Package 'gdpar'

July 15, 2026

Type Package

Title General Dynamic Parameter Models via Reference Anchoring

Version 0.1.0

Description Implements a unified predictive framework in which individual parameters are decomposed as theta_i equal to theta_ref plus Delta(x_i, theta_ref), with theta_ref a population reference and Delta an explicit deviation function. The decomposition follows the Additive-Multiplicative-Modulated canonical form and is estimated through three complementary paths: hierarchical Bayesian inference via 'Stan', varying-coefficient models via penalized splines, and amortized inference via hypernetworks in 'torch'. The package provides identifiability diagnostics, validity tests for the population reference, and benchmarks against canonical zero-inflated count datasets and avian abundance data from the eBird Status and Trends project. The framework and its estimation paths are described in Gomez Julian (2026) <doi:10.5281/zenodo.21046269>.

License GPL (>= 3)

Encoding UTF-8

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

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

Depends R (>= 4.2.0)

Imports posterior, stats, methods, withr

Suggests cmdstanr, loo, bayesplot, DHARMa, digest, mgcv, Matrix, pscl, AER, ebirdst, knitr, rmarkdown, testthat (>= 3.0.0), kableExtra, brms, INLA, rstanarm, scoringRules, gratia, grf, reticulate, mvnfast

Additional_repositories https://stan-dev.r-universe.dev, https://inla.r-inla-download.org/R/stable

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 8.0.0

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2026-07-15 18:10:02 UTC

Contents

amm_build *Start a chainable builder for an AMM specification*

Description

Initialise an empty amm_builder object that can be incrementally configured through a sequence of setter calls and finalised into an amm_spec via as_amm_spec. The builder pattern is intended for low-verbosity programmatic construction of specifications in user scripts and serialisation routines (see upcoming amm_save_spec and amm_load_spec). Functionally equivalent to a direct call to amm_spec; the builder adds no new modelling capability.

Usage

```
amm_build(p = 1L)
```

Arguments

p	Positive integer giving the dimension of the per-individual parameter vector θ_i. Defaults to 1L (scalar path, backward compatible with the direct amm_spec call without p).

Details

The builder is a thin wrapper around the existing dims_spec semantics: every amm_set_a_uniform / amm_set_b_uniform call mutates the base of the embedded dims_spec, and every amm_set_a / amm_set_b call layers a per-dimension override on top. Overrides survive subsequent uniform changes, in direct correspondence to dimwise composed with override.

Construction is bifurcated by p at the finalisation step, not at builder time: when as_amm_spec is called with p = 1L, the embedded dims_spec is resolved to a length-one list and the resulting a and b formulas are passed to amm_spec on the scalar path; with p > 1L the embedded dims_spec is passed directly to the multivariate path. This keeps a single source of truth for per-dimension semantics.

Value

An object of class amm_builder with components p, dims (a dims_spec object initialised with NULL base and no overrides), W (NULL), and x_vars (NULL). The object is intended to be passed through a chain of amm_set_* calls and terminated by as_amm_spec.

Methodological notes

The builder exposes the same structural asymmetry between per-dimension components (a, b) and the cross-dimension component (W) as amm_spec: dedicated per-dimension setters are provided for a and b, while W is set globally via amm_set_W. Declaring W per-dimension would silently restrict the model class to the separable sub-class, which is rejected by construction.

See Also

amm_set_a_uniform, amm_set_b_uniform, amm_set_a, amm_set_b, amm_set_W, amm_set_x_vars, as_amm_spec, amm_spec, dimwise, override

Examples

```
spec <- amm_build(p = 1L) |>
  amm_set_a_uniform(~ x1 + x2) |>
  amm_set_b_uniform(~ x1) |>
  amm_set_W(W_basis(type = "polynomial", degree = 2)) |>
  as_amm_spec()
print(spec)

spec_mv <- amm_build(p = 3L) |>
  amm_set_a_uniform(~ x1 + x2) |>
  amm_set_a(k = 2L, ~ x1) |>
  amm_set_b_uniform(~ x1) |>
  as_amm_spec()
print(spec_mv)
```

---

amm_load_spec *Load a canonical amm_spec file produced by amm_save_spec*

Description

Parse a file written by amm_save_spec and reconstruct the corresponding amm_spec object. Parsing is purely lexical (no source or eval of the file contents) and validates the version header strictly against the running package version.

