Package: spDBL (via r-universe)

November 16, 2026

Title Dynamic Bayesian Learning for Spatiotemporal Mechanistic Models

Version 1.2.0

Description Emulates the output of spatiotemporal mechanistic simulators, a field on a spatial grid at each time step, with a matrix-variate dynamic linear model whose posterior is available in closed form, processing large grids as sequences of spatial blocks, and predicts the output at untried inputs. Calibrates simulator parameters to noisy field data without further simulator runs: a design Gaussian process supplies the emulator's autoregressive inputs, and 'Stan' (through the suggested package 'rstan') samples the parameters conditionally on emulator posterior draws, with the emulator field and a dynamic spatial discrepancy integrated out. Follows Banerjee, Chen, Frankenburg and Zhou (2025) <https://jmlr.org/papers/v26/22-0896.html>.

License MIT + file LICENSE

URL https://xiangchen-stat.github.io/spDBL/, https://github.com/xiangchen-stat/spDBL

BugReports https://github.com/xiangchen-stat/spDBL/issues

Encoding UTF-8

Depends R (>= 4.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppEigen

Imports Rcpp, matrixsampling, mniw, utils, tools, methods, graphics, stats, readr, rlang (>= 1.0.0)

Suggests deSolve, ReacTran, ggplot2, ggpubr, scales, testthat (>= 3.3.2), here, knitr, rmarkdown, loo, rstan (>= 2.32.7)

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

Config/pak/sysreqs libx11-dev

Repository https://xiangchen-stat.r-universe.dev

Date/Publication 2026-11-16 21:41:52 UTC

RemoteUrl https://github.com/xiangchen-stat/spdbl

RemoteRef HEAD

RemoteSha 73d425a9eca03dc9d13be5ebc9ab9bf024991d05

Contents

BS . . . 4
BS_bigdata_R . . . 5
cal_Bt_bt . . . 6
cal_errorbar . . . 7
cal_errorbar_mean . . . 8
cal_jacobian_logit_uniform . . . 8
calibration_fit . . . 9
calibration_predict . . . 12
calibration_prepare . . . 13
calibration_validate . . . 15
check_pds . . . 16
compute_analytical_waic . . . 17
compute_sample_waic . . . 18
dMTig . . . 20
dt_calibration . . . 21
dt_emulation . . . 22
emulator_learn . . . 22
emulator_predict . . . 24
expit . . . 25
FF . . . 26
FF_1step_R_I . . . 27
FF_1step_R_sigma2R . . . 28
FF_bigdata_R . . . 29
FF_I . . . 32
FF_sigma2R . . . 33
FFBS . . . 34
FFBS_I . . . 35
FFBS_predict_bigdata_R . . . 36
FFBS_predict_exact . . . 38
FFBS_predict_MC . . . 39
FFBS_sampling . . . 41
FFBS_sampling_I . . . 42
FFBS_sampling_sigma2R . . . 43
FFBS_sigma2R . . . 45
gen_calibrate_data . . . 46
gen_calibrate_data_uncorr . . . 47

gen_exp_kernel . . . 48
gen_expsq_kernel . . . 49
gen_F_ls_AR1 . . . 49
gen_F_ls_AR1_EP . . . 50
gen_F_ls_AR2 . . . 51
gen_F_ls_AR2_EP . . . 52
gen_ffbs_csv . . . 53
gen_ffbs_data . . . 54
gen_gp_kernel . . . 55
gen_Jt . . . 55
gen_pd_matrix . . . 56
gen_pde . . . 57
gen_prior_u_tau2 . . . 58
gen_ran_matrix . . . 59
generate.grid.exact . . . 60
generate.grid.lr . . . 60
generate.grid.rowsnake . . . 61
generate_grid . . . 62
inv_chol . . . 63
lppd_id_1t . . . 64
lppd_IG_1t . . . 65
lppd_IW_1t . . . 66
make_pds . . . 67
MNIG_sampler . . . 67
MNIW_R . . . 68
MNIW_R_naiive . . . 70
MNIW_sampler . . . 71
plot.spDBL_calibration . . . 72
plot_panel_heatmap_9 . . . 73
plot_panel_heatmap_9_cal . . . 74
plot_panel_heatmap_9_cal_nolab . . . 75
prepare_data . . . 77
quick_heat . . . 78
quick_save . . . 79
read_big_csv_quick . . . 80
recover_from_EP_exact . . . 81
recover_from_EP_MC . . . 82
rmn_chol . . . 83
rmn_chol_more . . . 84
sample_y_eta_one . . . 85
scale_back_uniform . . . 86
scale_uniform . . . 87
SIR . . . 88
spDBL_calibration_input-methods . . . 89
spDBL_emulator-methods . . . 90
summary.spDBL_calibration . . . 92
update_muSigma_eta_one . . . 93
update_y_eta . . . 95

update_y_eta_one 97

Index **100**

BS *Backward sampler for the Forward Filter Backward Sampler (FFBS)*

Description

Performs the backward sampling pass of the FFBS algorithm using the filtered distributions produced by FF. Iterates from time T back to 1, drawing smoothed state samples at each step via BS_1step_cpp.

Usage

```
BS(res_ff, G_ls, nT, delta = 1, verbose = FALSE)
```

Arguments

res_ff	List of length nT (plus a prior element) returned by FF. Each element res_ff[[t]] must contain mt, Mt, at, and At.
G_ls	Either a single numeric matrix (state transition matrix G, constant over time) or a list of nT matrices (time-varying).
nT	Integer. Number of time steps.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list with:

st Array of dimension c(p, q, nT) containing the smoothed state means.

St Array of dimension c(p, p, nT) containing the smoothed state left-covariance matrices.

See Also

FF, FFBS

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res_ff <- FF(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
             par$n0, par$D0, nT = 4)
res_bs <- BS(res_ff, par$G0, nT = 4)
dim(res_bs$st)
# At the last time the smoothed and filtered means agree
all.equal(res_bs$st[, , 4], res_ff$T4$mt)
```

`BS_bigdata_R` *Out-of-core backward smoothing*

Description

Runs the backward smoothing pass over the filtered moments that `FF_bigdata_R()` wrote to disk, one time step at a time, and writes the smoothed moments to CSV files. The result equals `BS()` applied to the in-memory filter over the same blocks, without holding all episodes in memory.

Usage

```
BS_bigdata_R(
  ff_files,
  G_ls,
  nT,
  fnrow,
  fncol,
  bnrow,
  bncol,
  path_out,
  delta = 1,
  verbose = FALSE
)
```

Arguments

`ff_files`	The list of file paths returned by `FF_bigdata_R()`.
`G_ls`	The state transition file path(s) given to `FF_bigdata_R()`: one path, or one per time step.
`nT`, `fnrow`, `fncol`, `bnrow`, `bncol`	As given to `FF_bigdata_R()`.
`path_out`	Directory where the output CSV files are written.
`delta`	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
`verbose`	Logical. If `TRUE`, reports progress.

Value

A list of file paths with one file per time step:

st_ls Smoothed state means; the $p \times C_b$ blocks are bound by column in traversal order, as in the filter's `mt` files.

St_ls Smoothed left covariances, $p \times p$ blocks bound by column.

See Also

`FF_bigdata_R()`, `FFBS_predict_bigdata_R()`, `BS()`

Examples

```
set.seed(1)
dir <- file.path(tempdir(), "bs_big")
dir.create(dir, showWarnings = FALSE)
csv <- function(x, name) {
  path <- file.path(dir, name)
  utils::write.csv(x, path, row.names = FALSE)
  path
}
# 5 runs x 4 sites over 3 times, in two blocks of 2 sites (AR(1) designs)
Y <- lapply(1:3, function(t) matrix(rnorm(20, t), 5, 4))
F <- c(list(matrix(1, 5, 4)), Y[1:2])
ff <- FF_bigdata_R(Y_ls = mapply(csv, Y, paste0("Y", 1:3, ".csv")),
  F_ls = mapply(csv, F, paste0("F", 1:3, ".csv")),
  G_ls = csv(diag(2), "G.csv"), W_ls = csv(diag(2), "W.csv"),
  V_ls = csv(diag(5), "V.csv"), m0 = matrix(0, 2, 2), M0 = diag(2),
  n0 = 4, D0 = diag(2), nT = 3, fnrow = 5, fncol = 4, bnrow = 5,
  bncol = 2, path_out = dir)
bs <- BS_bigdata_R(ff, file.path(dir, "G.csv"), nT = 3, fnrow = 5,
  fncol = 4, bnrow = 5, bncol = 2, path_out = dir)
utils::read.csv(bs$st_ls[1])  # smoothed means at time 1, both blocks
unlink(dir, recursive = TRUE)
```

---

cal_Bt_bt *Update the posterior mean and covariance of the discrepancy field*

Description

Computes the posterior mean vector $B_t b_t$ and covariance matrix B_t of the discrepancy u_t given the current predictive mean μ_t, observations $z_t - u_t$, and variance τ_t^2.

Usage

```
cal_Bt_bt(mut, invSigma, tau2t, zt_ut, I_Stilde)
```

Arguments

mut	Numeric vector. Predictive mean at calibration locations.
invSigma	Numeric matrix. Inverse of the predictive spatial covariance.
tau2t	Numeric scalar. Current variance τ_t^2.
zt_ut	Numeric vector. Observation residuals $z_t - u_t$.
I_Stilde	Numeric matrix. Identity-like matrix for the discrepancy prior precision (N_sp_train x N_sp_train).

Value

A named list with:

Bt Numeric matrix. Posterior covariance of the discrepancy.

Btbt Numeric vector. Posterior mean of the discrepancy.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

Examples

```
set.seed(1)
Sigma <- matrix(c(1, 0.3, 0.1, 0.3, 1, 0.3, 0.1, 0.3, 1), 3)
mu <- rnorm(3)
zu <- rnorm(3)
post <- cal_Bt_bt(mu, invSigma = solve(Sigma), tau2t = 0.4, zt_ut = zu, I_Stilde = diag(3))
K <- Sigma %*% solve(Sigma + 0.4 * diag(3))  # Kalman gain
all.equal(post$Bt, Sigma - K %*% Sigma)
all.equal(as.vector(post$Btbt), as.vector(mu + K %*% (zu - mu)))
```

`cal_errorbar` *Compute median and 95% credible interval across rows*

Description

For each row of the matrix `X`, computes the median and the 2.5th and 97.5th percentiles across columns (samples).

Usage

```
cal_errorbar(X)
```

Arguments

`X`	Numeric matrix. Rows correspond to spatial locations or parameters; columns correspond to posterior samples.

Value

A data frame with columns:

med Row-wise median.

lower Row-wise 2.5th percentile.

upper Row-wise 97.5th percentile.

See Also

`cal_errorbar_mean`

Examples

```
set.seed(1)
X <- matrix(rnorm(3 * 200, mean = 1:3), nrow = 3)
cal_errorbar(X)
```

cal_errorbar_mean *Compute mean and 95% credible interval across rows*

Description

For each row of the matrix `X`, computes the mean and the 2.5th and 97.5th percentiles across columns (samples).

Usage

```
cal_errorbar_mean(X)
```

Arguments

`X`	Numeric matrix. Rows correspond to spatial locations or parameters; columns correspond to posterior samples.

Value

A data frame with columns:

med Row-wise mean.

lower Row-wise 2.5th percentile.

upper Row-wise 97.5th percentile.

See Also

`cal_errorbar`

Examples

```
set.seed(1)
X <- matrix(rexp(3 * 200, rate = 1:3), nrow = 3)
cal_errorbar_mean(X)
```

cal_jacobian_logit_uniform *Log absolute Jacobian of the logit-uniform transformation*

Description

Computes the log (or raw) absolute Jacobian of the transformation from the constrained parameter `eta` (bounded in `[eta_limit_low, eta_limit_high]`) to the unconstrained logit scale. Fixed coordinates (equal lower and upper bounds) are excluded from the product. Versions up to 1.1.1 excluded zero entries of `eta` instead, which wrongly dropped a free coordinate equal to zero.

Usage

```
cal_jacobian_logit_uniform(eta, eta_limit_low, eta_limit_high, log = FALSE)
```

Arguments

`eta`	Numeric vector. Constrained parameter values.
`eta_limit_low`	Numeric vector. Lower bounds for each element of `eta`.
`eta_limit_high`	Numeric vector. Upper bounds for each element of `eta`.
`log`	Logical. If `TRUE`, returns the log Jacobian. Defaults to `FALSE`.

Value

Numeric scalar (log absolute Jacobian or absolute Jacobian).

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

Examples

```
eta <- c(0.3, 2.5)
low <- c(0, 1)
high <- c(1, 4)
cal_jacobian_logit_uniform(eta, low, high)
cal_jacobian_logit_uniform(eta, low, high, log = TRUE)
# A fixed coordinate (low == high) does not contribute
all.equal(cal_jacobian_logit_uniform(c(0.3, 2), c(0, 2), c(1, 2)), 1 / (0.3 * 0.7))
```

`calibration_fit` *Fit modular calibration conditional on emulator posterior draws*

Description

Targets one conditional posterior per fixed emulator coefficient set and pools equal numbers of samples. The emulator is not updated by field observations. Gaussian emulator fields are integrated out. By default discrepancy states are also integrated out using a Kalman filter.

Usage

```
calibration_fit(
  object,
  z,
  U,
  locations = seq_len(object$S),
  backend = "rstan",
```

```
  draws = seq_len(object$M),
  chains = 1L,
  iter,
  warmup = iter%/%2L,
  draws_per_emulator = min(200L, chains * (iter - warmup)),
  n0 = 30,
  d0 = 2,
  discount = 1,
  m0 = rep(0, length(locations)),
  M0 = diag(length(locations)),
  seed = 1234L,
  cores = 1L,
  marginal_discrepancy = TRUE,
  keep_predictions = FALSE,
  keep_discrepancy = TRUE,
 control = if (warmup == 0L) list(adapt_engaged = FALSE, stepsize = 0.001, max_treedepth
    = 12L) else list(adapt_delta = 0.95, max_treedepth = 12L),
  refresh = 100L,
  checkpoint_dir = NULL,
  cache = TRUE,
  cache_dir = NULL,
  compiled_model = NULL,
  init = "mode",
  mode_starts = 20L,
  parallel_draws = 1L
)
```

Arguments

`object`	Prepared emulator from `calibration_prepare()`.
`z`	Finite time-by-observed-location matrix of field observations.
`U`	Full-grid symmetric positive-definite discrepancy covariance scale. Correlation matrices are supported, but the diagonal need not equal one. Discrepancy increments have covariance tau2t * U; observation errors retain covariance tau2t * I.
`locations`	Ordered original-grid indices corresponding to columns of z.
`backend`	Currently "rstan".
`draws`	Emulator draw indices to use, without duplicates.
`chains`, `iter`, `warmup`	Number of chains, total iterations per chain, and warmup iterations for each emulator-conditional fit. `iter` has no default and must be given. For inference use several chains and enough iterations, for example `chains = 4, iter = 2000`; `iter = 1` (with no warmup) is a quick execution check, not inference. `chains` defaults to one and `warmup` to `floor(iter / 2)`.
`draws_per_emulator`	Equal number of retained samples per conditional. Defaults to the smaller of 200 and the available post-warmup samples.

`n0`, `d0` Positive inverse-gamma shape and scale hyperparameters.

`discount` Calibration variance discount factor in (0, 1].

`m0`, `M0` Initial discrepancy mean and covariance scale at observed sites.

`seed` Nonnegative integer RNG seed, local to the function call.

`cores` Positive number of parallel chains. Defaults to one.

`marginal_discrepancy`
Whether to integrate out discrepancy states.

`keep_predictions`, `keep_discrepancy`
Whether to recover and retain latent fields, replicated observations, and discrepancy states.

`control` Named list passed to rstan sampling. With zero warmup the defaults disable adaptation and use stepsize = 0.001. With warmup they use adapt_delta = 0.95. Both use max_treedepth = 12.

`refresh` Stan progress reporting interval.

`checkpoint_dir` Optional directory for completed conditional fits. Each fit is saved as it completes; a fit already saved there with the same data, settings, seed and Stan program is reused instead of refitted, so an interrupted calibration resumes where it stopped.

`cache`, `cache_dir`
Whether to reuse compiled models and optional user cache directory. Installed package files are never modified.

`compiled_model` Optional compatible compiled Stan model for advanced use.

`init` Starting values of the free parameters for every chain. `"mode"` (the default) runs L-BFGS (`rstan::optimizing()`) from `mode_starts` random starts in each conditional fit and starts every chain, slightly jittered, at the optimum with the highest log posterior density, with the precisions of that optimum. Chains started far from the main mode can settle in local modes of negligible posterior mass, which this avoids; the optima found are recorded in `diagnostics[[j]]$mode_search`. `"random"` draws each chain's start uniformly within the bounds, reproducibly from `seed`, so that $\hat{R}$ can reveal chains that settle in different modes; `"midpoint"` starts every chain at the centre of the bounds; or give a numeric vector of parameter values in training-column order (named vectors are matched by name), strictly inside the bounds of the free parameters. Values given for fixed parameters are ignored. Each conditional fit starts afresh; except for `"mode"`, precisions start at n0 / d0.

`mode_starts` Number of random starts of the mode search for `init = "mode"`.

`parallel_draws` Number of conditional fits (one per emulator draw) run at the same time, in forked processes. Each forked fit runs its chains one after another, so `cores` is then not used and `parallel_draws` processes run at once. With a multithreaded BLAS (for example oneMKL or OpenBLAS), start R with `OMP_NUM_THREADS=1`: forking after the BLAS has used several threads can leave the forked fits hung, and `calibration_fit()` warns when it detects that setup. Results do not depend on it: every fit draws its random numbers from its own seeds, and rstan's draws do not depend on the number of cores. Forking is unavailable on Windows, where fits run one after another. Defaults to one.

Details

Requires the suggested package rstan and a C++ toolchain on first compilation. One iteration per emulator draw is an execution experiment, not converged posterior inference; its samples stay near `init`. When the midpoint of the bounds is close to the true value, as in `dt_calibration`, use `init = "random"` to check that results do not depend on the start. Missing observations and new time/grid locations are not supported in this initial interface.

Value

An ordinary serializable list of class spDBL_calibration containing samples, per-conditional diagnostics, settings, and prediction context. No compiled Stan model is stored in the result.

See Also

`calibration_predict()`, `calibration_validate()`

Examples

