# Package 'bayesianOU'

December 19, 2025

**Type** Package

**Title** Bayesian Nonlinear Ornstein-Uhlenbeck Models with Stochastic Volatility

**Version** 0.1.3

**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

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

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

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**URL** https://github.com/isadorenabi/bayesianOU

**BugReports** https://github.com/isadorenabi/bayesianOU/issues

**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 20:10:22 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)
```

## Arguments

| | |
|---|---|
| fit | Fitted Stan model object |
| zTMG_use | Numeric vector. Standardized TMG series used in fitting. |

## 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 z values.

### 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 z values to evaluate drift. |

### 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 |

## Value

Named list with one element per horizon...

## 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)
```

`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` *Extract convergence evidence for kappa parameters*

---

## Description

Computes 95 percent credible intervals for each kappa_s (mean reversion speed) and verifies formal convergence conditions.

## 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

**convergence** Logical indicating if all kappa in (0,1)

**prob_convergence** Posterior probability of joint convergence

## 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_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_nonlinear_tmg` *Fit Bayesian nonlinear OU model with TMG effect and SV*

---

## Description

Fits a Bayesian nonlinear Ornstein-Uhlenbeck model with cubic drift, stochastic volatility, and Student-t innovations using Stan.

## Usage

```
fit_ou_nonlinear_tmg(
  results_robust,
  Y,
  X,
  TMG,
  COM,
  CAPITAL_TOTAL,
  model = c("base"),
  priors = list(sigma_delta = 0.002),
  com_in_mean = TRUE,
  chains = 6,
  iter = 12000,
  warmup = 6000,
  thin = 2,
  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 = FALSE,
  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. Currently supports sigma_delta. |
| `com_in_mean` | Logical. Include COM effect in mean equation. Default TRUE. |
| `chains` | Integer. Number of MCMC chains. Default 6. |
| `iter` | Integer. Total iterations per chain. Default 12000. |
| `warmup` | Integer. Warmup iterations. Default 6000. |
| `thin` | Integer. Thinning interval. Default 2. |

| | |
|---|---|
| cores | Integer. Number of cores for parallel chains. |
| threads_per_chain | Integer. Threads per chain for within-chain parallelism. |
| 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. Default FALSE. |
| 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 hierarchical priors for sector-specific parameters. Training period is set to 70 percent of observations by default. All data standardization uses training period statistics only.

## Value

List containing:

**factor_ou** Model results including draws and parameter estimates

**beta_tmg** Time-varying beta estimates

**sv** Stochastic volatility summaries

**nonlinear** Nonlinearity diagnostics

**accounting** TMG accounting block

**diagnostics** MCMC diagnostics, LOO, and OOS metrics

## 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)
# We use very short chains just to demonstrate execution
if (requireNamespace("cmdstanr", quietly = TRUE) ||
```

```
    requireNamespace("rstan", quietly = TRUE)) {

  # Wrap in try to avoid failure if Stan is not configured locally
  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, # Short run for example
      verbose = FALSE
    )
  }, silent = TRUE)
}
```

---

`ou_nonlinear_tmg_stan_code` *Stan code for nonlinear OU model with SV and Student-t*

---

## Description

Returns the complete Stan code for the nonlinear Ornstein-Uhlenbeck model with cubic drift, stochastic volatility, and Student-t innovations. Includes numerical guardrails and parallel computation support.

## Usage

```
ou_nonlinear_tmg_stan_code()
```

## Details

The model implements:

- Cubic drift: $\kappa(\theta - Y + a_3(Y - \theta)^3)$
- Stochastic volatility with AR(1) log-variance
- Student-t innovations with estimated degrees of freedom
- Hierarchical priors for sector-specific parameters
- Optional soft constraint on TMG variable
- Parallel likelihood computation via reduce_sum

## Value

Character string containing Stan model code

## Examples

```
code <- ou_nonlinear_tmg_stan_code()
cat(substr(code, 1, 500))
```

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)
```

## 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

## 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