Usage

```
amm_load_spec(path)
```

Arguments

path	Character scalar giving the file path.

Value

An object of class amm_spec.

Errors

Aborts with a gdpar_input_error for: missing or unreadable file; absent or malformed version header; version mismatch; malformed record line (missing colon); unknown or missing required keys; invalid value grammar for a recognised key; W.type = "user" (not serialisable in the canonical form); dims.K.* records outside 1:p.

See Also

amm_save_spec, amm_spec

Examples

```
spec <- amm_spec(a = ~ x1 + x2,
                 W = W_basis(type = "polynomial", degree = 2))
tmp  <- tempfile(fileext = ".gdpar")
amm_save_spec(spec, tmp)
spec2 <- amm_load_spec(tmp)
print(spec2)
```

---

amm_save_spec *Serialize an amm_spec to a canonical plain-text file*

Description

Write an amm_spec object to a text file in a canonical, human-readable format suitable for version control, archival, and bit-exact reproducibility. The file uses a small key: value grammar with a mandatory version header and a hierarchical naming convention for the per-dimension entries on the multivariate path. The file is parsed by amm_load_spec via a dedicated parser (no source or eval of the file contents), so the serialized form is safe to load from untrusted locations.

Usage

```
amm_save_spec(spec, path)
```

Arguments

spec	An object of class amm_spec.
path	Character scalar giving the destination file path.

Details

The serialized format records exactly the constructor inputs of the specification, not derived state. In particular, when the modulating basis W has been materialised at a specific θ_{ref} dimension (via the internal materialize_W_basis), the materialised fields are not written; the reconstructed object after amm_load_spec is the unmaterialised W_basis corresponding to the same constructor arguments, which is the form normally produced by amm_spec.

User-defined W_basis objects (type = "user") cannot be serialised because the evaluator is an arbitrary R function whose definition cannot be canonised into the file format. An attempt to serialise such a spec aborts with an informative error.

Value

Invisibly returns path.

File format

The first non-empty line must be the version header `# gdpar_spec_version: <version>`. Subsequent lines are either comments starting with `#`, empty lines (ignored), or records of the form `key: value`. Recognised keys and value grammars:

- `p`: positive integer.
- `a`, `b`: `NULL` or a one-sided formula literal such as `~ x1 + x2`. Used on the scalar path; set to `NULL` on the multivariate path.
- `x_vars`: `NULL` or a literal of the form `c("x1", "x2", ...)`.
- `W.type`: `NULL`, `polynomial` or `bspline`. `user` is not serialisable.
- `W.degree`: positive integer (required for `polynomial` and `bspline`).
- `W.knots`: `c(...)` of numerics (bspline with interior knots only).
- `W.df`: positive integer (bspline with df only).
- `dims.K.a`, `dims.K.b`: same grammar as `a`, `b`, for `K` in `1:p`. Required when `p > 1`.

Version policy

The version field is checked strictly against the running package version. Until the package reaches its first stable release a mismatch is treated as an error; a subsequent release will introduce a forward-compatible upgrade path.

See Also

amm_load_spec, amm_spec, W_basis

Examples

```
spec <- amm_spec(a = ~ x1 + x2, b = ~ x1,
                 W = W_basis(type = "polynomial", degree = 2))
tmp  <- tempfile(fileext = ".gdpar")
amm_save_spec(spec, tmp)
spec2 <- amm_load_spec(tmp)
identical(spec$level, spec2$level)
```

---

amm_set_a *Set a per-dimension additive basis override on an AMM builder*

Description

Register a per-dimension override of the additive component for index `k`. The override replaces the uniform base (set via amm_set_a_uniform, or NULL if never set) for dimension `k` only. Calling `amm_set_a` twice with the same `k` replaces the previous override.

Usage

```
amm_set_a(builder, k, a)
```

Arguments

builder	An object of class amm_builder.
k	Positive integer in 1:p. Indexes the dimension to override.
a	One-sided formula or NULL. The latter disables the additive component for dimension k only.

Value

The modified amm_builder.

