Package 'bayesianOU'

June 18, 2026

Type Package

Title Bayesian Nonlinear Ornstein-Uhlenbeck Models with Stochastic Volatility

Version 0.2.0

Description Fits Bayesian nonlinear Ornstein-Uhlenbeck models with cubic drift, stochastic volatility, and Student-t innovations. The package implements hierarchical priors for sector-specific parameters and supports parallel MCMC sampling via 'Stan'. Model comparison is performed using Pareto Smoothed Importance Sampling Leave-One-Out (PSIS-LOO) cross-validation following Vehtari, Gelman, and Gabry (2017) <doi:10.1007/s11222-016-9696-4>. Prior specifications follow recommendations from Gelman (2006) <doi:10.1214/06-BA117A> for scale parameters.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, graphics, utils, parallel

Suggests cmdstanr, rstan (>= 2.21.0), loo (>= 2.5.0), posterior, ggplot2, tidyr, openxlsx, knitr, rmarkdown, testthat (>= 3.0.0)

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

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/IsadoreNabi/bayesianOU

BugReports https://github.com/IsadoreNabi/bayesianOU/issues

Config/roxygen2/version 8.0.0

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 2026-06-18 19:30:02 UTC

Contents

`build_accounting_block` *Build accounting block for TMG*

Description

Creates accounting information for TMG transformations.

Usage

```
build_accounting_block(
  TMG_raw,
  zTMG_exo,
  zTMG_use,
  mu_tmg,
  sd_tmg,
  hard,
  sigma_delta
)
```

Arguments

TMG_raw	Numeric vector. Original TMG series.
zTMG_exo	Numeric vector. Exogenous z-scored TMG.
zTMG_use	Numeric vector. TMG used in model (possibly orthogonalized).
mu_tmg	Numeric. Training mean of TMG.
sd_tmg	Numeric. Training SD of TMG.
hard	Logical. Whether hard sum-to-zero constraint was used.
sigma_delta	Numeric. Prior SD for wedge in original units.

Value

List with components:

tmg_byK Back-transformed TMG used in model

tmg_exo Back-transformed exogenous TMG

wedge_delta Difference (zero if hard=TRUE)

sigma_delta_prior Prior SD for wedge

note Description of constraint type

---

build_beta_tmg_table *Build beta(TMG_t) table by sector and time*

Description

Constructs the time-varying beta matrix using posterior medians.

Usage

```
build_beta_tmg_table(fit, zTMG_use, summ = NULL)
```

Arguments

`fit`	Fitted Stan model object
`zTMG_use`	Numeric vector. Standardized TMG series used in fitting.
`summ`	Optional list from `extract_posterior_summary`. If supplied, it is reused instead of recomputing the (expensive) summary.

Details

This is a deterministic point reconstruction $\beta_s(t) = \beta_{0,s} + \beta_1 \cdot zTMG_t$ evaluated at the posterior medians of `beta0_s` and `beta1`. It is NOT a sampled time-varying coefficient: the time variation comes only from `zTMG_t`.

Value

List with components:

beta_point Matrix (T x S) of beta values

meta List with description metadata

`compare_models_loo` *Compare models using PSIS-LOO*

Description

Compares two fitted models using PSIS-LOO cross-validation.

Usage

```
compare_models_loo(results_new, results_base)
```

Arguments

`results_new`	List. Results from new model.
`results_base`	List. Results from base model.

Value

List with components:

loo_table Comparison table from loo::loo_compare

deltaELPD Numeric difference in ELPD

Examples

```
if (requireNamespace("loo", quietly = TRUE)) {
  # 1. Create mock 'loo' objects manually to avoid computation errors
  # Structure required by loo_compare: list with 'estimates' and class 'psis_loo'

  # Mock Model 1: ELPD = -100
  est1 <- matrix(c(-100, 5), nrow = 1, dimnames = list("elpd_loo", c("Estimate", "SE")))
  # Pointwise data (required for diff calculation). 10 observations.
  pw1 <- matrix(c(rep(-10, 10)), ncol = 1, dimnames = list(NULL, "elpd_loo"))

  loo_obj1 <- list(estimates = est1, pointwise = pw1)
  class(loo_obj1) <- c("psis_loo", "loo")

  # Mock Model 2: ELPD = -102 (worse)
  est2 <- matrix(c(-102, 5), nrow = 1, dimnames = list("elpd_loo", c("Estimate", "SE")))
  pw2 <- matrix(c(rep(-10.2, 10)), ncol = 1, dimnames = list(NULL, "elpd_loo"))

  loo_obj2 <- list(estimates = est2, pointwise = pw2)
  class(loo_obj2) <- c("psis_loo", "loo")

  # 2. Wrap in the structure expected by your package
  res_new <- list(diagnostics = list(loo = loo_obj1))
  res_base <- list(diagnostics = list(loo = loo_obj2))

  # 3. Compare (This will now run cleanly without warnings)
  cmp <- compare_models_loo(res_new, res_base)
  print(cmp)
}
```

---

count_divergences *Count HMC divergences*

Description

Extracts the number of divergent transitions from a fitted Stan model.

Usage

```
count_divergences(fit)
```

Arguments

fit	Fitted Stan model object (CmdStanMCMC or stanfit)

Value

Integer. Number of divergent transitions (post-warmup).

Examples

```
# Create a "mock" CmdStanMCMC object for demonstration
# (This simulates a model with 0 divergences)
mock_fit <- structure(list(
  sampler_diagnostics = function() {
    # Return a 3D array: [iterations, chains, variables]
    # Variable 1 is usually accept_stat__, let's say var 2 is divergent__
    ar <- array(0, dim = c(100, 4, 6))
    dimnames(ar)[[3]] <- c("accept_stat__", "divergent__", "energy__",
                           "n_leapfrog__", "stepsize__", "treedepth__")
    return(ar)
  }
), class = "CmdStanMCMC")

# Now the example can run without errors:
n_div <- count_divergences(mock_fit)
print(n_div)
```

`drift_decomposition_grid` *Drift decomposition over grid*

Description

Computes the cubic OU drift function over a grid of *centered* deviations $z = Y - \theta_s$. The evaluated function is $\kappa_s(-z + a_{3,s}z^3)$, i.e. the drift expressed in the deviation coordinate, not in the original Y coordinate. To plot against Y, shift the grid by the corresponding `theta_s`.

Usage

```
drift_decomposition_grid(fit, summ, z_grid = seq(-2.5, 2.5, length.out = 101))
```

Arguments

`fit`	Fitted Stan model object (reserved for future use)
`summ`	List. Output from `extract_posterior_summary`
`z_grid`	Numeric vector. Grid of centered deviation values (Y - theta).

Value

List with components:

z The input z grid

drift Matrix (length(z) x S) of drift values by sector

`evaluate_oos` *Evaluate out-of-sample forecast metrics*

Description

Computes RMSE and MAE for multiple forecast horizons.

Usage

```
evaluate_oos(
  summ,
  Yz,
  Xz,
  zTMG,
  T_train,
  COM_ts,
  K_ts,
  com_in_mean = FALSE,
  horizons = c(1, 4, 8)
)
```

Arguments

`summ`	List. Posterior summary from `extract_posterior_summary`
`Yz`	Numeric matrix. Standardized Y values (T x S)
`Xz`	Numeric matrix. Standardized X values (T x S)
`zTMG`	Numeric vector. Standardized TMG series
`T_train`	Integer. End of training period
`COM_ts`	Numeric matrix. COM values by time and sector (T x S)
`K_ts`	Numeric matrix. Capital values by time and sector (T x S)
`com_in_mean`	Logical. Whether COM is included in mean equation
`horizons`	Integer vector. Forecast horizons to evaluate

Details

Two caveats matter for interpretation:

1. **Conditional forecast.** The recursion uses the *realized* future covariates X_{t-1} and $zTMG_t$ at each step. It is therefore a forecast of Y *conditional on* the future paths of X and TMG, not an unconditional forecast. If those covariates are not known ex ante, treat these numbers as conditional/nowcasting metrics.
2. **Plug-in medians.** The path is propagated with posterior medians of the parameters and ignores parameter uncertainty, the stochastic volatility and the Student-t innovations. It is a point (mean-equation) forecast, not the full posterior predictive distribution. For genuinely out-of-sample numbers, fit with `fit_window = "train"`.

