary`: each is a data frame with columns `model`, `type`, `config` (direction and lag), `stat`, and `p_value`, or `NULL` if no results for that type.

summarise_te_top3_posthoc *Summarise top-3 transfer entropy results (global)*

## Description

Produces a compact summary of the three most statistically significant transfer entropy estimates across directions and lags, optionally combining information from counts, rates, and binary specifications.

## Usage

```
summarise_te_top3_posthoc(te_tab, dir_csv)
```

## Arguments

| | |
|---|---|
| te_tab | Optional data frame with transfer entropy results, containing at least columns lag, TE_ItoC, TE_CtoI, p_ItoC, p_CtoI, and optionally type. If NULL or empty, the function attempts to read the data from CSV files via the internal helper .read_te_all(). |
| dir_csv | Character scalar; directory where the transfer entropy CSV files are stored (used when te_tab is missing). |

## Details

The function reshapes te_tab into a long format with directions "I->C" and "C->I", orders by p-value (ascending) and lag, and keeps the three rows with the smallest p-values.

## Value

A data frame with up to three rows and columns:

- model: constant string "TransferEntropy".
- config: textual description of direction, lag, and, if available, type (counts, rates, binary).
- stat: transfer entropy estimate.
- p_value: associated p-value.

If no results are available, a single-row data frame with NA entries is returned.

summarise_tvarstar_posthoc *Summarise nonlinear time-series models (TVAR and LSTAR)*

## Description

Produces a small summary table for nonlinear time-series models such as TVAR and LSTAR, focusing on model status and AIC.

## Usage

```
summarise_tvarstar_posthoc(tsdyn_res)
```

## Arguments

| | |
|---|---|
| `tsdyn_res` | A list of model objects, typically with elements `$TVAR`, `$LSTAR_I`, `$LSTAR_C`, as returned by a fitting routine based on the **tsDyn** package. |

## Details

For each of the three models (TVAR, LSTAR for I, LSTAR for C), the function extracts:

- A textual status (class names of the object).
- The AIC, if `stats::AIC()` can be computed.

If `tsdyn_res` is `NULL`, default rows with `NA` values are returned.

## Value

A data frame with one row per model and columns:

- `model`: "TVAR", "LSTAR_I", "LSTAR_C".
- `status`: model class string or `NA`.
- `aic`: numeric AIC value or `NA`.

summarise_varx_posthoc *Summarise VARX model fit and diagnostics*

## Description

Produces a compact summary of a VARX model, including information about lag order, exogenous variables, information criteria, and selected diagnostic p-values.

## Usage

```
summarise_varx_posthoc(varx_res)
```

## Arguments

| | |
|---|---|
| varx_res | A list returned by run_varx(), typically containing elements $fit, $serial, $normal, and $arch. |

## Details

The function extracts:

- Lag order p from fit$p, if available.
- AIC and BIC via stats::AIC() and stats::BIC().
- P-values from serial correlation, normality, and ARCH tests using the helper .first_pvalue().

If varx_res or varx_res$fit is NULL, a default row with NA values is returned.

## Value

A data frame with one row and columns:

- model: constant string "VARX".
- config: textual description of the lag order and exogenous variables.
- AIC, BIC: information criteria.
- p_serial, p_normal, p_arch: p-values from diagnostic tests.

# Index