See Also

amm_build, amm_set_a_uniform, override

Examples

```
b <- amm_build(p = 3L) |>
  amm_set_a_uniform(~ x1 + x2) |>
  amm_set_a(k = 2L, ~ x1) |>
  amm_set_a(k = 3L, NULL)
print(b)
```

---

amm_set_a_uniform *Set the uniform additive basis on an AMM builder*

Description

Replace the base additive formula of the embedded dims_spec with the supplied one-sided formula (or NULL to disable the additive component on the base). Per-dimension overrides previously registered via amm_set_a are preserved and continue to take precedence at their respective indices.

Usage

```
amm_set_a_uniform(builder, a)
```

Arguments

builder	An object of class amm_builder.
a	One-sided formula or NULL. Becomes the new base of the additive component, applied uniformly to every dimension that lacks an explicit override.

Value

The modified amm_builder, returned invisibly for the pipe convention but suitable for direct inspection.

See Also

amm_build, amm_set_a

Examples

```
b <- amm_build(p = 2L) |>
  amm_set_a(k = 2L, ~ x1) |>
  amm_set_a_uniform(~ x1 + x2)
print(b)
```

---

amm_set_b *Set a per-dimension multiplicative basis override on an AMM builder*

Description

Register a per-dimension override of the multiplicative component for index k. The override replaces the uniform base (set via amm_set_b_uniform, or NULL if never set) for dimension k only. Calling amm_set_b twice with the same k replaces the previous override.

Usage

```
amm_set_b(builder, k, b)
```

Arguments

builder	An object of class amm_builder.
k	Positive integer in 1:p.
b	One-sided formula or NULL.

Value

The modified amm_builder.

See Also

amm_build, amm_set_b_uniform, override

Examples

```
b <- amm_build(p = 2L) |>
  amm_set_b_uniform(~ x1) |>
  amm_set_b(k = 2L, NULL)
print(b)
```

`amm_set_b_uniform` *Set the uniform multiplicative basis on an AMM builder*

Description

Replace the base multiplicative formula of the embedded `dims_spec` with the supplied one-sided formula (or `NULL` to disable the multiplicative component on the base). Per-dimension overrides previously registered via `amm_set_b` are preserved.

Usage

```
amm_set_b_uniform(builder, b)
```

Arguments

`builder`	An object of class `amm_builder`.
`b`	One-sided formula or `NULL`. Becomes the new base of the multiplicative component, applied uniformly to every dimension that lacks an explicit override.

Value

The modified `amm_builder`.

See Also

`amm_build`, `amm_set_b`

Examples

```
b <- amm_build(p = 2L) |>
  amm_set_b_uniform(~ x1)
print(b)
```

amm_set_W — *Set the modulating basis on an AMM builder*

Description

Store a W_basis object as the modulating component of the specification under construction, or clear it by passing NULL. The modulating component is global to all dimensions of θ_i and is therefore stored as a single top-level slot of the builder.

Usage

```
amm_set_W(builder, W)
```

Arguments

builder	An object of class amm_builder.
W	A W_basis object, or NULL to disable the modulating component.

Value

The modified amm_builder.

See Also

amm_build, W_basis

Examples

```
wb <- W_basis(type = "polynomial", degree = 2)
b <- amm_build(p = 1L) |>
  amm_set_a_uniform(~ x1) |>
  amm_set_W(wb)
print(b)
```

amm_set_x_vars — *Set the covariate names used by the modulating component on an AMM builder*

Description

Record the character vector identifying the covariates that enter the modulating component as the linear factor x in $W(\theta)x$, or clear it by passing NULL. The value is forwarded to amm_spec at finalisation time. When NULL, the package uses the covariates derived from the right-hand side of the model formula passed to gdpar.

Usage

```
amm_set_x_vars(builder, x_vars)
```

Arguments

builder	An object of class amm_builder.
x_vars	Character vector with the names of the covariates, or NULL.

Value

The modified amm_builder.