```
data("dt_calibration")
d <- dt_calibration
em <- emulator_learn(d$training$input, d$training$responses, d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
                     seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
fit <- calibration_fit(cal, d$field$z, d$field$U,
                       locations = d$field$locations, draws = 1:2,
                       chains = 1, iter = 200, seed = 1)
summary(fit)
```

---

calibration_predict *Recover fields at the calibrated observation times and locations*

Description

Recovers discrepancy trajectories when needed, then draws latent emulator fields conditional on the observations and fresh replicated observations. This is not forecasting at new times or new spatial sites.

Usage

```
calibration_predict(object, seed = 1234L)
```

Arguments

`object`	Result from `calibration_fit()`.
`seed`	Local RNG seed for Gaussian recovery.

Value

A list containing y, u, and z_rep arrays, each with dimensions posterior draws by observation times by observed locations.

Examples

```
data("dt_calibration")
d <- dt_calibration
em <- emulator_learn(d$training$input, d$training$responses, d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
                     seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
fit <- calibration_fit(cal, d$field$z, d$field$U,
                       locations = d$field$locations, draws = 1:2,
                       chains = 1, iter = 200, seed = 1)
pred <- calibration_predict(fit)
dim(pred$y)
```

---

calibration_prepare *Prepare a fitted emulator for PDE-free calibration*

Description

Fits a Gaussian process (the design GP) to the recorded autoregressive design matrices, so that design rows can be predicted at untried inputs. Reuses the fitted emulator's coefficient samples or draws additional samples from saved filtering results, without refitting the dynamic emulator.

Usage

```
calibration_prepare(
  emulator,
  lower,
  upper,
  n_draws = NULL,
  seed = 1234,
  sampling = NULL,
  design_uncertainty = TRUE,
  design_kernel = c("matern52", "sqexp"),
  design_groups = c("time", "shared")
)
```

Arguments

emulator	A sigma2R object from emulator_learn().
lower, upper	Finite parameter bounds in training-column order. Equal bounds specify fixed coordinates. Named bounds are matched to the training parameter names.

`n_draws` Number of fresh coefficient sample sets. NULL reuses existing samples when the selected sampling method agrees with the fitted emulator.

`seed` Optional local seed used when drawing fresh samples.

`sampling` NULL inherits the fitted emulator's method (normally "marginal"). "marginal" draws independent smoothed episode coefficients conditional on a shared variance; "joint" draws coherent backward paths. Changing the method resamples the existing number of draws unless n_draws is supplied.

`design_uncertainty`
If TRUE (the default), the predictive covariance of the AR design GP is added to the field covariance used by `calibration_fit()` and `calibration_predict()`. FALSE uses the GP mean only, as spDBL 1.1.1 did; its intervals are too narrow.

`design_kernel` Kernel of the design GP: `"matern52"` (Matern 5/2, the default) or `"sqexp"` (squared exponential). Both are anisotropic, with one lengthscale per free parameter on the unit scale.

`design_groups` `"time"` (the default) fits separate kernel hyperparameters, by maximum likelihood, for the design columns of each time step; `"shared"` fits one set for all times. `design_kernel = "sqexp", design_groups = "shared"` reproduces the design GP of earlier spDBL versions.

Details

The AR design GP supplies the design rows at a new input. At each time step, all design columns share one kernel (with `design_groups = "time"`; one kernel for all times with `"shared"`), so their prediction errors at an input η are jointly Gaussian with covariance $C_{F,t}(\eta) = v_t(\eta)\hat{\Sigma}_{F,t}$: $v_t(\eta)$ is the GP's latent variance factor and $\hat{\Sigma}_{F,t}$ is the maximum-likelihood covariance between the design columns of time t. With `design_uncertainty = TRUE` this is propagated through each coefficient draw: at each time, the field covariance gains $\Theta^\top C_{F,t}(\eta)\Theta$ over the episodes of that time, including covariance between spatial blocks. Design errors are treated as independent across times and independent of the emulator's own prediction error. The emulator's conditional covariance and coefficient uncertainty are always retained. Calibration support is limited to the scalar sigma2R model on the original time/grid. Objects prepared by spDBL 1.1.1 or earlier have no design variance.

Value

A spDBL_calibration_input object containing the training data, coefficient samples, offline design predictor, and shared GP factors. At most one draw's residual weights are cached at a time.

See Also

`calibration_validate()`, `calibration_fit()`, `emulator_predict()`

Examples

```
data("dt_calibration")
d <- dt_calibration
steps <- 1:6  # the first 6 of 26 time steps, to keep the example fast
em <- emulator_learn(d$training$input, d$training$responses[steps], d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
```

```
                    seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
class(cal)
c(times = cal$T, locations = cal$S, draws = cal$M)
cal$parameter_names
```

`calibration_validate` *Validate offline calibration predictions against held-out simulations*

Description

Held-out trajectories are used only for this accuracy assessment. They are not added to either GP's training data.

Usage

```
calibration_validate(object, input_new, responses, seed = 1234)
```

Arguments

`object`	A prepared object from `calibration_prepare()`.
`input_new`	Matrix of held-out mechanistic inputs.
`responses`	Time-indexed list of held-out response matrices.
`seed`	Not used since spDBL 1.2.0, whose AR designs are deterministic; kept for compatibility.

Value

A list with episode design errors and per-input field RMSE using known versus GP-predicted autoregressive rows.

Examples

```
data("dt_calibration")
d <- dt_calibration
steps <- 1:6  # the first 6 of 26 time steps, to keep the example fast
em <- emulator_learn(d$training$input, d$training$responses[steps], d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
                     seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
held_out <- lapply(d$validation$responses[steps], function(y) y[1:3, , drop = FALSE])
val <- calibration_validate(cal, d$validation$input[1:3, ], held_out)
val$field
```

check_pds *Check and repair a matrix to be positive definite and symmetric*

Description

Verifies that a matrix is approximately symmetric (within a relative and absolute tolerance). If so, symmetrises it and, if still not positive definite, adds a small diagonal ridge. Raises an error if the asymmetry exceeds the tolerance.

Usage

```
check_pds(C, eps = 10^(-5), per = 0.05)
```

Arguments

`C`	Numeric matrix to check.
`eps`	Numeric scalar. Absolute tolerance for the Frobenius norm of `C - t(C)`. Defaults to `1e-5`.
`per`	Numeric scalar. Relative tolerance: the ratio of the asymmetry norm to the mean of `C`. Defaults to `0.05`.

Value

The (possibly corrected) positive definite symmetric matrix.

See Also

`make_pds`

Examples

```
# A covariance matrix with a tiny rounding asymmetry
A <- matrix(c(2, 1, 1 + 1e-8, 2), 2)
isSymmetric(A)
A_fixed <- check_pds(A)
isSymmetric(A_fixed)
A_fixed
```

`compute_analytical_waic` *Analytical WAIC for an MNIW multivariate regression posterior*

Description

Computes the log pointwise predictive density and the posterior-variance form of WAIC without posterior simulation. Each row of the response matrix is one multivariate observation.

Usage

```
compute_analytical_waic(Y, X, Mn, mn, Psi_n, nu_n)
```

Arguments

`Y`	Finite numeric response matrix with dimensions n by q.
`X`	Finite numeric design matrix with dimensions n by p. Include an intercept column explicitly when required.
`Mn`	Symmetric positive-definite p by p posterior row covariance of the regression coefficients, conditional on the response covariance.
`mn`	Finite numeric p by q posterior coefficient mean. Supply the actual mean, not its precision-weighted parameterization.
`Psi_n`	Symmetric positive-definite q by q inverse-Wishart posterior scale matrix.
`nu_n`	Finite scalar inverse-Wishart posterior degrees of freedom, strictly greater than q - 1.

Details

The likelihood and posterior are

$$Y_i \mid B, \Sigma \sim N_q(x_i B, \Sigma),$$

independently across rows, and

$$B \mid \Sigma, D \sim MN(m_n, M_n, \Sigma), \quad \Sigma \mid D \sim IW(\nu_n, \Psi_n).$$

The inverse-Wishart density is proportional to $|\Sigma|^{-(\nu_n+q+1)/2} \exp\{-\mathrm{tr}(\Psi_n \Sigma^{-1})/2\}$. In particular, `mn` is the coefficient mean itself. When using `MNIW_R()` with `H = diag(n)`, pass `Vbnew`, `Cnew`, `Snew`, and `vnew` as `Mn`, `mn`, `Psi_n`, and `nu_n`, respectively.

Let $h_i = x_i M_n x_i^T$ and $d_i = (y_i - x_i m_n)\Psi_n^{-1}(y_i - x_i m_n)^T$. The row predictive density is multivariate Student t with $\nu_n - q + 1$ degrees of freedom, mean $x_i m_n$, and scale $(1+h_i)\Psi_n/(\nu_n - q + 1)$. The WAIC penalty for row i is

$$p_i = \tfrac{1}{4}\left\{ \sum_{j=1}^{q} \psi_1((\nu_n - j + 1)/2) + 2qh_i^2 + 4h_i\nu_n d_i + 2\nu_n d_i^2 - 4d_i \right\}.$$

Here ψ_1 is the trigamma function. WAIC is $-2\sum_i(\mathrm{lppd}_i - p_i)$.

Use the observations that fitted the posterior to compute training-data WAIC. Scoring other observations evaluates the supplied posterior at those observations; it does not refit the model. These functions assume independent likelihood rows and do not directly compute WAIC for correlated spatial GP rows or a full state-space emulator. Compare models using the same response data and the same definition of an observation. Lower WAIC indicates higher estimated predictive accuracy among the compared models.

Value

A list containing the original scalar summaries `WAIC`, `lppd`, and `p_waic`; their difference `elpd_waic`; and an n by 4 `pointwise` matrix with columns `lppd`, `p_waic`, `elpd_waic`, and `WAIC`.

Author(s)

Sudipto Banerjee (original derivation and development implementation).

References

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. Journal of Machine Learning Research, 11, 3571–3594.

See Also

`compute_sample_waic()`, `MNIW_R()`, `lppd_IW_1t()`

Examples

```
X <- cbind(1, c(-1, 0, 1, 2))
Y <- cbind(c(0.1, 0.4, 1.2, 1.7), c(1.2, 0.8, 0.5, 0.1))
post <- MNIW_R(X, Y, H = diag(nrow(Y)), v = 4, S = diag(2),
               C = matrix(0, 2, 2), Vb = diag(2))
score <- compute_analytical_waic(Y, X, post$Vbnew, post$Cnew,
                                 post$Snew, post$vnew)
score$WAIC
score$pointwise
```

---

compute_sample_waic *WAIC from paired coefficient and covariance draws*

Description

Evaluates row-wise multivariate normal log likelihoods for paired posterior draws, using log-sum-exp aggregation for the predictive density and sample variances for the WAIC penalty.

Usage

```
compute_sample_waic(Y, X, B_samples, Sigma_samples, return_log_lik = FALSE)
```

Arguments

Y	Finite numeric response matrix with dimensions n by q.
X	Finite numeric design matrix with dimensions n by p. Include an intercept column explicitly when required.
B_samples	Finite numeric array with dimensions p by q by S, containing posterior coefficient draws. At least two draws are required.
Sigma_samples	Finite numeric array with dimensions q by q by S, containing symmetric positive-definite posterior response covariance draws. Slice s must be paired with slice s of B_samples.
return_log_lik	Logical scalar. If TRUE, also return the S by n log_lik matrix, suitable for loo::waic() or loo::loo(). The loo package is optional and is not called by this function.

Details

Each response row is one observation, with likelihood $Y_i \mid B, \Sigma \sim N_q(x_i B, \Sigma)$ independently across rows. The draws can come from any posterior with this likelihood; they need not follow an MNIW law. The analytical comparison requires an MNIW posterior. Arrays retain three dimensions even when p, q, or n equals one. The penalty uses sample variance with denominator S - 1, matching loo::waic() for a matrix of log likelihoods. Correlated draws can increase Monte Carlo error; this function does not diagnose convergence.

Value

The same summaries and pointwise matrix as compute_analytical_waic(), with an additional log_lik matrix when return_log_lik = TRUE.

Author(s)

Sudipto Banerjee (original development implementation).

See Also

compute_analytical_waic()

Examples

```
X <- matrix(1, 2, 1)
Y <- matrix(c(0.1, 0.3), 2, 1)
B <- array(c(0, 0.2, 0.4), c(1, 1, 3))
Sigma <- array(c(0.5, 1, 1.5), c(1, 1, 3))
score <- compute_sample_waic(Y, X, B, Sigma, return_log_lik = TRUE)
score$WAIC
dim(score$log_lik)
```

dMTig *Log density of the matrix-T distribution with inverse-gamma right covariance*

Description

Evaluates the log (or raw) density of the matrix-T distribution that arises when the right covariance is $\sigma^2 R$ and $\sigma^2 \sim IG(\nu, d)$.

Usage

```
dMTig(Y, m, M, nu, d, R, log = TRUE)
```

Arguments

Y	Numeric matrix. Observation matrix ($p \times S$).
m	Numeric matrix. Mean matrix ($p \times S$).
M	Numeric matrix. Left covariance matrix ($p \times p$, positive definite).
nu	Numeric scalar. Degrees of freedom (shape parameter of the inverse-gamma on σ^2).
d	Numeric scalar. Rate parameter of the inverse-gamma on σ^2).
R	Numeric matrix. Right correlation matrix ($S \times S$, positive definite).
log	Logical. If TRUE (default), returns the log density.

Value

Numeric scalar. Log density (or density if log = FALSE).

Examples

```
set.seed(1)
Y <- matrix(rnorm(6), 2, 3)
R <- 0.5^abs(outer(1:3, 1:3, "-"))
dMTig(Y, m = matrix(0, 2, 3), M = diag(2), nu = 3, d = 2, R = R)
dMTig(Y, m = matrix(0, 2, 3), M = diag(2), nu = 3, d = 2, R = R, log = FALSE)
```

dt_calibration *Saved SIR simulations and synthetic field observations for calibration*

Description

A 6 by 6 grid with 26 observation times, including physical time zero. Training uses 40 simulator inputs; ten further inputs are held out. Responses and observations are on the log(I + 1) scale. This is a different simulation design from dt_emulation: eta1 bounds are 2 to 4 here, whereas dt_emulation uses inputs near 1 to 2.

Usage

```
dt_calibration
```

Format

A list with training and validation lists (input matrices and time-indexed response matrices); named lower and upper parameter bounds; Nx, Ny, time, and grid metadata; a field list containing observations, original-grid indices, discrepancy correlation, and prior settings; a separate truth list used only for validation/plots; and provenance containing input identifiers, response scale, seed, and source checksums.

Source

Saved SIR simulations following Equation (26) in Banerjee, Chen, Frankenburg and Zhou (2025), JMLR 26(146). The field-generation/import script is maintained in data-raw.

See Also

calibration_prepare(), calibration_fit()

Examples

```
data("dt_calibration")
d <- dt_calibration
rbind(lower = d$lower, upper = d$upper, truth = d$truth$eta)
dim(d$field$z)
length(d$training$responses)
```

dt_emulation *Example emulation dataset*

Description

A list of example training and testing objects used to illustrate the emulation workflow in the package.

Usage

```
dt_emulation
```

Format

A list with 4 elements:

pde_para_train Training set of PDE parameter values.

pde_para_test Test set of PDE parameter values.

dt_pde_train Training set of PDE-based outputs corresponding to `pde_para_train`.

dt_pde_test Test set of PDE-based outputs corresponding to `pde_para_test`.

Source

Simulated data / package example data

Examples

```
data("dt_emulation")
d <- dt_emulation
dim(d$pde_para_train)
length(d$dt_pde_train)
dim(d$dt_pde_train$T1)
```

emulator_learn *Fit an FFBS-based dynamic emulator*

Description

Fits an episode-based emulator to saved simulator runs. The default inverse-Wishart model is retained for compatibility. Use covariance = "sigma2R" for scalar-variance coefficient samples that can be passed to `calibration_prepare()`.

Usage

```
emulator_learn(
  pde_para_train,
  dt_pde_train,
  Nx,
  Ny,
  F_ls_train = "default",
  F_ls_test = "default",
  N_people = 10000,
  nsam = 10,
  AR_choice = 2,
  episode_window = 1,
  gp_tune = 0.5,
  gp_sigma2 = 1.1,
  gp_tau2 = 1e-04,
  covariance = c("iw", "sigma2R"),
  R_episode = NULL,
  R_field = NULL,
  seed = NULL,
  verbose = FALSE,
  sampling = c("marginal", "joint"),
  keep_draws = TRUE
)
```

Arguments

`pde_para_train`	Finite numeric matrix of training parameter inputs.
`dt_pde_train`	Time-indexed list of training response matrices, with simulator runs in rows and original-grid locations in columns.
`Nx`, `Ny`	Positive integer grid dimensions.
`F_ls_train`	"default" or a list of episode design matrices.
`F_ls_test`	Unused compatibility argument.
`N_people`	Population scale recorded in the emulator metadata.
`nsam`	Positive integer number of posterior coefficient draws.
`AR_choice`	Autoregressive order, 1 or 2.
`episode_window`	Number of spatial strips per block; must divide Nx.
`gp_tune`	Positive range tuning factor.
`gp_sigma2`, `gp_tau2`	Positive input/spatial kernel amplitude and nugget.
`covariance`	"iw" (default) or "sigma2R".
`R_episode`, `R_field`	Optional spatial covariance matrices for one block and for the full grid. `R_episode` defaults to the exponential kernel over the sites of a block. `R_field` is used only by calibration. Unless it is supplied it is not formed here, since an S x S matrix costs $O(S^2)$ memory and its check $O(S^3)$ time; calibration builds the same exponential kernel at the observed sites when it needs it.

seed	Optional local RNG seed. NULL follows the session RNG.
verbose	Whether to report progress.
sampling	"marginal" (default) uses smoothed episode marginals conditional on a shared covariance draw, following Eq. (9) and Algorithm 4. "joint" uses backward conditionals to preserve dependence across episodes.
keep_draws	If TRUE (default), store every coefficient draw. If FALSE (marginal sampling only), store only where each episode's draws start in the random-number stream and regenerate identical draws when prediction or calibration needs them. This removes the largest part of a large emulator's memory at the cost of re-drawing (about the original sampling time per full pass over the draws). Each episode then costs about 2.5 KB instead of its draws, so it pays off once an episode's draws exceed that.

Value

An object of class spDBL_emulator, retaining para_ffbs, res_ffbs, and a complete setup list for prediction and calibration.

See Also

emulator_predict(), calibration_prepare()

Examples