Value

Named list with one element per horizon, each a list with `h`, `RMSE`, `MAE` and `n_obs` (number of pooled sector-by-origin errors; `0` / `NA` when the horizon exceeds the test window).

Examples

```
# 1. Generate dummy data for testing
T_obs <- 20
S <- 2
Yz <- matrix(rnorm(T_obs * S), nrow = T_obs, ncol = S)
Xz <- matrix(rnorm(T_obs * S), nrow = T_obs, ncol = S)
COM_ts <- matrix(abs(rnorm(T_obs * S)), nrow = T_obs, ncol = S)
K_ts <- matrix(abs(rnorm(T_obs * S)) + 1, nrow = T_obs, ncol = S)
zTMG <- rnorm(T_obs)

# 2. Create a dummy summary list (mimicking extract_posterior_summary)
summ <- list(
  theta_s = runif(S),
  kappa_s = runif(S),
  a3_s = runif(S),
  beta0_s = runif(S),
  beta1 = 0.5,
  gamma = 0.1
)

# 3. Run the function
metrics <- evaluate_oos(summ, Yz, Xz, zTMG, T_train = 15,
                        COM_ts, K_ts, horizons = c(1, 2))
print(metrics)
```

---

evaluate_oos_nested *Out-of-sample forecast metrics for the 2-level nested model*

Description

Out-of-sample analogue of `evaluate_oos` for the nested model. The decisive difference from the single-level recursion is the attractor: in the single-level model the market price reverts to a constant level with a linear forcing by the exogenous production-price covariate X; in the nested model the market price φ reverts to the *latent* production price Φ, which is itself propagated forward through its own Level-2 Ornstein-Uhlenbeck equation toward the G'-driven mean $\mu_{s,t} = m_{0,s} + m_1 G'_t$. Both states are advanced jointly; the market never sees the realized future X.

Usage

```
evaluate_oos_nested(
  meds,
  Yz,
  Phi_med,
```

```
  Gprime_z,
  zTMG,
  T_train,
  COM_ts,
  K_ts,
  kappa_cap,
  com_in_mean = FALSE,
  horizons = c(1, 4, 8),
  V_z = NULL
)
```

Arguments

meds	List of posterior medians with components kappa_tilde (length S, the per-sector pre-link market reversion), a3_s (length S), beta1 (scalar), gamma (scalar), kappa_p (length S), mu_const (length S, the Level-2 mean intercept $m_{0,s}$) and m1 (scalar, the G' slope). For the Level-3 value model it may also carry m_v (scalar, the value coupling); paired with V_z it adds the value term to the Level-2 attractor.
Yz	Numeric matrix (T x S). Standardized market price φ.
Phi_med	Numeric matrix (T x S). Posterior-median latent production-price path $\hat{\Phi}$ (seeds the Level-2 recursion).
Gprime_z	Numeric vector (length T). Standardized aggregate profit rate G' (training statistics), the Level-2 mean driver.
zTMG	Numeric vector (length T). Standardized TMG series used in fitting.
T_train	Integer. End of the training window.
COM_ts	Numeric matrix (T x S). Composition of capital by sector.
K_ts	Numeric matrix (T x S). Total capital (training weights for the COM standardization, matching the Stan transformed-data block).
kappa_cap	Numeric > 0. Stability cap used in fitting.
com_in_mean	Logical. Whether the COM term enters the mean. Default FALSE.
horizons	Integer vector. Forecast horizons to evaluate.
V_z	Numeric matrix (T x S) or NULL. Standardized value anchor (direct prices $c+v+p$) for the Level-3 model. When supplied together with a meds$m_v coupling, the Level-2 attractor gains the value term so $\mu_{s,t} = m_{0,s} + m_1 G'_t + m_v V_{s,t}$ (faithful out-of-sample recursion for n_levels = 3). NULL (default) reproduces the Level-2 G'-driven mean exactly.

Details

At each forecast origin t_0 the recursion is seeded with the realized standardized market price φ_{t_0-1} and the posterior-median latent production price $\hat{\Phi}_{t_0-1}$, then for $h = 1, \ldots$ steps:

$$\Phi_t = \Phi_{t-1} + \kappa_s^p(\mu_{s,t} - \Phi_{t-1}), \quad \mu_{s,t} = m_{0,s} + m_1 G'_t,$$

$$\varphi_t = \varphi_{t-1} + \kappa_s^m(-d_{t-1} + a_{3,s} d_{t-1}^3) + \gamma\,\mathrm{COM}_{t-1,s}, \quad d_{t-1} = \varphi_{t-1} - \Phi_{t-1},$$

with $\kappa_s^m = \texttt{kappa\_cap} \cdot \text{logit}^{-1}(\tilde{\kappa}_s + \beta_1 zTMG_t)$ the bounded gravitation speed (the same logit link used in fitting). The market step uses Φ_{t-1} (one-step lag), matching the Stan likelihood.

For the Level-3 value model (`n_levels = 3`), the Level-2 attractor also tracks the value index, $\mu_{s,t} = m_{0,s} + m_1 G'_t + m_v V_{s,t}$ (production prices gravitate around values, Capital III ch. IX), recovered by supplying the standardized value anchor `V_z` and the coupling `meds$m_v`. Without them the recursion is the plain Level-2 mean.

Two caveats, identical in spirit to `evaluate_oos`:

1. **Conditional forecast.** The recursion still consumes the realized future G'_t and $zTMG_t$ (the macro drivers), so it is a forecast *conditional* on those aggregate paths, not on the sectoral production prices (which are now latent and forecast endogenously).
2. **Plug-in medians.** States are propagated with posterior medians and ignore parameter uncertainty, the latent-Φ innovation, the stochastic volatility and the Student-t tails. It is a point (mean-equation) forecast, not the full posterior predictive distribution. Fit with `fit_window = "train"` for genuinely held-out numbers.

Value

Named list with one element per horizon, each a list with `h`, `RMSE`, `MAE` and `n_obs` (pooled sector-by-origin errors; `0` / `NA` when the horizon exceeds the test window). Errors are in standardized φ units, directly comparable with `evaluate_oos`.

See Also

`evaluate_oos` (single-level), `fit_ou_nested`.

`export_model_comparison` *Export model comparison to Excel*

Description

Creates an Excel workbook with model comparison results, parameter summaries, and fit information.

Usage

```
export_model_comparison(
  results_new,
  results_base,
  path = file.path(tempdir(), "model_comparison.xlsx"),
  verbose = FALSE
)
```

Arguments

results_new	List. Results from new model.
results_base	List. Results from base model.
path	Character. Output file path. Default "model_comparison.xlsx".
verbose	Logical. Print progress messages. Default FALSE.

Details

Creates three worksheets:

- loo_comparison: PSIS-LOO comparison table
- param_summary: Sector-specific parameter estimates
- fit_info: Model configuration and diagnostics

Value

TRUE invisibly on success.

Examples

```
if (requireNamespace("openxlsx", quietly = TRUE) &&
    requireNamespace("loo", quietly = TRUE)) {

  # 1. Create mock results objects
  # Mock Model New
  res_new <- list(
    factor_ou = list(
      kappa_s = c(0.5, 0.6), a3_s = c(-0.1, -0.2), beta0_s = c(1, 2),
      gamma = 0.05, model = "TestModel", beta1 = 0.3, nu = 4,
      factor_ou_info = list(T_train = 50, com_in_mean = TRUE)
    ),
    diagnostics = list(
      divergences = 0,
      # Mock LOO object
      loo = list(
      estimates = matrix(c(-100, 2), 1, 2, dimnames=list("elpd_loo", c("Estimate","SE"))),
         pointwise = matrix(rep(-2, 50), ncol=1)
      )
    )
  )
  class(res_new$diagnostics$loo) <- c("psis_loo", "loo")

  # Mock Model Base
  res_base <- list(
    diagnostics = list(
      loo = list(
      estimates = matrix(c(-110, 2), 1, 2, dimnames=list("elpd_loo", c("Estimate","SE"))),
         pointwise = matrix(rep(-2.2, 50), ncol=1)
      )
    )
```

```
  )
  class(res_base$diagnostics$loo) <- c("psis_loo", "loo")

  # 2. Define a safe temporary path
  out_path <- file.path(tempdir(), "comparison.xlsx")

  # 3. Run export (This writes to tempdir, allowed by CRAN)
  try({
    export_model_comparison(res_new, res_base, path = out_path)
    # unlink(out_path) # Cleanup
  })
}
```

`extract_convergence_evidence` *Mean-reversion (dynamic stability) evidence for kappa parameters*

Description

Computes 95 percent credible intervals for each `kappa_s` (mean reversion speed) and the posterior probability that every sector reverts.