See Also

amm_build, amm_spec

Examples

```
b <- amm_build(p = 1L) |>
  amm_set_a_uniform(~ x1 + x2) |>
  amm_set_x_vars(c("x1", "x2"))
print(b)
```

---

amm_spec *Specify the AMM canonical decomposition*

Description

Declare which components of the Additive-Multiplicative-Modulated canonical form $\Delta(x, \theta) = a(x) + b(x) \odot \theta + W(\theta)x$ are active and how they are parametrized. The result is consumed by gdpar to assemble the design matrices and the Stan model.

Usage

```
amm_spec(a = NULL, b = NULL, W = NULL, x_vars = NULL, p = 1L, dims = NULL)
```

Arguments

a	Scalar path only (p = 1L). One-sided formula declaring the basis of the additive component on the covariate space, evaluated via model.matrix. Use NULL to disable the additive component (degenerate Level 0 or Level 1 with only the modulating term). Must be NULL when p > 1L.
b	Scalar path only (p = 1L). One-sided formula declaring the basis of the multiplicative component on the covariate space. Must be NULL when p > 1L.

`W` An object of class `W_basis` declaring the basis of the modulating component on the parameter space, or `NULL` to disable the modulating component. See `W_basis`. The modulating component couples all dimensions of θ_{ref} in the canonical form and is therefore declared as a single basis object regardless of `p`.

`x_vars` Character vector with the names of the covariates that enter the modulating component as the linear factor x in $W(\theta)x$. When `NULL` (default), the package uses the same covariates as those entering the right-hand side of the model formula passed to `gdpar`.

`p` Positive integer giving the dimension of the per-individual parameter vector θ_i. Defaults to `1L` (scalar path, backward compatible with all previous specifications).

`dims` Multivariate path only (`p > 1L`). Either a `dims_spec` object produced by `dimwise` (possibly composed with `override`), or a plain list of length `p` where each entry is itself a list with components `a` and `b` (each a one-sided formula or `NULL`). Must be `NULL` when `p == 1L`. Bare formulas are rejected, to prevent silent recycling.

Details

Two construction paths exist, selected by the dimension `p` of the per-individual parameter vector θ_i: the scalar path (`p = 1L`, the default) uses `a` and `b` as one-sided formulas directly; the multivariate path (`p > 1L`) uses the `dims` argument together with the `dimwise` and `override` helpers to declare per-dimension specifications. The two paths are deliberately bifurcated to keep contract semantics clean and backward compatibility absolute.

The additive and multiplicative bases are declared as one-sided R formulas; the package uses `model.matrix` to build the design matrices from the data frame supplied to `gdpar`. Each formula is evaluated without an intercept column; the centering conditions (C2) and (C3) of Block 1 are enforced empirically by column-wise centering of the design matrices Z_a and Z_b inside the constructor (each column has its sample mean subtracted, so that colMeans(Z_a) = 0 and colMeans(Z_b) = 0 exactly). Because E_mu[a(X)] = colMeans(Z_a) * a_coef under the empirical mu used by Path 1, the centering of Z_a alone satisfies (C2) for any choice of a_coef in the full J_a-dimensional Euclidean space; analogously for (C3). No additional restriction on the basis coefficients is imposed.

On the multivariate path, the additive and multiplicative components factor per dimension of θ_i because they depend only on the covariates x_i (and therefore on coordinates of $\mathcal{X}$, not of Θ). The modulating component W depends on the full vector θ_{ref} and therefore couples the dimensions; declaring W per-dimension would silently restrict the model class to the separable sub-class. The package therefore enforces the structural asymmetry between (`a`, `b`) and `W` at the API level: `dims` for the former, a single top-level `W` argument for the latter.

Value

An object of class `amm_spec` with components `a`, `b`, `W`, `x_vars`, `level`, `p`, and `dims`. On the scalar path (`p = 1L`), `dims` is `NULL` and `a`, `b` hold the formulas. On the multivariate path (`p > 1L`), `a` and `b` are `NULL` and `dims` holds the resolved per-dimension list of length `p`, each entry being a list with `a` and `b`.

Methodological notes

The AMM level implied by the specification is recorded on the returned object: Level 0 if every component is NULL across all dimensions, Level 1 if only the additive component is active in any dimension and no multiplicative or modulating term is active anywhere, and Level 2 otherwise. The identifiability theorems of Block 1 apply at each level under the linearity assumption (LIN), which holds automatically for formula-based bases (linear subspaces of L^2_0(\mu)) and for the polynomial and B-spline cases of `W_basis`.

