Package: PESTO (via r-universe)

June 3, 2026

Type Package

Title Parameter Estimation, Surrogates, and Tooling for Optimisation

Version 0.6.0.9000

Description High-performance parameter estimation, uncertainty quantification, and inverse modelling toolkit built on modernised PEST++ algorithms. Provides native R interfaces to iterative ensemble smoothers (IES), Gauss-Levenberg-Marquardt (GLM) solvers, global sensitivity analysis, and multi-objective optimisation under uncertainty. Includes surrogate-accelerated IES via Gaussian Process and Random Fourier Features, adaptive SVD backends (randomised SVD, LAPACK, Eigen), and convergence-aware adaptive ensemble sizing. Designed for large-scale environmental, hydrological, and agricultural models with support for highly-parameterised problems (>100,000 parameters).

License GPL (>= 3)

URL https://github.com/max578/PESTO, https://max578.github.io/PESTO

BugReports https://github.com/max578/PESTO/issues

Encoding UTF-8

Language en-AU

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements C++17

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.12), data.table, ggplot2, parallel, S7 (>= 0.2.0), yaml (>= 2.3.0), digest (>= 0.6.0)

LinkingTo Rcpp, RcppEigen

Suggests testthat (>= 3.0.0), knitr, rmarkdown, viridis, microbenchmark, Matrix, apsimx (>= 2.7.0)

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://max578.r-universe.dev

Date/Publication 2026-06-03 00:25:56 UTC

RemoteUrl https://github.com/max578/PESTO

RemoteRef main

RemoteSha 9275e524be986cb4d78c8d969d906052e956475d

Contents

`accelerate_svd` *Hardware-Accelerated SVD via LAPACK*

Description

Uses R's linked LAPACK (which on macOS is Apple Accelerate/AMX, on Linux is typically OpenBLAS or MKL) for hardware-optimised SVD computation.

Usage

```
accelerate_svd(A, thin = TRUE)
```

Arguments

`A`	Matrix (m x n). Input matrix.
`thin`	Logical. If TRUE (default), compute thin SVD.

Value

A list with components U, d, V.

Examples

```
set.seed(1L)
A <- matrix(rnorm(8 * 5), nrow = 8, ncol = 5)
res <- accelerate_svd(A, thin = TRUE)
length(res$d)
all.equal(sort(res$d, decreasing = TRUE), svd(A)$d)
```

`adaptive_ensemble_size` *Adaptive Ensemble Sizing*

Description

Dynamically determines the optimal ensemble size based on convergence diagnostics and information-theoretic criteria.

Usage

```
adaptive_ensemble_size(
  phi_values,
  current_size,
  min_size = 20L,
  max_size = 500L,
  cv_target = 0.3
)
```

Arguments

`phi_values`	Numeric vector. Current phi values per realisation.
`current_size`	Integer. Current ensemble size.
`min_size`	Integer. Minimum ensemble size (default 20).
`max_size`	Integer. Maximum ensemble size (default 500).
`cv_target`	Numeric. Target coefficient of variation for phi (default 0.3).

Details

Uses the effective sample size (ESS) and coefficient of variation of phi to determine whether the ensemble is too large (wasting compute) or too small (poor UQ coverage).

Value

A list with recommended_size, reasoning, and diagnostics.

Examples

```
set.seed(1L)
phi_values <- rnorm(50L, mean = 100, sd = 20)^2
res <- adaptive_ensemble_size(
  phi_values   = phi_values,
  current_size = 50L
)
res$recommended_size
res$cv_phi
res$ess_ratio
```

adaptive_svd *Adaptive SVD with Automatic Backend Selection*

Description

Automatically selects the optimal SVD algorithm based on matrix dimensions and available hardware:

Usage

```
adaptive_svd(A, k = 0L, method = "auto")
```

Arguments

A	Matrix (m x n). Input matrix.
k	Integer. Target rank. If 0 (default), computes full SVD.
method	Character. Force a specific method: "auto" (default), "rsvd", "accelerate", "eigen", "cuda".

Details

- **Randomised SVD** (Halko et al., 2011): when target rank k « min(m,n)
- **Apple Accelerate**: on macOS, for full SVD leveraging AMX coprocessor
- **Eigen BDCSVD**: cross-platform fallback with divide-and-conquer
- **CUDA cuSOLVER**: on GPU-equipped systems (when compiled with PESTO_USE_CUDA)

Value

A list with components U (m x k), d (k), V (n x k), plus `method_used` and `time_ms`.

Examples

```
set.seed(1L)
A <- matrix(rnorm(20 * 12), nrow = 20, ncol = 12)
res <- adaptive_svd(A, k = 5L, method = "auto")
length(res$d)
is.character(res$method_used)
```

apsim_callback *apsimx Forward-Model Adapter for PESTO IES*

Description

Builds an in-process forward-model closure for pesto_ies_callback() that drives APSIM (Next Gen .apsimx or Classic .apsim) through the apsimx package without going via the .pst-file path. Used as Year-1 §D4 of the UQ ag-stack roadmap (uq_ag_stack_roadmap_v0.md).

Usage

```
apsim_callback(
  template,
  param_map,
  output_extractor,
  workdir = tempfile("apsim_cb_"),
  param_writer = NULL,
  simulation_runner = NULL,
  verbose = FALSE
)
```

Arguments

template	Character. Path to a working .apsimx (Next Gen) or .apsim (Classic) template file. Per-realisation copies are made into workdir; the template itself is never modified.
param_map	Named list. Names are the parameter columns expected in theta; values are character node paths understood by the appropriate apsimx::edit_* function (e.g. "Manager.SowingRule.Rule.Population" for Next Gen).
output_extractor	Function. Takes the object returned by apsimx::apsimx() / apsim() (typically a data.frame of report variables) and returns a length-nobs numeric vector. The first successful realisation defines nobs.
workdir	Character. Per-run working directory. Created if it does not exist; not cleaned automatically (so failures are inspectable). Default: a fresh tempfile("apsim_cb_").
param_writer	Optional function with signature function(file, src.dir, node, value). Overrides the default apsimx editor dispatch. Useful for unusual node paths or non-standard apsimx versions.
simulation_runner	Optional function with signature function(file, src.dir) returning the simulation object. Overrides the default apsimx::apsimx() / apsim() dispatch.
verbose	Logical. Forward verbose flag into apsimx calls and print per-realisation status (default FALSE).

Details

The returned closure has signature `function(theta) -> obs`, where `theta` is an `nreal x npar` matrix with column names matching `names(param_map)`, and `obs` is an `nreal x nobs` matrix. Each row is produced by:

1. Copying `template` into a fresh per-realisation file under `workdir`.
2. For each `(par_name, node_path)` in `param_map`, calling the appropriate `apsimx::edit_*` function to write `theta[i, par_name]` to that node.
3. Calling `apsimx::apsimx()` (or `apsim()` for Classic) and passing the returned simulation object to `output_extractor()`, which must return a length-`nobs` numeric vector.

Per-realisation failures (APSIM crash, missing output, extractor error) populate an `NA` row; `pesto_ies_callback()` then carries that realisation forward unchanged or aborts, depending on its `on_failure` setting.

Value

A closure of signature `function(theta) -> obs` suitable for the `forward_model =` argument of `pesto_ies_callback()`.

APSIM version compatibility

Tested against the `apsimx` package API as of CRAN 2.7.x. The exact editor function differs between APSIM Next Gen (`edit_apsimx*`) and APSIM Classic (`edit_apsim`); selection is by file extension of `template`. If your installed `apsimx` version exposes a different editor signature, supply `param_writer` to override the default per-parameter writer.

Concurrency

The returned closure evaluates whatever block of realisations it is handed and writes each to a **unique** per-realisation file under `workdir`, so it is safe to drive in parallel. To run an ensemble concurrently, wrap the closure in a `pesto_forward_model()` with `parallel = "multicore"` (or a custom `map_fn`); the PESTO evaluation engine then dispatches realisations across forked workers, each invoking APSIM on its own input file. Called directly (the bulk path), realisations run serially. `apsimx`'s own thread-safety under heavy ensemble load has not been independently verified, so start with a modest `n_cores`.

See Also

`pesto_ies_callback()` for the IES driver; `pesto_ies()` for the classic `.pst`-file path.

Examples

```
## Not run:
# Requires apsimx and a working APSIM installation
fm <- apsim_callback(
  template  = "wheat_wagga.apsimx",
  param_map = list(
    RUE       = "Wheat.Leaf.Photosynthesis.RUE.FixedValue",
    CN2       = "Soil.SoilWater.CN2Bare"
  ),
```

```
  output_extractor = function(sim) {
    # sim is a data.frame; extract end-of-season yield trajectory
    as.numeric(sim$Wheat.Grain.Total.Wt)
  }
)
prior <- matrix(c(runif(40, 1.0, 2.0), runif(40, 60, 90)),
                ncol = 2, dimnames = list(NULL, c("RUE", "CN2")))
fit <- pesto_ies_callback(
  forward_model  = fm,
  prior_ensemble = prior,
  obs            = c(y1 = 4500, y2 = 5200),
  obs_sd         = 200,
  noptmax        = 4
)

## End(Not run)
```

`as_forward_model` *Coerce an object into a* `pesto_forward_model`

Description

Generic used by the IES driver so a caller can pass either a bare `function(theta) -> obs` or a fully-specified `pesto_forward_model()`. A bare function is wrapped with the supplied policy arguments; an existing `pesto_forward_model` is returned unchanged (its own policy is authoritative and the `...` are ignored).

Usage

```
as_forward_model(x, ...)
```

Arguments

`x`	A function or a `pesto_forward_model`.
`...`	Policy arguments forwarded to `pesto_forward_model()` when x is a bare function (e.g. `on_failure`, `parallel`, `n_obs`).

Value

A `pesto_forward_model` S7 object.

See Also

`pesto_forward_model()`, `pesto_evaluate()`.

Examples

```
fm <- as_forward_model(function(theta) theta, on_failure = "stop")
S7::S7_inherits(fm, pesto_forward_model)
```

as_manifest *Convert a PESTO ensemble result into a* pesto_ensemble_manifest

Description

Wraps the plain-list result returned by pesto_ies_callback() (and, eventually, pesto_ies()) in the S7 manifest contract object so downstream packages can consume it via S7 dispatch without reaching into PESTO-specific list internals.

Usage

```
as_manifest(x, ...)
```

Arguments

x	A pesto_ies_callback_result (or any object with a method registered against this generic).
...	Method-specific arguments. For pesto_ies_callback_result: run_id (character, defaults to a timestamp+hash slug), seed (integer, defaults to NA_integer_), fidelity (structured provenance list or NULL; defaults to the record captured by pesto_ies_callback() for a multi-fidelity run), apsim_version (character, defaults to NA_character_).

Value

A pesto_ensemble_manifest S7 object.

check_surrogate_regime *Check whether a surrogate-IES regime is favourable*

Description

Issues a warning when the ratio of training points to parameters falls below an empirical threshold, where the Gaussian-process surrogate inside pesto_surrogate_ies() and surrogate_ensemble_update() is unlikely to repay its training cost. The check is a soft guardrail — it does not modify the run, only flags an unfavourable regime so the caller can decide whether to fall back to pure IES.

Usage

```
check_surrogate_regime(n_params, n_train, threshold = 5)
```

Arguments

n_params	Integer. Number of estimated parameters.
n_train	Integer. Number of training samples available to the surrogate (typically the ensemble size).
threshold	Numeric. Minimum acceptable n_train / n_params ratio. Default 5, the empirical soft floor from the surrogate-IES vignette.

Details

The default threshold of 5 corresponds to the soft floor n_train >= 5 * n_params documented in vignette("surrogate-ies", package = "PESTO"). Below that floor the GP posterior variance typically stays above the uncertainty-driven switching threshold and surrogate savings collapse to near zero.

This is exposed as a stand-alone helper so users can call it explicitly before scheduling an expensive ensemble. It is **not** invoked automatically by pesto_surrogate_ies() in the current release; that wiring is tracked as a v0.2 enhancement candidate.

Value

Invisibly returns TRUE when the regime is favourable (n_train >= threshold * n_params) and FALSE otherwise. Called for the warning side-effect.