```
data("dt_emulation")
d <- dt_emulation
em <- emulator_learn(d$pde_para_train, d$dt_pde_train, Nx = 10, Ny = 10,
                     nsam = 5, seed = 1)
class(em)
c(n_train = em$setup$n_train, episodes = em$setup$nT, locations = em$setup$N_sp)
```

---

emulator_predict *Predict simulator output from a fitted emulator*

Description

With an ordinary emulator_learn() result, supply either new design matrices or saved simulator outputs used to construct them. With a calibration_prepare() result, the offline GP supplies those rows and dt_pde_test is unnecessary.

Usage

```
emulator_predict(
  emulator,
  input_new,
  dt_pde_test = NULL,
  F_new_ls = "default",
  MC = FALSE
)
```

Arguments

emulator	An emulator or prepared calibration-input object.
input_new	Matrix or vector of new mechanistic parameter inputs.
dt_pde_test	Optional saved responses at new inputs, used only to construct default AR designs for an ordinary emulator.
F_new_ls	"default" or explicit episode design matrices.
MC	If TRUE, return coefficient-conditioned mean samples, not full draws of the emulator field. If FALSE, return smoothed means for an ordinary emulator, or draw-averaged means for a prepared emulator.

Value

A time-indexed list of matrices, or arrays with a final posterior draw dimension when MC = TRUE.

See Also

FFBS_predict_exact(), calibration_predict()

Examples

```
data("dt_emulation")
d <- dt_emulation
em <- emulator_learn(d$pde_para_train, d$dt_pde_train, Nx = 10, Ny = 10,
                     nsam = 5, seed = 1)
pred <- emulator_predict(em, d$pde_para_test, dt_pde_test = d$dt_pde_test)
dim(pred[[2]])
sqrt(mean((unlist(pred) - unlist(d$dt_pde_test))^2))
```

---

expit *Logistic (expit) function*

Description

Computes the logistic sigmoid $1/(1+e^{-x})$.

Usage

```
expit(x)
```

Arguments

x	Numeric vector or matrix.

Value

Numeric vector or matrix with values in $(0, 1)$.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

Examples

```
expit(c(-2, 0, 2))
all.equal(expit(1.5), stats::plogis(1.5))
```

`FF` *Forward Filter for the MNIW dynamic linear model*

Description

Runs the full forward filtering pass over `nT` time steps under the Matrix Normal Inverse Wishart (MNIW) model. At each step, calls `FF_1step_cpp` to update the state mean and covariance as well as the inverse-Wishart parameters for the right-covariance matrix Σ.

Usage

```
FF(Y, F_ls, G_ls, W_ls, V_ls, m0, M0, n0, D0, nT, delta = 1, verbose = FALSE)
```

Arguments

`Y`	List of length `nT`. Each element `Y[[t]]` is the $N \times q$ data matrix at time t.
`F_ls`	Either a single $N \times p$ covariate matrix (constant over time) or a list of `nT` such matrices (time-varying).
`G_ls`	Either a single $p \times p$ state transition matrix or a list of `nT` such matrices.
`W_ls`	Either a single $p \times p$ state noise left-covariance matrix or a list of `nT` such matrices.
`V_ls`	Either a single $N \times N$ observation noise left-covariance matrix or a list of `nT` such matrices.
`m0`	Numeric matrix. Prior mean of the state $\beta_0 \mid D_0$ ($p \times q$).
`M0`	Numeric matrix. Prior left-covariance of $\beta_0 \mid D_0$ ($p \times p$).
`n0`	Numeric scalar. Prior degrees of freedom of the inverse-Wishart on $\Sigma \mid D_0$.
`D0`	Numeric matrix. Prior scale matrix of the inverse-Wishart on $\Sigma \mid D_0$ ($q \times q$).
`nT`	Integer. Number of time steps.
`delta`	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to `1`.
`verbose`	Logical. If `TRUE`, emits progress messages. Defaults to `FALSE`.

Value

A named list of length nT + 1. Elements "T1" through "T<nT>" each contain a list with filtered parameters nt, Dt, at, At, mt, Mt. The additional element prior stores m0, M0, n0, D0.

See Also

BS, FFBS

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res_ff <- FF(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
             par$n0, par$D0, nT = 4)
names(res_ff)
res_ff$T4$mt  # filtered state mean at the last time (p x S)
```

FF_1step_R_I *Single forward filter step (identity right-covariance)*

Description

Performs one step of the forward filter under the model where the right covariance of the state and observation is the identity matrix (no inverse-Wishart update for Σ).

Usage

```
FF_1step_R_I(Yt, Ft, Gt, Wt, Vt, mt_1, Mt_1, delta = 1)
```

Arguments

Yt	Numeric matrix. Data matrix at time t ($N \times S$).
Ft	Numeric matrix. Covariate matrix at time t ($N \times p$).
Gt	Numeric matrix. State transition matrix ($p \times p$).
Wt	Numeric matrix. State noise left-covariance ($p \times p$).
Vt	Numeric matrix. Observation noise left-covariance ($N \times N$).
mt_1	Numeric matrix. Filtered state mean at $t-1$ ($p \times S$).
Mt_1	Numeric matrix. Filtered state left-covariance at $t-1$ ($p \times p$).
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.

Value

A named list with:

at One-step-ahead state mean.

At One-step-ahead state left-covariance.

mt Filtered state mean.

Mt Filtered state left-covariance.

delta Always 1; discounting is not implemented.

See Also

FF_I

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 1)
par <- dat$para
one <- FF_1step_R_I(dat$Y[[1]], par$F0, par$G0, par$W0, par$V0,
                    par$m0, par$M0)
one$mt
one$Mt
```

FF_1step_R_sigma2R *Single forward filter step (scalar right-covariance, sigma-squared times R)*

Description

Performs one step of the forward filter under the model where the right covariance of Y is $\sigma^2 R$ with $\sigma^2 \sim IG(n, d)$. The inverse of R (`Rinv`) must be precomputed and passed in.

Usage

```
FF_1step_R_sigma2R(Yt, Ft, Gt, Wt, Vt, mt_1, Mt_1, nt_1, Dt_1, Rinv, delta = 1)
```

Arguments

`Yt`	Numeric matrix. Data matrix at time t ($N \times S$).
`Ft`	Numeric matrix. Covariate matrix at time t ($N \times p$).
`Gt`	Numeric matrix. State transition matrix at time t ($p \times p$).
`Wt`	Numeric matrix. State noise left-covariance at time t ($p \times p$).
`Vt`	Numeric matrix. Observation noise left-covariance at time t ($N \times N$).
`mt_1`	Numeric matrix. Filtered state mean at $t-1$ ($p \times S$).
`Mt_1`	Numeric matrix. Filtered state left-covariance at $t-1$ ($p \times p$).

nt_1	Numeric scalar. Shape parameter of $\sigma^2 \mid D_{t-1}$.
Dt_1	Numeric scalar. Rate parameter of $\sigma^2 \mid D_{t-1}$.
Rinv	Numeric matrix. Precomputed inverse of the spatial correlation matrix R ($S \times S$).
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.

Value

A named list with updated filtering parameters:

nt Updated shape parameter.

Dt Updated rate parameter.

at One-step-ahead state mean ($p \times S$).

At One-step-ahead state left-covariance ($p \times p$).

mt Filtered state mean ($p \times S$).

Mt Filtered state left-covariance ($p \times p$).

delta Always 1; discounting is not implemented.

See Also

FF_sigma2R

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 1)
par <- dat$para
R <- 0.5^abs(outer(1:3, 1:3, "-"))
one <- FF_1step_R_sigma2R(dat$Y[[1]], par$F0, par$G0, par$W0, par$V0,
                          par$m0, par$M0, nt_1 = 3, Dt_1 = 1,
                          Rinv = solve(R))
c(nt = one$nt, Dt = one$Dt)
one$mt
```

FF_bigdata_R *Forward Filter for big data stored in CSV files (MNIW model)*

Description

Runs the forward filtering pass for large datasets that are partitioned into spatial blocks and stored as CSV files on disk. At each time step the spatial domain is traversed block-by-block using a snake traversal order (see generate.grid.rowsnake), and results are written to disk to avoid exhausting memory.

Usage

```
FF_bigdata_R(
  Y_ls,
  F_ls,
  G_ls,
  W_ls,
  V_ls,
  m0,
  M0,
  n0,
  D0,
  nT,
  fnrow,
  fncol,
  bnrow,
  bncol,
  path_out,
  delta = 1,
  verbose = FALSE
)
```

Arguments

Y_ls	Character vector of length nT. File paths to the response CSV files, one per time step.
F_ls	Either a single file path (constant F_t) or a character vector of length nT (time-varying). Can also contain a second set of columns (AR2 lags) for the same file at offset fncol. Each design block is read with the same rows and columns as the response block, so the design must have as many columns as a block (bncol), as with the autoregressive designs.
G_ls	Either a single file path (constant G_t) or a character vector of length nT (time-varying).
W_ls	Either a single file path (constant W_t) or a character vector of length nT (time-varying).
V_ls	Either a single file path (constant V_t) or a character vector of length nT (time-varying). Expected to be a square block submatrix indexed by the row range of each spatial block.
m0	Numeric matrix. Prior state mean ($p \times q$).
M0	Numeric matrix. Prior left-covariance of the state ($p \times p$).
n0	Numeric scalar. Prior degrees of freedom of the inverse-Wishart.
D0	Numeric matrix. Prior scale matrix of the inverse-Wishart.
nT	Integer. Number of time steps (files).
fnrow	Integer. Total number of rows in each data file.
fncol	Integer. Total number of columns in each data file.
bnrow	Integer. Number of rows per spatial block.

bncol	Integer. Number of columns per spatial block.
path_out	Character. Directory path where output CSV files are written.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list of character vectors with paths to the output CSV files:

nt_ls Paths to filtered degrees-of-freedom files.

Dt_ls Paths to filtered scale-matrix files.

mt_ls Paths to filtered state-mean files.

Mt_ls Paths to filtered left-covariance files.

at_ls Paths to predicted state-mean files.

At_ls Paths to predicted left-covariance files.

FT Path to the covariate matrix at the final time step.

VT Path to the noise covariance matrix at the final time step.

See Also

FF, generate.grid.rowsnake

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 2, p = 2, nT = 3)
par <- dat$para
dir <- file.path(tempdir(), "ff_big")
dir.create(dir, showWarnings = FALSE)
csv <- function(x, name) {
  f <- file.path(dir, name)
  write.csv(x, f, row.names = FALSE)
  f
}
Y_files <- mapply(csv, dat$Y, paste0("Y", 1:3, ".csv"))
files <- FF_bigdata_R(Y_files, csv(par$F0, "F.csv"), csv(par$G0, "G.csv"),
                      csv(par$W0, "W.csv"), csv(par$V0, "V.csv"), par$m0,
                      par$M0, par$n0, par$D0, nT = 3, fnrow = 5, fncol = 2,
                      bnrow = 5, bncol = 2, path_out = dir)
# With a single block this matches the in-memory filter
res_ff <- FF(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
             par$n0, par$D0, nT = 3)
all.equal(unname(as.matrix(read.csv(files$mt_ls[3]))), unname(res_ff$T3$mt))
unlink(dir, recursive = TRUE)
```

FF_I *Forward Filter with identity right-covariance*

Description

Runs the full forward filtering pass over nT time steps under the model where the right covariance is fixed at the identity (no inverse-Wishart update). Calls FF_1step_R_I at each step.

Usage

```
FF_I(Y, F_ls, G_ls, W_ls, V_ls, m0, M0, nT, delta = 1, verbose = FALSE)
```

Arguments

Y	List of length nT. Each element is the data matrix at time t.
F_ls	Covariate matrix or list of matrices (see FF).
G_ls	State transition matrix or list (see FF).
W_ls	State noise left-covariance matrix or list (see FF).
V_ls	Observation noise left-covariance matrix or list (see FF).
m0	Numeric matrix. Prior state mean.
M0	Numeric matrix. Prior state left-covariance.
nT	Integer. Number of time steps.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list of length nT + 1. Elements "T1" through "T<nT>" contain at, At, mt, Mt. The element prior stores m0 and M0.

See Also

FF_1step_R_I, FFBS_I

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res_ff <- FF_I(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0, nT = 4)
# The first step equals a single call of FF_1step_R_I
one <- FF_1step_R_I(dat$Y[[1]], par$F0, par$G0, par$W0, par$V0,
                    par$m0, par$M0)
all.equal(res_ff$T1, one)
```

FF_sigma2R *Forward Filter for the scalar-sigma-squared-times-R model*

Description

Runs the full forward filtering pass over nT time steps under the model where the right covariance of Y is $\sigma^2 R$ with $\sigma^2 \sim IG(n_0, d_0)$.

Usage

```
FF_sigma2R(
  Y,
  F_ls,
  G_ls,
  W_ls,
  V_ls,
  m0,
  M0,
  n0,
  D0,
  nT,
  R,
  delta = 1,
  verbose = FALSE
)
```

Arguments

Y	List of length nT. Each element is the $N \times S$ data matrix at the corresponding time step.
F_ls	Covariate matrix or list of matrices (see FF).
G_ls	State transition matrix or list (see FF).
W_ls	State noise left-covariance matrix or list (see FF).
V_ls	Observation noise left-covariance matrix or list (see FF).
m0	Numeric matrix. Prior state mean ($p \times S$).
M0	Numeric matrix. Prior state left-covariance ($p \times p$).
n0	Numeric scalar. Prior shape of σ^2.
D0	Numeric scalar. Prior rate of σ^2.
nT	Integer. Number of time steps.
R	Numeric matrix. Fixed spatial correlation matrix ($S \times S$). Its Cholesky inverse is used internally.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list of length nT + 1 (same structure as FF) with an additional scalar Dt in place of a matrix.

See Also

FF_1step_R_sigma2R, FFBS_sigma2R

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
R <- 0.5^abs(outer(1:3, 1:3, "-"))
res_ff <- FF_sigma2R(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
                     n0 = 3, D0 = 1, nT = 4, R = R)
c(nt = res_ff$T4$nt, Dt = res_ff$T4$Dt)  # IG posterior of sigma^2
```

FFBS *Forward Filter Backward Sampler (MNIW model)*

Description

Runs the complete FFBS algorithm under the Matrix Normal Inverse Wishart (MNIW) model: first applies the forward filter (FF) and then the backward sampler (BS).

Usage

```
FFBS(Y, F_ls, G_ls, W_ls, V_ls, m0, M0, n0, D0, nT, delta = 1, verbose = FALSE)
```

Arguments

Y	List of length nT. Each element is the $N \times q$ data matrix at the corresponding time step.
F_ls	Covariate matrix or list of matrices (see FF).
G_ls	State transition matrix or list (see FF).
W_ls	State noise left-covariance matrix or list (see FF).
V_ls	Observation noise left-covariance matrix or list (see FF).
m0	Numeric matrix. Prior state mean ($p \times q$).
M0	Numeric matrix. Prior state left-covariance ($p \times p$).
n0	Numeric scalar. Prior degrees of freedom of the inverse-Wishart.
D0	Numeric matrix. Prior scale matrix of the inverse-Wishart.
nT	Integer. Number of time steps.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list with:

ff Output of FF: filtered distributions for each time step.

bs Output of BS: smoothed state means (st) and left-covariances (St).

See Also

FF, BS, FFBS_sampling

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
            par$n0, par$D0, nT = 4)
res$bs$st[, , 1]  # smoothed state mean at time 1
# The forward part equals a separate call of FF
all.equal(res$ff, FF(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0,
                     par$M0, par$n0, par$D0, nT = 4))
```

FFBS_I *Forward Filter Backward Sampler (identity right-covariance)*

Description

Runs the complete FFBS algorithm with the right covariance fixed at the identity matrix (no inverse-Wishart update for Σ).

Usage

```
FFBS_I(Y, F_ls, G_ls, W_ls, V_ls, m0, M0, nT, delta = 1, verbose = FALSE)
```

Arguments

Y	List of length nT. Data matrices.
F_ls	Covariate matrix or list (see FF).
G_ls	State transition matrix or list (see FF).
W_ls	State noise left-covariance matrix or list (see FF).
V_ls	Observation noise left-covariance matrix or list (see FF).
m0	Numeric matrix. Prior state mean.
M0	Numeric matrix. Prior state left-covariance.
nT	Integer. Number of time steps.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list with `ff` (output of `FF_I`) and `bs` (output of `BS`).