Cross-component non-identifiability is not detected by this constructor; it is detected at the Gram-matrix diagnostic of `gdpar_check_identifiability`, which is called automatically before fitting. For `p > 1L`, an additional cross-dimension identifiability condition applies (see the planned vignette `vop02_arbitrary_p`).

Dependencies

Uses `terms` for parsing the formulas and `model.matrix` downstream for building the design matrices.

References

See `vignette("v01_amm_identifiability", package = "gdpar")`, Section 3 (the AMM hierarchy) and Section 5 (standing assumptions (C1)-(C6)).

See Also

`W_basis`, `dimwise`, `override`, `gdpar_check_identifiability`, `gdpar`

Examples

```
spec <- amm_spec(
  a = ~ x1 + x2,
  b = ~ x1,
  W = W_basis(type = "polynomial", degree = 2)
)
print(spec)

spec_mv <- amm_spec(
  p    = 2L,
  dims = dimwise(a = ~ x1 + x2, b = ~ x1),
  W    = W_basis(type = "polynomial", degree = 2)
)
print(spec_mv)
```

as.data.frame.gdpar_coef

Coerce a gdpar_coef object to a long-tidy data.frame

Description

Flattens the hierarchical per-component, per-coordinate structure into a single data.frame with columns (`component`, `k`, `identifier`, `x_name`, `mean`, `q05`, `q50`, `q95`). The `identifier` column carries the `term` for `a` and `b` slots, the `basis_idx` (formatted as a string) for `W`, and `NA` for `theta_ref`. The `x_name` column is `NA` except for `W` rows. Useful as input to **dplyr** / **ggplot2** pipelines.

Usage

```
## S3 method for class 'gdpar_coef'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	Object of class `gdpar_coef`.
`row.names`	Ignored; required by the generic.
`optional`	Ignored; required by the generic.
`...`	Unused.

Value

A data.frame with one row per scalar coefficient summarized.

as.data.frame.gdpar_preflight_report

as.data.frame method for gdpar_preflight_report

Description

Returns the tidy per-dim data frame. Suitable for use with `subset`, `aggregate`, `dplyr::filter`, etc.

Usage

```
## S3 method for class 'gdpar_preflight_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	An object of class gdpar_preflight_report.
row.names	NULL or character vector forwarded as row.names of the returned data frame.
optional	Ignored; present for S3 generic compatibility.
...	Unused; present for S3 generic compatibility.

Value

Data frame with one row per (component, dim).

---

as_amm_spec *Finalise an AMM builder into an amm_spec*

Description

Validate the accumulated state of an amm_builder and convert it to an amm_spec object. Dispatches on p: the scalar path (p = 1L) resolves the embedded dims_spec to the single per-dimension entry and forwards a, b, W, x_vars to amm_spec; the multivariate path (p > 1L) forwards the embedded dims_spec directly. All structural validation (range of override indices, consistency between p and the supplied components) is delegated to amm_spec.

Usage

```
as_amm_spec(builder)
```

Arguments

builder	An object of class amm_builder.

Value

An object of class amm_spec; see amm_spec for the slot layout.

See Also

amm_build, amm_spec

Examples

```
spec <- amm_build(p = 2L) |>
  amm_set_a_uniform(~ x1 + x2) |>
  amm_set_b_uniform(~ x1) |>
  as_amm_spec()
print(spec)
```

as_per_k *Split a separable multivariate W_basis into per-coordinate sub-bases*

Description

For separable multivariate W bases (polynomial and bspline with $p > 1$), return a list of length p whose k-th entry is a univariate W_basis describing the contribution of coordinate k of theta_ref. Useful for introspection, per-coordinate diagnostics (e.g., the per-k CP/NCP toggle of the pre-flight pipeline), and downstream methods that operate on one coordinate at a time.

Usage

```
as_per_k(wb)
```

Arguments

wb	A W_basis object with p populated (either constructed with p = ... or processed by materialize_W_basis()).

Details

Polynomial and bspline bases are separable by construction: the total basis is the concatenation, in coordinate order, of the univariate bases applied independently to each coordinate of theta_ref. as_per_k() simply reconstructs that decomposition as explicit W_basis objects, each materialized with p = 1L.