References

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

See Also

pesto_surrogate_ies(), surrogate_ensemble_update(), vignette("surrogate-ies", package = "PESTO")

Examples

```
# Favourable regime: 100 training points for 10 parameters.
check_surrogate_regime(n_params = 10L, n_train = 100L)

# Unfavourable regime: 30 training points for 30 parameters
# (the curse-of-dimensionality case from Scenario C of the
# comparison-and-simulation vignette). Emits a warning.
suppressWarnings(
  check_surrogate_regime(n_params = 30L, n_train = 30L)
)

# Custom threshold for users with a smoother forward model.
check_surrogate_regime(n_params = 20L, n_train = 60L, threshold = 3)
```

`compute_phi` *Compute Phi (Objective Function) for Ensemble*

Description

Calculates the weighted sum of squared residuals for each realisation in the ensemble.

Usage

```
compute_phi(residuals, weights)
```

Arguments

`residuals`	Matrix (nobs x nreal). Observation residuals.
`weights`	Numeric vector (nobs). Observation weights.

Value

Numeric vector (nreal). Phi value per realisation.

Examples

```
set.seed(1L)
residuals <- matrix(rnorm(5 * 4), 5, 4)
weights   <- rep(1, 5)
phi <- compute_phi(residuals, weights)
length(phi)
phi
```

`correlation_localisation`
Correlation-Based Automatic Localisation Taper

Description

Builds an `npar` x `nobs` localisation taper directly from the ensemble, with no parameter / observation coordinates required. This is the iterative-ensemble-smoother-native localisation of Luo & Bhakta (2020): spurious sample correlations between a parameter and an observation (an artefact of finite ensemble size) are damped, while genuine correlations that stand above an estimated noise floor are retained.

Usage

```
correlation_localisation(
  par_diff,
  obs_diff,
  threshold = -1,
  taper = "hard",
  n_shuffle = 1L,
  quantile = 0.95
)
```

Arguments

par_diff	Matrix (npar x nreal). Parameter anomalies.
obs_diff	Matrix (nobs x nreal). Observation anomalies.
threshold	Numeric. Noise floor on $\|\rho\|$. Negative (default -1) triggers automatic estimation by permutation.
taper	Character. "hard" (indicator) or "soft" (linear ramp above the floor).
n_shuffle	Integer. Number of permutation replicates for the automatic floor (default 1; each replicate yields npar * nobs spurious samples). Ignored when threshold >= 0.
quantile	Numeric in (0, 1). Quantile of the spurious-correlation distribution used as the floor (default 0.95). Ignored when threshold >= 0.

Details

The sample correlation ρ_{ij} between parameter-anomaly row i and observation-anomaly row j is compared against a noise floor θ. When threshold < 0 the floor is estimated by destroying the parameter-observation link — the realisation order of the observation anomalies is randomly permuted, independently per replicate, and the floor is taken as a high quantile (default 0.95) of the resulting spurious $|\rho|$ values. The permutation uses R's RNG, so the estimate is reproducible under set.seed().

Two tapers are offered. "hard" keeps correlations above the floor unchanged (weight 1) and zeroes the rest. "soft" applies a smooth, monotone ramp $w_{ij} = \mathrm{clip}((|\rho_{ij}| - \theta)/(1 - \theta), 0, 1)$, which downweights near-floor correlations rather than thresholding them sharply.

Value

A list with rho (the npar x nobs taper), threshold (the floor used), n_active (count of entries with non-zero weight), and frac_active (that count over npar * nobs).

References

Luo, X. & Bhakta, T. (2020). Automatic and adaptive localization for ensemble-based history matching. *Journal of Petroleum Science and Engineering*, 184, 106559.

Examples

```
set.seed(1L)
npar <- 8L; nobs <- 5L; nreal <- 40L
pd <- matrix(rnorm(npar * nreal), npar, nreal)
# Make parameter 1 genuinely correlated with observation 1.
od <- matrix(rnorm(nobs * nreal), nobs, nreal)
od[1L, ] <- od[1L, ] + 2 * pd[1L, ]
loc <- correlation_localisation(pd, od)
loc$threshold
loc$rho[1L, 1L]
```

---

create_pest_scenario *Create a PEST Scenario Programmatically*

Description

Builds a pesto_pst object from R data structures, without requiring an existing .pst file.

Usage

```
create_pest_scenario(
  parameters,
  observations,
  model_command,
  template_files = NULL,
  instruction_files = NULL,
  pestpp_options = list()
)
```

Arguments

parameters	data.table. Parameter definitions with columns: parnme, partrans, parchglim, parval1, parlbnd, parubnd, pargp.
observations	data.table. Observation definitions with columns: obsnme, obsval, weight, obgnme.
model_command	Character. Model command line(s).
template_files	data.table. Template/model input file pairs with columns template_file, model_file.
instruction_files	data.table. Instruction/model output file pairs with columns instruction_file, model_file.
pestpp_options	Named list. PEST++ options.

Value

A pesto_pst object.

Examples

```
pars <- data.table::data.table(
  parnme = c("k1", "k2"),
  partrans = "log",
  parchglim = "factor",
  parval1 = c(1.0, 0.5),
  parlbnd = c(0.01, 0.001),
  parubnd = c(100, 50),
  pargp = "hydraulic"
)
obs <- data.table::data.table(
  obsnme = c("h1", "h2"),
  obsval = c(1.0, 2.0),
  weight = c(1.0, 1.0),
  obgnme = "head"
)
pst <- create_pest_scenario(
  parameters    = pars,
  observations  = obs,
  model_command = "python model.py"
)
inherits(pst, "pesto_pst")
pst$control_data$npar
```

---

ensemble_solution *Ensemble Solution Kernel (GLM form)*

Description

Implements the core IES ensemble update equation using the Gauss-Levenberg-Marquardt (GLM) formulation from Chen & Oliver (2013). This is the computational hotspot of the iterative ensemble smoother.

Usage

```
ensemble_solution(
  par_diff,
  obs_diff,
  obs_resid,
  par_resid,
  weights,
  parcov_inv,
  Am,
  cur_lam,
  eigthresh = 1e-06,
  use_approx = TRUE,
  use_prior_scaling = FALSE,
  iter = 1L,
```

```
  reg_factor = -1
)
```

Arguments

par_diff	Matrix (npar x nreal). Parameter anomalies (deviations from mean).
obs_diff	Matrix (nobs x nreal). Observation anomalies.
obs_resid	Matrix (nobs x nreal). Observation residuals, simulated minus observed (sim - obs). This sign convention is required so that the leading negative in the GLM update ($\Delta\theta = -\Delta\theta' V s(s^2 + \lambda I)^{-1} U^T r$) yields a descent step on $\Phi = \|W(g(\theta) - d^{obs})\|^2$. Passing (obs - sim) inverts the gradient and causes phi to diverge.
par_resid	Matrix (npar x nreal). Parameter residuals (par - prior mean).
weights	Numeric vector (nobs). Observation weights (1/sqrt(variance)).
parcov_inv	Numeric vector (npar). Diagonal of inverse parameter covariance.
Am	Matrix (npar x nreal-1). Random Am matrix for upgrade_2 (optional).
cur_lam	Numeric. Current Marquardt lambda.
eigthresh	Numeric. Eigenvalue truncation threshold (0-1).
use_approx	Logical. If TRUE, skip upgrade_2 (prior-scaling correction).
use_prior_scaling	Logical. Scale by prior covariance.
iter	Integer. Current iteration number.
reg_factor	Numeric. Regularisation factor for upgrade_2 blending.

Details

The update equation solves:

$$\Delta\theta = -\Delta\theta' V s(s^2 + \lambda I)^{-1} U^T r$$

where the SVD is performed on the scaled observation difference matrix.

Value

Matrix (nreal x npar). Parameter upgrade vectors (one row per realisation). The returned matrix is the *negative-direction* step from the Chen-Oliver 2013 GLM update formula. To advance the ensemble, apply by subtraction: `par_new = par_old - upgrade`. The R-side driver `pesto_ies_callback()` handles this convention internally.

Examples

```
set.seed(1L)
npar  <- 4L
nreal <- 20L
nobs  <- 30L
par_diff  <- matrix(rnorm(npar * nreal), npar, nreal)
obs_diff  <- matrix(rnorm(nobs * nreal), nobs, nreal)
```

```
obs_resid <- matrix(rnorm(nobs * nreal, sd = 0.5), nobs, nreal)
par_resid <- matrix(rnorm(npar * nreal, sd = 0.1), npar, nreal)
weights    <- rep(1, nobs)
parcov_inv <- rep(1, npar)
Am         <- matrix(0, 0, 0)
upgrade <- ensemble_solution(
  par_diff   = par_diff,
  obs_diff   = obs_diff,
  obs_resid  = obs_resid,
  par_resid  = par_resid,
  weights    = weights,
  parcov_inv = parcov_inv,
  Am         = Am,
  cur_lam    = 1.0
)
dim(upgrade)
```

`ensemble_solution_gpu` *Ensemble Solution with Adaptive SVD Backend*

Description

Enhanced version of `ensemble_solution()` that uses the adaptive SVD backend for automatic hardware acceleration. On macOS, uses Apple Accelerate (AMX coprocessor). On CUDA systems, uses cuSOLVER. For low-rank problems, automatically switches to randomised SVD.

Usage

```
ensemble_solution_gpu(
  par_diff,
  obs_diff,
  obs_resid,
  par_resid,
  weights,
  parcov_inv,
  Am,
  cur_lam,
  eigthresh = 1e-06,
  use_approx = TRUE,
  use_prior_scaling = FALSE,
  iter = 1L,
  reg_factor = -1,
  svd_method = "auto",
  target_rank = 0L
)
```

Arguments

par_diff	Matrix (npar x nreal). Parameter anomalies.
obs_diff	Matrix (nobs x nreal). Observation anomalies.
obs_resid	Matrix (nobs x nreal). Observation residuals.
par_resid	Matrix (npar x nreal). Parameter residuals.
weights	Numeric vector (nobs). Observation weights.
parcov_inv	Numeric vector (npar). Inverse parameter covariance diagonal.
Am	Matrix. Random Am matrix for upgrade_2.
cur_lam	Numeric. Marquardt lambda.
eigthresh	Numeric. Eigenvalue truncation threshold.
use_approx	Logical. Skip upgrade_2.
use_prior_scaling	Logical. Scale by prior covariance.
iter	Integer. Current iteration.
reg_factor	Numeric. Regularisation factor.
svd_method	Character. SVD method: "auto", "rsvd", "accelerate", "eigen".
target_rank	Integer. Target rank for randomised SVD (0 = auto).

Value

A list with upgrade matrix and performance diagnostics.

Examples

```
set.seed(1L)
npar  <- 4L
nreal <- 20L
nobs  <- 30L
par_diff  <- matrix(rnorm(npar * nreal), npar, nreal)
obs_diff  <- matrix(rnorm(nobs * nreal), nobs, nreal)
obs_resid <- matrix(rnorm(nobs * nreal, sd = 0.5), nobs, nreal)
par_resid <- matrix(rnorm(npar * nreal, sd = 0.1), npar, nreal)
weights    <- rep(1, nobs)
parcov_inv <- rep(1, npar)
Am         <- matrix(0, 0, 0)
res <- ensemble_solution_gpu(
  par_diff   = par_diff,
  obs_diff   = obs_diff,
  obs_resid  = obs_resid,
  par_resid  = par_resid,
  weights    = weights,
  parcov_inv = parcov_inv,
  Am         = Am,
  cur_lam    = 1.0,
  svd_method = "auto"
)
dim(res$upgrade)
```

`ensemble_solution_localised` *Localised Ensemble Solution Kernel (explicit-gain GLM form)*

Description

Computes the IES Gauss-Levenberg-Marquardt update with state-space covariance localisation applied as a Schur (elementwise) product on the explicit Kalman gain. The standard SVD kernel `ensemble_solution()` works in the reduced observation-anomaly subspace and never forms the `npar x nobs` gain, so it cannot host localisation; this kernel reconstructs the gain

$$K = \Delta\Theta\, V\, \mathrm{diag}(s)\, \mathrm{diag}((s^2 + (\lambda + 1))^{-1})\, U^T,$$

(with UsV^T the thin SVD of the weight-scaled observation anomalies) tapers it as $K \circ \rho$, and applies it to the weighted residuals.