See Also

`FF_I`, `BS`, `FFBS_sampling_I`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS_I(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
              nT = 4)
res$bs$st[, , 1]
all.equal(res$ff, FF_I(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0,
                       par$M0, nT = 4))
```

FFBS_predict_bigdata_R

Out-of-core exact prediction

Description

Predicts simulator output at new inputs from the smoothed coefficients written by `BS_bigdata_R()`, streaming one block of data at a time. For each block it computes the same mean as `FFBS_predict_exact()`: $F_{new}\hat{\Theta} + J^\top V^{-1}(Y - F\hat{\Theta})$, with the exponential input kernel. Every block must contain all training runs (`bnrow = fnrow` in `FF_bigdata_R()`), as in `emulator_learn()`.

Usage

```
FFBS_predict_bigdata_R(
  Y_ls,
  F_ls,
  F_new_ls,
  input,
  input_new,
  bs_files,
  nT,
  fnrow,
  fncol,
  bncol,
  phi_para,
  gp_sigma2 = 1.1,
  gp_tau2 = 1e-04,
  path_out,
  verbose = FALSE
)
```

Arguments

Y_ls, F_ls	The response and design file paths given to FF_bigdata_R().
F_new_ls	Design file path(s) for the new inputs, laid out like F_ls: one row per new input, fncol lag-1 columns and, for AR(2) designs, fncol lag-2 columns after them. One path or one per time step.
input	Matrix of training inputs, one row per run (fnrow rows).
input_new	Matrix (or vector, for one input) of new inputs.
bs_files	The list returned by BS_bigdata_R().
nT, fnrow, fncol, bncol	As given to FF_bigdata_R().
phi_para, gp_sigma2, gp_tau2	Range, amplitude and nugget of the exponential input kernel, as in FFBS_predict_exact().
path_out	Directory where the prediction files are written.
verbose	Logical. If TRUE, reports progress.

Value

A character vector with one CSV path per time step; each file holds the predicted means, one row per new input and one column per site.

See Also

FF_bigdata_R(), BS_bigdata_R(), FFBS_predict_exact()

Examples

```
set.seed(1)
dir <- file.path(tempdir(), "pred_big")
dir.create(dir, showWarnings = FALSE)
csv <- function(x, name) {
  path <- file.path(dir, name)
  utils::write.csv(x, path, row.names = FALSE)
  path
}
# 5 training runs and 2 new inputs; 4 sites in two blocks; 3 times
input <- matrix(runif(10), 5, 2)
input_new <- matrix(runif(4), 2, 2)
Y <- lapply(1:3, function(t) matrix(rnorm(20, t), 5, 4))
F <- c(list(matrix(1, 5, 4)), Y[1:2])
V <- 1.1 * exp(-2 * as.matrix(dist(input))) + diag(1e-4, 5)
ff <- FF_bigdata_R(Y_ls = mapply(csv, Y, paste0("Y", 1:3, ".csv")),
  F_ls = mapply(csv, F, paste0("F", 1:3, ".csv")),
  G_ls = csv(diag(2), "G.csv"), W_ls = csv(diag(2), "W.csv"),
  V_ls = csv(V, "V.csv"), m0 = matrix(0, 2, 2), M0 = diag(2), n0 = 4,
  D0 = diag(2), nT = 3, fnrow = 5, fncol = 4, bnrow = 5, bncol = 2,
  path_out = dir)
bs <- BS_bigdata_R(ff, file.path(dir, "G.csv"), nT = 3, fnrow = 5,
  fncol = 4, bnrow = 5, bncol = 2, path_out = dir)
```

```
# Designs for the new inputs, e.g. built from their own lagged outputs
F_new <- c(list(matrix(1, 2, 4)), lapply(1:2, function(t) matrix(t, 2, 4)))
pred <- FFBS_predict_bigdata_R(file.path(dir, paste0("Y", 1:3, ".csv")),
  file.path(dir, paste0("F", 1:3, ".csv")),
  mapply(csv, F_new, paste0("Fnew", 1:3, ".csv")), input, input_new, bs,
  nT = 3, fnrow = 5, fncol = 4, bncol = 2, phi_para = 2, path_out = dir)
utils::read.csv(pred[3])  # predicted means at time 3
unlink(dir, recursive = TRUE)
```

`FFBS_predict_exact` *Exact posterior predictive mean using FFBS smoothed states (MNIW model)*

Description

Computes the analytical posterior predictive mean at new spatial locations using the smoothed state means s_t from `FFBS` and an exponential GP kernel for the cross-covariance.

Usage

```
FFBS_predict_exact(
  Y,
  para_ffbs = NULL,
  F_ls,
  F_new_ls,
  input,
  input_new,
  nT,
  phi_para,
  gp_sigma2 = 1.1,
  gp_tau2 = 10^(-4),
  delta = 1,
  theta = NULL,
  V_chol = NULL,
  residual_weights = NULL,
  return_covariance = FALSE
)
```

Arguments

`Y`	List of length `nT`. Observed data matrices.
`para_ffbs`	List. Output of `FFBS`, containing `ff` and `bs`.
`F_ls`	Covariate matrix or list for observed locations.
`F_new_ls`	Covariate matrix or list for new locations.
`input`	Numeric matrix. Coordinates of observed locations.
`input_new`	Numeric matrix or vector. Coordinates of new locations.

nT	Integer. Number of time steps.
phi_para	Numeric scalar. Range parameter of the exponential kernel.
gp_sigma2	Numeric scalar. Variance parameter of the GP kernel. Defaults to 1.1.
gp_tau2	Numeric scalar. Nugget variance of the GP kernel. Defaults to 1e-4.
delta	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to 1.
theta	Optional list of fixed coefficient matrices, one per episode.
V_chol	Optional cached upper Cholesky factor of the input covariance.
residual_weights	Optional cached inverse-covariance-weighted residuals for the supplied coefficient trajectory.
return_covariance	If TRUE, return a list with mean and input_covariance before scaling by the emulator variance and spatial covariance.

Value

A named list of length nT. Each element "T<t>" is a matrix of dimension c(N_new, q) with the posterior predictive mean at the new locations.

See Also

FFBS_predict_MC, gen_exp_kernel

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
            par$n0, par$D0, nT = 4)
x_new <- matrix(c(-0.5, 0.5))  # two new inputs
pred <- FFBS_predict_exact(dat$Y, res, F_ls = par$F0,
                           F_new_ls = cbind(1, x_new), input = par$F0[, 2],
                           input_new = x_new, nT = 4, phi_para = 1)
pred$T4  # predictive mean at the new inputs (2 x S)
```

FFBS_predict_MC *Monte Carlo prediction using FFBS output (MNIW model)*

Description

Estimates posterior predictive means at new spatial locations using Monte Carlo integration over the posterior samples of the state Θ_t. Uses an exponential GP kernel to compute the cross-covariance between observed and new locations.

Usage

```
FFBS_predict_MC(
  nsam,
  Y,
  res_ffbs,
  F_ls,
  F_new_ls,
  input,
  input_new,
  nT,
  phi_para,
  gp_sigma2 = 1.1,
  gp_tau2 = 1e-04,
  delta = 1
)
```

Arguments

`nsam`	Integer. Number of posterior samples to average over.
`Y`	List of length `nT`. Observed data matrices.
`res_ffbs`	List. Posterior samples of Θ_t, as returned by `FFBS_sampling`.
`F_ls`	Covariate matrix or list for observed locations.
`F_new_ls`	Covariate matrix or list for new (prediction) locations.
`input`	Numeric matrix. Coordinates of observed locations ($N \times d$).
`input_new`	Numeric matrix or vector. Coordinates of new locations.
`nT`	Integer. Number of time steps.
`phi_para`	Numeric scalar. Range parameter of the exponential kernel.
`gp_sigma2`	Numeric scalar. Variance parameter of the GP kernel. Defaults to `1.1`.
`gp_tau2`	Numeric scalar. Nugget variance of the GP kernel. Defaults to `1e-4`.
`delta`	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to `1`.

Value

A named list of length `nT`. Each element `"T<t>"` is an array of dimension `c(N_new, q, nsam)` with posterior predictive mean samples at the new locations.

See Also

`FFBS_predict_exact`, `gen_exp_kernel`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
            par$n0, par$D0, nT = 4)
sam <- FFBS_sampling(50, res, par$F0, par$G0, nT = 4, seed = 1)
x_new <- matrix(c(-0.5, 0.5))  # two new inputs
pred <- FFBS_predict_MC(50, dat$Y, sam, F_ls = par$F0,
                        F_new_ls = cbind(1, x_new), input = par$F0[, 2],
                        input_new = x_new, nT = 4, phi_para = 1)
dim(pred$T4)
apply(pred$T4, 1:2, mean)  # Monte Carlo predictive mean at time 4
```

`FFBS_sampling` *Draw posterior samples from FFBS output (MNIW model)*

Description

Given the filtered and smoothed distributions from `FFBS`, draws `nsam` posterior sample sets of the state matrices $\Theta_1, \ldots, \Theta_T$ and the covariance Σ using the MNIW sampler.

Usage

```
FFBS_sampling(
  nsam,
  para_ffbs,
  F_ls,
  G_ls,
  nT,
  delta = 1,
  verbose = FALSE,
  sampling = c("marginal", "joint"),
  seed = NULL,
  keep_draws = TRUE
)
```

Arguments

`nsam`	Integer. Number of posterior samples to draw.
`para_ffbs`	List. Output of `FFBS`, containing elements `ff` and `bs`.
`F_ls`	Covariate matrix or list of matrices (see `FF`).
`G_ls`	State transition matrix or list (see `FF`).
`nT`	Integer. Number of time steps.
`delta`	Discount factor. Must be 1: discounting is not implemented. Defaults to `1`.
`verbose`	Logical. If `TRUE`, emits progress messages. Defaults to `FALSE`.

sampling	"marginal" (default) samples the smoothed marginal at each episode conditional on the same covariance draw. "joint" samples backward conditionals to obtain coherent trajectories.
seed	Optional local RNG seed; NULL follows the session RNG.
keep_draws	If TRUE (default), store every episode's coefficient draws. If FALSE (marginal sampling only), store the random-number state at the start of each episode's draws instead; spDBL's prediction and calibration functions then regenerate identical draws on demand, which cuts the memory of large emulators. Indexing the returned draws by episode ([[e]]) then fails.

Value

A named list of length nT + 1. Elements "T1" through "T<nT>" are arrays of dimension c(p, q, nsam) containing posterior samples of Θ_t. The element Sigma is an array of dimension c(q, q, nsam) containing posterior samples of Σ. Attributes sampling_method and joint_trajectories identify the chosen law.

See Also

FFBS, MNIW_sampler

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
            par$n0, par$D0, nT = 4)
sam <- FFBS_sampling(500, res, par$F0, par$G0, nT = 4, seed = 1)
dim(sam$T2)
dim(sam$Sigma)
# Sample mean versus smoothed mean at time 2
round(apply(sam$T2, 1:2, mean) - res$bs$st[, , 2], 2)
```

FFBS_sampling_I *Draw posterior samples from FFBS output (identity right-covariance)*

Description

Draws nsam posterior samples of the state Θ_t from the smoothed distributions produced by FFBS_I, assuming an identity right-covariance matrix.

Usage

```
FFBS_sampling_I(nsam, para_ffbs, F_ls, G_ls, nT, delta = 1, verbose = FALSE)
```

Arguments

nsam	Integer. Number of posterior samples.
para_ffbs	List. Output of FFBS_I.
F_ls, G_ls, delta	Not used: each draw comes from the smoothed marginal at its own time. Kept for compatibility with the other samplers.
nT	Integer. Number of time steps.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.

Value

A named list of length nT. Each element "T<t>" is an array of dimension c(p, q, nsam) with posterior samples of the state at time t.

See Also

FFBS_I

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS_I(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
              nT = 4)
sam <- FFBS_sampling_I(500, res, par$F0, par$G0, nT = 4)
dim(sam$T2)
round(apply(sam$T2, 1:2, mean) - res$bs$st[, , 2], 2)
```

FFBS_sampling_sigma2R *Draw posterior samples from FFBS output (scalar sigma-squared-times-R model)*

Description

Draws nsam posterior samples of the state Θ_t and the scalar variance σ^2 using the MNIG sampler, given the FFBS output under the scalar-sigma model.

Usage

```
FFBS_sampling_sigma2R(
  nsam,
  para_ffbs,
  F_ls,
  G_ls,
  nT,
  R,
```

```
  delta = 1,
  verbose = FALSE,
  seed = NULL,
  sampling = c("marginal", "joint"),
  keep_draws = TRUE
)
```

Arguments

nsam	Integer. Number of posterior samples.
para_ffbs	List. Output of FFBS_sigma2R.
F_ls	Covariate matrix or list (see FF).
G_ls	State transition matrix or list (see FF).
nT	Integer. Number of time steps.
R	Numeric matrix. Fixed spatial correlation matrix.
delta	Discount factor. Must be 1: discounting is not implemented. Defaults to 1.
verbose	Logical. If TRUE, emits progress messages. Defaults to FALSE.
seed	Optional local seed. NULL uses the session RNG.
sampling	"marginal" (default) draws smoothed episode marginals conditional on a shared variance. "joint" preserves cross-episode dependence by sampling backward conditionals.
keep_draws	If TRUE (default), store every episode's coefficient draws. If FALSE (marginal sampling only), store the random-number state at the start of each episode's draws instead; spDBL's prediction and calibration functions then regenerate identical draws on demand, which cuts the memory of large emulators. Indexing the returned draws by episode ([[e]]) then fails.

Value

A named list of length nT + 1. Elements "T1" through "T<nT>" are arrays of dimension c(p, q, nsam) with state samples. The element Sigma is a vector of length nsam with samples of σ^2. Attributes sampling_method and joint_trajectories identify the chosen law.

See Also

FFBS_sigma2R, MNIG_sampler

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
R <- 0.5^abs(outer(1:3, 1:3, "-"))
res <- FFBS_sigma2R(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
                    n0 = 3, D0 = 1, nT = 4, R = R)
sam <- FFBS_sampling_sigma2R(500, res, par$F0, par$G0, nT = 4, R = R,
                             seed = 1)
```

```
dim(sam$T2)
summary(sam$Sigma)  # draws of sigma^2
round(apply(sam$T2, 1:2, mean) - res$bs$st[, , 2], 2)
```

`FFBS_sigma2R` *Forward Filter Backward Sampler (scalar sigma-squared-times-R model)*

Description

Runs the complete FFBS algorithm under the model where the right covariance is $\sigma^2 R$ with $\sigma^2 \sim IG(n_0, d_0)$.

Usage

```
FFBS_sigma2R(
  Y,
  F_ls,
  G_ls,
  W_ls,
  V_ls,
  m0,
  M0,
  n0,
  D0,
  nT,
  R,
  delta = 1,
  verbose = FALSE
)
```

Arguments

`Y`	List of length `nT`. Data matrices.
`F_ls`	Covariate matrix or list (see `FF`).
`G_ls`	State transition matrix or list (see `FF`).
`W_ls`	State noise left-covariance matrix or list (see `FF`).
`V_ls`	Observation noise left-covariance matrix or list (see `FF`).
`m0`	Numeric matrix. Prior state mean.
`M0`	Numeric matrix. Prior state left-covariance.
`n0`	Numeric scalar. Prior shape of σ^2.
`D0`	Numeric scalar. Prior rate of σ^2.
`nT`	Integer. Number of time steps.
`R`	Numeric matrix. Fixed spatial correlation matrix.
`delta`	Discount factor. Not implemented: values other than 1 give a warning and are ignored. Defaults to `1`.
`verbose`	Logical. If `TRUE`, emits progress messages. Defaults to `FALSE`.

Value

A named list with `ff` (output of `FF_sigma2R`) and `bs` (output of `BS`).

See Also

`FF_sigma2R`, `BS`, `FFBS_sampling_sigma2R`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
R <- 0.5^abs(outer(1:3, 1:3, "-"))
res <- FFBS_sigma2R(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
                    n0 = 3, D0 = 1, nT = 4, R = R)
res$bs$st[, , 1]
all.equal(res$ff, FF_sigma2R(dat$Y, par$F0, par$G0, par$W0, par$V0,
                             par$m0, par$M0, 3, 1, nT = 4, R = R))
```

---

gen_calibrate_data *Generate synthetic calibration data with correlated discrepancy*

Description

Simulates observed calibration data z_t by adding a spatially correlated random discrepancy u_t and an independent noise term to the computer model output y_t. The variance τ_t^2 decays over time via the discount factor b.

Usage

```
gen_calibrate_data(ycal_mat, para_gen_cal, U_gen)
```

Arguments

`ycal_mat`	Numeric matrix of dimension `c(t_cal, s_cal)`. Computer model output at calibration locations and time steps.
`para_gen_cal`	Named list with elements:
	b Numeric scalar. Discount factor in $(0, 1)$.
	n0 Numeric scalar. Prior shape of τ_0^2.
	d0 Numeric scalar. Prior scale of τ_0^2.
	u0 Numeric vector of length `s_cal`. Initial discrepancy.
`U_gen`	Numeric matrix. Spatial covariance matrix for the discrepancy (`s_cal` x `s_cal`).

Value

A named list with:

u Matrix of dimension `c(t_cal, s_cal)` of discrepancy realisations.

z Matrix of dimension `c(t_cal, s_cal)` of simulated observations.

tau2 Numeric vector of length `t_cal` with sampled variances.

See Also

`gen_prior_u_tau2`

Examples

```
set.seed(1)
ycal <- matrix(rnorm(3 * 4), nrow = 3, ncol = 4)
para <- list(b = 0.9, n0 = 20, d0 = 2, u0 = rep(0, 4))
sim <- gen_calibrate_data(ycal, para, U_gen = 0.5 * diag(4))
names(sim)
dim(sim$z)
sim$tau2
```

---

gen_calibrate_data_uncorr

Generate synthetic calibration data with uncorrelated discrepancy

---

Description

Like `gen_calibrate_data` but uses independent (uncorrelated) discrepancy increments drawn from a univariate normal with variance $\tau_t^2 \cdot U_{11}$.

Usage

```
gen_calibrate_data_uncorr(ycal_mat, para_gen_cal, U_gen)
```

Arguments

`ycal_mat`	Numeric matrix. Computer model output (`t_cal` x `s_cal`).
`para_gen_cal`	Named list (same structure as in `gen_calibrate_data`).
`U_gen`	Numeric matrix. Only the `[1,1]` entry is used as the marginal variance scale.

Value

A named list with:

z Matrix of dimension `c(t_cal, s_cal)` of simulated observations.

tau2 Numeric vector of length `t_cal`.

See Also

gen_calibrate_data

Examples

```
set.seed(1)
ycal <- matrix(rnorm(3 * 4), nrow = 3, ncol = 4)
para <- list(b = 0.9, n0 = 20, d0 = 2, u0 = rep(0, 4))
sim <- gen_calibrate_data_uncorr(ycal, para, U_gen = 0.5 * diag(4))
names(sim)
dim(sim$z)
sim$tau2
```

---

gen_exp_kernel *Compute an exponential GP kernel matrix*

Description

Evaluates the exponential kernel $k(s, s') = \sigma^2 \exp(-\phi \|s - s'\|) + \tau^2 \mathbf{1}_{s=s'}$ for all pairs of rows in `loc`.

Usage

```
gen_exp_kernel(loc, phi, sigma2 = 1, tau2 = 0)
```

Arguments

`loc`	Numeric matrix. Spatial locations ($n \times d$).
`phi`	Numeric scalar. Range (decay) parameter.
`sigma2`	Numeric scalar. Marginal variance. Defaults to `1`.
`tau2`	Numeric scalar. Nugget variance. Defaults to `0`.

Value

A symmetric positive definite matrix of dimension `c(n, n)`.

See Also

gen_gp_kernel, gen_expsq_kernel

Examples

```
loc <- cbind(c(0, 1, 0), c(0, 0, 2))
K <- gen_exp_kernel(loc, phi = 0.5)
round(K, 3)
all.equal(K[1, 3], exp(-0.5 * 2))
```

gen_expsq_kernel *Compute a squared-exponential (Gaussian) GP kernel matrix*

Description

Evaluates the squared-exponential kernel $k(s, s') = \sigma^2 \exp(-\phi \|s - s'\|^2) + \tau^2 \mathbf{1}_{s=s'}$ for all pairs of rows in `loc`.

Usage