Usage

```
extract_convergence_evidence(fit_res, verbose = TRUE)
```

Arguments

`fit_res`	List returned by `fit_ou_nonlinear_tmg`
`verbose`	Logical. Print summary to console. Default TRUE.

Value

List with components:

kappa_ic95 Matrix (S x 3) with columns q2.5, median, q97.5

stable Logical indicating if all kappa CIs fall in (0,1)

convergence Deprecated alias of `stable` (kept for back-compat)

prob_stable Posterior probability of joint mean reversion

prob_convergence Deprecated alias of `prob_stable`

Important - this is NOT MCMC convergence

Despite the historical name, this function does NOT assess sampler convergence. It evaluates a *dynamic* property of the estimated process: whether the mean-reversion speed lies in a range consistent with a stable, reverting (gravitating) system. For sampler convergence use R-hat, ESS and divergences (see `validate_ou_fit`). The threshold `kappa < 1` is a (conservative) monotone-reversion criterion; under the Euler discretization the linear map is stable for `0 < kappa < 2`. The alias `kappa_stability_evidence` is preferred.

Examples

```
# 1. Create a mock fit object with kappa draws
# kappa_tilde is log(kappa), so we use log(0.5) roughly -0.69
n_draws <- 100
S <- 2
kappa_tilde_draws <- matrix(rnorm(n_draws * S, mean = -0.7, sd = 0.1),
                            nrow = n_draws, ncol = S)
colnames(kappa_tilde_draws) <- c("kappa_tilde[1]", "kappa_tilde[2]")

mock_fit <- structure(list(
  draws = function(vars, format="matrix") {
    if (vars == "kappa_tilde") return(kappa_tilde_draws)
    return(NULL)
  }
), class = "CmdStanMCMC")

# 2. Wrap in the results list structure
results_mock <- list(
  factor_ou = list(
    stan_fit = mock_fit
  )
)

# 3. Extract evidence
conv <- extract_convergence_evidence(results_mock)
print(conv$prob_convergence)
```

`extract_mu_trajectory` *Latent Level-2 mean trajectory mu_t = m0_s + m1 * G'_t*

Description

Reconstructs the time-varying Level-2 mean toward which the latent production price Φ reverts, $\mu_{s,t} = m_{0,s} + m_1 G'_t$, with credible bands obtained by evaluating the expression over the posterior draws of `mu_const` ($m_{0,s}$) and `m1`. This is the structural hook to the (not-yet-formalized) Level 3 of values: μ is the slow attractor of the production price, here driven by the aggregate profit rate G'.

Usage

```
extract_mu_trajectory(fit, Gprime_z, sector_names = NULL, V_z = NULL)
```

Arguments

`fit`	Fitted Stan object (`CmdStanMCMC` or `stanfit`) from a 2-level fit.
`Gprime_z`	Numeric vector (length T). Standardized aggregate profit rate used in fitting (training statistics).
`sector_names`	Character vector (length S) or NULL. Column names for the returned matrices.
`V_z`	Numeric matrix (T x S) or NULL. Standardized value anchor (direct prices $c + v + p$) for the Level-3 model (`n_levels = 3`). When supplied and the fit carries the value coupling `m_v`, the mean trajectory gains the value term $m_v V_{s,t}$, so $\mu_{s,t} = m_{0,s} + m_1 G'_t + m_v V_{s,t}$ (production prices gravitate around values, Capital III ch. IX). NULL (default) reproduces the Level-2 G'-driven mean.

Value

List with `median`, `q2.5`, `q97.5` (each a T x S matrix) and `Gprime_z` (the driver), or `NULL` if the Level-2 parameters are not present in the fit.

See Also

`fit_ou_nested`.

---

extract_posterior_summary

Extract posterior summary from fitted model

Description

Extracts median point estimates and credible intervals for all model parameters from a fitted Stan model.

Usage

```
extract_posterior_summary(fit)
```

Arguments

`fit`	Fitted Stan model object (CmdStanMCMC or stanfit)

Value

List with components:

beta1 Median of global TMG effect

beta0_s Vector of sector-specific intercepts

kappa_s Vector of mean reversion speeds

a3_s Vector of cubic drift coefficients

theta_s Vector of equilibrium levels

rho_s Vector of SV persistence parameters

alpha_s Vector of SV level parameters

sigma_eta_s Vector of SV volatility parameters

nu Degrees of freedom for Student-t

gamma COM effect in mean

rhat R-hat convergence diagnostics

ess Effective sample sizes

Examples

```
# 1. Create a mock CmdStanMCMC object
# We simulate a posterior distribution for 2 sectors
S <- 2
n_draws <- 100

# Helper to generate random draws
mock_draws <- function(name, n_cols=1) {
  m <- matrix(rnorm(n_draws * n_cols), nrow = n_draws, ncol = n_cols)
  if (n_cols > 1) {
    colnames(m) <- paste0(name, "[", 1:n_cols, "]")
  } else {
    colnames(m) <- name
  }
  as.data.frame(m)
}

# Combine draws into one data frame
df_draws <- cbind(
  mock_draws("beta1", 1),
  mock_draws("beta0_s", S),
  mock_draws("kappa_tilde", S), # Note: function expects log-scale kappa
  mock_draws("a3_tilde", S),    # Note: function expects log-scale a3
  mock_draws("theta_s", S),
  mock_draws("rho_s", S),
  mock_draws("alpha_s", S),
  mock_draws("sigma_eta_s", S),
  mock_draws("nu_tilde", 1),
  mock_draws("gamma", 1)
)
```

```
# Mock fit object
mock_fit <- structure(list(
  draws = function(vars, format="df") {
     # Simple regex matching for the mock
     if (length(vars) == 1) {
       # Check if it's a scalar or vector parameter request
       if (vars %in% names(df_draws)) return(df_draws[vars])
       # Pattern match for vectors like "beta0_s" -> "beta0_s[1]", "beta0_s[2]"
       cols <- grep(paste0("^", vars, "\\["), names(df_draws), value = TRUE)
       if (length(cols) > 0) return(df_draws[cols])
     }
     return(df_draws)
  },
  summary = function() {
    data.frame(
      variable = names(df_draws),
      rhat = rep(1.0, ncol(df_draws)),
      ess_bulk = rep(400, ncol(df_draws))
    )
  }
), class = "CmdStanMCMC")

# 2. Run extraction
summ <- extract_posterior_summary(mock_fit)
print(summ$kappa_s)
```

`fit_ou_nested` *Fit the unified nonlinear OU model: single-level or 2-level nested*

Description

Single engine behind the package. `n_levels = 1` reproduces the legacy single-level model exactly (market price reverts to a constant level with a linear forcing by the production price and a TMG interaction); `n_levels = 2` fits the nested model in which the market price φ reverts to a *latent* production price Φ (Level 1), and Φ has its own Ornstein-Uhlenbeck dynamics reverting to a G'-driven mean (Level 2). The constructed production-price index enters as a noisy measurement (anchor) of the latent Φ.