Usage

```
ensemble_solution_localised(
  par_diff,
  obs_diff,
  obs_resid,
  weights,
  rho,
  cur_lam,
  eigthresh = 1e-06
)
```

Arguments

`par_diff`	Matrix (npar x nreal). Parameter anomalies.
`obs_diff`	Matrix (nobs x nreal). Observation anomalies.
`obs_resid`	Matrix (nobs x nreal). Observation residuals (sim - obs); see `ensemble_solution()` for the sign rationale.
`weights`	Numeric vector (nobs). Observation weights (1 / sqrt(variance)).
`rho`	Matrix (npar x nobs). Localisation taper in $[0, 1]$, e.g. from `correlation_localisation()` or `gaspari_cohn()`.
`cur_lam`	Numeric. Current Marquardt lambda.
`eigthresh`	Numeric. Eigenvalue truncation threshold (0-1).

Details

When $\rho \equiv 1$ the result is identical (to truncation tolerance) to `ensemble_solution()` with `use_approx = TRUE`; the prior-scaling null-space correction (`upgrade_2`) is not part of the localised path. The returned matrix follows the same sign convention as `ensemble_solution()` — it is the negative-direction step, applied to the ensemble by subtraction (`par_new = par_old - upgrade`).

Value

Matrix (nreal x npar). Negative-direction parameter upgrade, applied by subtraction.

References

Chen, Y. & Oliver, D.S. (2013). Levenberg-Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17(4), 689–703.

Examples

```
set.seed(1L)
npar <- 4L; nreal <- 20L; nobs <- 6L
par_diff  <- matrix(rnorm(npar * nreal), npar, nreal)
obs_diff  <- matrix(rnorm(nobs * nreal), nobs, nreal)
obs_resid <- matrix(rnorm(nobs * nreal, sd = 0.5), nobs, nreal)
weights   <- rep(1, nobs)
rho       <- matrix(1, npar, nobs)          # no localisation
upg <- ensemble_solution_localised(
  par_diff, obs_diff, obs_resid, weights, rho, cur_lam = 1.0
)
dim(upg)
```

---

ensemble_solution_mda *Ensemble Solution Kernel (MDA / Evensen form)*

Description

Implements the Multiple Data Assimilation (MDA) update from Evensen (2018), Section 14.3.2. Uses low-rank representation of the error covariance.

Usage

```
ensemble_solution_mda(
  par_diff,
  obs_diff,
  obs_resid,
  obs_err,
  cur_lam = 1,
  eigthresh = 1e-06
)
```

Arguments

par_diff	Matrix (npar x nreal). Parameter anomalies.
obs_diff	Matrix (nobs x nreal). Observation anomalies.
obs_resid	Matrix (nobs x nreal). Observation residuals.

obs_err	Matrix (nobs x nreal). Observation error realisations.
cur_lam	Numeric. Inflation factor.
eigthresh	Numeric. Eigenvalue truncation threshold.

Value

Matrix (nreal x npar). Parameter upgrade vectors.

Examples

```
set.seed(1L)
npar  <- 4L
nreal <- 20L
nobs  <- 30L
par_diff  <- matrix(rnorm(npar * nreal), npar, nreal)
obs_diff  <- matrix(rnorm(nobs * nreal), nobs, nreal)
obs_resid <- matrix(rnorm(nobs * nreal, sd = 0.5), nobs, nreal)
obs_err   <- matrix(rnorm(nobs * nreal, sd = 0.5), nobs, nreal)
upgrade <- ensemble_solution_mda(
  par_diff  = par_diff,
  obs_diff  = obs_diff,
  obs_resid = obs_resid,
  obs_err   = obs_err,
  cur_lam   = 1.0
)
dim(upgrade)
```

---

ensemble_spread_ess *Spectral Spread Effective Sample Size of a Parameter Ensemble*

Description

Diagnoses ensemble collapse (under-dispersion) by the participation ratio of the parameter-anomaly covariance eigenspectrum. Given the parameter anomalies $\Delta\Theta$ (deviations from the ensemble mean, npar x nreal), the anomaly covariance is $C = \Delta\Theta\Delta\Theta^T/(N-1)$ with eigenvalues $\lambda_i = s_i^2/(N-1)$, where s_i are the singular values of $\Delta\Theta$.

Usage

```
ensemble_spread_ess(par_diff)
```

Arguments

par_diff	Matrix (npar x nreal). Parameter anomalies (deviations from the ensemble mean). At least 2 columns are required.

Details

The spectral spread-ESS is the participation ratio

$$\mathrm{ESS} = \frac{(\sum_i \lambda_i)^2}{\sum_i \lambda_i^2} = \frac{(\sum_i s_i^2)^2}{\sum_i s_i^4},$$

the effective number of directions carrying variance. It is bounded in $[1, r_{\max}]$ with $r_{\max} = \min(\mathrm{npar}, N-1)$: equal to $r_{\max}$ when variance is spread isotropically across all modes, and approaching 1 when the ensemble collapses onto a single direction. Because the ratio is invariant to a global rescaling of the anomalies, it isolates the *shape* of the collapse (directional degeneracy) from its *magnitude*; magnitude is tracked separately by the R-side spread-retention ratio.

Value

A list with components `ess` (the spectral spread-ESS), `r_max` (the maximum attainable value $\min(\mathrm{npar}, N-1)$), and `ess_ratio` (`ess / r_max`, in $(0, 1]$).

References

Bretherton, C.S., Widmann, M., Dymnikov, V.P., Wallace, J.M. & Blade, I. (1999). The effective number of spatial degrees of freedom of a time-varying field. *Journal of Climate*, 12(7), 1990–2009.

Examples

```
set.seed(1L)
# Healthy isotropic spread -> ESS near r_max
good <- matrix(rnorm(6L * 40L), 6L, 40L)
ensemble_spread_ess(good)$ess_ratio
# Collapsed onto one direction -> ESS near 1
v <- rnorm(6L)
bad <- outer(v, rnorm(40L)) + matrix(rnorm(6L * 40L, sd = 1e-3), 6L, 40L)
ensemble_spread_ess(bad)$ess_ratio
```

`gaspari_cohn` *Gaspari-Cohn Localisation Taper*

Description

Evaluates the Gaspari & Cohn (1999) fifth-order piecewise-rational compactly-supported correlation function on a matrix of distances. This is the classical distance-based localisation taper: a smooth bump that equals 1 at zero distance, decays to 0 at twice the localisation radius, and is identically 0 beyond. Used to taper the Kalman gain when the parameters and observations carry a spatial (or otherwise metric) coordinate.

Usage

```
gaspari_cohn(distances, radius)
```

Arguments

distances	Matrix (npar x nobs). Non-negative parameter-to- observation distances.
radius	Numeric scalar (> 0). Localisation radius c; the taper reaches 0 at distance $2c$.

Details

With $z = d/c$ (distance over localisation radius c):

$$G(z) = \begin{cases} -\frac{1}{4}z^5 + \frac{1}{2}z^4 + \frac{5}{8}z^3 - \frac{5}{3}z^2 + 1 & 0 \le z \le 1 \\ \frac{1}{12}z^5 - \frac{1}{2}z^4 + \frac{5}{8}z^3 + \frac{5}{3}z^2 - 5z + 4 - \frac{2}{3}z^{-1} & 1 < z \le 2 \\ 0 & z > 2. \end{cases}$$

Value

Matrix (npar x nobs) of taper weights in $[0, 1]$.

References

Gaspari, G. & Cohn, S.E. (1999). Construction of correlation functions in two and three dimensions. *Quarterly Journal of the Royal Meteorological Society*, 125(554), 723–757.

Examples

```
d <- matrix(c(0, 0.5, 1, 1.5, 2, 3), nrow = 2L)
gaspari_cohn(d, radius = 1.0)
```

---

mf_control_variate *Affine control-variate bias correction across fidelities*

Description

Debiases a cheap-fidelity output ensemble against a sparse expensive sample, per observation dimension. For each observation `j` it fits the first-order autoregressive control variate (the linear term of the Kennedy-O'Hagan multi-fidelity model) `high_j ~ a_j + b_j * low_j` on the paired subset, with `b_j = cov(high_j, low_j) / var(low_j)` the variance-minimising coefficient, then predicts the corrected high-fidelity output for every realisation as `a_j + b_j * low_all_j`.

Usage

```
mf_control_variate(low_all, high_sub, low_sub)
```

Arguments

low_all	Numeric matrix, `nreal x nobs`. Cheap-fidelity output for every realisation.
high_sub	Numeric matrix, `nsub x nobs`. Expensive-fidelity output for the paired subset.
low_sub	Numeric matrix, `nsub x nobs`. Cheap-fidelity output for the same subset, row-aligned with `high_sub`.

Details

The estimator degrades gracefully: where the cheap output has zero variance on the subset it falls back to the expensive subset mean (`b_j = 0`), and where the two fidelities are weakly correlated the correction shrinks toward that mean rather than amplifying noise.

This is the plug-in primitive for surrogate cascades: a cascade runs the cheap level over the full ensemble, the expensive level over a chosen subset, and calls this to lift the cheap ensemble toward the expensive one at a fraction of the cost.

Value

A `nreal x nobs` matrix of bias-corrected outputs, with attributes `"intercept"` (`a_j`), `"slope"` (`b_j`), and `"subset_cor"` (per-dimension subset correlation; `NA` for a degenerate dimension).

References

Kennedy, M. C. & O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1), 1–13.

See Also

`pesto_multifidelity_model()`.

Examples

```
set.seed(1L)
low_all  <- matrix(rnorm(40L), ncol = 2L)
sub      <- 1:5
low_sub  <- low_all[sub, , drop = FALSE]
high_sub <- 0.3 + 1.2 * low_sub + matrix(rnorm(10L, sd = 0.01), ncol = 2L)
corrected <- mf_control_variate(low_all, high_sub, low_sub)
attr(corrected, "slope")
```

`pesto_ensemble_manifest` *PESTO Ensemble Manifest (S7 class)*

Description

A versioned, hashed, provenance-tracked container for the result of a PESTO ensemble run. The manifest is the cross-package contract object that downstream consumers (`kernR`, `proxymix`, paperskill) read; it carries enough provenance to make a run independently reproducible and independently verifiable (via `verify_manifest()`).

Arguments

schema_version	Character. Semantic version of the manifest schema itself (default "1.0.0").
run_id	Character. Short identifier (auto-generated by as_manifest() if not supplied).
params	data.frame / data.table. Parameter ensemble; first column real_name, subsequent columns one per parameter.
outputs	data.frame / data.table. Simulated observation ensemble (same shape conventions as params).
weights	Named numeric. IES observation weights (1/obs_sd).
obs_target	Named numeric. Target observations.
seed	Integer scalar. RNG seed used (NA_integer_ if unrecorded).
data_hash	Character. sha256:... checksum over (params, outputs, weights, obs_target, seed).
fidelity	Structured provenance list or NULL. For a multi-fidelity run pesto_ies_callback() records list(type, schedule, final_level, n_levels, costs); NULL for a single-fidelity run. A legacy named-numeric tag is still accepted.
apsim_version	Character. APSIM version string (NA_character_ when forward model is non-APSIM).
pesto_version	Character. PESTO package version that produced the run.
timestamp	POSIXct. Construction time.
method	Character. One of "ies_callback", "ies_pst", "mda", "surrogate".
noptmax	Integer. Number of IES iterations.
lambda_schedule	Numeric vector. Marquardt lambdas per iteration.
failure_rate	Numeric in [0, 1]. Fraction of forward evaluations that returned NA.

Details

Construct directly via the class constructor, or — preferred — via as_manifest() applied to a pesto_ies_callback_result.

Persistence

See write_manifest() / read_manifest() for YAML serialisation and verify_manifest() for integrity checking. The YAML file is the authoritative human-readable entry; params, outputs, and the assimilation context (weights + obs_target) are emitted as sidecar files whose relative paths are recorded inside the YAML. Three sidecar modes:

- format = "rds" (default, **verifiable**) — RDS only. IEEE 754 doubles round-trip bit-exactly; SHA-256 integrity check holds.
- format = "both" (**verifiable**) — RDS sidecars plus parallel *_inspection.csv files. Hash bound to RDS; CSVs are decorative.
- format = "csv_unverified" (**not verifiable**, renamed from "csv" in PESTO 0.3.2) — CSV-only sidecars. Hash is recorded but cannot be recomputed from disk (CSV formatter precision loss ~1 ULP). verify_manifest() returns ok = NA + message. Use only for one-way export to non-R analysts; round-trip integrity is by construction broken.