```
gen_expsq_kernel(loc, phi, sigma2 = 1, tau2 = 0)
```

Arguments

`loc`	Numeric matrix. Spatial locations ($n \times d$).
`phi`	Numeric scalar. Inverse length-scale parameter.
`sigma2`	Numeric scalar. Marginal variance. Defaults to `1`.
`tau2`	Numeric scalar. Nugget variance. Defaults to `0`.

Value

A symmetric positive definite matrix of dimension `c(n, n)`.

See Also

`gen_gp_kernel`, `gen_exp_kernel`

Examples

```
loc <- cbind(c(0, 1, 0), c(0, 0, 2))
K <- gen_expsq_kernel(loc, phi = 0.5, tau2 = 0.01)
round(K, 3)
all.equal(K[1, 3], exp(-0.5 * 2^2))
```

gen_F_ls_AR1 *Build AR(1) covariate list from a list of response matrices*

Description

Constructs the time-varying covariate list F_t for a first-order autoregressive model: $F_t = Y_{t-1}$. No earlier output exists at $t = 1$, so F_1 is a matrix of ones (a per-site intercept). The design is causal and deterministic: F_t uses only outputs before time t. (Versions up to 1.1.1 used $F_1 = Y_1$, the response itself.)

Usage

```
gen_F_ls_AR1(Y, nT)
```

Arguments

Y	List of length nT. Each element is the $N \times S$ data matrix at time t.
nT	Integer. Number of time steps.

Value

A list of length nT where element t is the covariate matrix F_t.

See Also

gen_F_ls_AR2, gen_F_ls_AR1_EP

Examples

```
Y <- lapply(1:4, function(t) matrix(t, 2, 3))
F_ls <- gen_F_ls_AR1(Y, nT = 4)
length(F_ls)
sapply(F_ls, function(Ft) Ft[1, 1])  # ones at t = 1, then lags: 1 1 2 3
```

---

gen_F_ls_AR1_EP *Build AR(1) covariate list for the episode-block model*

Description

Generates a flattened covariate list for the episode-partition (EP) model with AR(1) lags. The output list is indexed by $(t-1) \times n_b + j$, where j is the block index and t is the time index.

Usage

```
gen_F_ls_AR1_EP(Y, nT, n_b, ind)
```

Arguments

Y	List of length nT. Each element is the full spatial data matrix at time t.
nT	Integer. Number of time steps.
n_b	Integer. Number of spatial blocks.
ind	Data frame of block indices as returned by generate_grid.

Value

A list of length nT * n_b with block-wise AR(1) covariate matrices.

See Also

gen_F_ls_AR1, gen_F_ls_AR2_EP

Examples

```
# Three time steps on six sites, split into three blocks of two sites
Y <- lapply(1:3, function(t) matrix(10 * t + 1:6, nrow = 1))
ind <- generate_grid(1, 6, 1, 2)
F_ep <- gen_F_ls_AR1_EP(Y, nT = 3, n_b = 3, ind = ind)
length(F_ep)  # nT * n_b
F_ep[[5]]     # time 2, block 2: sites 3-4 at time 1
```

---

gen_F_ls_AR2 *Build AR(2) covariate list from a list of response matrices*

Description

Constructs the time-varying covariate list F_t for a second-order autoregressive model by column-binding the two most recent lags: $F_t = [Y_{t-2}, Y_{t-1}]$. A lag that does not exist yet is replaced by a matrix of ones (a per-site intercept): $F_1 = [1, 1]$ and $F_2 = [1, Y_1]$. The design is causal and deterministic. (Versions up to 1.1.1 used Y_1 plus random noise at $t = 1$, and the future value Y_3 at $t = 2$.)

Usage

```
gen_F_ls_AR2(Y, nT)
```

Arguments

Y	List of length nT. Each element is the $N \times S$ data matrix at time t.
nT	Integer. Number of time steps.

Value

A list of length nT where element t is the $N \times 2S$ covariate matrix F_t.

See Also

gen_F_ls_AR1, gen_F_ls_AR2_EP

Examples

```
Y <- lapply(1:4, function(t) matrix(t, 2, 3))
F_ls <- gen_F_ls_AR2(Y, nT = 4)
dim(F_ls[[4]])
F_ls[[2]][1, ]  # ones (no lag 2 yet), then lag 1
F_ls[[4]][1, ]  # lags 2 and 3
```

`gen_F_ls_AR2_EP` *Build AR(2) covariate list for the episode-block model*

Description

Generates a flattened covariate list for the episode-partition (EP) model with AR(2) lags. Both lag-1 and lag-2 column blocks are extracted and column-bound for each spatial block.

Usage

```
gen_F_ls_AR2_EP(Y, nT, n_b, ind)
```

Arguments

`Y`	List of length `nT`. Each element is the full spatial data matrix at time t.
`nT`	Integer. Number of time steps.
`n_b`	Integer. Number of spatial blocks.
`ind`	Data frame of block indices as returned by `generate_grid`.

Value

A list of length `nT * n_b` with block-wise AR(2) covariate matrices (each of width $2 \times$ bncol).

See Also

`gen_F_ls_AR2`, `gen_F_ls_AR1_EP`

Examples

```
set.seed(1)
Y <- lapply(1:3, function(t) matrix(10 * t + 1:6, nrow = 1))
ind <- generate_grid(1, 6, 1, 2)
F_ep <- gen_F_ls_AR2_EP(Y, nT = 3, n_b = 3, ind = ind)
length(F_ep)
F_ep[[8]]  # time 3, block 2: sites 3-4 at times 1 and 2
```

gen_ffbs_csv *Generate synthetic FFBS data and write to CSV files*

Description

Simulates data from the MNIW dynamic linear model for `nT` time steps and writes the observations $Y_1, \ldots, Y_T$ and all model parameters to CSV files in `path`.

Usage

```
gen_ffbs_csv(N, S, p, nT, path)
```

Arguments

`N`	Integer. Number of spatial locations (rows of Y_t).
`S`	Integer. Number of response variables (columns of Y_t).
`p`	Integer. Dimension of the state vector (must be ≥ 2).
`nT`	Integer. Number of time steps.
`path`	Character. Directory path (with trailing `/`) where CSV files are written.

Value

Invisibly returns `NULL`. Files written: `m0.csv`, `MM0.csv`, `G0.csv`, `F0.csv`, `Sigma.csv`, `RSigma.csv`, `V0.csv`, `RV0.csv`, `W0.csv`, and `Y1.csv` through `Y<nT>.csv`.

See Also

gen_ffbs_data

Examples

```
set.seed(1)
out_dir <- file.path(tempdir(), "ffbs_csv")
dir.create(out_dir)
gen_ffbs_csv(N = 10, S = 3, p = 2, nT = 3, path = paste0(out_dir, "/"))
list.files(out_dir)
dim(read.csv(file.path(out_dir, "Y1.csv")))
unlink(out_dir, recursive = TRUE)
```

`gen_ffbs_data` *Generate synthetic FFBS data in memory*

Description

Simulates data from the MNIW dynamic linear model for `nT` time steps and returns the generated observations and model parameters as in-memory objects.

Usage

```
gen_ffbs_data(N, S, p, nT)
```

Arguments

`N`	Integer. Number of spatial locations (rows of Y_t).
`S`	Integer. Number of response variables (columns of Y_t).
`p`	Integer. Dimension of the state vector (must be ≥ 2).
`nT`	Integer. Number of time steps.

Value

A named list with:

Y Named list of length `nT` (`"Y1"`, ..., `"Y<nT>"`), each element an $N \times S$ data matrix.

para Named list of model parameters: `loc`, `dist`, `m0`, `M0`, `G0`, `F0`, `Sigma`, `RSigma`, `V0`, `RV0`, `W0`, `n0`, `D0`.

See Also

`gen_ffbs_csv`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 10, S = 3, p = 2, nT = 4)
names(dat$Y)
dim(dat$Y$Y1)
dim(dat$para$V0)
```

gen_gp_kernel *Compute a Gaussian Process covariance kernel matrix*

Description

Evaluates the isotropic GP exponential kernel $k(s, s') = \sigma^2 \exp(-\phi\|s - s'\|) + \tau^2 \mathbf{1}_{s=s'}$ for all pairs of rows in `loc`.

Usage

```
gen_gp_kernel(loc, phi, sigma2, tau2)
```

Arguments

`loc`	Numeric matrix. Spatial locations ($n \times d$).
`phi`	Numeric scalar. Range (decay) parameter of the exponential kernel.
`sigma2`	Numeric scalar. Marginal variance parameter.
`tau2`	Numeric scalar. Nugget (white-noise) variance.

Value

A symmetric positive definite matrix of dimension `c(n, n)`.

See Also

`gen_exp_kernel`, `gen_expsq_kernel`

Examples

```
loc <- matrix(c(0, 1, 3), ncol = 1)
K <- gen_gp_kernel(loc, phi = 1, sigma2 = 2, tau2 = 0.1)
K
all.equal(K[1, 2], 2 * exp(-1))
```

gen_Jt *Compute the cross-covariance matrix between observed and new locations*

Description

Evaluates the exponential GP kernel between observed and new locations, giving the $N \times N_{\text{new}}$ cross-covariance block J_t. The nugget does not enter a cross-covariance, even when a new location coincides with an observed one; this matches `FFBS_predict_exact`.

Usage

```
gen_Jt(input, input_new, phi_para, gp_sigma2 = 1.1, gp_tau2 = 10^(-4))
```

Arguments

input	Numeric matrix. Coordinates of observed locations ($N \times d$).
input_new	Numeric matrix or vector. Coordinates of new locations ($N_{\text{new}} \times d$).
phi_para	Numeric scalar. Range parameter of the exponential kernel.
gp_sigma2	Numeric scalar. Marginal variance of the kernel. Defaults to 1.1.
gp_tau2	Numeric scalar. Nugget variance. Not used; kept for compatibility. Defaults to 1e-4.

Value

Numeric matrix of dimension c(N, N_new).

See Also

gen_exp_kernel

Examples

```
set.seed(1)
input <- matrix(runif(10), nrow = 5, ncol = 2)
J <- gen_Jt(input, input_new = c(0.5, 0.5), phi_para = 2)
length(J)
# At an observed input the cross-covariance equals gp_sigma2
gen_Jt(input, input[1, ], phi_para = 2)[1]
```

---

gen_pd_matrix *Generate a random positive definite matrix*

Description

Constructs a positive definite matrix of dimension dim by forming $A^\top A$ from a random normal matrix A and then rescaling so the maximum entry is 1.

Usage

```
gen_pd_matrix(dim)
```

Arguments

dim	Integer. Number of rows (and columns) of the output matrix.

Value

A symmetric positive definite numeric matrix of dimension `c(dim, dim)`.

Examples

```
set.seed(1)
S <- gen_pd_matrix(4)
isSymmetric(S)
max(S)
all(eigen(S, symmetric = TRUE)$values > 0)
```

---

gen_pde *Simulate a spatially extended SIR PDE model*

Description

Numerically integrates the spatial SIR ODE (`SIR`) on a two-dimensional grid using `deSolve::ode.2D` and returns the log(1 + I) values of the infected compartment.

Usage

```
gen_pde(eta, Nx, Ny, N, nT_ori)
```

Arguments

`eta`	Numeric vector of length 5. PDE parameters: transmission rate, recovery rate, and diffusion coefficients for S, I, R.
`Nx`	Integer. Number of grid cells in the x-direction (rows).
`Ny`	Integer. Number of grid cells in the y-direction (columns).
`N`	Numeric. Total population size.
`nT_ori`	Integer. Number of time steps to simulate (times $0, 1, \ldots,$ nT_ori $- 1$).

Value

Numeric matrix of dimension `c(nT_ori, Nx * Ny)` containing $\log(1 + I_t)$ for each time step and grid cell.

See Also

`SIR`, `update_y_eta`

Examples

```
if (requireNamespace("deSolve", quietly = TRUE) &&
    requireNamespace("ReacTran", quietly = TRUE)) {
y <- gen_pde(eta = c(0.5, 0.1, 0.05, 0.05, 0.05), Nx = 4, Ny = 4, N = 100,
             nT_ori = 3)
dim(y)  # nT_ori x (Nx * Ny)
round(rowSums(y), 3)  # total log(1 + I) grows over time
}
```

`gen_prior_u_tau2` *Sample prior discrepancy trajectory and variance sequence*

Description

Simulates a prior trajectory $u_1, \ldots, u_T$ of the spatial discrepancy field and the associated inverse-gamma variance sequence $\tau_1^2, \ldots, \tau_T^2$ from the dynamic calibration model.

Usage

```
gen_prior_u_tau2(n0, d0, b, m0, M0, U, nT_cal)
```

Arguments

`n0`	Numeric scalar. Prior shape of τ_0^2.
`d0`	Numeric scalar. Prior scale of τ_0^2.
`b`	Numeric scalar. Discount factor in $(0, 1)$.
`m0`	Numeric vector. Prior mean of the initial discrepancy u_0.
`M0`	Numeric matrix. Prior covariance of u_0.
`U`	Numeric matrix. Spatial covariance matrix for the discrepancy increments.
`nT_cal`	Integer. Number of calibration time steps.

Value

A named list with:

u0 Numeric vector. Sampled initial discrepancy.

u Matrix of dimension `c(nT_cal, s_cal)` of discrepancy realisations.

tau2_0 Numeric scalar. Sampled τ_0^2.

tau2 Numeric vector of length `nT_cal`.

See Also

`gen_calibrate_data`

Examples

```
set.seed(1)
prior <- gen_prior_u_tau2(n0 = 20, d0 = 2, b = 0.95, m0 = rep(0, 4),
                          M0 = diag(4), U = 0.5 * diag(4), nT_cal = 3)
names(prior)
dim(prior$u)
prior$tau2
```

---

gen_ran_matrix *Generate a random matrix with entries scaled to* $[-1, 1]$

Description

Draws a matrix of independent standard normal entries and divides by the largest absolute entry, so all entries lie in $[-1, 1]$. (Versions up to 1.1.1 divided by the absolute value of the largest entry, so entries could fall below -1.)

Usage

```
gen_ran_matrix(nrow, ncol)
```

Arguments

`nrow`	Integer. Number of rows.
`ncol`	Integer. Number of columns.

Value

A numeric matrix of dimension `c(nrow, ncol)`.

Examples

```
set.seed(1)
M <- gen_ran_matrix(3, 4)
dim(M)
max(abs(M))  # the largest absolute entry is scaled to 1
```

generate.grid.exact *Generate an exact block grid analytically*

Description

Computes block indices when block dimensions divide file dimensions exactly. Uses vectorised operations rather than a loop, making it fast for large grids.

Usage

```
generate.grid.exact(fnrow, fncol, bnrow, bncol, traversal.mode)
```

Arguments

fnrow	Integer. Number of rows in the file or matrix.
fncol	Integer. Number of columns in the file or matrix.
bnrow	Integer. Number of rows per block. Must divide fnrow exactly.
bncol	Integer. Number of columns per block. Must divide fncol exactly.
traversal.mode	Character. "rowsnake" or "lr". See generate_grid for details.

Value

A data.frame with columns block_no, U, D, L, R. See generate_grid for column descriptions.

See Also

generate_grid, generate.grid.rowsnake, generate.grid.lr

Examples

```
# Block sizes divide the grid exactly
generate.grid.exact(4, 6, 2, 3, traversal.mode = "rowsnake")
generate.grid.exact(4, 6, 2, 3, traversal.mode = "lr")
```

generate.grid.lr *Generate a flexible block grid with left-to-right traversal*

Description

Iteratively computes block indices using a strict left-to-right traversal order across all rows. Handles cases where block dimensions do not evenly divide file dimensions by allowing the last block in each row or column to be smaller.

Usage

```
generate.grid.lr(fnrow, fncol, bnrow, bncol)
```

Arguments

fnrow	Integer. Number of rows in the file or matrix.
fncol	Integer. Number of columns in the file or matrix.
bnrow	Integer. Number of rows per block.
bncol	Integer. Number of columns per block.

Value

A data.frame with columns block_no, U, D, L, R. See generate_grid for column descriptions.

See Also

generate_grid, generate.grid.exact, generate.grid.rowsnake

Examples

```
# Last block row and column are smaller; every block row runs left to right
generate.grid.lr(5, 7, 2, 3)
```

---

generate.grid.rowsnake

Generate a flexible block grid with snake traversal

Description

Iteratively computes block indices using a snake (boustrophedon) traversal order. Handles cases where block dimensions do not evenly divide file dimensions by allowing the last block in each row or column to be smaller.

Usage

```
generate.grid.rowsnake(fnrow, fncol, bnrow, bncol)
```

Arguments

fnrow	Integer. Number of rows in the file or matrix.
fncol	Integer. Number of columns in the file or matrix.
bnrow	Integer. Number of rows per block.
bncol	Integer. Number of columns per block.

Value

A `data.frame` with columns `block_no`, `U`, `D`, `L`, `R`. See `generate_grid` for column descriptions.

See Also

`generate_grid`, `generate.grid.exact`, `generate.grid.lr`

Examples

```
# Last block row and column are smaller; even block rows run right to left
generate.grid.rowsnake(5, 7, 2, 3)
```

`generate_grid` *Generate block indices for big data grid traversal*

Description

Partitions a matrix or file into rectangular blocks and returns a data frame of block indices numbered from 1 to K (total number of blocks). Supports exact and flexible partitioning, and multiple traversal orders.

Usage

```
generate_grid(
  fnrow,
  fncol,
  bnrow,
  bncol,
  traversal.mode = "rowsnake",
  is.flexible = NULL,
  is.flexible.row = FALSE,
  is.flexible.col = FALSE
)
```

Arguments

`fnrow`	Integer. Number of rows in the file or matrix.
`fncol`	Integer. Number of columns in the file or matrix.
`bnrow`	Integer. Number of rows per block.
`bncol`	Integer. Number of columns per block.
`traversal.mode`	Character. Traversal order for the grid. `"rowsnake"` uses alternating left-right traversal (snake/boustrophedon order); `"lr"` uses strict left-to-right traversal throughout. Defaults to `"rowsnake"`.

`is.flexible` Logical or `NULL`. If non-`NULL`, overrides both `is.flexible.row` and `is.flexible.col`. When `TRUE`, allows block dimensions that do not evenly divide the file dimensions. Defaults to `NULL`.

`is.flexible.row` Logical. If `FALSE`, an error is thrown when `bnrow` does not evenly divide `fnrow`. Defaults to `FALSE`.

`is.flexible.col` Logical. If `FALSE`, an error is thrown when `bncol` does not evenly divide `fncol`. Defaults to `FALSE`.

Value

A `data.frame` with columns:

block_no Block index from 1 to K.