This decomposition is the natural input to the per-coordinate CP/NCP toggle of the pre-flight pipeline (Path B'), which operates on each coordinate of theta_ref independently when p > 1.

For type = "user" the separability is unverifiable and the function returns NULL. Callers that need per-coordinate decompositions for user bases must construct them explicitly.

Value

A list of length wb$p. Each entry is itself a W_basis of the same type as wb, materialized with p = 1L. For type = "user" the function returns NULL: separability of a user-supplied evaluator cannot be inferred automatically. A warning is emitted in that case.

See Also

W_basis

Examples

```
wb <- W_basis(type = "polynomial", degree = 2, p = 3L)
subs <- as_per_k(wb)
length(subs)
subs[[1]]$dim
```

coef.gdpar_eb_fit *Coefficient extraction for gdpar_eb_fit*

Description

Returns an object analogous to `coef.gdpar_fit`: a `gdpar_coef` list with components `theta_ref`, `a`, `b`, `W`. The EB version reports `theta_ref$method == "EB"` and includes a flag `theta_ref$eb_correction_applied` so downstream consumers can tell that the credible intervals are EB-corrected (or that the correction was disabled).

Usage

```
## S3 method for class 'gdpar_eb_fit'
coef(object, ...)
```

Arguments

`object`	A `gdpar_eb_fit` object.
`...`	Forwarded to the underlying `coef.gdpar_fit`-like accessor on the conditional fit (none recognized in v0).

Value

A list with class `c("gdpar_coef_eb", "gdpar_coef", "list")`.

coef.gdpar_fit *Coefficients of a fitted gdpar model*

Description

Returns posterior summaries (mean and 5%/50%/95% quantiles) of `theta_ref` and the basis coefficients (additive `a_coef`, multiplicative `b_coef`/`c_b`, modulating `W_raw`) as an object of class `gdpar_coef`. The format is consistent across `p = 1` (scalar) and `p > 1` (multi) fits.

Usage

```
## S3 method for class 'gdpar_fit'
coef(object, ...)
```

Arguments

`object`	An object of class `gdpar_fit`.
`...`	Unused; present for S3 generic compatibility.

Details

For multi fits, the modulating slot uses the effective per-sample product `W_raw * sigma_W` (when `cp_W` is FALSE) so that the coefficients are reported on the natural modulating scale. The per-coordinate `b` slot draws from the multi parameter `c_b` (already on the theta_ref-multiplied scale; see Recovery 2, handoff 4).

For K-individual fits (`K > 1`, distributional regression on the per-slot template `amm_distrib_K.stan`) the function returns a named list of length `K` whose entries are `gdpar_coef` objects (each with `p = 1L`), one per slot (decision E4.A of sub-phase 8.3.10). The modulating block `W_raw` is globally shared across slots in the K-individual template and its contribution enters `eta_k` for every slot (see `amm_distrib_K.stan`, lines 500-523, and the canonical decision "Scope of W: global" in the likelihood K > 1 decision of handoff 28); accordingly, when any slot declared a non-NULL `W` the resulting `W` component is attached to every slot's `gdpar_coef` (and the per-sample product `W_raw * sigma_W` is reported when `cp_W` is FALSE, identically to the scalar path). K-individual fits with grouping (`J_groups > 1`) are queued for Session 8.4 and raise `gdpar_unsupported_feature_error`.

Value

For scalar (`p = 1`) and multi (`p > 1`) fits an object of class `gdpar_coef`; for K-individual fits (`K > 1`) a named list of length `K` of `gdpar_coef` objects. Use `as.data.frame()` on a single `gdpar_coef` to obtain a long-tidy table; for K-individual fits use `lapply(coef(fit), as.data.frame)`.

`diagnostics` *Extract diagnostics from a fitted gdpar model*

Description

Convenience accessor for the convergence diagnostics computed at fit time and stored in a `gdpar_fit` object.

Usage

```
diagnostics(fit)
```

Arguments

`fit`	An object of class `gdpar_fit`.

Details

The diagnostics are computed once at fit time and stored. Calling this function does not re-run any computation.