The YAML carries an explicit `integrity:` field (`"verifiable"` / `"not_verifiable"`) derived from `format` so downstream non-R tools can branch on the integrity contract without needing to know the kernR/PESTO-specific format vocabulary.

Examples

```
set.seed(1)
npar <- 2L; nobs <- 3L; nreal <- 30L
G <- matrix(stats::rnorm(nobs * npar), nobs, npar)
y <- as.numeric(G %*% c(0.5, -1.0)) + stats::rnorm(nobs, sd = 0.05)
names(y) <- paste0("o", seq_len(nobs))
prior <- matrix(stats::rnorm(nreal * npar), nreal, npar,
                dimnames = list(NULL, c("a", "b")))
fit <- pesto_ies_callback(function(t) t %*% t(G), prior, y, 0.05,
                          noptmax = 3, verbose = FALSE)
m <- as_manifest(fit, seed = 1L)
verify_manifest(m)$ok
```

---

pesto_evaluate *Evaluate a PESTO forward model*

Description

Runs the forward model on a parameter matrix under its own failure policy and concurrency strategy, returning a shape-guaranteed `nreal x nobs` observation matrix. Failed realisations populate `NA` rows (when `on_failure = "na"`). The returned matrix carries two attributes: `"n_failures"` (integer count of `NA` rows) and `"fail_idx"` (integer realisation indices that failed).

Usage

```
pesto_evaluate(model, theta, ...)
```

Arguments

`model`	A `pesto_forward_model` (or, for the multi-fidelity method, a `pesto_multifidelity_model`).
`theta`	Numeric matrix, `nreal x npar`. Column names, when present, are checked against `model@param_names`.
`...`	Method-specific arguments. The multi-fidelity method accepts `level` (integer fidelity level to evaluate).

Value

An `nreal x nobs` numeric matrix with attributes `"n_failures"` and `"fail_idx"`.

See Also

`pesto_forward_model()`, `pesto_multifidelity_model()`.

Examples

```
fm <- pesto_forward_model(fn = function(theta) theta[, 1, drop = FALSE],
                          n_obs = 1L)
pesto_evaluate(fm, matrix(c(1, 2, 3), ncol = 1L))
```

`pesto_forward_model` *Forward-Model Contract (S7 class)*

Description

A `pesto_forward_model` wraps a user callable of signature `function(theta) -> obs` – where `theta` is an `nreal x npar` numeric matrix and `obs` is an `nreal x nobs` numeric matrix – in a typed object that owns the evaluation contract: output dimensionality, expected parameter names, the failure policy, the concurrency strategy, and a fidelity tag.

Usage

```
pesto_forward_model(
  fn,
  n_obs = NA_integer_,
  param_names = character(0),
  on_failure = "na",
  parallel = "serial",
  n_cores = NA_integer_,
  map_fn = NULL,
  max_fail_frac = 1,
  fidelity = 0L,
  label = NA_character_
)
```

Arguments

`fn`	Function. The forward model, signature `function(theta) -> obs`.
`n_obs`	Integer or `NA`. Known observation dimensionality. If `NA` (default) it is inferred from the first successful evaluation.
`param_names`	Character. Expected parameter column names. Empty (default) disables the column check.
`on_failure`	Character. `"na"` (default) records failed realisations as `NA` rows and proceeds; `"stop"` aborts on any failure.
`parallel`	Character. `"serial"` (default) or `"multicore"`.
`n_cores`	Integer or `NA`. Worker count for `"multicore"`. `NA` (default) resolves to `parallel::detectCores() - 1L` at evaluation time.
`map_fn`	Function or `NULL`. Optional `lapply`-shaped override `function(X, FUN, ...)`; when supplied it drives per-realisation dispatch regardless of `parallel`.

max_fail_frac	Numeric in [0, 1]. Abort if the fraction of failed realisations in any single evaluation exceeds this. Default 1 (never abort on fraction; on_failure still governs the zero-success case).
fidelity	Integer. Fidelity level tag (default 0L).
label	Character. Optional human label carried into diagnostics.

Details

This is the single contract both PESTO adapter modes honour. The native callback driver pesto_ies_callback() accepts either a bare function (auto-wrapped via as_forward_model() with that driver's failure policy) or a pesto_forward_model built here; the apsimx adapter apsim_callback() can emit one directly. Evaluation is via pesto_evaluate(), which guarantees the returned shape and accounts for failed realisations as NA rows.

Value

A pesto_forward_model S7 object.

Concurrency

With parallel = "serial" (the default) and no map_fn, evaluation attempts a single bulk call fn(theta) and falls back to a serial per-realisation loop only if the bulk call errors – preserving the fast path for vectorised forward models. With parallel = "multicore" realisations are dispatched per row through parallel::mclapply() (fork-based; silently serial on Windows). A custom map_fn (an lapply-shaped function(X, FUN, ...)) overrides both and lets callers plug in future.apply::future_lapply, mirai, or a cluster backend. For reproducible parallel runs set RNGkind("L'Ecuyer-CMRG") and set.seed() before evaluating; parallel::mclapply() then draws independent streams per realisation.

Fidelity

fidelity is an integer level tag (0L = base / cheapest by convention; higher = more expensive / higher resolution). A single-fidelity model leaves it at 0L. The tag is what pesto_multifidelity_model() uses to order levels and what the manifest emitter threads into provenance.

See Also

pesto_evaluate() to evaluate one; as_forward_model() to coerce a bare function; pesto_multifidelity_model() to compose several across fidelity levels; pesto_ies_callback() for the IES driver that consumes it.

Examples

```
# A vectorised linear forward model wrapped as a contract object.
G  <- matrix(c(1, 0, 0, 1, 1, -1), nrow = 3L, byrow = TRUE)
fm <- pesto_forward_model(
  fn          = function(theta) theta %*% t(G),
  n_obs       = 3L,
  param_names = c("a", "b")
)
```

```
theta <- matrix(c(1, 2, 3, 4), nrow = 2L, byrow = TRUE,
                dimnames = list(NULL, c("a", "b")))
pesto_evaluate(fm, theta)
```

`pesto_glm` *Run PEST++ GLM (Gauss-Levenberg-Marquardt)*

Description

Executes the pestpp-glm algorithm for deterministic parameter estimation.

Usage

```
pesto_glm(
  pst_file,
  exe = NULL,
  noptmax = 20,
  extra_args = list(),
  working_dir = NULL,
  verbose = TRUE
)
```

Arguments

`pst_file`	Character. Path to the .pst control file.
`exe`	Character. Path to pestpp-glm executable.
`noptmax`	Integer. Maximum number of iterations.
`extra_args`	Named list. Additional PEST++ arguments.
`working_dir`	Character. Working directory.
`verbose`	Logical. Print output.

Value

A list of class `pesto_glm_result`.

Examples

```
if (nzchar(Sys.which("pestpp-glm"))) {
  pars <- data.table::data.table(
    parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
    parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
    parubnd = c(100, 50), pargp = "hydraulic"
  )
  obs <- data.table::data.table(
    obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
    weight = c(1.0, 1.0), obgnme = "head"
  )
```

```
  pst <- create_pest_scenario(pars, obs, model_command = "echo run")
  tf <- tempfile(fileext = ".pst")
  on.exit(unlink(tf), add = TRUE)
  write_pst(pst, tf)
  res <- pesto_glm(tf, noptmax = 1, verbose = FALSE)
  res$exit_code
}
```

`pesto_ies` *Run PEST++ IES (Iterative Ensemble Smoother)*

Description

Executes the pestpp-ies algorithm on a PEST control file. This is the primary method for ensemble-based parameter estimation and uncertainty quantification.

Usage

```
pesto_ies(
  pst_file,
  exe = NULL,
  num_reals = 50,
  noptmax = 4,
  lambda_scale_fac = c(0.1, 0.5, 1),
  ies_par_en = NULL,
  extra_args = list(),
  working_dir = NULL,
  verbose = TRUE
)
```

Arguments

`pst_file`	Character. Path to the .pst control file.
`exe`	Character. Path to pestpp-ies executable. If NULL, uses the bundled binary.
`num_reals`	Integer. Number of ensemble realisations (overrides the value in the .pst file).
`noptmax`	Integer. Maximum number of iterations.
`lambda_scale_fac`	Numeric vector. Lambda scaling factors.
`ies_par_en`	Character. Path to existing parameter ensemble file.
`extra_args`	Named list. Additional PEST++ arguments.
`working_dir`	Character. Working directory for the run. Defaults to the directory containing the .pst file.
`verbose`	Logical. Print stdout/stderr from pestpp-ies.

Value

A list of class `pesto_ies_result` containing:

phi data.table of objective function values per iteration

par_ensemble Final parameter ensemble (data.table)

obs_ensemble Final observation ensemble (data.table)

exit_code Integer exit code from pestpp-ies

runtime_seconds Total wall-clock runtime

Examples

```
if (nzchar(Sys.which("pestpp-ies"))) {
  pars <- data.table::data.table(
    parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
    parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
    parubnd = c(100, 50), pargp = "hydraulic"
  )
  obs <- data.table::data.table(
    obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
    weight = c(1.0, 1.0), obgnme = "head"
  )
  pst <- create_pest_scenario(pars, obs, model_command = "echo run")
  tf <- tempfile(fileext = ".pst")
  on.exit(unlink(tf), add = TRUE)
  write_pst(pst, tf)
  res <- pesto_ies(tf, num_reals = 3, noptmax = 1, verbose = FALSE)
  res$exit_code
}
```

---

pesto_ies_callback *Run IES with an In-Process R Callback Forward Model*

Description

Drives an Iterative Ensemble Smoother entirely in R, using a user-supplied forward model callable instead of the PEST++ `.pst`-file invocation cycle. Each iteration:

Usage

```
pesto_ies_callback(
  forward_model,
  prior_ensemble,
  obs,
  obs_sd,
  noptmax = 4L,
  lambda = 1,
```

```
  fidelity_schedule = NULL,
  parcov = NULL,
  eigthresh = 1e-06,
  use_approx = TRUE,
  inflation = NULL,
  localisation = NULL,
  on_failure = c("na", "stop"),
  verbose = TRUE
)
```

Arguments

forward_model	One of: a function with signature `function(theta) -> obs` (where `theta` is an `nreal x npar` numeric matrix and `obs` an `nreal x nobs` numeric matrix); a `pesto_forward_model()` (the typed contract object, e.g. one carrying a `parallel = "multicore"` evaluation strategy); or a `pesto_multifidelity_model()` (then see `fidelity_schedule`). A bare function is auto-wrapped via `as_forward_model()` with this call's `on_failure`. Failed realisations may return rows of `NA`; the driver tolerates them (see `on_failure`). To run realisations in parallel, pass a `pesto_forward_model` built with `parallel = "multicore"` rather than a bare function.
prior_ensemble	Matrix or data.table, `nreal x npar`. Columns are parameters; an optional `real_name` column is preserved if present. Column names supply parameter names.
obs	Named numeric vector. Target observations.
obs_sd	Numeric scalar or vector of length `nobs`. Observation standard deviation(s); the IES weights are `1/obs_sd`.
noptmax	Integer. Number of IES iterations (default 4).
lambda	Numeric scalar or vector. Marquardt lambda per iteration. A scalar is recycled; a vector shorter than `noptmax` is right-padded with its last value (default 1.0).
fidelity_schedule	Integer vector or `NULL`. Only consulted when `forward_model` is a `pesto_multifidelity_model()`: the fidelity level to evaluate at each iteration (recycled / right-padded to `noptmax`). `NULL` (default) uses the highest fidelity every iteration. Ignored otherwise.
parcov	Numeric vector of length `npar`, the diagonal of the prior parameter covariance. Defaults to the column-wise variance of `prior_ensemble`; zero or negative entries are replaced with 1.0.
eigthresh	Numeric. SVD eigenvalue truncation threshold (default 1e-6).
use_approx	Logical. If TRUE (default), skip the prior-scaling correction (upgrade_2); matches the typical `pestpp-ies` default.
inflation	A `pesto_inflation()` specification, or `NULL` (default) for no covariance inflation. Inflation counteracts ensemble under-dispersion – the progressive collapse of posterior spread a finite-ensemble smoother suffers. `NULL` leaves the update identical to the un-inflated smoother.

localisation	A pesto_localisation() specification, or NULL (default) for no localisation. Localisation tapers the Kalman gain to suppress spurious finite-sample parameter-observation correlations; the active path uses ensemble_solution_localised() (approximate / upgrade_1 form), so a non-NULL localisation with use_approx = FALSE warns and drops the null-space correction.
on_failure	Character. "na" (default) carries failed realisations forward unchanged and proceeds; "stop" aborts on any failure.
verbose	Logical. Print per-iteration phi summaries.