U Upper (first) row index of the block.

D Lower (last) row index of the block.

L Left (first) column index of the block.

R Right (last) column index of the block.

Examples

```
# Exact partition with snake traversal
generate_grid(100, 100, 10, 10)

# Flexible partition (block size does not divide file size evenly)
generate_grid(105, 103, 10, 10, is.flexible = TRUE)
```

---

inv_chol *Invert a matrix via its Cholesky factorisation*

Description

Computes X^{-1} using `chol` followed by `chol2inv` for numerical stability.

Usage

```
inv_chol(X)
```

Arguments

`X` Numeric symmetric positive definite matrix.

Value

The inverse of `X`.

Examples

```
A <- matrix(c(4, 1, 1, 3), 2)
inv_chol(A)
all.equal(inv_chol(A), solve(A))
```

`lppd_id_1t` *One-step log posterior predictive density (identity right-covariance model)*

Description

Computes the log posterior predictive density of Y_t at a single time step under the model with an identity right-covariance matrix, using a matrix-normal density.

Usage

```
lppd_id_1t(Yt, Ft, Vt, st, St)
```

Arguments

`Yt`	Numeric matrix. Observed data at time t ($N \times S$).
`Ft`	Numeric matrix. Covariate matrix at time t ($N \times p$).
`Vt`	Numeric matrix. Observation noise left-covariance ($N \times N$).
`st`	Numeric matrix. Smoothed state mean at time t ($p \times S$).
`St`	Numeric matrix. Smoothed state left-covariance at time t ($p \times p$).

Value

Numeric scalar. Log posterior predictive density.

See Also

`lppd_IW_1t`, `lppd_IG_1t`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS_I(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
              nT = 4)
lppd_id_1t(dat$Y[[2]], par$F0, par$V0, st = res$bs$st[, , 2],
           St = res$bs$St[, , 2])
```

`lppd_IG_1t` *One-step log posterior predictive density (scalar sigma-squared-times-R model)*

Description

Computes the log posterior predictive density of Y_t at a single time step under the scalar-sigma model ($\sigma^2 \sim IG$), using `dMTig`.

Usage

```
lppd_IG_1t(Yt, Ft, Vt, st, St, nt, Dt, R)
```

Arguments

`Yt`	Numeric matrix. Observed data at time t ($N \times S$).
`Ft`	Numeric matrix. Covariate matrix at time t ($N \times p$).
`Vt`	Numeric matrix. Observation noise left-covariance ($N \times N$).
`st`	Numeric matrix. Smoothed state mean at time t ($p \times S$).
`St`	Numeric matrix. Smoothed state left-covariance at time t ($p \times p$).
`nt`	Numeric scalar. Filtered shape parameter at time t.
`Dt`	Numeric scalar. Filtered rate parameter at time t.
`R`	Numeric matrix. Fixed right correlation matrix ($S \times S$).

Value

Numeric scalar. Log posterior predictive density.

See Also

`dMTig`, `lppd_IW_1t`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
R <- 0.5^abs(outer(1:3, 1:3, "-"))
res <- FFBS_sigma2R(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
                    n0 = 3, D0 = 1, nT = 4, R = R)
lppd_IG_1t(dat$Y[[2]], par$F0, par$V0, st = res$bs$st[, , 2],
           St = res$bs$St[, , 2], nt = res$ff$T2$nt, Dt = res$ff$T2$Dt,
           R = R)
```

`lppd_IW_1t` *One-step log posterior predictive density (MNIW / inverse-Wishart model)*

Description

Computes the log posterior predictive density of Y_t at a single time step under the MNIW model, evaluated at the smoothed state parameters from `FFBS`.

Usage

```
lppd_IW_1t(Yt, Ft, Vt, st, St, nt, Dt)
```

Arguments

`Yt`	Numeric matrix. Observed data at time t ($N \times S$).
`Ft`	Numeric matrix. Covariate matrix at time t ($N \times p$).
`Vt`	Numeric matrix. Observation noise left-covariance ($N \times N$).
`st`	Numeric matrix. Smoothed state mean at time t ($p \times S$).
`St`	Numeric matrix. Smoothed state left-covariance at time t ($p \times p$).
`nt`	Numeric scalar. Filtered degrees of freedom at time t.
`Dt`	Numeric matrix. Filtered scale matrix at time t ($S \times S$).

Value

Numeric scalar. Log posterior predictive density.

See Also

`lppd_IG_1t`, `lppd_id_1t`

Examples

```
set.seed(1)
dat <- gen_ffbs_data(N = 5, S = 3, p = 2, nT = 4)
par <- dat$para
res <- FFBS(dat$Y, par$F0, par$G0, par$W0, par$V0, par$m0, par$M0,
            par$n0, par$D0, nT = 4)
lppd_IW_1t(dat$Y[[2]], par$F0, par$V0, st = res$bs$st[, , 2],
           St = res$bs$St[, , 2], nt = res$ff$T2$nt, Dt = res$ff$T2$Dt)
```

make_pds — *Force a matrix to be positive definite and symmetric*

Description

Symmetrises a matrix by averaging it with its transpose and, if the result is not positive definite, adds a small diagonal ridge.

Usage

```
make_pds(C, eps = 10^(-4))
```

Arguments

C	Numeric matrix.
eps	Numeric scalar. Size of the diagonal ridge added when C is not positive definite. Defaults to 1e-4.

Value

The corrected positive definite symmetric matrix.

See Also

check_pds

Examples

```
A <- matrix(c(2, 0.9, 1.1, 2), 2)
make_pds(A)  # averaged with its transpose
isSymmetric(make_pds(A))
```

MNIG_sampler — *Sample from the Matrix Normal Inverse Gamma (MNIG) distribution*

Description

Draws nsam posterior samples of the regression coefficient matrix B and the scalar variance σ^2 under the MNIG model where the right covariance of $B|\sigma^2$ is $\sigma^2 R$:

$$\sigma^2 \sim IG(v, S), \quad B|\sigma^2 \sim MN(C, V_b, \sigma^2 R).$$

Usage

```
MNIG_sampler(nsam, X, v, S, C, Vb, R)
```

Arguments

nsam	Integer. Number of samples to draw.
X	Numeric matrix. Covariate (design) matrix ($n \times p$).
v	Numeric scalar. Shape parameter of the inverse-gamma prior on σ^2.
S	Numeric scalar. Rate parameter of the inverse-gamma prior on σ^2.
C	Numeric matrix. Prior mean of $B\|\sigma^2$ ($p \times q$).
Vb	Numeric matrix. Prior row covariance of $B\|\sigma^2$ ($p \times p$, positive definite).
R	Numeric matrix. Fixed right correlation matrix ($q \times q$, positive definite).

Value

A named list with:

sigma2 Numeric vector of length nsam with samples of σ^2.

B Array of dimension c(p, q, nsam) with samples of B.

See Also

MNIW_sampler, FFBS_sampling_sigma2R

Examples

```
set.seed(1)
draws <- MNIG_sampler(nsam = 2000, X = matrix(1, 5, 1), v = 5, S = 2,
                      C = matrix(c(1, -1), nrow = 1), Vb = matrix(1), R = diag(2))
dim(draws$B)
mean(draws$sigma2)  # close to S / (v - 1) = 0.5
round(rowMeans(draws$B[1, , ]), 2)  # close to C
```

MNIW_R *MNIW posterior update*

Description

Computes the posterior parameters of the Matrix Normal Inverse Wishart (MNIW) model using crossprod for improved numerical efficiency.

Usage

```
MNIW_R(X, Y, newX = NULL, newY = NULL, H, v, S, C, Vb)
```

Arguments

X	Numeric matrix. Covariate (design) matrix of dimension $n \times p$.
Y	Numeric matrix. Response matrix of dimension $n \times q$.
newX	Unused legacy parameter. Defaults to NULL.
newY	Unused legacy parameter. Defaults to NULL.
H	Numeric matrix. Row covariance matrix of the error term ($n \times n$).
v	Numeric scalar. Prior degrees of freedom of the inverse-Wishart on Σ.
S	Numeric matrix. Prior scale matrix of the inverse-Wishart on Σ.
C	Numeric matrix. Prior mean matrix of $B\|\Sigma$ ($p \times q$).
Vb	Numeric matrix. Prior row covariance matrix of $B\|\Sigma$ ($p \times p$).

Details

The model is $Y = XB+E$, where $E \sim MN(0, H, \Sigma)$, $\Sigma \sim IW(v, S)$, and $B|\Sigma \sim MN(C, V_b, \Sigma)$.

Value

A named list with posterior parameters:

vnew Posterior degrees of freedom.

Snew Posterior scale matrix of the inverse-Wishart.

Cnew Posterior mean matrix of $B|\Sigma$.

Vbnew Posterior row covariance matrix of $B|\Sigma$.

See Also

MNIW_R_naiive

Examples

```
set.seed(1)
n <- 30
X <- cbind(1, rnorm(n))
B <- matrix(c(1, -1, 0.5, 2, 0, 1), nrow = 2)
Y <- X %*% B + matrix(rnorm(n * 3, sd = 0.1), n, 3)
post <- MNIW_R(X, Y, H = diag(n), v = 5, S = diag(3),
               C = matrix(0, 2, 3), Vb = diag(10, 2))
post$vnew
round(post$Cnew, 2)  # close to B
```

`MNIW_R_naiive` *Naive MNIW posterior update*

Description

Computes the posterior parameters of the Matrix Normal Inverse Wishart (MNIW) model using explicit matrix transposes. Equivalent to `MNIW_R` but without `crossprod` optimisations. Intended for reference and testing.

Usage

```
MNIW_R_naiive(X, Y, newX = NULL, newY = NULL, H, v, S, C, Vb)
```

Arguments

`X`	Numeric matrix. Covariate (design) matrix of dimension $n \times p$.
`Y`	Numeric matrix. Response matrix of dimension $n \times q$.
`newX`	Unused legacy parameter. Defaults to `NULL`.
`newY`	Unused legacy parameter. Defaults to `NULL`.
`H`	Numeric matrix. Row covariance matrix of the error term ($n \times n$).
`v`	Numeric scalar. Prior degrees of freedom of the inverse-Wishart on Σ.
`S`	Numeric matrix. Prior scale matrix of the inverse-Wishart on Σ.
`C`	Numeric matrix. Prior mean matrix of $B\|\Sigma$ ($p \times q$).
`Vb`	Numeric matrix. Prior row covariance matrix of $B\|\Sigma$ ($p \times p$).

Details

The model is $Y = XB+E$, where $E \sim MN(0, H, \Sigma)$, $\Sigma \sim IW(v, S)$, and $B|\Sigma \sim MN(C, V_b, \Sigma)$.

Value

A named list with posterior parameters:

vnew Posterior degrees of freedom.

Snew Posterior scale matrix of the inverse-Wishart.

Cnew Posterior mean matrix of $B|\Sigma$.

Vbnew Posterior row covariance matrix of $B|\Sigma$.

See Also

`MNIW_R`

Examples

```
set.seed(1)
n <- 30
X <- cbind(1, rnorm(n))
Y <- X %*% matrix(c(1, -1, 0.5, 2), 2) + matrix(rnorm(n * 2, sd = 0.1), n, 2)
prior <- list(H = diag(n), v = 4, S = diag(2), C = matrix(0, 2, 2), Vb = diag(10, 2))
post <- do.call(MNIW_R_naiive, c(list(X = X, Y = Y), prior))
round(post$Cnew, 2)
# Same result as the crossprod version
all.equal(post, do.call(MNIW_R, c(list(X = X, Y = Y), prior)))
```

`MNIW_sampler` *Sample from the Matrix Normal Inverse Wishart (MNIW) distribution*

Description

Draws `nsam` posterior samples of the regression coefficient matrix B and the covariance matrix Σ under the MNIW model:

$$Y = XB + E, \quad E \sim MN(0, H, \Sigma)$$
$$\Sigma \sim IW(v, S), \quad B|\Sigma \sim MN(C, V_b, \Sigma).$$

Usage

```
MNIW_sampler(nsam, X, v, S, C, Vb)
```

Arguments

`nsam`	Integer. Number of samples to draw.
`X`	Numeric matrix. Covariate (design) matrix ($n \times p$).
`v`	Numeric scalar. Degrees of freedom of the inverse-Wishart prior on Σ.
`S`	Numeric matrix. Scale matrix of the inverse-Wishart prior ($q \times q$, positive definite).
`C`	Numeric matrix. Prior mean of $B|\Sigma$ ($p \times q$).
`Vb`	Numeric matrix. Prior row covariance of $B|\Sigma$ ($p \times p$, positive definite).

Value

A named list with:

sigma Array of dimension `c(q, q, nsam)` with samples of Σ.

B Array of dimension `c(p, q, nsam)` with samples of B.

See Also

`MNIG_sampler`, `FFBS_sampling`

Examples

```
set.seed(1)
C <- matrix(c(1, -1), nrow = 1)
draws <- MNIW_sampler(nsam = 2000, X = matrix(1, 5, 1), v = 10, S = diag(2),
                      C = C, Vb = matrix(1))
dim(draws$B)
# E[Sigma] = S / (v - q - 1) = diag(2) / 7
round(apply(draws$sigma, 1:2, mean), 2)
round(rowMeans(draws$B[1, , ]), 2)  # close to C
```

---

plot.spDBL_calibration

Plot calibrated mechanistic parameter samples

Description

Plot calibrated mechanistic parameter samples

Usage

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

Arguments

x	Calibration result.
...	Graphical arguments passed to hist.

Value

Invisibly returns x.

Examples

```
data("dt_calibration")
d <- dt_calibration
em <- emulator_learn(d$training$input, d$training$responses, d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
                     seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
fit <- calibration_fit(cal, d$field$z, d$field$U,
                       locations = d$field$locations, draws = 1:2,
                       chains = 1, iter = 200, seed = 1)
plot(fit)
```

plot_panel_heatmap_9 *Plot a 3-by-3 panel of heatmaps across selected time stamps*

Description

Produces nine raster heatmaps arranged in a 3-by-3 grid, one for each time stamp in `tstamp`. Each panel shows the spatial field for a given input (row) index and time step.

Usage

```
plot_panel_heatmap_9(
  dat,
  input_num,
  tstamp,
  max_y,
  Nx,
  Ny,
  nT,
  filename = "plot_panel",
  savei = TRUE,
  col_bgr = spdbl_default_col_bgr(),
  path_fig = "."
)
```

Arguments

`dat`	List of length $\geq \max($`tstamp`$)$. Each element is a matrix of dimension `c(n_inputs, Nx * Ny)`.
`input_num`	Integer. Row index within each time-step matrix to plot.
`tstamp`	Integer vector of length 9. Time step indices to display.
`max_y`	Numeric scalar. Upper limit of the colour scale.
`Nx`	Integer. Number of grid points in the x-direction.
`Ny`	Integer. Number of grid points in the y-direction.
`nT`	Integer. Total number of time steps in `dat`.
`filename`	Character. Base name for the saved PNG (used when `savei = TRUE`). Defaults to `"plot_panel"`.
`savei`	Logical. Whether to save the panel to disk. Defaults to `TRUE`.
`col_bgr`	Character vector of colours for the heatmap gradient.
`path_fig`	Character. Directory path where the file is saved when `savei = TRUE`. Defaults to the current working directory.

Value

A list of nine `ggplot` objects (one per panel).

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggpubr", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE)) {
  data("dt_emulation")
  d <- dt_emulation
  p <- plot_panel_heatmap_9(d$dt_pde_test, input_num = 1,
                            tstamp = seq(2, 26, by = 3), max_y = 10,
                            Nx = 10, Ny = 10, nT = 26, savei = FALSE)
  length(p)
  p[[9]]
}
```

`plot_panel_heatmap_9_cal`

Plot a 3-by-3 panel of calibration heatmaps

Description

Produces nine raster heatmaps for calibration data arranged in a 3-by-3 grid, one for each time stamp in `tstamp`. Locations are specified via `loc_cal` rather than a regular grid.

Usage

```
plot_panel_heatmap_9_cal(
  dat,
  tstamp,
  max_y,
  loc_cal,
  Nx,
  Ny,
  filename = "plot_panel",
  savei = TRUE,
  col_bgr = spdbl_default_col_bgr(),
  path_fig = "."
)
```

Arguments

`dat`	Numeric matrix of dimension `c(nT, n_locations)`. Rows are time steps; columns are calibration locations.
`tstamp`	Integer vector of length 9. Time-step indices (rows of `dat`) to display.
`max_y`	Numeric scalar. Upper limit of the colour scale.
`loc_cal`	Data frame with columns `row` and `col` giving the spatial coordinates of calibration locations.
`Nx`	Integer. Number of grid points in the x-direction (used for reference only).

Ny	Integer. Number of grid points in the y-direction (used for reference only).
filename	Character. Base name for the saved PNG. Defaults to "plot_panel".
savei	Logical. Whether to save the panel to disk. Defaults to TRUE.
col_bgr	Character vector of colours for the heatmap gradient.
path_fig	Character. Directory path where the file is saved when savei = TRUE. Defaults to the current working directory.

Value

A list of nine ggplot objects.

See Also

plot_panel_heatmap_9_cal_nolab

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggpubr", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE)) {
  data("dt_calibration")
  d <- dt_calibration
  p <- plot_panel_heatmap_9_cal(d$field$z, tstamp = seq(2, 26, by = 3),
                                max_y = max(d$field$z), loc_cal = d$grid,
                                Nx = d$Nx, Ny = d$Ny, savei = FALSE)
  length(p)
  p[[1]]
}
```

---

plot_panel_heatmap_9_cal_nolab

Plot a 3-by-3 panel of calibration heatmaps without axis labels

---

Description

Like plot_panel_heatmap_9_cal but suppresses x- and y-axis labels and supports a custom lower colour-scale limit (min_y).

Usage

```
plot_panel_heatmap_9_cal_nolab(
  dat,
  tstamp,
  min_y = 0,
  max_y,
  loc_cal,
  Nx,
```

```
  Ny,
  filename = "plot_panel",
  savei = TRUE,
  col_bgr = spdbl_default_col_bgr(),
  path_fig = "."
)
```

Arguments

dat	Numeric matrix of dimension c(nT, n_locations).
tstamp	Integer vector of length 9. Time-step indices to display.
min_y	Numeric scalar. Lower limit of the colour scale. Defaults to 0.
max_y	Numeric scalar. Upper limit of the colour scale.
loc_cal	Data frame with columns row and col.
Nx	Integer. Number of grid points in the x-direction.
Ny	Integer. Number of grid points in the y-direction.
filename	Character. Base name for the saved PNG. Defaults to "plot_panel".
savei	Logical. Whether to save the panel to disk. Defaults to TRUE.
col_bgr	Character vector of colours for the heatmap gradient.
path_fig	Character. Directory path where the file is saved when savei = TRUE. Defaults to the current working directory.