Value

An object of class `gdpar_diagnostics` with R-hat, effective sample size, divergent-transition and tree-depth summaries, and a logical `converged` flag.

Dependencies

Diagnostics are computed with **posterior** at fit time.

See Also

gdpar

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  df <- data.frame(x1 = rnorm(100), y = rnorm(100))
  fit <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df,
               iter_warmup = 200, iter_sampling = 200, chains = 2)
  diagnostics(fit)
}
```

---

dimwise *Broadcast a uniform per-component specification to multiple dimensions*

Description

Construct a `dims_spec` object that records the same additive and multiplicative formulas for every dimension $k = 1, \ldots, p$ of a multivariate `theta_i` vector. Used as the value of the `dims` argument of `amm_spec` when all dimensions share the same per-component specification. Per-dimension overrides can be layered on top via `override`.

Usage

```
dimwise(a = NULL, b = NULL)
```

Arguments

a	One-sided formula declaring the additive basis applied uniformly to every dimension of `theta_i`. Use `NULL` to disable the additive component for all dimensions.
b	One-sided formula declaring the multiplicative basis applied uniformly to every dimension of `theta_i`. Use `NULL` to disable the multiplicative component for all dimensions.

Details

This helper enables a low-verbosity declaration of p > 1 specs for the common case where all dimensions of the multivariate theta_i share the same a_k and b_k structures. Broadcasting is explicit: the user must wrap the per-component formulas in dimwise() to opt in. Bare formulas passed directly to the dims argument of amm_spec when p > 1 are rejected, to avoid silent recycling masking bugs.

The dimension p is not stored in the dims_spec; it is resolved at the point of consumption by amm_spec, which takes p as an explicit argument and validates coherence across the spec, the multivariate W basis, and any overrides.

The modulating component W is not part of dims_spec because W couples all dimensions of theta_ref in the canonical AMM form and therefore cannot be declared per-dimension. W is supplied to amm_spec as a separate top-level argument via a multivariate basis.

Value

An object of class dims_spec with components base (the uniform template, a list with a and b) and overrides (an initially empty named list of per-dimension overrides keyed by character form of the integer index k).

Methodological notes

The asymmetry between per-dimension components (a, b) and the cross-dimension component (W) reflects the canonical AMM form

$$\theta_i[k] = \theta_{\text{ref}}[k] + a_k(x_i) + b_k(x_i)\theta_{\text{ref}}[k] + (W_k(\theta_{\text{ref}}) - W_k(\theta_{\text{anchor}}))x_i, \quad k = 1, \ldots, p,$$

in which a_k and b_k depend only on the covariates x_i (and therefore factor per k) while each W_k depends on the full vector θ_{ref} (and therefore couples the dimensions). Representing W as a list of independent per-dimension bases would silently restrict the model class to the separable sub-class. dims_spec therefore excludes W by construction.

See Also

override, amm_spec

Examples

```
base <- dimwise(a = ~ x1 + x2, b = ~ x1)
print(base)

with_override <- override(base, k = 2L, a = ~ x1)
print(with_override)
```

format.gdpar_coef *One-line formatter for gdpar_coef objects*

Description

Useful for logs and condensed printouts.

Usage

```
## S3 method for class 'gdpar_coef'
format(x, ...)
```

Arguments

x	Object of class gdpar_coef.
...	Unused.

Value

A length-1 character vector.

format.gdpar_preflight_report *Format method for gdpar_preflight_report*

Description

One-line representation of the report. Used by format in contexts where a print is not desired.

Usage

```
## S3 method for class 'gdpar_preflight_report'
format(x, ...)
```

Arguments

x	An object of class gdpar_preflight_report.
...	Unused; present for S3 generic compatibility.

Value

Character scalar.

gdpar *Fit an AMM canonical model*

Description

Main entry point of the package. Fits the AMM canonical decomposition of the individual parameter via one of three estimation paths. In this version of the package, only Path 1 (hierarchical Bayesian inference via Stan) is fully implemented; Path 2 and Path 3 wrappers are placeholders that signal the development status.

Usage