Details

1. Evaluates forward_model(par_ensemble) to obtain simulated observations.
2. Forms parameter / observation anomalies and residuals.
3. Calls the C++ kernel ensemble_solution() (GLM form, Chen & Oliver 2013) to compute an nreal x npar upgrade.
4. Adds the upgrade to the current ensemble.

The classic .pst-file path remains available via pesto_ies() for full PEST++ compatibility. Use this callback driver when the forward model is itself an R function (e.g. an apsimx wrapper from apsim_callback(), a Python bridge, or a synthetic test problem) and the per-realisation file-I/O overhead of the .pst path is the bottleneck.

Phase-1 behaviour: single lambda per iteration (or a user-supplied schedule). A line-search over ies_lambda_mults matching pestpp-ies is a planned Phase-2 enhancement; for the common case of a well-behaved forward model with lambda = 1, the GLM update reduces phi reliably (see vignette apsim-callback).

Value

A list of class c("pesto_ies_callback_result", "pesto_ies_result") with components:

phi data.table of per-realisation phi by iteration.

par_ensemble Final parameter ensemble (data.table).

obs_ensemble Final simulated-observation ensemble (data.table).

iterations List of per-iteration metadata: lambda, mean_phi, n_failures, and the dispersion diagnostics spread_ess / spread_ess_ratio (ensemble_spread_ess()), plus inflation_method / inflation_factor / retention and localisation / loc_threshold / loc_frac_active when those countermeasures are active.

runtime_seconds Total wall-clock runtime.

n_forward_evals Total number of realisation-level forward evaluations across all iterations (including the final refresh).

failure_rate Fraction of forward evaluations that returned NA.

fidelity For a pesto_multifidelity_model() run, a provenance list list(type, schedule, final_level, n_levels, costs) recording the realised per-iteration fidelity schedule; NULL for a single-fidelity run. Consumed by as_manifest() to populate the manifest contract.

Multi-fidelity

When `forward_model` is a `pesto_multifidelity_model()`, `fidelity_schedule` selects which fidelity level each iteration evaluates – a recycled / padded integer vector, one entry per iteration (default: the highest fidelity every iteration, i.e. exactly the single-fidelity behaviour). This supports fidelity ramping (cheap early iterations, expensive late ones); the final ensemble refresh always uses the highest fidelity so the returned posterior is at full resolution. The control-variate combiner `mf_control_variate()` is the plug-in point for bias-corrected surrogate cascades.

References

Chen, Y. & Oliver, D.S. (2013). Levenberg-Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17(4), 689–703.

See Also

`pesto_ies()` for the `.pst`-file path; `apsim_callback()` for the apsimx adapter.

Examples

```
# Linear-Gaussian recovery toy
set.seed(1)
npar <- 3; nobs <- 6; nreal <- 80
G <- matrix(rnorm(nobs * npar), nobs, npar)
theta_true <- c(1.0, -0.5, 2.0)
y <- as.numeric(G %*% theta_true) + rnorm(nobs, sd = 0.05)
f <- function(theta) theta %*% t(G)
prior <- matrix(rnorm(nreal * npar), nreal, npar,
                dimnames = list(NULL, paste0("p", 1:npar)))
fit <- pesto_ies_callback(
  forward_model = f, prior_ensemble = prior,
  obs = setNames(y, paste0("o", 1:nobs)), obs_sd = 0.05,
  noptmax = 5, verbose = FALSE
)
colMeans(as.matrix(fit$par_ensemble[, -1]))  # should approach theta_true
```

---

pesto_ies_filter *Run a Sequential (Filter-Mode) Iterative Ensemble Smoother*

Description

Assimilates observations in time-ordered **windows** against a static parameter ensemble. Each window runs an IES Gauss-Levenberg-Marquardt update using only that window's observation block; the updated ensemble is carried forward as the prior for the next window. This is the filtering analogue of `pesto_ies_callback()`: instead of one batch assimilation of all observations, the posterior is refined window by window and is available after each.

Usage

```
pesto_ies_filter(
  forward_model,
  prior_ensemble,
  obs,
  obs_sd,
  windows,
  window_noptmax = 1L,
  lambda = 1,
  fidelity_schedule = NULL,
  parcov = NULL,
  eigthresh = 1e-06,
  use_approx = TRUE,
  inflation = NULL,
  localisation = NULL,
  on_failure = c("na", "stop"),
  verbose = TRUE
)
```

Arguments

forward_model	A function `function(theta) -> obs`, a `pesto_forward_model()`, or a `pesto_multifidelity_model(` (then see `fidelity_schedule`). The model maps the full `nreal x npar` parameter matrix to the full `nreal x nobs` observation matrix; windows select columns of that output. A bare function is auto-wrapped via `as_forward_model()`. See `pesto_ies_callback()` for the contract.
prior_ensemble	Matrix or data.table, `nreal x npar`.
obs	Named numeric vector of length `nobs`. The full set of target observations; `windows` indexes into it.
obs_sd	Numeric scalar or length-`nobs` vector. Observation standard deviation(s); weights are `1 / obs_sd`.
windows	A list of integer vectors. Each element gives the observation indices (into `obs`, 1-based) assimilated at that window, in assimilation order. Windows must be **disjoint** (no observation assimilated twice) and need not cover all of `obs`.
window_noptmax	Integer scalar or vector. IES iterations *within* each window (default `1L`, the pure filter; `> 1` gives an iterated filter per window). A scalar is recycled; a short vector is right-padded with its last value.
lambda	Numeric scalar or per-window vector. Marquardt lambda (default `1.0`; recycled / right-padded across windows).
fidelity_schedule	Integer vector or `NULL`. Per-window fidelity level for a `pesto_multifidelity_model()` (see *Multi-fidelity*).
parcov	Numeric vector of length `npar`, the diagonal of the prior parameter covariance. Defaults to the prior ensemble's column-wise variance (non-positive entries replaced with 1).
eigthresh	Numeric. SVD eigenvalue truncation (default `1e-6`).

use_approx	Logical. If TRUE (default) skip the prior-scaling correction, matching the pestpp-ies default.
inflation	A pesto_inflation() specification, or NULL (default). Applied within each window's inner update to counteract ensemble under-dispersion. See pesto_ies_callback().
localisation	A pesto_localisation() specification, or NULL (default). Tapers the per-window Kalman gain to suppress spurious finite-sample correlations. See pesto_ies_callback().
on_failure	Character. "na" (default) tolerates failed realisations; "stop" aborts on any failure.
verbose	Logical. Print a per-window phi summary.

Value

A list of class c("pesto_ies_filter_result", "pesto_ies_result") with components:

phi data.table of per-realisation phi by window (on that window's observation block).

par_ensemble Final parameter ensemble (data.table).

obs_ensemble Final simulated-observation ensemble, full nobs columns (data.table).

windows List of per-window metadata: assimilated indices, lambda, mean phi, per-parameter ensemble mean and standard deviation (the sd trace shows the posterior tightening), and failure count.

runtime_seconds, n_forward_evals, failure_rate Run totals.

fidelity Multi-fidelity provenance (or NULL), as in pesto_ies_callback().

Filter vs smoother

pesto_ies_callback() forms parameter residuals against the *original* prior mean and assimilates every observation together. Here each window forms residuals against the *current* (carried-forward) ensemble mean and assimilates only its own block, so information accrues sequentially. With the default use_approx = TRUE the carried-forward ensemble itself encodes the background covariance, so the sequential behaviour comes from propagating the ensemble, not from an explicit prior term.

Multi-fidelity

When forward_model is a pesto_multifidelity_model(), fidelity_schedule selects the fidelity level evaluated at each *window* (recycled / padded to the number of windows; default: highest fidelity throughout). The final ensemble refresh always uses the highest fidelity.

References

Chen, Y. & Oliver, D.S. (2013). Levenberg-Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17(4), 689–703.

See Also

pesto_ies_callback() for the batch smoother; pesto_multifidelity_model() for fidelity stacks; as_manifest() to wrap the result in the ensemble-manifest contract.

Examples

```
# Linear-Gaussian recovery, assimilated in three observation windows.
set.seed(1)
npar <- 3; nobs <- 9; nreal <- 80
G <- matrix(rnorm(nobs * npar), nobs, npar)
theta_true <- c(1.0, -0.5, 2.0)
y <- as.numeric(G %*% theta_true) + rnorm(nobs, sd = 0.05)
f <- function(theta) theta %*% t(G)
prior <- matrix(rnorm(nreal * npar), nreal, npar,
                dimnames = list(NULL, paste0("p", 1:npar)))
fit <- pesto_ies_filter(
  forward_model  = f, prior_ensemble = prior,
  obs = setNames(y, paste0("o", 1:nobs)), obs_sd = 0.05,
  windows = list(1:3, 4:6, 7:9), verbose = FALSE
)
# Posterior sd should shrink window over window:
vapply(fit$windows, function(w) mean(w$par_sd), numeric(1))
```

`pesto_inflation` *Covariance Inflation Specification for IES*

Description

Builds a control object that tells `pesto_ies_callback()` and `pesto_ies_filter()` how to counteract ensemble under-dispersion — the progressive collapse of posterior spread that an iterative ensemble smoother suffers with a finite ensemble. Inflation re-expands the post-update parameter spread each iteration; the default `method = "none"` leaves the update byte-identical to the uninflated smoother.

Usage

```
pesto_inflation(
  method = c("none", "rtps", "adaptive", "multiplicative"),
  alpha = 0.5,
  factor = 1,
  retention_floor = 0.5,
  max_factor = 5
)
```

Arguments

`method`	Character. One of `"none"` (default), `"rtps"`, `"adaptive"`, `"multiplicative"`.
`alpha`	Numeric in [0, 1]. RTPS relaxation coefficient (default 0.5); used only when `method = "rtps"`.
`factor`	Numeric ≥ 1. Fixed inflation factor for `method = "multiplicative"` (default 1, i.e. no inflation).

`retention_floor`
: Numeric in (0, 1]. Target floor on the mean spread-retention ratio for `method = "adaptive"` (default 0.5).

`max_factor`
: Numeric ≥ 1. Upper bound on any single-iteration inflation factor for `"rtps"` and `"adaptive"` (default 5).

Details

Four methods are offered. `"rtps"` is relaxation-to-prior-spread (Whitaker & Hamill 2012): each parameter's posterior anomalies are rescaled by $\alpha(\sigma^b - \sigma^a)/\sigma^a + 1$, where σ^b and σ^a are the background (pre-update) and analysis (post-update) standard deviations. Being per-parameter, it re-inflates the directions that collapsed hardest, so it is the spectrally-aware workhorse. `"adaptive"` is a global, magnitude-targeting scheme: it measures the mean spread-retention ratio $q = \mathrm{mean}_j(\sigma^a_j/\sigma^b_j)$ and, when q falls below `retention_floor`, applies a single multiplicative factor $\min(\texttt{max\_factor}, \texttt{retention\_floor}/q)$ to restore the lost variance magnitude. `"multiplicative"` applies a fixed `factor` every iteration. `"none"` disables inflation.

The companion *diagnostic* is the spectral spread-ESS (`ensemble_spread_ess()`), recorded each iteration regardless of method: it reports the effective number of variance-carrying directions and is what detects directional collapse. Because that participation ratio is invariant to a global rescaling, a global (`"multiplicative"` / `"adaptive"`) inflation restores variance *magnitude* but not the spectral *shape*; `"rtps"` is the method that reshapes the spectrum. The two compose well.