Usage

```
fit_ou_nested(
  Y,
  X,
  TMG,
  COM,
  CAPITAL_TOTAL,
  n_levels = 1L,
```

```
  Gprime = NULL,
  k_cost = NULL,
  V_value = NULL,
  level_spec = NULL,
  theta_separation = c("soft", "hard"),
  k_uncertainty = c("meas", "recon"),
  sigma_phi_meas_fixed = NULL,
  kappa_cap = 2,
  results_robust = list(),
  priors = list(),
  com_in_mean = TRUE,
  train_frac = 0.7,
  fit_window = c("train", "full"),
  chains = 6,
  iter = 12000,
  warmup = 6000,
  thin = 1,
  cores = max(1, parallel::detectCores() - 1),
  threads_per_chain = 2,
  hard_sum_zero = TRUE,
  orthogonalize_tmg = TRUE,
  factor_from = c("X", "Y"),
  use_train_loadings = TRUE,
  adapt_delta = 0.97,
  max_treedepth = 12,
  seed = 1234,
  init = NULL,
  moment_match = NULL,
  verbose = FALSE
)
```

Arguments

`Y`	Numeric matrix (T x S). Market prices/values φ by sector.
`X`	Numeric matrix (T x S). Production prices Φ. In single-level mode this is a linear forcing covariate; in 2-level mode it is the constructed production-price index that anchors the latent Φ.
`TMG`	Numeric vector (length T). Aggregate profit-rate series used as the TMG interaction (modulates the gravitation speed).
`COM`	Numeric matrix (T x S). Composition of capital by sector.
`CAPITAL_TOTAL`	Numeric matrix (T x S). Total capital by sector.
`n_levels`	Integer, 1, 2 or 3. Model depth. `1` single-level (legacy); `2` nested (market φ reverts to latent production Φ, which reverts to a G'-driven mean); `3` adds the value level (D-IMPL-10): the latent production price reverts toward a mean that also tracks the value anchor V (direct prices $c + v + p$), so prices of production gravitate around values (Marx, Capital III ch. IX). `n_levels = 3` requires `V_value` and `Gprime`; it inherits all of the 2-level machinery and adds the value coupling m_v. Default 1.

Gprime — Numeric vector (length T) or NULL. Aggregate profit rate $G'_t = \sum EBO_t / \sum K_t$ that drives the Level-2 mean. Required when n_levels = 2; ignored otherwise. In k_uncertainty = "recon" the same (raw) series also enters the price identity $\Phi = k + KG'$.

k_cost — Numeric matrix (T x S) or NULL. Cost price $k = c + v$ by sector (raw units). Required only when k_uncertainty = "recon", where the latent production-price anchor is reconstructed as $\Phi = k + KG'$ with the total capital advanced K (taken from CAPITAL_TOTAL) treated as lognormally uncertain.

V_value — Numeric matrix (T x S) or NULL. Value anchor (direct prices $V = c + v + p =$ DirectPrices_Index, constructed directly from k_cost + EBO, indexed). Required only when n_levels = 3, where the latent production price reverts toward a mean that tracks V with the estimated coupling m_v (Capital III ch. IX). V is standardized on the training window and enters as a *datum* (direct empirical construction), never solved simultaneously (no Leontief inverse).

level_spec — List or NULL. Per-level richness toggles, each a list level1/level2 of four logical switches: cubic (cubic drift), sv (stochastic volatility), student_t (Student-t tails) and hierarchy (cross-sector partial pooling). NULL selects the canonical specification (Level 1 full, Level 2 lean). The switches act only in the 2-level mode; in single-level mode Level 1 is always full (pass NULL). Use ou_level_spec for the named experiment configurations ("canonical", "both_full", "both_lean", "n1_lean").

theta_separation — Character. Time-scale separation $\kappa^m > \kappa^p$ between market and production reversion: "soft" (a prior that favours it, falsifiable) or "hard" (imposed by construction). Only used when n_levels = 2. Default "soft".

k_uncertainty — Character. Treatment of the methodological uncertainty about the capital K (eq. 11), two contrastable modes. "meas" feeds the constructed production-price index as a noisy measurement of the latent Φ with scale σ_Φ (first-order measurement error). "recon" reconstructs the anchor as $\Phi = k + KG'$ with K lognormally uncertain (prior f_Y around the point estimate CAPITAL_TOTAL, scale priors$sigma_K_recon); this propagates the uncertainty about K into the latent production price. The reconstruction is standardized so that the $\sigma_K \to 0$ limit reproduces the "meas" anchor exactly (recon nests meas). "recon" requires k_cost and n_levels = 2. Default "meas".

sigma_phi_meas_fixed — Numeric > 0 or NULL. Latency dial for the anchor measurement SD σ_Φ (2-level only; decision D-IMPL-9.4). When NULL (default) the SD is *estimated* as a parameter with the half-normal prior priors$sigma_phi_meas_sd (the K-stochastic / recon regime, looser prior). When a positive number it is held *fixed* to that value as a datum and no parameter is sampled: this is the $\sigma_\Phi \to 0$ limit done well for the K-deterministic regime ($\Phi \approx$ the constructed index), and it removes the boundary funnel (G6) the estimated SD fell into in the deterministic limit, where the anchor-minus-Φ residuals vanish and the posterior of σ_Φ piles up at 0 (Session-9 CP smoke: rhat 2.208 on sigma_phi_meas). A small value such as 0.05 reproduces the tight-latency K-deterministic anchor. Ignored when n_levels = 1.

kappa_cap — Numeric > 0. Stability cap for the 2-level reversion speeds: both the market speed κ^m and the production speed κ^p are constrained to $(0, \texttt{kappa\_cap})$ through a logit link (with the TMG modulation inside the link). The Euler-discretized OU is stable for $\kappa < 2$ and monotone (no overshoot) for $\kappa < 1$; the default 2 admits the full stable region, oscillatory convergence included. Only used when `n_levels = 2`.

results_robust — List. Previous results object to extend (or empty list).

priors — List. Prior overrides (partial; missing entries use robust defaults). In addition to the single-level names (`sigma_delta`, `beta1_mean`/`beta1_sd`, `nu_shape`/`nu_rate`, `rho_mean`/`rho_sd`) the 2-level mode reads `sigma_phi_meas_sd` (half-normal scale of the anchor measurement SD, standardized units; default 0.5) and, in `"recon"` mode, `sigma_K_recon` (lognormal scale of the methodological uncertainty about K, in log units; default 0.10, i.e. roughly a 10% multiplicative uncertainty on the total capital advanced).

com_in_mean — Logical. Include COM effect in the mean. Default TRUE.

train_frac — Numeric in (0,1). Training-window fraction. Default 0.70.

fit_window — Character. `"train"` fits the likelihood on the training window only (honest forecasting); `"full"` uses all observations. Default `"train"`.

chains, iter, warmup, thin — Integer MCMC controls. Defaults 6 / 12000 / 6000 / 1.

cores — Integer. Cores for parallel chains.

threads_per_chain — Integer. Within-chain threads (capped so `parallel_chains * threads <= cores`).

hard_sum_zero — Logical. Fix the TMG wedge at zero. Default TRUE.

orthogonalize_tmg — Logical. Orthogonalize TMG w.r.t. the common factor. Default TRUE.

factor_from — Character. Source for the common factor: `"X"` or `"Y"`. Default `"X"`.

use_train_loadings — Logical. Compute factor loadings from training only. Default TRUE.

adapt_delta — Numeric in (0,1). NUTS target acceptance. Default 0.97.

max_treedepth — Integer. NUTS maximum tree depth. Default 12.

seed — Integer. Random seed.

init — Numeric, list or function. Initial values (passed to the backend).

moment_match — Logical or NULL. Accepted for API parity; not applied to the array-based PSIS-LOO (see `fit_ou_nonlinear_tmg`).

verbose — Logical. Print progress. Default FALSE.

Details

`fit_ou_nonlinear_tmg` is a thin backward-compatible wrapper around this engine with `n_levels = 1`; this function is the single source of truth.