Value

A list of nine ggplot objects.

See Also

plot_panel_heatmap_9_cal

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggpubr", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE)) {
  data("dt_calibration")
  d <- dt_calibration
  u <- d$truth$u
  p <- plot_panel_heatmap_9_cal_nolab(u, tstamp = seq(2, 26, by = 3),
                                      min_y = min(u), max_y = max(u),
                                      loc_cal = d$grid, Nx = d$Nx, Ny = d$Ny,
                                      savei = FALSE)
  length(p)
  p[[1]]
}
```

`prepare_data` *Prepare PDE emulator training and testing data from CSV files*

Description

Reads a matrix of PDE input parameters and a set of PDE output CSV files, splits the inputs into training and testing sets, and reshapes the simulator outputs into time-indexed lists for emulator fitting and evaluation.

Usage

```
prepare_data(file_para, file_data, prop_train = 0.8)
```

Arguments

`file_para`	Character scalar. Path to a CSV file containing the PDE input parameter matrix. Rows correspond to simulator runs.
`file_data`	Character vector. Paths to CSV files containing PDE outputs, one file per simulator run. The length should match the number of rows in `file_para`.
`prop_train`	Numeric scalar in $(0, 1)$. Proportion of simulator runs used for training. Defaults to `0.8`.

Details

The function assumes that each file in `file_data` contains a matrix whose rows correspond to time steps and whose columns correspond to spatial locations. The number of time steps is inferred from the first PDE output file.

Value

A named list with:

pde_para_train Numeric matrix of training input parameters.

pde_para_test Numeric matrix of testing input parameters.

dt_pde_train Named list of training PDE outputs by time step.

dt_pde_test Named list of testing PDE outputs by time step.

n_train Integer. Number of training simulator runs.

n_test Integer. Number of testing simulator runs.

See Also

`emulator_learn`, `emulator_predict`

Examples

```
data("dt_emulation")
d <- dt_emulation
dir <- tempfile("spdbl_")
dir.create(dir)
file_para <- file.path(dir, "para.csv")
utils::write.csv(d$pde_para_train[1:5, ], file_para, row.names = FALSE)
file_data <- file.path(dir, paste0("run_", 1:5, ".csv"))
for (i in 1:5) utils::write.csv(t(sapply(d$dt_pde_train, function(y) y[i, ])),
                                file_data[i], row.names = FALSE)
dat <- prepare_data(file_para, file_data, prop_train = 0.6)
c(n_train = dat$n_train, n_test = dat$n_test)
dim(dat$dt_pde_train$T1)
unlink(dir, recursive = TRUE)
```

`quick_heat` *Quick raster heatmap*

Description

Creates a `ggplot2` raster heatmap using the `col_bgr` colour palette with a squished colour scale capped at `max_y`.

Usage

```
quick_heat(dt, max_y, col_bgr = spdbl_default_col_bgr())
```

Arguments

`dt`	Data frame with columns `x` (horizontal position), `y` (vertical position), and `fill` (value to display).
`max_y`	Numeric scalar. Upper limit of the colour scale. Values above this are squished to the maximum colour.
`col_bgr`	Character vector of colours for the heatmap gradient.

Value

A `ggplot` object.

See Also

`quick_save`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggpubr", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE)) {
  data("dt_calibration")
  d <- dt_calibration
  dt <- data.frame(x = d$grid$col, y = d$grid$row, fill = d$field$z[10, ])
  p <- quick_heat(dt, max_y = max(d$field$z))
  class(p)
  p
}
```

`quick_save` *Save a ggplot to a timestamped PNG file*

Description

Saves `plot` to a PNG file in `path_fig` whose name is `filename` followed by the current Unix timestamp, ensuring unique file names across repeated calls.

Usage

```
quick_save(filename, path_fig, plot)
```

Arguments

`filename`	Character. Base name for the output file (no extension).
`path_fig`	Character. Directory path where the file is saved.
`plot`	A `ggplot` object to save.

Value

The path of the saved file, invisibly (the value of `ggplot2::ggsave()`). Called for its side effect.

See Also

`quick_heat`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggpubr", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE)) {
  data("dt_calibration")
  d <- dt_calibration
  dt <- data.frame(x = d$grid$col, y = d$grid$row, fill = d$field$z[10, ])
  p <- quick_heat(dt, max_y = max(d$field$z))
  path_fig <- tempfile("spdbl_fig_")
```

```
  dir.create(path_fig)
  quick_save("heat", path_fig, p)
  list.files(path_fig)
  unlink(path_fig, recursive = TRUE)
}
```

`read_big_csv_quick` *Read a rectangular block from a large CSV file*

Description

Efficiently reads a contiguous rectangular block of rows and columns from a CSV file using `readr::read_csv`. Useful for processing big data in chunks without loading the entire file into memory.

Usage

```
read_big_csv_quick(filename, rows = c(1, Inf), cols = c(1, Inf), header = TRUE)
```

Arguments

`filename`	Character. Path to the CSV file.
`rows`	Integer vector of length 2 giving the first and last row indices to read (1-based, excluding the header row if `header = TRUE`). Use `c(1, Inf)` to read all rows. Defaults to `c(1, Inf)`.
`cols`	Integer vector of length 2 giving the first and last column indices to read. Use `c(1, Inf)` to read all columns. Defaults to `c(1, Inf)`.
`header`	Logical. Whether the file has a header row. Defaults to `TRUE`.

Value

A `tibble` (from `readr`) containing the requested block.

Examples

```
f <- file.path(tempdir(), "big.csv")
write.csv(matrix(1:20, 5, 4), f, row.names = FALSE)
read_big_csv_quick(f, rows = c(2, 4), cols = c(2, 3))
unlink(f)
```

recover_from_EP_exact *Recover episode-partitioned data to original time dimension (exact)*

Description

Reassembles a flattened episode-partitioned list (as produced by the EP model) back into a list indexed by the original `nT_ori` time steps. Assumes the number of blocks per season `n_block = nT / nT_ori` is an integer.

Usage

```
recover_from_EP_exact(dat_EP, nT_ori, nT, block_cols = NULL)
```

Arguments

`dat_EP`	List of length `nT`. Flattened episode-partitioned data.
`nT_ori`	Integer. Number of original (non-partitioned) time steps.
`nT`	Integer. Total number of episode time steps (`nT = nT_ori * n_block`).
`block_cols`	Optional block-by-column mapping to the original grid.

Value

A named list of length `nT_ori` (`"T1"`, ..., `"T<nT_ori>"`), each element a matrix covering one original time step.

See Also

`recover_from_EP_MC`

Examples

```
# Two time steps on six sites, split into three episodes of two sites each
Y <- lapply(1:2, function(t) matrix(10 * t + 1:6, nrow = 1))
ind <- generate_grid(1, 6, 1, 2)
Y_ep <- unlist(lapply(Y, function(y) lapply(1:3, function(j)
  y[, ind$L[j]:ind$R[j], drop = FALSE])), recursive = FALSE)
length(Y_ep)
rec <- recover_from_EP_exact(Y_ep, nT_ori = 2, nT = 6)
all.equal(unname(rec$T2), Y[[2]])
```

`recover_from_EP_MC` *Recover episode-partitioned posterior samples to original time dimension*

Description

Reassembles a flattened episode-partitioned list of posterior sample arrays back into a list indexed by the original `nT_ori` time steps.

Usage

```
recover_from_EP_MC(dat_EP, nT_ori, nT, nsam)
```

Arguments

`dat_EP`	List of length `nT`. Each element is an array of dimension `c(p, bncol, nsam)`.
`nT_ori`	Integer. Number of original time steps.
`nT`	Integer. Total number of episode time steps.
`nsam`	Integer. Number of posterior samples.

Value

A named list of length `nT_ori`. Each element is an array of dimension `c(p, bncol * n_block, nsam)`.

See Also

`recover_from_EP_exact`

Examples

```
set.seed(1)
# 2 original times x 3 episodes; each episode holds 4 draws of a 2 x 2 block
ep <- lapply(1:6, function(e) array(rnorm(2 * 2 * 4), c(2, 2, 4)))
rec <- recover_from_EP_MC(ep, nT_ori = 2, nT = 6, nsam = 4)
names(rec)
dim(rec$T1)
all.equal(rec$T2[, 3:4, 1], ep[[5]][, , 1])
```

`rmn_chol` *Draw one sample from a matrix-normal distribution (Cholesky parameterisation)*

Description

Samples $\Theta \sim MN(m, U, \Sigma)$ using the upper-triangular Cholesky factors `RM` and `RSigma` of U and Σ respectively: $\Theta = m + \text{RM}^\top Z \,\text{RSigma}$, where Z has i.i.d. standard normal entries.

Usage

```
rmn_chol(m, RM, RSigma)
```

Arguments

`m`	Numeric matrix. Mean matrix ($p \times S$).
`RM`	Numeric matrix. Upper-triangular Cholesky factor of the row covariance U ($p \times p$).
`RSigma`	Numeric matrix. Upper-triangular Cholesky factor of the column covariance Σ ($S \times S$).

Value

A numeric matrix of dimension `c(p, S)`.

See Also

`rmn_chol_more`

Examples

```
set.seed(1)
U <- matrix(c(1, 0.5, 0.5, 1), 2)
Sigma <- diag(c(1, 2, 3))
draw <- rmn_chol(m = matrix(0, 2, 3), RM = chol(U), RSigma = chol(Sigma))
dim(draw)
draw
```

`rmn_chol_more` *Draw multiple samples from a matrix-normal distribution (Cholesky parameterisation)*

Description

Draws `nsam` independent samples from $MN(m, U, \Sigma)$ using the Cholesky factors of U and Σ.

Usage

```
rmn_chol_more(nsam, m, RM, RSigma)
```

Arguments

`nsam`	Integer. Number of samples to draw.
`m`	Numeric matrix. Mean matrix ($p \times S$).
`RM`	Numeric matrix. Upper-triangular Cholesky factor of the row covariance U ($p \times p$).
`RSigma`	Numeric matrix. Upper-triangular Cholesky factor of the column covariance Σ ($S \times S$).

Value

An array of dimension `c(p, S, nsam)`.

See Also

`rmn_chol`

Examples

```
set.seed(1)
U <- matrix(c(1, 0.5, 0.5, 1), 2)
Sigma <- diag(c(1, 2, 3))
draws <- rmn_chol_more(2000, m = matrix(0, 2, 3), RM = chol(U), RSigma = chol(Sigma))
dim(draws)
var(draws[1, 3, ])               # close to U[1, 1] * Sigma[3, 3] = 3
cor(draws[1, 1, ], draws[2, 1, ]) # close to U[1, 2] = 0.5
```

`sample_y_eta_one` *Draw predictive samples from a precomputed mean and covariance*

Description

Draws one sample from the predictive distribution $y_t \sim N(\mu_t, \Sigma)$ at each calibration time step, given precomputed mean and covariance (e.g., from `update_muSigma_eta_one`).

Usage

```
sample_y_eta_one(ind_sam, nT_cal, N_sp_train, mu_arry_train, Sigma_eta_train)
```

Arguments

`ind_sam`	Integer. Sample index (used only for loop iteration; single sample is drawn).
`nT_cal`	Integer. Number of calibration time steps.
`N_sp_train`	Integer. Number of training spatial locations.
`mu_arry_train`	Numeric array of dimension `c(nT_cal, N_sp_train)`. Predictive means.
`Sigma_eta_train`	Numeric matrix. Predictive covariance (`N_sp_train` x `N_sp_train`).

Value

Numeric matrix of dimension `c(nT_cal, N_sp_train)` with one draw per time step.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

`update_muSigma_eta_one`

Examples

```
set.seed(1)
mu <- matrix(1:6, nrow = 2, ncol = 3)
y <- sample_y_eta_one(ind_sam = 1, nT_cal = 2, N_sp_train = 3,
                      mu_arry_train = mu, Sigma_eta_train = 0.01 * diag(3))
dim(y)
round(y - mu, 2)
```

`scale_back_uniform` *Invert a uniform scaling transformation*

Description

Maps `x` from the unit interval back to the original range $[\text{low}, \text{high}]$: $x \cdot (\text{high} - \text{low}) + \text{low}$.

Usage

```
scale_back_uniform(x, low, high)
```

Arguments

`x`	Numeric vector or matrix. Values in $[0, 1]$.
`low`	Numeric. Lower bound of the target range.
`high`	Numeric. Upper bound of the target range.

Value

Numeric vector or matrix rescaled to $[\text{low}, \text{high}]$.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

`scale_uniform`

Examples

```
u <- c(0, 0.25, 1)
x <- scale_back_uniform(u, low = -1, high = 3)
x
all.equal(scale_uniform(x, low = -1, high = 3), u)
```

`scale_uniform` *Scale a vector to the unit interval via a uniform transformation*

Description

Maps `x` to $(x - \text{low})/(\text{high} - \text{low})$. Raises an error if `x` is outside `[low, high]`. Sets `NaN` results to 0.

Usage

```
scale_uniform(x, low, high)
```

Arguments

`x`	Numeric vector or matrix. Values to scale.
`low`	Numeric. Lower bound of the original range.
`high`	Numeric. Upper bound of the original range.

Value

Numeric vector or matrix with values in $[0, 1]$.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

`scale_back_uniform`

Examples

```
x <- c(2, 3.5, 5)
u <- scale_uniform(x, low = 2, high = 5)
u
all.equal(scale_back_uniform(u, low = 2, high = 5), x)
```

SIR *Right-hand side of the spatially extended SIR ODE*

Description

Defines the right-hand side of a spatial SIR (Susceptible-Infected-Recovered) partial differential equation suitable for use with `deSolve::ode.2D`. Diffusion is implemented via `ReacTran::tran.2D` with zero-flux boundary conditions.

Usage

```
SIR(t, y, parms)
```

Arguments

`t`	Numeric. Current time (passed by the ODE solver).
`y`	Numeric vector. Flattened state vector of length $3N_xN_y$: S values, then I values, then R values.
`parms`	Named list of parameters:

Nx Integer. Number of grid cells in the x-direction.

Ny Integer. Number of grid cells in the y-direction.

N Numeric. Total population size.

eta Numeric vector of length 5: `eta[1]` transmission rate, `eta[2]` recovery rate, `eta[3]` diffusion coefficient for S, `eta[4]` diffusion coefficient for I, `eta[5]` diffusion coefficient for R.

Gridx 1D grid object for x (from `ReacTran::setup.grid.1D`).

Gridy 1D grid object for y.

Value

A list with one element: a numeric vector of derivatives of the same length as `y`, as required by `deSolve`.

See Also

gen_pde

Examples

```
if (requireNamespace("deSolve", quietly = TRUE) &&
    requireNamespace("ReacTran", quietly = TRUE)) {
set.seed(1)
parms <- list(eta = c(0.3, 0.1, 0, 0, 0), Nx = 2, Ny = 2, N = 100,
              Gridx = ReacTran::setup.grid.1D(x.up = 0, x.down = 2, N = 2),
              Gridy = ReacTran::setup.grid.1D(x.up = 0, x.down = 2, N = 2))
y <- c(runif(4, 50, 100), runif(4, 0, 10), rep(0, 4))
```

```
d <- SIR(t = 0, y = y, parms = parms)[[1]]
length(d)
all.equal(sum(d), 0)  # without diffusion, S + I + R is conserved
}
```

`spDBL_calibration_input-methods`

Print and predict from a prepared calibration object

Description

Methods for the result of `calibration_prepare()`. `print()` shows the parameter bounds, the design GP and the emulator draws. `predict()` calls `emulator_predict()`, which needs no new simulator runs, or with `interval = "prediction"` returns pointwise predictive distributions.

Usage

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

## S3 method for class 'spDBL_calibration_input'
predict(
  object,
  newdata,
  MC = FALSE,
  interval = c("none", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

`x`, `object`	An `spDBL_calibration_input` from `calibration_prepare()`.
`...`	Unused.
`newdata`	Matrix (or vector, for one input) of mechanistic inputs.
`MC`	Passed to `emulator_predict()`.
`interval`	`"none"` (the default) returns what `emulator_predict()` returns. `"prediction"` returns the pointwise predictive distribution of the field: given each emulator draw it is Gaussian, with the design-row uncertainty when `calibration_prepare()` propagates it; the equal-weight mixture over draws is summarised by its mean and variance, and the intervals are Gaussian with those moments.
`level`	Coverage of the central prediction intervals.

Value

`print()` returns `x` invisibly; `predict()` returns what `emulator_predict()` returns, or, with `interval = "prediction"`, a list with time-indexed lists `mean`, `sd`, `lower` and `upper` of input by site matrices, plus `level`, `distribution` and `n_draws`.

See Also

`calibration_prepare()`, `emulator_predict()`

Examples

```
data("dt_calibration")
d <- dt_calibration
steps <- 1:6  # the first 6 of 26 time steps, to keep the example fast
em <- emulator_learn(d$training$input, d$training$responses[steps], d$Nx, d$Ny,
  covariance = "sigma2R", episode_window = 2, nsam = 3, seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
cal
band <- predict(cal, d$validation$input[1:2, ], interval = "prediction")
round(band$sd[[6]][, 1:4], 3)
pred <- predict(cal, d$validation$input[1:2, ])
dim(pred[[5]])
```

---

spDBL_emulator-methods

Print, summarize and predict from a fitted emulator

Description

Methods for the result of `emulator_learn()`. `print()` shows the model settings. `summary()` adds the in-sample fit of the smoothed coefficients and the posterior mean of the residual variance. `predict()` calls `emulator_predict()`, or with `interval = "prediction"` returns pointwise predictive distributions.

Usage

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

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

## S3 method for class 'spDBL_emulator'
predict(
  object,
  newdata,
  dt_pde_test = NULL,
  F_new_ls = "default",
```

```
  MC = FALSE,
  interval = c("none", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

x, object	An spDBL_emulator from emulator_learn().
...	Unused.
newdata	Matrix (or vector, for one run) of new mechanistic inputs.
dt_pde_test, F_new_ls, MC	Passed to emulator_predict().
interval	"none" (the default) returns what emulator_predict() returns. "prediction" returns the pointwise predictive distribution given the autoregressive rows: for each site and time, a Student-t distribution from integrating the coefficients, the residual covariance and the input Gaussian process analytically.
level	Coverage of the central prediction intervals.

Value

print() returns x invisibly. summary() returns an object of class summary.spDBL_emulator, a list with

settings One-row data frame of model settings.

rmse_by_time Root mean squared error of the smoothed fit $F_e\hat{\Theta}_e$ to the training responses, at each time.

rmse The same over all times.

response_sd Standard deviation of the training responses.

residual_variance Posterior mean of the residual variance, averaged over the sites of a block: $D/(n - C - 1)$ for the inverse-Wishart model and $D/(n - 1)$ times the mean diagonal of R_episode for the $\sigma^2 R$ model, from the final filter.

predict() returns what emulator_predict() returns, or, with interval = "prediction", a list with time-indexed lists mean, sd, lower and upper of new-run by site matrices (the bounds of central intervals with coverage level), plus level, distribution and the degrees of freedom df.

See Also

emulator_learn(), emulator_predict()

Examples