Value

An object of class `"pesto_inflation"`.

References

Whitaker, J.S. & Hamill, T.M. (2012). Evaluating methods to account for system errors in ensemble data assimilation. *Monthly Weather Review*, 140(9), 3078–3089.

See Also

`pesto_localisation()`, `ensemble_spread_ess()`, `pesto_ies_callback()`.

Examples

```
inf <- pesto_inflation("rtps", alpha = 0.5)
inf
```

`pesto_localisation` *Covariance Localisation Specification for IES*

Description

Builds a control object that tells `pesto_ies_callback()` and `pesto_ies_filter()` how to taper the ensemble Kalman gain, suppressing the spurious long-range parameter-observation correlations that a finite ensemble manufactures. Localisation is applied as a Schur (elementwise) product on the explicit gain inside `ensemble_solution_localised()`; the default `method = "none"` leaves the standard SVD update untouched.

Usage

```
pesto_localisation(
  method = c("none", "correlation", "distance"),
  taper = c("hard", "soft"),
  threshold = -1,
  n_shuffle = 1L,
  quantile = 0.95,
  distances = NULL,
  par_coords = NULL,
  obs_coords = NULL,
  radius = NULL
)
```

Arguments

`method`	Character. One of `"none"` (default), `"correlation"`, `"distance"`.
`taper`	Character. `"hard"` (default) or `"soft"`; passed to `correlation_localisation()` for `method = "correlation"`.
`threshold`	Numeric. Correlation noise floor; negative (default -1) triggers automatic per-iteration estimation. `method = "correlation"`.
`n_shuffle`	Integer ≥ 1. Permutation replicates for the automatic floor (default 1). `method = "correlation"`.
`quantile`	Numeric in (0, 1). Quantile of the spurious-correlation distribution used as the floor (default 0.95). `method = "correlation"`.
`distances`	Matrix (npar x nobs) or `NULL`. Precomputed parameter-to-observation distances for `method = "distance"`.
`par_coords, obs_coords`	Matrices (npar x d, nobs x d) or `NULL`. Parameter / observation coordinates; Euclidean distances are derived when `distances` is `NULL`. `method = "distance"`.
`radius`	Numeric (> 0) or `NULL`. Gaspari-Cohn localisation radius; required for `method = "distance"`.

Details

`"correlation"` is the iterative-ensemble-smoother-native automatic localisation of Luo & Bhakta (2020): it needs no parameter or observation coordinates, estimating a noise floor from the ensemble itself and damping sample correlations that fall below it (see `correlation_localisation()`). This is the recommended default for parameter-estimation problems whose parameters carry no spatial metric. `"distance"` is classical distance-based localisation: a Gaspari-Cohn taper (`gaspari_cohn()`)

of a parameter-to-observation distance matrix, for problems where such a metric exists — supply either `distances` directly or `par_coords` + `obs_coords` (Euclidean distances are then computed), together with `radius`.

Value

An object of class `"pesto_localisation"`.

References

Luo, X. & Bhakta, T. (2020). Automatic and adaptive localization for ensemble-based history matching. *Journal of Petroleum Science and Engineering*, 184, 106559.

See Also

`pesto_inflation()`, `correlation_localisation()`, `gaspari_cohn()`.

Examples

```
loc <- pesto_localisation("correlation", taper = "soft")
loc
```

---

pesto_multifidelity_model

Multi-Fidelity Forward Model (S7 class)

Description

Bundles an ordered stack of `pesto_forward_model()` levels – cheapest (`level = 0`) to most expensive (`level = n - 1`) – with their relative per-evaluation costs. This is the first-class form of the bridge's `(cheap, expensive)` fidelity vector: the IES driver `pesto_ies_callback()` selects a level per iteration via `fidelity_schedule`, and `mf_control_variate()` debiases a cheap level against a sparse expensive sample for surrogate cascades.

Usage

```
pesto_multifidelity_model(levels, costs = NULL, label = NA_character_)
```

Arguments

`levels`	List of `pesto_forward_model()` objects (or bare functions, which are coerced), ordered cheapest first.
`costs`	Numeric vector of relative per-evaluation costs, one per level, ascending by convention. Defaults to `seq_along(levels)`. Carried for cost-aware allocation; not yet used to schedule automatically (that is the documented extension point).
`label`	Character. Optional human label.

Details

Each element of `levels` may be a bare `function(theta) -> obs` or a fully-specified `pesto_forward_model()`; bare functions are coerced via `as_forward_model()`. Levels must be ordered by ascending fidelity (cheapest first) – the convention the `level` index and the `costs` vector both follow.

Value

A `pesto_multifidelity_model` S7 object.

See Also

`pesto_forward_model()`, `pesto_evaluate()`, `mf_control_variate()`, `pesto_ies_callback()`.

Examples

```
cheap     <- function(theta) theta %*% c(1, 1)        # fast, biased
expensive <- function(theta) theta %*% c(1, 1) + 0.5  # slow, truth
mf <- pesto_multifidelity_model(
  levels = list(
    pesto_forward_model(fn = cheap,     n_obs = 1L, fidelity = 0L),
    pesto_forward_model(fn = expensive, n_obs = 1L, fidelity = 1L)
  ),
  costs = c(1, 25)
)
theta <- matrix(c(1, 0, 0, 1), nrow = 2L, byrow = TRUE)
pesto_evaluate(mf, theta, level = 0L)  # cheap
pesto_evaluate(mf, theta, level = 1L)  # expensive
```

---

pesto_reference_ies *Pure-R Reference IES Update — Chen & Oliver (2013) eq. 12*

Description

A textbook, pure-R implementation of one Iterative Ensemble Smoother (IES) parameter upgrade as published in Chen & Oliver (2013) eq. 12. This function is independent of `ensemble_solution()` (the C++ kernel) and exists for two purposes:

Usage

```
pesto_reference_ies(
  par_ensemble,
  obs_ensemble,
  obs_target,
  weights,
  lambda = 1
)
```

Arguments

par_ensemble	Numeric matrix (n_par x n_real). Current parameter ensemble (each column is one realisation).
obs_ensemble	Numeric matrix (n_obs x n_real). Simulated observations from the current ensemble.
obs_target	Numeric vector (n_obs). Target (measured) observation values.
weights	Numeric vector (n_obs). Observation weights (typically 1 / sd(obs_noise)).
lambda	Numeric scalar. Marquardt damping parameter. 1.0 is the textbook GLM update; larger values dampen more aggressively.

Details

1. **Independent numerical validation.** PESTO's C++ kernel can be cross-checked against this reference at machine precision (the package's regression test tests/testthat/test-ensemble-solution-sign.R and the paired-seed protocol in inst/scripts/i2_paired_seed_check.R both rely on this).
2. **Self-contained** pestpp-ies **comparison.** Vignettes can compare PESTO native IES to this textbook reference without requiring the upstream pestpp-ies binary, making the comparison story reproducible on CRAN check farms and other binary-free environments.

The implementation is deliberately pedagogical — readability over speed. For production work use ensemble_solution() (the C++ kernel; $\sim 35\times$ faster on production-scale ensembles).

Value

A numeric matrix (n_par x n_real): the parameter upgrade (add to par_ensemble to get the next iterate).

Sign convention

This reference uses the textbook sign obs_resid = obs - sim with a positive leading sign in the upgrade. PESTO's C++ kernel uses the equivalent obs_resid = sim - obs with a leading negative sign (see ?ensemble_solution). Both yield identical upgrades; the difference is purely cosmetic. The two are cross-validated to machine precision in inst/scripts/i2_paired_seed_check.R.

References

Chen, Y. & Oliver, D. S. (2013). Levenberg-Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17(4), 689–703. doi:10.1007/s1059601393515

Evensen, G. (2018). Analysis of iterative ensemble smoothers for solving inverse problems. *Computational Geosciences*, 22(3), 885–908. doi:10.1007/s105960189731y

See Also

ensemble_solution() for the production C++ kernel.

Examples

```
# Simple linear inverse problem: identify the true theta given noisy obs.
set.seed(20260425L)
n_par <- 4L; n_obs <- 12L; n_real <- 30L
G <- matrix(rnorm(n_obs * n_par) / sqrt(n_par), n_obs, n_par)
theta_true <- rnorm(n_par)
y_obs <- as.numeric(G %*% theta_true) + rnorm(n_obs, sd = 0.05)
weights <- rep(1 / 0.05, n_obs)

par_ens <- matrix(rnorm(n_par * n_real, sd = 1), n_par, n_real)
obs_ens <- G %*% par_ens

# One textbook IES upgrade.
upg <- pesto_reference_ies(
  par_ensemble = par_ens,
  obs_ensemble = obs_ens,
  obs_target   = y_obs,
  weights      = weights,
  lambda       = 1.0
)
dim(upg)

# Apply the upgrade and check phi reduces.
par_next <- par_ens + upg
obs_next <- G %*% par_next
Y <- matrix(rep(y_obs, n_real), n_obs, n_real)
phi0 <- mean(compute_phi(Y - obs_ens,  weights))
phi1 <- mean(compute_phi(Y - obs_next, weights))
phi1 < phi0
```

`pesto_sensitivity` *Run PEST++ SEN (Global Sensitivity Analysis)*

Description

Executes pestpp-sen for Morris or Sobol sensitivity analysis.

Usage

```
pesto_sensitivity(
  pst_file,
  method = c("morris", "sobol"),
  exe = NULL,
  extra_args = list(),
  working_dir = NULL,
  verbose = TRUE
)
```

Arguments

pst_file	Character. Path to the .pst control file.
method	Character. "morris" or "sobol".
exe	Character. Path to pestpp-sen executable.
extra_args	Named list. Additional options.
working_dir	Character. Working directory.
verbose	Logical. Print output.

Value

A list of class pesto_sen_result.

Examples

```
if (nzchar(Sys.which("pestpp-sen"))) {
  pars <- data.table::data.table(
    parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
    parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
    parubnd = c(100, 50), pargp = "hydraulic"
  )
  obs <- data.table::data.table(
    obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
    weight = c(1.0, 1.0), obgnme = "head"
  )
  pst <- create_pest_scenario(pars, obs, model_command = "echo run")
  tf <- tempfile(fileext = ".pst")
  on.exit(unlink(tf), add = TRUE)
  write_pst(pst, tf)
  res <- pesto_sensitivity(tf, method = "morris", verbose = FALSE)
  res$method
}
```

---

pesto_surrogate_ies *Surrogate-Accelerated IES Iteration*

Description

Performs a single IES iteration using a Gaussian Process surrogate to reduce the number of expensive full-model evaluations.

Usage

```
pesto_surrogate_ies(
  par_ensemble,
  obs_ensemble,
  obs_target,
```

```
  weights,
  parcov_inv,
  lambda = 1,
  uncertainty_threshold = 0.1,
  eigthresh = 1e-06
)
```

Arguments

par_ensemble	data.table or matrix. Current parameter ensemble (rows = realisations, columns = parameters).
obs_ensemble	data.table or matrix. Current observation ensemble from model evaluations.
obs_target	Named numeric vector. Target observation values.
weights	Numeric vector. Observation weights.
parcov_inv	Numeric vector. Diagonal of inverse parameter covariance.
lambda	Numeric. Marquardt lambda (default 1.0).
uncertainty_threshold	Numeric. Threshold for surrogate/model switching. Fraction of signal variance (default 0.1 = 10%).
eigthresh	Numeric. SVD eigenvalue threshold.

Details

How it works:

1. A GP surrogate is trained on existing parameter-observation pairs
2. The surrogate predicts model outputs for all ensemble members
3. Only members with high prediction uncertainty trigger full model runs
4. Control-variate bias correction blends surrogate and model results
5. Standard IES update is computed on the blended ensemble

This typically saves 50-90% of model evaluations per iteration.

Value

A list containing:

upgrade Matrix of parameter upgrades

n_model_runs Number of full model evaluations needed

n_surrogate_runs Number of surrogate-only evaluations

savings_pct Percentage of model runs saved

gp_diagnostics GP training diagnostics

References

Rasmussen, C.E. & Williams, C.K.I. (2006). Gaussian Processes for Machine Learning. MIT Press.

Liu, F. & Guillas, S. (2017). Dimension reduction for Gaussian process emulation. *Statistics and Computing*, 27(3), 785-802.