Value

A list. For `n_levels = 1` the structure matches the legacy single-level fit, with the inner `factor_ou` carrying the historical S3 class `"ou_nonlinear_tmg"`. For `n_levels = 2` the *top-level* object carries the S3 class `"ou_nested_2level"` (so `print`, `summary` and `plot` dispatch on it) and adds `factor_ou$level2` (production reversion speed `kappa_p`, market speed at $zTMG = 0$ `kappa_m_base`, mean intercept `mu_const`, slope `m1`, innovation scale `sigma_p`, measurement scale `sigma_phi_meas`, and the Level-2 richness quantities that are present only when their switch is on: the cubic restoring coefficient `a3_p` (`l2_cubic`), the Student-t degrees of freedom `nu_p` (`l2_studentt`) and the time-varying SV scale path `sigma_p_t` = $\exp(h_p/2)$ (`l2_sv`); see decision D-IMPL-10); `phi_latent` (the latent Φ trajectory with credible bands); `mu_path` (the G'-driven Level-2 mean trajectory $\mu_{s,t} = m_{0,s} + m_1 G'_t$ with bands); a `separation` block with the posterior evidence for $\kappa^m > \kappa^p$; and `diagnostics$oos`, the 2-level out-of-sample metrics in which the market reverts to the latent Φ propagated forward (see `evaluate_oos_nested`).

See Also

`fit_ou_nonlinear_tmg` for the single-level wrapper and `fit_ou_nested_mi` for the multiple-imputation driver that couples disaggregation draws to the 2-level model via Rubin's rule.

Examples

```
T_obs <- 24; S <- 2
Y <- matrix(rnorm(T_obs * S), T_obs, S)
X <- matrix(rnorm(T_obs * S), T_obs, S)
TMG <- rnorm(T_obs)
COM <- matrix(runif(T_obs * S), T_obs, S)
K <- matrix(runif(T_obs * S, 100, 1000), T_obs, S)
Gp <- rnorm(T_obs)
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  try(fit_ou_nested(Y, X, TMG, COM, K, n_levels = 2, Gprime = Gp,
                    chains = 1, iter = 100, warmup = 50), silent = TRUE)
}
```

`fit_ou_nested_mi` *Fit the nested OU model under multiple imputation of the market price*

Description

Couples the disaggregation posterior to the OU model by multiple imputation (Rubin's rule). The disaggregation step produces M posterior draws of the sector-level market price φ (each a T x S matrix); this driver refits the OU model once per imputation and combines the analyses, propagating the disaggregation uncertainty into the OU posterior. Two complementary summaries are returned: Rubin's moment-pooled estimates (within + between imputation variance) and the mixture posterior over the imputations (the "complete-draws" pooling used for joint probabilities such as the time-scale separation).

Usage

```
fit_ou_nested_mi(
  phi_draws,
  X,
  TMG,
  COM,
  CAPITAL_TOTAL,
  Gprime = NULL,
  V_value = NULL,
  n_levels = 2L,
  M = 25,
  pool_params = NULL,
  keep_fits = FALSE,
  keep_kappa_mixture = FALSE,
  kappa_mixture_draws = 4000L,
  seed = 1234,
  verbose = FALSE,
  checkpoint_dir = NULL,
  ...
)
```

Arguments

phi_draws	Imputations of the market price. Either a list of D numeric matrices (each T x S) or a 3-D array [T, S, D]. These are posterior draws of φ from the disaggregation model.
X	Production-price index Φ (the Level-2 anchor). Either a fixed numeric matrix (T x S) – used verbatim for every imputation, the legacy behaviour – or a per-imputation set (a list of D matrices or a 3-D array [T, S, D]) paired one-to-one with phi_draws: draw m of X is used with draw m of phi_draws.
TMG	Numeric vector (length T). Aggregate profit-rate series. Fixed across imputations (observed aggregate).
COM	Composition of capital by sector. A fixed matrix (T x S) or a per-imputation list/array [T, S, D] paired with phi_draws (see X).
CAPITAL_TOTAL	Total capital advanced K by sector. A fixed matrix (T x S) or a per-imputation list/array [T, S, D] paired with phi_draws (see X).
Gprime	Numeric vector (length T) or NULL. Aggregate profit rate driving the Level-2 mean; required when n_levels >= 2. Fixed across imputations (observed aggregate, preserved by construction).
V_value	Value anchor (direct prices $c + v + p$ = DirectPrices_Index) for the Level-3 model; required when n_levels = 3, NULL otherwise. A fixed matrix (T x S) or a per-imputation list/array [T, S, D] paired with phi_draws (see X); passed to fit_ou_nested, which standardizes it on the training window.
n_levels	Integer, 1, 2 or 3. Passed to fit_ou_nested.
M	Integer. Number of imputations to use. Default 25. If fewer draws are supplied, all are used (with a warning); if more, M are taken on an evenly spaced grid.

`pool_params` Character vector or NULL. Parameters to Rubin-pool. NULL selects a sensible set for the chosen `n_levels`.

`keep_fits` Logical. Keep the per-imputation fit objects (memory heavy). Default FALSE (only compact per-imputation summaries are retained).

`keep_kappa_mixture` Logical. When TRUE (and `n_levels >= 2`), retain in `$kappa_mixture` a thinned sample of the mixture posterior of the per-sector reversion speeds `kappa_m_base` and `kappa_p` (stacked across imputations), for plotting the sectoral distribution of half-lives and computing low-kappa-trap probabilities at a chosen horizon. Default FALSE.

`kappa_mixture_draws` Integer. Target number of thinned mixture draws to retain when `keep_kappa_mixture = TRUE`. Default 4000.

`seed` Integer. Base seed; imputation `m` uses `seed + m`.

`verbose` Logical. Print progress. Default FALSE.

`checkpoint_dir` Character path or NULL. When NULL (default) the function behaves exactly as before (no files written). When a directory is given, each imputation's compact pooling contribution is persisted to `checkpoint_dir/imp_<j>.rds` as soon as it is computed, and a later call with the same directory reloads the completed imputations and skips their (expensive) Stan refit – resuming a run interrupted by a crash or shutdown. Only the pooling contributions are stored, never the full fits, so it is incompatible with `keep_fits = TRUE`. A stored checkpoint is honoured only when its draw index still matches the current imputation grid, so changing `M` (or the number of available draws) invalidates stale checkpoints automatically. Because the post-loop Rubin pooling is untouched and each per-imputation fit is deterministic given `seed + m`, a resumed run yields an object identical to an uninterrupted one.

`...` Passed to `fit_ou_nested` (e.g. `chains`, `iter`, `warmup`, `theta_separation`, `kappa_cap`).

Details

Unified external multiple imputation (D-IMPL-13.5). Beyond the market price, `X` (the production-price anchor Φ), `V_value` (the value anchor), `CAPITAL_TOTAL` (K) and `COM` may also vary by imputation. Each of these four accepts either a fixed matrix (the legacy behaviour: used verbatim for every imputation, bit-identical results) or a per-imputation set – a list of D matrices or a 3-D array `[T, S, D]` – paired one-to-one with `phi_draws`. This lets a single external imputation generator propagate ALL the construction uncertainty (disaggregation of the market, the imputed split of v and p that feeds K and V, and the reconstructed Φ) through the convergent K-DET / Variant-A engine, pooling by Rubin's rule. The aggregates `TMG` and `Gprime` stay fixed (they are observed, preserved by the aggregate-preserving renormalization of the generator). The engine is untouched (R-side only): $n \in \{1, 2\}$ and the all-fixed path remain bit-identical.

Value

An object of class `"ou_nested_mi"`: a list with

rubin Data frame of Rubin-pooled estimates per parameter (estimate, total SD, 95% CI, Barnard-Rubin df, fraction of missing information).

phi_latent_pooled (`n_levels >= 2`) Rubin-pooled latent Φ path: `mean`, `sd`, `q2.5`, `q97.5` as T x S matrices.

separation_pooled (`n_levels >= 2`) mixture-posterior evidence for $\kappa^m > \kappa^p$: `prob_sep_by_sector` (per-sector $P(\kappa^m > \kappa^p)$, the sectoral distribution to report – at `n_levels = 3` the joint probability typically collapses because κ^p is re-identified once the value term absorbs the production mean, see D-IMPL-10.1, which is NOT "no separation"), `prob_sep_joint`, and the per-sector reversion summaries `kappa_m_median` / `kappa_p_median` and half-lives $(\ln 2)/\kappa$ (`halflife_*_median`, `halflife_*_q2.5`, `halflife_*_q97.5`) from the mixture posterior. Since κ is positive by construction, $P(\kappa > 0) = 1$ is uninformative; the half-life (and whether it exceeds the observed span – the low-kappa trap) is the substantive per-sector quantity.

kappa_mixture (only when `keep_kappa_mixture = TRUE`) a thinned mixture posterior of `kappa_m_base` and `kappa_p` (draws x S), stacked across imputations, for sectoral half-life distributions and low-kappa-trap probabilities at a chosen horizon.

per_imputation List of compact per-imputation summaries or full fits if `keep_fits = TRUE`. Each summary carries the chain-aware convergence over the pooled parameters (`rhat_max_pooled`, `ess_bulk_min_pooled`, `ess_tail_min_pooled`) – the binding diagnostic for Rubin's pooling – plus the global `rhat_max`, `rhat_share`, `ess_bulk_min` (over all parameters, latent states included; reference only) and `divergences`.

config The imputation/pooling configuration.