```
data("dt_emulation")
d <- dt_emulation
em <- emulator_learn(d$pde_para_train, d$dt_pde_train, Nx = 10, Ny = 10,
                     nsam = 5, seed = 1)
em
```

```
summary(em)
pred <- predict(em, d$pde_para_test, dt_pde_test = d$dt_pde_test)
dim(pred[[3]])
band <- predict(em, d$pde_para_test, dt_pde_test = d$dt_pde_test,
                interval = "prediction")
# Share of held-out outputs at time 3 inside the 95% prediction intervals
mean(d$dt_pde_test[[3]] >= band$lower[[3]] & d$dt_pde_test[[3]] <= band$upper[[3]])
```

`summary.spDBL_calibration` *Summarize modular calibration samples*

Description

Summarize modular calibration samples

Usage

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

Arguments

`object`	A calibration result.
`...`	Unused.

Value

A list of parameter summaries, conditional diagnostics, and settings.

Examples

```
data("dt_calibration")
d <- dt_calibration
em <- emulator_learn(d$training$input, d$training$responses, d$Nx, d$Ny,
                     covariance = "sigma2R", episode_window = 2, nsam = 3,
                     seed = 1)
cal <- calibration_prepare(em, d$lower, d$upper)
fit <- calibration_fit(cal, d$field$z, d$field$U,
                       locations = d$field$locations, draws = 1:2,
                       chains = 1, iter = 200, seed = 1)
s <- summary(fit)
s$parameters
```

update_muSigma_eta_one *Compute posterior predictive mean and covariance without sampling (single sample)*

Description

Like update_y_eta_one but skips the final sampling step, returning only the predictive mean and covariance. Useful when sampling is done separately (e.g., in sample_y_eta_one).

Usage

```
update_muSigma_eta_one(
  eta,
  ind_sam,
  Nx,
  Ny,
  N_people,
  nT_ori,
  nT,
  nT_cal,
  N_sp,
  N_sp_train,
  AR_choice,
  bncol,
  res_ffbs,
  n_b_cal,
  ind_block_cal,
  t_VinvY_FTheta_ls,
  Vinv,
  R_cal_train,
  input,
  input_cal,
  phi_para,
  gp_sigma2,
  gp_tau2,
  ind_train_cal
)
```

Arguments

eta	Numeric vector. PDE parameters (length 5).
ind_sam	Integer. Index of the FFBS sample to use.
Nx	Integer. Grid cells in x-direction.
Ny	Integer. Grid cells in y-direction.
N_people	Numeric. Total population size.

nT_ori	Integer. Number of original time steps.
nT	Integer. Number of episode time steps.
nT_cal	Integer. Number of calibration time steps.
N_sp	Integer. Total number of spatial locations.
N_sp_train	Integer. Number of training spatial locations.
AR_choice	Integer. AR order (1 or 2).
bncol	Integer. Number of columns per block.
res_ffbs	List. FFBS posterior samples.
n_b_cal	Integer. Number of calibration blocks.
ind_block_cal	Data frame. Block indices for calibration.
t_VinvY_FTheta_ls	List. Precomputed $V^{-1}(Y - F\Theta)$ terms.
Vinv	Numeric matrix. Inverse of the observation covariance.
R_cal_train	Numeric matrix. Correlation matrix for training locations.
input	Numeric matrix. Observed location coordinates.
input_cal	Numeric matrix. Calibration location coordinates.
phi_para	Numeric scalar. GP range parameter.
gp_sigma2	Numeric scalar. GP marginal variance.
gp_tau2	Numeric scalar. GP nugget variance.
ind_train_cal	Integer vector. Indices of training locations.

Value

A named list with `mu` and `Sigma`.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

update_y_eta_one, sample_y_eta_one

Examples

```
set.seed(1)
input <- matrix(runif(10), nrow = 5, ncol = 2)
res_ffbs <- lapply(1:12, function(e) array(rnorm(8), c(2, 2, 2)))
res_ffbs$Sigma <- c(0.2, 0.3)
args <- list(eta = c(0.3, 0.1, 0.01, 0.01, 0.01), Nx = 2, Ny = 3, N_people = 100,
  nT_ori = 4, nT = 12, nT_cal = 3, N_sp = 6, N_sp_train = 4, AR_choice = 1,
  bncol = 2, res_ffbs = res_ffbs, n_b_cal = 3,
  ind_block_cal = generate_grid(1, 6, 1, 2),
  t_VinvY_FTheta_ls = lapply(1:3, function(t) array(rnorm(60), c(6, 5, 2))),
```

```
  Vinv = solve(gen_exp_kernel(input, phi = 2, sigma2 = 1.1, tau2 = 1e-4)),
  R_cal_train = diag(4), input = input, input_cal = c(0.4, 0.6), phi_para = 2,
  gp_sigma2 = 1.1, gp_tau2 = 1e-4, ind_train_cal = c(1, 3, 4, 6))
out <- do.call(update_muSigma_eta_one, c(args, ind_sam = 1))
dim(out$mu)  # nT_cal x N_sp_train
y <- sample_y_eta_one(1, nT_cal = 3, N_sp_train = 4, out$mu, out$Sigma)
dim(y)
```

`update_y_eta` *Update the likelihood of observations given PDE parameters (Monte Carlo)*

Description

For a given candidate PDE parameter vector `eta`, simulates the computer model output, computes the posterior predictive mean and covariance at calibration locations using all `nsam` FFBS samples, and draws new observations from the resulting predictive distribution.

Usage

```
update_y_eta(
  eta,
  nsam,
  Nx,
  Ny,
  N_people,
  nT_ori,
  nT,
  nT_cal,
  N_sp,
  N_sp_train,
  AR_choice,
  bncol,
  res_ffbs,
  n_b_cal,
  ind_block_cal,
  t_VinvY_FTheta_ls,
  Vinv,
  R_cal_train,
  input,
  input_cal,
  phi_para,
  gp_sigma2,
  gp_tau2,
  ind_train_cal
)
```

Arguments

`eta`	Numeric vector. PDE parameters (length 5).
`nsam`	Integer. Number of posterior samples.
`Nx`	Integer. Grid cells in x-direction.
`Ny`	Integer. Grid cells in y-direction.
`N_people`	Numeric. Total population size.
`nT_ori`	Integer. Number of original time steps.
`nT`	Integer. Number of episode time steps.
`nT_cal`	Integer. Number of calibration time steps.
`N_sp`	Integer. Total number of spatial locations.
`N_sp_train`	Integer. Number of training spatial locations.
`AR_choice`	Integer. AR order (1 or 2).
`bncol`	Integer. Number of columns per block.
`res_ffbs`	List. FFBS posterior samples.
`n_b_cal`	Integer. Number of calibration blocks.
`ind_block_cal`	Data frame. Block indices for calibration.
`t_VinvY_FTheta_ls`	List. Precomputed $V^{-1}(Y - F\Theta)$ terms.
`Vinv`	Numeric matrix. Inverse of the observation covariance.
`R_cal_train`	Numeric matrix. Correlation matrix for training locations.
`input`	Numeric matrix. Observed location coordinates.
`input_cal`	Numeric matrix. Calibration location coordinates.
`phi_para`	Numeric scalar. GP range parameter.
`gp_sigma2`	Numeric scalar. GP marginal variance.
`gp_tau2`	Numeric scalar. GP nugget variance.
`ind_train_cal`	Integer vector. Indices of training locations within calibration locations.

Value

A named list with:

y Array `c(nT_cal, N_sp_train, nsam)` of predictive samples.

mu Array `c(nT_cal, N_sp_train, nsam)` of predictive means.

sigma2_cal Numeric scalar. Predictive variance constant.

Sigma Numeric matrix. Predictive covariance matrix.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

update_y_eta_one, gen_pde

Examples

```
set.seed(1)
input <- matrix(runif(10), nrow = 5, ncol = 2)
res_ffbs <- lapply(1:12, function(e) array(rnorm(8), c(2, 2, 2)))
res_ffbs$Sigma <- c(0.2, 0.3)
args <- list(eta = c(0.3, 0.1, 0.01, 0.01, 0.01), Nx = 2, Ny = 3, N_people = 100,
  nT_ori = 4, nT = 12, nT_cal = 3, N_sp = 6, N_sp_train = 4, AR_choice = 1,
  bncol = 2, res_ffbs = res_ffbs, n_b_cal = 3,
  ind_block_cal = generate_grid(1, 6, 1, 2),
  t_VinvY_FTheta_ls = lapply(1:3, function(t) array(rnorm(60), c(6, 5, 2))),
  Vinv = solve(gen_exp_kernel(input, phi = 2, sigma2 = 1.1, tau2 = 1e-4)),
  R_cal_train = diag(4), input = input, input_cal = c(0.4, 0.6), phi_para = 2,
  gp_sigma2 = 1.1, gp_tau2 = 1e-4, ind_train_cal = c(1, 3, 4, 6))
out <- do.call(update_y_eta, c(args, nsam = 2))
dim(out$y)  # nT_cal x N_sp_train x nsam
out$sigma2_cal
```

---

update_y_eta_one *Update the likelihood of observations given PDE parameters (single sample)*

Description

Like update_y_eta but uses only one FFBS sample (indexed by ind_sam) instead of averaging over all samples, making it faster for MCMC proposals.

Usage

```
update_y_eta_one(
  eta,
  ind_sam,
  Nx,
  Ny,
  N_people,
  nT_ori,
  nT,
  nT_cal,
  N_sp,
  N_sp_train,
  AR_choice,
  bncol,
  res_ffbs,
  n_b_cal,
```

```
  ind_block_cal,
  t_VinvY_FTheta_ls,
  Vinv,
  R_cal_train,
  input,
  input_cal,
  phi_para,
  gp_sigma2,
  gp_tau2,
  ind_train_cal
)
```

Arguments

`eta`	Numeric vector. PDE parameters (length 5).
`ind_sam`	Integer. Index of the FFBS sample to use.
`Nx`	Integer. Grid cells in x-direction.
`Ny`	Integer. Grid cells in y-direction.
`N_people`	Numeric. Total population size.
`nT_ori`	Integer. Number of original time steps.
`nT`	Integer. Number of episode time steps.
`nT_cal`	Integer. Number of calibration time steps.
`N_sp`	Integer. Total number of spatial locations.
`N_sp_train`	Integer. Number of training spatial locations.
`AR_choice`	Integer. AR order (1 or 2).
`bncol`	Integer. Number of columns per block.
`res_ffbs`	List. FFBS posterior samples.
`n_b_cal`	Integer. Number of calibration blocks.
`ind_block_cal`	Data frame. Block indices for calibration.
`t_VinvY_FTheta_ls`	List. Precomputed $V^{-1}(Y - F\Theta)$ terms.
`Vinv`	Numeric matrix. Inverse of the observation covariance.
`R_cal_train`	Numeric matrix. Correlation matrix for training locations.
`input`	Numeric matrix. Observed location coordinates.
`input_cal`	Numeric matrix. Calibration location coordinates.
`phi_para`	Numeric scalar. GP range parameter.
`gp_sigma2`	Numeric scalar. GP marginal variance.
`gp_tau2`	Numeric scalar. GP nugget variance.
`ind_train_cal`	Integer vector. Indices of training locations.

Value

A named list with `y`, `mu`, and `Sigma`.

Lifecycle

Deprecated since spDBL 1.2.0 and will be removed in a future release. The first call in a session warns; set `options(spDBL.warn_deprecated = FALSE)` to silence it.

See Also

update_y_eta

Examples

```
set.seed(1)
input <- matrix(runif(10), nrow = 5, ncol = 2)
res_ffbs <- lapply(1:12, function(e) array(rnorm(8), c(2, 2, 2)))
res_ffbs$Sigma <- c(0.2, 0.3)
args <- list(eta = c(0.3, 0.1, 0.01, 0.01, 0.01), Nx = 2, Ny = 3, N_people = 100,
  nT_ori = 4, nT = 12, nT_cal = 3, N_sp = 6, N_sp_train = 4, AR_choice = 1,
  bncol = 2, res_ffbs = res_ffbs, n_b_cal = 3,
  ind_block_cal = generate_grid(1, 6, 1, 2),
  t_VinvY_FTheta_ls = lapply(1:3, function(t) array(rnorm(60), c(6, 5, 2))),
  Vinv = solve(gen_exp_kernel(input, phi = 2, sigma2 = 1.1, tau2 = 1e-4)),
  R_cal_train = diag(4), input = input, input_cal = c(0.4, 0.6), phi_para = 2,
  gp_sigma2 = 1.1, gp_tau2 = 1e-4, ind_train_cal = c(1, 3, 4, 6))
out <- do.call(update_y_eta_one, c(args, ind_sam = 2))
dim(out$y)  # nT_cal x N_sp_train
dim(out$Sigma)
```

Index

∗ **datasets**
dt_calibration, 21
dt_emulation, 22

BS, 4, *27*, *34–36*, *46*
BS(), *5*
BS_bigdata_R, 5
BS_bigdata_R(), *36, 37*

cal_Bt_bt, 6
cal_errorbar, 7, *8*
cal_errorbar_mean, *7*, 8
cal_jacobian_logit_uniform, 8
calibration_fit, 9
calibration_fit(), *12*, *14*, *21*
calibration_predict, 12
calibration_predict(), *12*, *14*, *25*
calibration_prepare, 13
calibration_prepare(), *10*, *15*, *21, 22*, *24*, *89, 90*
calibration_validate, 15
calibration_validate(), *12*, *14*
check_pds, 16, *67*
compute_analytical_waic, 17
compute_analytical_waic(), *19*
compute_sample_waic, 18
compute_sample_waic(), *18*

dMTig, 20, *65*
dt_calibration, 21
dt_emulation, 22

emulator_learn, 22, *77*
emulator_learn(), *13*, *24*, *36*, *90, 91*
emulator_predict, 24, *77*
emulator_predict(), *14*, *24*, *89–91*
expit, 25

FF, *4*, 26, *31–35*, *41*, *44, 45*
FF_1step_R_I, 27, *32*
FF_1step_R_sigma2R, 28, *34*
FF_bigdata_R, 29
FF_bigdata_R(), *5*, *36, 37*
FF_I, *28*, 32, *36*
FF_sigma2R, *29*, 33, *46*
FFBS, *4*, *27*, 34, *38*, *41, 42*, *66*
FFBS_I, *32*, 35, *42, 43*
FFBS_predict_bigdata_R, 36
FFBS_predict_bigdata_R(), *5*
FFBS_predict_exact, 38, *40*, *55*
FFBS_predict_exact(), *25*, *36, 37*
FFBS_predict_MC, 39, *39*
FFBS_sampling, *35*, *40*, 41, *71*
FFBS_sampling_I, *36*, 42
FFBS_sampling_sigma2R, 43, *46*, *68*
FFBS_sigma2R, *34*, *44*, 45

gen_calibrate_data, 46, *47, 48*, *58*
gen_calibrate_data_uncorr, 47
gen_exp_kernel, *39, 40*, 48, *49*, *55, 56*
gen_expsq_kernel, *48*, 49, *55*
gen_F_ls_AR1, 49, *51*
gen_F_ls_AR1_EP, 50, *50*, *52*
gen_F_ls_AR2, *50*, 51, *52*
gen_F_ls_AR2_EP, *51*, 52
gen_ffbs_csv, 53, *54*
gen_ffbs_data, *53*, 54
gen_gp_kernel, *48, 49*, 55
gen_Jt, 55
gen_pd_matrix, 56
gen_pde, 57, *88*, *97*
gen_prior_u_tau2, *47*, 58
gen_ran_matrix, 59
generate.grid.exact, 60, *61, 62*
generate.grid.lr, 60, *60*, *62*
generate.grid.rowsnake, *29*, *31*, *60*, 61, *61*
generate_grid, *50*, *52*, *60, 61*, 62, *62*

inv_chol, 63

lppd_id_1t, 64, *66*

lppd_IG_1t, *64*, 65, *66*
lppd_IW_1t, *64*, *65*, 66
lppd_IW_1t(), *18*

make_pds, *16*, 67
MNIG_sampler, *44*, 67, *71*
MNIW_R, 68, *70*
MNIW_R(), *17*, *18*
MNIW_R_naiive, *69*, 70
MNIW_sampler, *42*, *68*, 71

plot.spDBL_calibration, 72
plot_panel_heatmap_9, 73
plot_panel_heatmap_9_cal, 74, *75*, *76*
plot_panel_heatmap_9_cal_nolab, 75, *75*
predict.spDBL_calibration_input
(spDBL_calibration_input-methods), 89
predict.spDBL_emulator
(spDBL_emulator-methods), 90
prepare_data, 77
print.spDBL_calibration_input
(spDBL_calibration_input-methods), 89
print.spDBL_emulator
(spDBL_emulator-methods), 90

quick_heat, 78, *79*
quick_save, *78*, 79

read_big_csv_quick, 80
recover_from_EP_exact, 81, *82*
recover_from_EP_MC, *81*, 82
rmn_chol, 83, *84*
rmn_chol_more, *83*, 84

sample_y_eta_one, 85, *94*
scale_back_uniform, 86, *87*
scale_uniform, *86*, 87
SIR, *57*, 88
spDBL_calibration_input-methods, 89
spDBL_emulator-methods, 90
summary.spDBL_calibration, 92
summary.spDBL_emulator
(spDBL_emulator-methods), 90

t, *10*

update_muSigma_eta_one, *85*, 93
update_y_eta, *57*, 95, *97*, *99*
update_y_eta_one, *93*, *94*, 97, *97*