Examples

```
set.seed(7L)
n_real <- 15L; n_par <- 5L; n_obs <- 8L
par_ens <- matrix(rnorm(n_real * n_par), n_real, n_par,
                  dimnames = list(NULL, paste0("k", 1:n_par)))
obs_ens <- matrix(rnorm(n_real * n_obs), n_real, n_obs,
                  dimnames = list(NULL, paste0("h", 1:n_obs)))
obs_target <- rnorm(n_obs)
weights    <- rep(1.0, n_obs)
parcov_inv <- rep(1.0, n_par)
res <- pesto_surrogate_ies(
  par_ensemble = par_ens,
  obs_ensemble = obs_ens,
  obs_target   = obs_target,
  weights      = weights,
  parcov_inv   = parcov_inv,
  lambda       = 1.0
)
res$savings_pct
```

`pesto_sweep` *Run PEST++ SWP (Parametric Sweep)*

Description

Executes pestpp-swp for embarrassingly parallel model runs across a parameter ensemble.

Usage

```
pesto_sweep(
  pst_file,
  par_ensemble,
  exe = NULL,
  working_dir = NULL,
  verbose = TRUE
)
```

Arguments

pst_file	Character. Path to the .pst control file.
par_ensemble	data.table or path. Parameter ensemble.
exe	Character. Path to pestpp-swp executable.
working_dir	Character. Working directory.
verbose	Logical. Print output.

Value

A list containing observation outputs for each realisation.

Examples

```
if (nzchar(Sys.which("pestpp-swp"))) {
  pars <- data.table::data.table(
    parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
    parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
    parubnd = c(100, 50), pargp = "hydraulic"
  )
  obs <- data.table::data.table(
    obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
    weight = c(1.0, 1.0), obgnme = "head"
  )
  pst <- create_pest_scenario(pars, obs, model_command = "echo run")
  tf <- tempfile(fileext = ".pst")
  on.exit(unlink(tf), add = TRUE)
  write_pst(pst, tf)
  par_ens <- data.table::data.table(
    real_name = c("r1", "r2", "r3"),
    k1 = c(0.8, 1.0, 1.2),
    k2 = c(0.4, 0.5, 0.6)
  )
  res <- pesto_sweep(tf, par_ensemble = par_ens, verbose = FALSE)
  res$exit_code
}
```

---

pesto_version *Get PESTO package version information*

Description

Returns version info for both the PESTO R package and the bundled PEST++ binaries.

Usage

```
pesto_version()
```

Value

A list with version strings.

Examples

```
v <- pesto_version()
v$pesto_version
v$platform
```

`plot_ensemble` *Plot Ensemble Parameter Distributions*

Description

Visualises the prior and/or posterior parameter ensemble distributions as violin plots or density ridges.

Usage

```
plot_ensemble(
  ensemble,
  parameters = NULL,
  prior_ensemble = NULL,
  max_params = 20L,
  title = "Parameter Ensemble Distributions"
)
```

Arguments

`ensemble`	data.table. Parameter ensemble (rows = realisations).
`parameters`	Character vector. Parameter names to plot. If NULL, selects up to 20 parameters with highest variance.
`prior_ensemble`	data.table. Optional prior ensemble for comparison.
`max_params`	Integer. Maximum parameters to display.
`title`	Character. Plot title.

Value

A ggplot2 object.

Examples

```
posterior <- data.table::data.table(
  real_name = sprintf("real_%02d", 1:50),
  k1 = rnorm(50, 1.0, 0.15),
  k2 = rnorm(50, 0.5, 0.05),
  k3 = rnorm(50, 2.0, 0.40),
  k4 = rnorm(50, 0.1, 0.02)
)
prior <- data.table::data.table(
  real_name = sprintf("real_%02d", 1:50),
  k1 = rnorm(50, 1.0, 0.50),
  k2 = rnorm(50, 0.5, 0.20),
  k3 = rnorm(50, 2.0, 1.00),
  k4 = rnorm(50, 0.1, 0.08)
)
p <- plot_ensemble(posterior, prior_ensemble = prior)
inherits(p, "ggplot")
```

---

plot_identifiability *Plot Parameter Identifiability*

Description

Creates a bar plot of parameter identifiability based on the singular value decomposition of the Jacobian matrix. Accepts either a numeric Jacobian matrix in memory or a path to a `.jco` (PEST binary) file.

Usage

```
plot_identifiability(
  jacobian = NULL,
  jco_file = NULL,
  pst = NULL,
  n_sv = NULL,
  title = "Parameter Identifiability"
)
```

Arguments

`jacobian`	Numeric matrix (n_obs x n_par). The Jacobian / sensitivity matrix. Column names, if present, are used as parameter labels; otherwise `p1`, `p2`, ... are generated. Either `jacobian` or `jco_file` must be supplied.
`jco_file`	Character. Path to a `.jco` (Jacobian) binary file. Mutually exclusive with `jacobian`.
`pst`	A `pesto_pst` object for parameter names. Optional; only used when reading from a `.jco` file and column names are absent.
`n_sv`	Integer. Number of singular values to retain.
`title`	Character. Plot title.

Value

A ggplot2 object.

Examples

```
J <- matrix(rnorm(30 * 8), nrow = 30, ncol = 8)
J[, 7] <- 0.5 * J[, 1] + 0.5 * J[, 2]
J[, 8] <- 1e-6 * rnorm(30)
colnames(J) <- paste0("k", 1:8)
p <- plot_identifiability(jacobian = J)
inherits(p, "ggplot")
```

---

plot_phi *Plot Objective Function (Phi) Convergence*

Description

Creates a publication-quality plot of objective function values across iterations, showing mean, min, max, and individual realisation traces.

Usage

```
plot_phi(
  result,
  log_scale = TRUE,
  show_reals = FALSE,
  title = "Objective Function Convergence"
)
```

Arguments

result	A `pesto_ies_result` or `pesto_glm_result` object, or a data.table with columns `iteration` and `phi` (at minimum).
log_scale	Logical. Use log10 scale for y-axis (default TRUE).
show_reals	Logical. Show individual realisation traces.
title	Character. Plot title.

Value

A ggplot2 object.

Examples

```
phi_dt <- data.table::data.table(
  iteration = 0:4,
  total_runs = c(50L, 100L, 150L, 200L, 250L),
  mean = c(1200, 450, 180, 95, 72),
  min  = c(900, 320, 130, 70, 55),
  max  = c(1700, 680, 260, 140, 105),
  median = c(1180, 440, 175, 92, 70),
  std    = c(180, 80, 35, 18, 12)
)
p <- plot_phi(phi_dt, log_scale = TRUE,
              title = "Synthetic Phi Convergence")
inherits(p, "ggplot")
```

---

plot_surrogate_diagnostics

Plot Surrogate Diagnostics

Description

Visualises the surrogate-accelerated IES performance including model savings, uncertainty distribution, and GP quality metrics.

Usage

```
plot_surrogate_diagnostics(results, title = "Surrogate IES Diagnostics")
```

Arguments

results	List of surrogate update results from multiple iterations.
title	Character. Plot title.

Value

A ggplot2 object.

Examples

```
iter1 <- list(n_model_runs = 12L, n_surrogate_runs = 38L,
              savings_pct = 76.0, mean_uncertainty = 0.18)
iter2 <- list(n_model_runs = 8L,  n_surrogate_runs = 42L,
              savings_pct = 84.0, mean_uncertainty = 0.11)
iter3 <- list(n_model_runs = 5L,  n_surrogate_runs = 45L,
              savings_pct = 90.0, mean_uncertainty = 0.07)
p <- plot_surrogate_diagnostics(list(iter1, iter2, iter3))
inherits(p, "ggplot")
```

predict_gp_surrogate *Predict with GP Surrogate (with Uncertainty)*

Description

Generates predictions and prediction uncertainties for new parameter sets using a trained GP surrogate. The uncertainty estimates are crucial for the adaptive switching criterion.

Usage

```
predict_gp_surrogate(gp, X_new)
```

Arguments

gp	A trained GP model (from `train_gp_surrogate`).
X_new	Matrix (m x npar). New parameter sets to predict.

Value

A list with:

mean Matrix (m x nobs). Predicted observations.

variance Matrix (m x nobs). Prediction variance per output.

uncertainty Numeric vector (m). Mean prediction uncertainty per realisation.

Examples

```
set.seed(1L)
X_train <- matrix(rnorm(20 * 4), 20, 4)
Y_train <- matrix(rnorm(20 * 6), 20, 6)
gp      <- train_gp_surrogate(X_train, Y_train)
X_new   <- matrix(rnorm(5 * 4), 5, 4)
pred    <- predict_gp_surrogate(gp, X_new)
dim(pred$mean)
length(pred$uncertainty)
```

predict_rff_surrogate *Predict with RFF Sparse GP Surrogate*

Description

Predict with RFF Sparse GP Surrogate

Usage

```
predict_rff_surrogate(rff, X_new)
```

Arguments

rff	A trained RFF model (from train_rff_surrogate).
X_new	Matrix (m x npar). New parameter sets.

Value

A list with mean predictions and approximate uncertainties.

Examples

```
set.seed(1L)
X_train <- matrix(rnorm(30 * 4), 30, 4)
Y_train <- matrix(rnorm(30 * 6), 30, 6)
rff     <- train_rff_surrogate(X_train, Y_train, n_features = 100L)
X_new   <- matrix(rnorm(5 * 4), 5, 4)
pred    <- predict_rff_surrogate(rff, X_new)
dim(pred$mean)
length(pred$uncertainty)
```

---

print.pesto_pst *Print method for pesto_pst objects*

Description

Print method for pesto_pst objects

Usage

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

Arguments

x	A pesto_pst object.
...	Ignored.

Value

Invisibly returns x. Called for the side effect of printing.

Examples

```
pars <- data.table::data.table(
  parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
  parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
  parubnd = c(100, 50), pargp = "hydraulic"
)
obs <- data.table::data.table(
  obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
  weight = c(1.0, 1.0), obgnme = "head"
)
pst <- create_pest_scenario(pars, obs, model_command = "echo run")
print(pst)
```

---

read_ensemble *Read an Ensemble File*

Description

Reads PEST++ ensemble files in CSV or binary (.jcb/.jco) format.

Usage

```
read_ensemble(file, format = c("csv", "binary"))
```

Arguments

file	Character. Path to ensemble file.
format	Character. One of "csv" (default) or "binary".

Value

A data.table with realisations as rows and parameters/observations as columns. The first column `real_name` contains realisation names.

See Also

`write_ensemble()`

Examples

```
ens <- data.table::data.table(
  real_name = sprintf("real_%02d", 1:10),
  k1 = rnorm(10, mean = 1.0, sd = 0.2),
  k2 = rnorm(10, mean = 0.5, sd = 0.1),
  k3 = rnorm(10, mean = 2.0, sd = 0.3)
)
tf <- tempfile(fileext = ".csv")
on.exit(unlink(tf), add = TRUE)
write_ensemble(ens, tf)
```

```
ens_back <- read_ensemble(tf, format = "csv")
identical(names(ens_back), names(ens))
nrow(ens_back) == nrow(ens)
```

`read_manifest` *Read a manifest from YAML + sidecar data files*

Description

Inverse of `write_manifest()`. Reads the YAML, loads the three sidecar data files (paths resolved relative to the YAML file), and reconstructs the `pesto_ensemble_manifest` S7 object. The file extensions in the YAML's `artefacts:` block determine the read path (`.rds` via `readRDS`, `.csv` via `utils::read.csv`).

Usage

```
read_manifest(file)
```

Arguments

`file`	Character. Path to the YAML manifest file.

Value

A `pesto_ensemble_manifest`.

`read_pst` *Read a PEST Control File (.pst)*

Description

Parses a PEST/PEST++ control file and returns a structured list containing all sections: control data, parameter groups, parameter data, observation groups, observation data, model command, and template/instruction file pairs.

Usage

```
read_pst(file)
```

Arguments

`file`	Character. Path to the .pst file.