References

Rubin DB (1987) Multiple Imputation for Nonresponse in Surveys. Barnard J, Rubin DB (1999) Small-sample degrees of freedom with multiple imputation. Biometrika 86(4):948-955.

See Also

`fit_ou_nested`.

`fit_ou_nonlinear_tmg` *Fit Bayesian nonlinear OU model with TMG effect and SV (single-level)*

Description

Backward-compatible single-level entry point. This is a thin wrapper around `fit_ou_nested` with `n_levels = 1`: the nested engine is the single source of truth, and `n_levels = 1` reproduces the legacy single-level model exactly (market price reverts to a constant level with a linear forcing by the production price `X` and a TMG interaction). The returned object keeps the historical structure and S3 class `"ou_nonlinear_tmg"` so existing extractors, plots and validators work unchanged.

Usage

```
fit_ou_nonlinear_tmg(
  results_robust,
  Y,
  X,
  TMG,
  COM,
  CAPITAL_TOTAL,
  model = c("base"),
  priors = list(),
  com_in_mean = TRUE,
  train_frac = 0.7,
  fit_window = c("train", "full"),
  chains = 6,
  iter = 12000,
  warmup = 6000,
  thin = 1,
  cores = max(1, parallel::detectCores() - 1),
  threads_per_chain = 2,
  hard_sum_zero = TRUE,
  orthogonalize_tmg = TRUE,
  factor_from = c("X", "Y"),
  use_train_loadings = TRUE,
  adapt_delta = 0.97,
  max_treedepth = 12,
  seed = 1234,
  init = NULL,
  moment_match = NULL,
  verbose = FALSE
)
```

Arguments

`results_robust`	List. Previous results object to extend (can be empty list).
`Y`	Numeric matrix (T x S). Dependent variable (prices/values by sector).
`X`	Numeric matrix (T x S). Independent variable (production prices).
`TMG`	Numeric vector (length T). Aggregate TMG series.
`COM`	Numeric matrix (T x S). Composition of capital by sector.
`CAPITAL_TOTAL`	Numeric matrix (T x S). Total capital by sector.
`model`	Character. Model type. Currently only "base" supported.
`priors`	List. Prior specifications (partial override allowed; missing entries fall back to robust defaults). Supported names: `sigma_delta` (wedge SD, original units, default 0.002); `beta1_mean`/`beta1_sd` (prior for the global TMG effect, default 0 / 0.5 – neutral, no sign baked in); `nu_shape`/`nu_rate` (gamma prior for the Student-t degrees of freedom shift, default 2 / 0.1 -> weakly informative, prior mean nu ~ 22); `rho_mean`/`rho_sd` (SV persistence prior, default 0.7 / 0.2).

com_in_mean	Logical. Include COM effect in mean equation. Default TRUE.
train_frac	Numeric in (0,1). Fraction of observations used for the training window. Default 0.70.
fit_window	Character. "train" (default) fits the likelihood and computes log_lik ONLY on the training window (2:T_train), so the test window is genuinely held out and evaluate_oos is a real out-of-sample evaluation. "full" fits on all observations (full-information / rstanarm-style); then PSIS-LOO is valid over all points but evaluate_oos becomes in-sample. See the README methodology note.
chains	Integer. Number of MCMC chains. Default 6.
iter	Integer. Total iterations per chain (must exceed warmup). Default 12000.
warmup	Integer. Warmup iterations. Default 6000.
thin	Integer. Thinning interval. Default 1 (thinning is statistically wasteful in HMC; increase only to save memory).
cores	Integer. Number of cores for parallel chains.
threads_per_chain	Integer. Threads per chain for within-chain parallelism. Capped internally so parallel_chains * threads <= cores.
hard_sum_zero	Logical. If TRUE, TMG wedge is fixed at zero. Default TRUE.
orthogonalize_tmg	Logical. Orthogonalize TMG w.r.t. common factor. Default TRUE.
factor_from	Character. Source for common factor: "X" or "Y". Default "X".
use_train_loadings	Logical. Compute factor loadings from training only (avoids look-ahead leakage). Default TRUE.
adapt_delta	Numeric. Target acceptance rate (0-1). Default 0.97.
max_treedepth	Integer. Maximum tree depth for NUTS. Default 12.
seed	Integer. Random seed for reproducibility.
init	Numeric or function. Initial values for parameters.
moment_match	Logical. Use moment matching for LOO. Default NULL.
verbose	Logical. Print progress messages. Default FALSE.

Details

The model uses a non-centered hierarchical (partial-pooling) parameterization for the sector-specific parameters theta_s, kappa_s, a3_s and beta0_s: each is built as hyper_intercept + hyper_slope * COM_s + hyper_sd * z, with z ~ N(0,1). The training window is floor(T * train_frac). All data standardization, the COM weighting, and (by default) the common factor loadings are computed from the training window only. The likelihood window is controlled by fit_window to keep train/test coherent.

The cubic drift enforces a3 < 0 (a restoring force that strengthens with the deviation); this is a stability *assumption*, not an estimated result, and it precludes detecting locally expansive (self-amplifying) regimes. The Euler discretization uses dt = 1, so the discrete persistence of the linear part is 1 - kappa; interpret half-lives accordingly.

Value

List containing `factor_ou` (model results and parameter estimates), `beta_tmg`, `sv`, `nonlinear`, `accounting` and `diagnostics` (MCMC diagnostics, PSIS-LOO and OOS metrics). See `fit_ou_nested` for the underlying engine and the 2-level model.

Examples

```
# 1. Prepare dummy data
T_obs <- 20
S_sectors <- 2
Y <- matrix(rnorm(T_obs * S_sectors), nrow = T_obs, ncol = S_sectors)
X <- matrix(rnorm(T_obs * S_sectors), nrow = T_obs, ncol = S_sectors)
TMG <- rnorm(T_obs)
COM <- matrix(runif(T_obs * S_sectors), nrow = T_obs, ncol = S_sectors)
K <- matrix(runif(T_obs * S_sectors, 100, 1000), nrow = T_obs, ncol = S_sectors)

# 2. Run model (conditional on Stan backend availability)
if (requireNamespace("cmdstanr", quietly = TRUE) ||
    requireNamespace("rstan", quietly = TRUE)) {
  try({
    results <- fit_ou_nonlinear_tmg(
      results_robust = list(),
      Y = Y, X = X, TMG = TMG, COM = COM, CAPITAL_TOTAL = K,
      chains = 1, iter = 100, warmup = 50,
      verbose = FALSE
    )
  }, silent = TRUE)
}
```

---

kappa_stability_evidence

Mean-reversion (dynamic stability) evidence for kappa parameters

---

Description

Preferred alias of `extract_convergence_evidence`. See that function for details. The name avoids conflating dynamic mean reversion with MCMC sampler convergence.

Usage

```
kappa_stability_evidence(fit_res, verbose = TRUE)
```

Arguments

`fit_res`	List returned by `fit_ou_nonlinear_tmg`
`verbose`	Logical. Print summary to console. Default TRUE.

Value

See extract_convergence_evidence.

---

ou_geom_bridge *Bridge a methods-enabled cmdstan model to the geometry engine*

Description

Decoupled core (the improvement over the gdpar bridge, which required a fitted object of a specific class and recompiled from its source): takes a CmdStanModel ALREADY compiled with compile_model_methods = TRUE together with its Stan data, samples one iteration to expose the standalone log_prob/grad_log_prob/hessian methods, reads the unconstrained dimension and a posterior-mean warm-start, and returns an ou_geom_target plus a reference position. Nothing in the regular fit path is touched.

Usage

```
ou_geom_bridge(
  model,
  stan_data,
  dim = NULL,
  reference = NULL,
  hessian = TRUE,
  methods_seed = 1L
)
```

Arguments

model	A CmdStanModel compiled with compile_model_methods = TRUE.
stan_data	The Stan data list.
dim	Optional unconstrained dimension (derived from the one-iteration fit when NULL).
reference	Optional unconstrained warm-start (posterior mean of the one-iteration fit when NULL).
hessian	Logical; request the standalone Hessian (best-effort: falls back to gradient-only when higher-order autodiff does not compile).
methods_seed	Integer seed forwarded to init_model_methods().

Value

A list with target (an ou_geom_target), dim, reference, fit (the one-iteration cmdstan fit with methods) and has_hessian.

ou_geom_hmc *Static Hamiltonian Monte Carlo with a pluggable geometry*

Description

Fixed step-size / fixed trajectory-length HMC with a Metropolis correction over a pluggable metric. The default Euclidean metric is textbook HMC; a Riemannian metric reuses the same loop with the generalised implicit leapfrog of Girolami & Calderhead (2011). The metric is a preconditioner only, so the returned draws are exact regardless of which metric is chosen.

Usage

```
ou_geom_hmc(
  target,
  metric = NULL,
  epsilon = 0.1,
  L = 20L,
  n_iter = 1000L,
  n_warmup = 500L,
  init = NULL,
  seed = NULL,
  fp_tol = 1e-09,
  fp_max = 100L
)
```

Arguments

target	A ou_geom_target (or object accepted by it).
metric	A ou_geom_metric_euclidean / ou_geom_metric_riemannian. Defaults to the identity metric.
epsilon	Leapfrog step size.
L	Leapfrog steps per proposal.
n_iter	Retained iterations.
n_warmup	Burn-in iterations discarded (no adaptation).
init	Optional initial position (defaults to zeros).
seed	Optional integer seed (RNG state set and restored).
fp_tol, fp_max	Fixed-point tolerance / max iterations for the implicit leapfrog (position-dependent metric only).

Value

A list of class ou_geom_hmc with draws, accept_rate, n_divergent, energy, ebfmi, epsilon, L and metric_type.

ou_geom_metric_euclidean

Euclidean (constant) metric for the geometry engine

Description

A position-independent mass matrix: the identity (diagonal) by default, or a supplied symmetric positive-definite matrix / positive variance vector (dense preconditioner; the remedy for a straight anisotropic canyon).

Usage

```
ou_geom_metric_euclidean(dim = NULL, M = NULL)
```

Arguments

`dim`	Integer dimension (required when `M` is `NULL`).
`M`	Optional `dim` x `dim` SPD matrix or length-`dim` positive diagonal vector.

Value

A `ou_geom_metric` (position-independent).

ou_geom_metric_riemannian

Riemannian (position-dependent) metric for the geometry engine

Description

A position-dependent mass $M(\theta)$ adapting the sampler's local distance to the curvature of the log-posterior – the remedy for a funnel (variable curvature). `curvature = "softabs"` maps the eigenvalues of the Hessian of $-\log\pi$ through $\lambda\coth(\alpha\lambda)$ (Betancourt 2013), flooring nearly flat directions; it needs only the Hessian (from the target or finite-differenced). `curvature = "fisher"` uses a supplied expected-Fisher function (positive-definite by construction).

Usage

```
ou_geom_metric_riemannian(
  target,
  curvature = c("softabs", "fisher"),
  fisher = NULL,
  dfisher = NULL,
  alpha = 1e+06,
  floor = 1e-08,
  fd_step = 1e-04
)
```

Arguments

target	A ou_geom_target.
curvature	"softabs" (default; observed Hessian) or "fisher" (supplied expected Fisher).
fisher, dfisher	For "fisher": fisher(theta) returning the expected Fisher matrix and optionally its derivative list.
alpha	SoftAbs softening (larger tracks curvature more faithfully).
floor	Minimum eigenvalue keeping $M(\theta)$ positive-definite.
fd_step	Finite-difference step for the Hessian / metric derivative.

Value

A ou_geom_metric (position-dependent).

---

ou_geom_target *Build a geometry-engine sampling target*

Description

Wrap an unconstrained log-density (and its gradient, optionally its Hessian) into the target object ou_geom_hmc integrates. Accepts either a cmdstan fit/model compiled with compile_model_methods = TRUE (whose $log_prob / $grad_log_prob / $hessian act on the unconstrained scale) or explicit R closures.

Usage

```
ou_geom_target(
  object = NULL,
  log_prob = NULL,
  grad_log_prob = NULL,
  dim = NULL,
  hessian = NULL,
  param_names = NULL,
  data = NULL
)
```

Arguments

object	Optional cmdstan fit/model (with methods) or a list carrying log_prob/grad_log_prob/dim.
log_prob, grad_log_prob	R closures of the unconstrained parameter vector (used when object is NULL).
dim	Integer unconstrained dimension (required for closures or a cmdstan model).
hessian	Optional Hessian closure (used by the SoftAbs Riemannian metric); finite-differenced from the gradient when absent.
param_names	Optional parameter names.
data	Stan data list (used when object is an uncompiled-to-fit CmdStanModel that must be sampled once to expose its methods).

Value

An object of class `ou_geom_target`.

`ou_level_spec` *Named per-level richness configurations for the comparative experiment*

Description

Convenience builder for the `level_spec` argument of `fit_ou_nested`. Each level has four richness switches: `cubic` (cubic drift), `sv` (stochastic volatility), `student_t` (Student-t innovations) and `hierarchy` (cross-sector partial pooling). The switches act only in the 2-level mode.

Usage

```
ou_level_spec(
  config = c("canonical", "n1_full_n2_lean", "both_full", "both_lean", "n1_lean",
    "n1_lean_n2_lean")
)
```

Arguments

`config`	Character. One of `"canonical"`, `"both_full"`, `"both_lean"`, `"n1_lean"` (and the explicit aliases above).

Details

The comparative experiment of the design contrasts these configurations by parameter recovery and out-of-sample / leave-cluster-out predictive performance, reporting which one the data support (no configuration is claimed uniquely correct):

`"canonical"` **(alias** `"n1_full_n2_lean"`**)** Level 1 full (cubic + SV + Student-t + hierarchy), Level 2 lean (linear Gaussian OU with a hierarchical mean). The default.

`"both_full"` Both levels full: the latent production price also gets cubic restoring, stochastic volatility and Student-t innovations.

`"both_lean"` Both levels lean: linear Gaussian OU at each level, hierarchy retained.

`"n1_lean"` **(alias** `"n1_lean_n2_lean"`**)** Level 1 lean, Level 2 lean.

The fifth experiment arm, the single-level model, is obtained directly with `fit_ou_nested(..., n_levels = 1)` (it takes no `level_spec`).

Value

A `level_spec` list with `level1` and `level2` entries, each a list of the four logical switches, ready to pass to `fit_ou_nested`.

See Also

fit_ou_nested.

Examples

```
ou_level_spec("both_full")
ou_level_spec("both_lean")
```

---

ou_nested_stan_code *Stan code for the unified nonlinear OU model*

Description

Returns the complete Stan code for the unified Ornstein-Uhlenbeck model: a single file (`inst/stan/ou_nested.stan`) covering the single-level mode (`n_levels = 1`, cubic drift, stochastic volatility, Student-t innovations, non-centered hierarchical priors and a TMG interaction) and the 2-level nested mode (`n_levels = 2`, market price reverting to a latent production price with its own G'-driven OU). The code is read from the canonical file (single source of truth); this function does not embed a duplicate copy.

Usage

```
ou_nested_stan_code()
```

Value

Character string containing the Stan model code.