Value

A list of class `pesto_pst` containing:

control_data Control section parameters (NPAR, NOBS, etc.)

parameter_groups data.table of parameter group definitions

parameters data.table of parameter data

observation_groups data.table of observation group definitions

observations data.table of observation data

model_command Character vector of model command lines

template_files data.table of template/model input file pairs

instruction_files data.table of instruction/model output file pairs

prior_information data.table of prior information (if present)

pestpp_options Named list of ++ options

See Also

`write_pst()`, `create_pest_scenario()`

Examples

```
pars <- data.table::data.table(
  parnme = c("k1", "k2", "k3"),
  partrans = c("log", "log", "none"),
  parchglim = "factor",
  parval1 = c(1.0, 0.5, 0.1),
  parlbnd = c(0.01, 0.001, 0.0),
  parubnd = c(100, 50, 1.0),
  pargp = c("hydraulic", "hydraulic", "storage")
)
obs <- data.table::data.table(
  obsnme = c("h1", "h2", "h3"),
  obsval = c(1.0, 2.0, 1.5),
  weight = c(1.0, 1.0, 0.5),
  obgnme = "head"
)
pst <- create_pest_scenario(pars, obs, model_command = "echo run")
tf <- tempfile(fileext = ".pst")
on.exit(unlink(tf), add = TRUE)
write_pst(pst, tf)
pst_back <- read_pst(tf)
pst_back$control_data$npar
pst_back$control_data$nobs
```

rsvd *Randomised SVD (Halko-Martinsson-Tropp Algorithm)*

Description

Computes a rank-k approximation to the SVD using randomised projections. This is asymptotically faster than full SVD for problems where the target rank k is much smaller than min(m,n).

Usage

```
rsvd(A, k, p = 10L, q = 2L)
```

Arguments

A	Matrix (m x n). Input matrix.
k	Integer. Target rank (number of singular values to compute).
p	Integer. Oversampling parameter (default 10). Higher = more accurate.
q	Integer. Number of power iterations (default 2). Higher = better for matrices with slowly decaying singular values.

Details

Complexity: O(mn*k) vs O(mn*min(m,n)) for full SVD.

Value

A list with components U (m x k), d (k), V (n x k).

References

Halko, N., Martinsson, P.G., & Tropp, J.A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2), 217-288.

Examples

```
set.seed(1L)
A <- matrix(rnorm(10 * 6), nrow = 10, ncol = 6)
res <- rsvd(A, k = 3L)
length(res$d)
A_hat <- res$u %*% diag(res$d) %*% t(res$v)
mean((A - A_hat)^2)
```

surrogate_ensemble_update *Surrogate-Accelerated Ensemble Update*

Description

Performs an IES update step using a GP surrogate for cheap pre-screening, with adaptive switching to the full model based on prediction uncertainty.

Usage

```
surrogate_ensemble_update(
  par_ensemble,
  obs_ensemble,
  obs_target,
  weights,
  parcov_inv,
  cur_lam = 1,
  uncertainty_threshold = 0.1,
  eigthresh = 1e-06
)
```

Arguments

par_ensemble	Matrix (nreal x npar). Current parameter ensemble.
obs_ensemble	Matrix (nreal x nobs). Current observation ensemble (from model).
obs_target	Numeric vector (nobs). Target observations.
weights	Numeric vector (nobs). Observation weights.
parcov_inv	Numeric vector (npar). Inverse parameter covariance diagonal.
cur_lam	Numeric. Marquardt lambda.
uncertainty_threshold	Numeric. Threshold for surrogate/model switching. Realisations with GP uncertainty above this are re-evaluated with full model. Default 0.1 (10% of signal variance).
eigthresh	Numeric. SVD eigenvalue threshold.

Details

Algorithm:

1. Train GP surrogate from current ensemble evaluations
2. Generate candidate upgrades using surrogate predictions
3. Evaluate uncertainty of surrogate predictions
4. Run full model only for realisations where uncertainty exceeds threshold
5. Blend surrogate and model results using control-variate correction

This typically reduces full model evaluations by 50-90%.

Value

A list with:

upgrade Matrix. Parameter upgrades.

n_model_runs Integer. Number of full model evaluations needed.

n_surrogate_runs Integer. Number of surrogate evaluations.

savings_pct Numeric. Percentage of model runs saved.

gp_diagnostics List. GP training diagnostics.

Examples

```
set.seed(1L)
npar  <- 5L
nreal <- 15L
nobs  <- 8L
par_ensemble <- matrix(rnorm(nreal * npar), nreal, npar)
obs_ensemble <- matrix(rnorm(nreal * nobs), nreal, nobs)
obs_target   <- rnorm(nobs)
weights      <- rep(1, nobs)
parcov_inv   <- rep(1, npar)
res <- surrogate_ensemble_update(
  par_ensemble = par_ensemble,
  obs_ensemble = obs_ensemble,
  obs_target   = obs_target,
  weights      = weights,
  parcov_inv   = parcov_inv,
  cur_lam      = 1.0
)
dim(res$upgrade)
res$savings_pct
```

---

train_gp_surrogate *Train a Gaussian Process Surrogate*

Description

Trains a GP surrogate model from parameter-observation pairs. Uses squared exponential (RBF) kernel with automatic relevance determination (ARD) via median heuristic for length scale.

Usage

```
train_gp_surrogate(
  X_train,
  Y_train,
  length_scale = 0,
  signal_var = 0,
  noise_var = 1e-04
)
```

Arguments

X_train	Matrix (n x npar). Training parameter sets.
Y_train	Matrix (n x nobs). Corresponding model outputs.
length_scale	Numeric. Kernel length scale. If 0 (default), uses the median heuristic (median pairwise distance).
signal_var	Numeric. Signal variance. If 0, uses variance of Y.
noise_var	Numeric. Observation noise variance.

Details

The GP learns the mapping: parameters -> observations, enabling cheap prediction of model outputs for new parameter sets.

Value

A list of class step_gp containing trained GP components: K_inv (inverse kernel matrix), alpha (weight vectors), hyperparameters.

Examples

```
set.seed(1L)
X_train <- matrix(rnorm(20 * 4), 20, 4)
Y_train <- matrix(rnorm(20 * 6), 20, 6)
gp <- train_gp_surrogate(X_train, Y_train)
pred <- predict_gp_surrogate(gp, X_train)
dim(pred$mean)
length(pred$uncertainty)
```

---

train_rff_surrogate *Train a Sparse GP Surrogate via Random Fourier Features*

Description

Approximates the RBF kernel GP using random Fourier features, reducing training cost from O(n^3) to O(n * D^2) where D is the number of random features (typically 100-500). This enables GP surrogates for ensembles of 1,000+ realisations.

Usage

```
train_rff_surrogate(
  X_train,
  Y_train,
  n_features = 200L,
  length_scale = 0,
  noise_var = 1e-04
)
```

Arguments

X_train	Matrix (n x npar). Training parameter sets.
Y_train	Matrix (n x nobs). Corresponding model outputs.
n_features	Integer. Number of random Fourier features (default 200).
length_scale	Numeric. Kernel length scale (0 = median heuristic).
noise_var	Numeric. Observation noise variance.

Details

The RBF kernel k(x,x') = sigma^2 exp(-||x-x'||^2 / 2l^2) is approximated by k(x,x') ~ z(x)^T z(x') where z(x) = sqrt(2/D) * cos(W*x* + *b), where W are random frequencies. with w_j ~ N(0, I/l^2) and b_j ~ Uniform(0, 2*pi).

Value

A list containing the trained RFF model.

Examples

```
set.seed(1L)
X_train <- matrix(rnorm(30 * 4), 30, 4)
Y_train <- matrix(rnorm(30 * 6), 30, 6)
rff <- train_rff_surrogate(X_train, Y_train, n_features = 100L)
rff$train_mse
X_new <- matrix(rnorm(5 * 4), 5, 4)
pred  <- predict_rff_surrogate(rff, X_new)
dim(pred$mean)
```

---

verify_manifest *Verify the integrity of a manifest*

Description

Recomputes the SHA-256 hash over `(params, outputs, weights, obs_target, seed)` and compares against the stored `data_hash`. Use this after `read_manifest()` to confirm the data files have not been tampered with or silently re-saved.

Usage

```
verify_manifest(manifest, ...)
```

Arguments

manifest	A `pesto_ensemble_manifest`.
...	Reserved.

Details

When the manifest's `format` slot is `"csv"`, the on-disk data has been round-tripped through `utils::read.csv()` and IEEE 754 doubles have been truncated at the formatter's precision (~15-17 digits). That precision loss is enough to flip the SHA-256 hash, so `verify_manifest()` returns `ok = NA` with an explanatory `message` field rather than reporting a spurious `FALSE`.

Value

A list with `ok` (`TRUE`, `FALSE`, or `NA` — see Details), `stored` (the manifest's recorded hash), `recomputed` (the hash computed from current data), and `message` (`NULL` for verifiable formats, otherwise an explanation).

`write_ensemble` *Write an Ensemble File*

Description

Writes an ensemble data.table to CSV format compatible with PEST++.

Usage

```
write_ensemble(ensemble, file, format = "csv")
```

Arguments

`ensemble`	A data.table with realisation data.
`file`	Character. Output file path.
`format`	Character. Currently only "csv" is supported.

Value

Invisible `NULL`.

See Also

`read_ensemble()`

Examples

```
ens <- data.table::data.table(
  real_name = sprintf("real_%02d", 1:5),
  k1 = runif(5, 0.1, 10),
  k2 = runif(5, 0.01, 1)
)
tf <- tempfile(fileext = ".csv")
on.exit(unlink(tf), add = TRUE)
write_ensemble(ens, tf)
file.exists(tf)
```

`write_manifest` *Write a manifest to YAML + sidecar data files*

Description

Serialises a `pesto_ensemble_manifest` as a YAML file at `file` plus three data sidecars (`<basename>_params.<ext>`, `<basename>_outputs.<ext>`, `<basename>_assim.<ext>`) in the same directory. `<ext>` depends on `format`:

Usage

```
write_manifest(manifest, ...)
```

Arguments

`manifest`	A `pesto_ensemble_manifest`.
`...`	Method-specific arguments. For `pesto_ensemble_manifest`: `file` (character path to the YAML output; parent directory must exist; sidecars are written next to it) and `format` (one of `"rds"`, `"both"`, `"csv_unverified"`).

Details

- `"rds"` (default) — RDS sidecars only. IEEE 754 doubles round-trip bit-exactly; `verify_manifest()` recomputes the SHA-256 hash and confirms integrity.
- `"both"` — RDS sidecars **plus** parallel inspection CSVs (`<basename>_params_inspection.csv`, etc.). The hash is still bound to the RDS form; the CSVs are decorative only and are recorded in the YAML's `inspection_csv:` block.
- `"csv_unverified"` — CSV sidecars only. The hash is still recorded (computed from the in-memory binary representation) but `verify_manifest()` cannot recompute it from disk: CSV write-formatter precision loss (~1 ULP at IEEE 754 epsilon) would falsely fail the check. The YAML carries `integrity: not_verifiable` so downstream tools can branch accordingly. Renamed from `"csv"` in PESTO 0.3.2 (post critical review): the old name was indistinguishable at a glance from the verifiable modes, which the review judged a footgun. Old YAMLs with `format: csv` continue to read back correctly.

Value

Invisible character vector of the written paths (YAML + sidecars, in write order).

`write_pst` *Write a PEST Control File (.pst)*

Description

Writes a `pesto_pst` object to a PEST-format control file.

Usage

```
write_pst(pst, file)
```

Arguments

`pst`	A `pesto_pst` object (as returned by `read_pst()`).
`file`	Character. Output file path.

Value

Invisible `NULL`. File is written as a side effect.

See Also

`read_pst()`

Examples

```
pars <- data.table::data.table(
  parnme = c("k1", "k2"), partrans = "log", parchglim = "factor",
  parval1 = c(1.0, 0.5), parlbnd = c(0.01, 0.001),
  parubnd = c(100, 50), pargp = "hydraulic"
)
obs <- data.table::data.table(
  obsnme = c("h1", "h2"), obsval = c(1.0, 2.0),
  weight = c(1.0, 1.0), obgnme = "head"
)
pst <- create_pest_scenario(pars, obs, model_command = "echo run")
tf <- tempfile(fileext = ".pst")
on.exit(unlink(tf), add = TRUE)
write_pst(pst, tf)
file.exists(tf)
```

Index