Examples

```
code <- ou_nested_stan_code()
cat(substr(code, 1, 300))
```

---

plot.ou_nested_2level *Plot a 2-level nested OU fit*

Description

Plot a 2-level nested OU fit

Usage

```
## S3 method for class 'ou_nested_2level'
plot(x, type = c("phi", "mu", "sv_p", "separation"), sector = 1, ...)
```

Arguments

x	An ou_nested_2level object from fit_ou_nested.
type	Character. "phi" (latent production-price path with 95% band), "mu" (the G'-driven Level-2 mean trajectory with band), "sv_p" (the time-varying Level-2 stochastic-volatility scale $\sigma_p(t)$ with band; only available when the Level-2 SV switch is on) or "separation" (per-sector posterior probability of $\kappa^m > \kappa^p$). Default "phi".
sector	Integer. Sector index for "phi" / "mu" / "sv_p". Default 1.
...	Passed to the underlying base graphics call.

Value

NULL invisibly; draws as a side effect.

---

plot.ou_nested_mi *Plot a multiple-imputation nested OU fit*

Description

Plot a multiple-imputation nested OU fit

Usage

```
## S3 method for class 'ou_nested_mi'
plot(x, type = c("phi", "fmi"), sector = 1, ...)
```

Arguments

x	An ou_nested_mi object from fit_ou_nested_mi.
type	Character. "phi" (Rubin-pooled latent production price with 95% band) or "fmi" (fraction of missing information per pooled parameter). Default "phi".
sector	Integer. Sector index for "phi". Default 1.
...	Passed to the underlying base graphics call.

Value

NULL invisibly; draws as a side effect.

plot_beta_tmg *Plot beta(TMG_t) evolution by sector*

Description

Creates a line plot showing the evolution of time-varying beta coefficients for selected sectors.

Usage

```
plot_beta_tmg(results, sectors = NULL)
```

Arguments

results	List returned by fit_ou_nonlinear_tmg
sectors	Character or integer vector. Sectors to plot. If NULL, plots all sectors.

Value

A ggplot2 object if ggplot2 is available, otherwise NULL with a base R plot produced as side effect.

Examples

```
# 1. Create mock data (T x S matrix)
T_obs <- 50
S <- 3
beta_mat <- matrix(rnorm(T_obs * S), nrow = T_obs, ncol = S)
colnames(beta_mat) <- paste0("Sector_", 1:S)

# 2. Wrap in list structure expected by function
results_mock <- list(
  beta_tmg = list(
    beta_point = beta_mat
  )
)

# 3. Plot
plot_beta_tmg(results_mock)
```

plot_drift_curves *Plot cubic OU drift curves*

Description

Displays the drift function for selected sectors showing mean reversion with cubic nonlinearity.

Usage

```
plot_drift_curves(results, sectors = NULL)
```

Arguments

results	List returned by fit_ou_nonlinear_tmg
sectors	Integer vector. Sector indices to plot. If NULL, plots all sectors.

Value

NULL invisibly. Produces a base R plot as side effect.

Examples

```
# 1. Create mock data
# z: vector of state deviations
z_seq <- seq(-3, 3, length.out = 100)
# drift: matrix (rows=z, cols=sectors)
drift_mat <- cbind(
  -0.5 * z_seq - 0.1 * z_seq^3, # Sector 1
  -0.8 * z_seq - 0.05 * z_seq^3 # Sector 2
)

# 2. Wrap in list structure
results_mock <- list(
  nonlinear = list(
    drift_decomp = list(
      z = z_seq,
      drift = drift_mat
    )
  )
)

# 3. Plot
plot_drift_curves(results_mock)
```

`plot_sv_evolution` *Plot stochastic volatility evolution*

Description

Displays the estimated volatility path for a selected sector.

Usage

```
plot_sv_evolution(results, sector = 1)
```

Arguments

`results`	List returned by `fit_ou_nonlinear_tmg`
`sector`	Integer. Sector index to plot. Default 1.

Value

NULL invisibly. Produces a base R plot as side effect.

Examples

```
# 1. Create mock data (Volatility must be positive)
T_obs <- 50
sigma_mat <- matrix(exp(rnorm(T_obs * 2)), nrow = T_obs, ncol = 2)

# 2. Wrap in list structure
results_mock <- list(
  sv = list(
    h_summary = list(
      sigma_t = sigma_mat
    )
  )
)

# 3. Plot sector 1
plot_sv_evolution(results_mock, sector = 1)
```

print.ou_nested_2level *Print a 2-level nested OU fit*

Description

Compact one-screen overview: data dimensions, MCMC health, the time-scale separation evidence $\kappa^m > \kappa^p$, and the median ranges of the market and production reversion speeds.

Usage

```
## S3 method for class 'ou_nested_2level'
print(x, ...)
```

Arguments

x	An ou_nested_2level object from fit_ou_nested.
...	Ignored.

Value

x, invisibly.

print.ou_nested_mi *Print a multiple-imputation nested OU fit*

Description

Print a multiple-imputation nested OU fit

Usage

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

Arguments

x	An ou_nested_mi object from fit_ou_nested_mi.
...	Ignored.

Value

x, invisibly.

summarize_sv_sigmas *Summarize stochastic volatility sigmas*

Description

Extracts median volatility paths from the SV component.

Usage

```
summarize_sv_sigmas(fit)
```

Arguments

fit	Fitted Stan model object

Value

List with component:

sigma_t Matrix (T x S) of median volatilities

summary.ou_nested_2level *Summarize a 2-level nested OU fit*

Description

Assembles the Level-2 parameter table (production reversion speed, market speed at $zTMG = 0$, mean intercept, innovation scale, measurement scale), the separation evidence, the MCMC diagnostics and the out-of-sample metrics.

Usage

```
## S3 method for class 'ou_nested_2level'
summary(object, ...)

## S3 method for class 'summary.ou_nested_2level'
print(x, ...)
```

Arguments

object	An ou_nested_2level object from fit_ou_nested.
...	Ignored.
x	A summary.ou_nested_2level object.

Value

An object of class summary.ou_nested_2level.

summary.ou_nested_mi *Summarize a multiple-imputation nested OU fit*

Description

Summarize a multiple-imputation nested OU fit

Usage

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

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

Arguments

object	An ou_nested_mi object from fit_ou_nested_mi.
...	Ignored.
x	A summary.ou_nested_mi object.

Value

An object of class summary.ou_nested_mi.

validate_ou_fit *Validate OU model fit*

Description

Prints diagnostic summaries and hypothesis tests for a fitted model.

Usage

```
validate_ou_fit(fit_res, verbose = TRUE)
```

Arguments

fit_res	List returned by fit_ou_nonlinear_tmg
verbose	Logical. Print detailed output. Default TRUE.

Value

Invisibly returns the diagnostics list

Examples

```
# Create a dummy results list that mimics the output of fit_ou_nonlinear_tmg
dummy_results <- list(
  diagnostics = list(
    rhat = c(alpha = 1.01, beta = 1.00),
    ess = c(alpha = 400, beta = 350),
    loo = list(estimates = matrix(c(1, 0.1), ncol=2,
               dimnames=list("elpd_loo", c("Estimate", "SE")))),
    oos = list(h1 = list(RMSE = 0.5))
  ),
  factor_ou = list(beta1 = 0.3),
  nonlinear = list(a3 = -0.5),
  sv = list(rho_s = 0.2)
)

# Run validation on the dummy object
validate_ou_fit(dummy_results)
```

---

zscore_train *Z-score standardization using training period statistics*

Description

Standardizes a matrix using mean and standard deviation computed from the training period only.

Usage

```
zscore_train(M, T_train, eps = 1e-08)
```

Arguments

M	Numeric matrix of dimensions T x S (time by sectors)
T_train	Integer. Number of observations in training period
eps	Numeric. Minimum standard deviation threshold to avoid division by zero. Default is 1e-8.

Value

A list with components:

Mz Standardized matrix of same dimensions as M

mu Vector of training means for each column

sd Vector of training standard deviations for each column

Examples

```
M <- matrix(rnorm(100), nrow = 20, ncol = 5)
result <- zscore_train(M, T_train = 14)
str(result)
```

Index