```
gdpar(
  formula,
  family = gdpar_family("gaussian"),
  amm = amm_spec(),
  W = NULL,
  data,
  prior = NULL,
  path = c("bayes", "vcm", "hyper"),
  anchor = "prior_mean",
  skip_id_check = FALSE,
  chains = 4L,
  iter_warmup = 1000L,
  iter_sampling = 1000L,
  adapt_delta = 0.95,
  max_treedepth = 12L,
  refresh = 100L,
  verbose = TRUE,
  seed = NULL,
  group = NULL,
  parametrization = c("auto", "ncp", "cp"),
  parametrization_a = NULL,
  parametrization_W = NULL,
  parametrization_aggregation = NULL,
  id_check_rigor = c("full", "fast"),
  ...
)
```

Arguments

formula	Either a two-sided formula or an object of class gdpar_formula_set. In the legacy two-sided form, the left-hand side is the outcome variable; the right-hand side either lists the covariates that enter the modulating component as the linear factor x in $W(\theta)x$ (the AMM components are declared via the amm argument), or it contains the AMM wrapper calls a(...), b(...) and/or W() themselves (canonized in sub-phase 8.3.3 as the K = 1 entry to the high-level parser path).

	A gdpar_formula_set (constructed via gdpar_formula_set or via the brms-style sugar gdpar_bf) is the multi-parameter form: one slot per K-individual parameter, with AMM wrappers in each slot's right-hand side.
family	An object of class gdpar_family produced by gdpar_family or gdpar_family_custom. Defaults to gdpar_family("gaussian").
amm	Either an object of class amm_spec produced by amm_spec (legacy single-parameter path, K = 1) or a named list of amm_spec objects (K-individual low-level path closed in sub-phase 8.3.3, names matching the eligible parameters of family). Defaults to a Level 0 (degenerate) single amm_spec when not supplied. When formula is a gdpar_formula_set or contains AMM wrapper calls (a()/b()/W()) in its right-hand side, the amm argument must remain at its default; the canonical specifications come from the formula in those paths.
W	An object of class W_basis produced by W_basis, or NULL (default). Used by the K-individual paths (formula set, named list of amm_spec, or classic formula with AMM wrappers in the RHS) to supply the modulating basis when one or more slots declare W(). Ignored in the legacy single-amm_spec path, where the modulating basis travels through amm$W.
data	A data frame containing the variables referenced by formula and amm.
prior	An object of class gdpar_prior produced by gdpar_prior. When NULL (default), the package defaults are used.
path	Character scalar identifying the estimation path. One of "bayes" (Path 1, default), "vcm" (Path 2), "hyper" (Path 3).
anchor	Either a numeric scalar with the anchor value theta_0 in the linear-predictor scale, or one of "prior_mean" (default) and "empirical_y" (the linkfun applied to the outcome mean).
skip_id_check	Logical scalar. If TRUE, skip the automatic Gram-matrix identifiability diagnostic. Defaults to FALSE; turning it off is intended for advanced users who have verified identifiability separately.
chains	Integer scalar with the number of Hamiltonian Monte Carlo chains. Defaults to 4.
iter_warmup	Integer scalar with the number of warmup iterations per chain. Defaults to 1000.
iter_sampling	Integer scalar with the number of sampling iterations per chain. Defaults to 1000.
adapt_delta	Numeric scalar in (0, 1) controlling the target acceptance probability of the No-U-Turn Sampler. Defaults to 0.95 (more conservative than the Stan default 0.8) to reduce divergent transitions in hierarchical models.
max_treedepth	Integer scalar bounding the tree depth of the No-U-Turn Sampler. Defaults to 12.
refresh	Integer scalar with the progress-reporting frequency passed to cmdstanr. Defaults to 100, which prints a status line every 100 iterations.
verbose	Logical scalar controlling the verbosity of informational messages from the package itself (independent of refresh). Defaults to TRUE.
seed	Optional integer scalar passed to cmdstanr for reproducibility. Defaults to NULL, which lets cmdstanr pick a seed. The same seed is forwarded to the pre-flight diagnostic when parametrization = "auto".

group
: Optional one-sided formula identifying the grouping variable in data that anchors per-group hierarchical estimation of theta_ref. When NULL (default) the model uses a single global theta_ref, matching the Block 6 behavior bit-exactly. When supplied, theta_ref is promoted to a vector indexed by group with a Normal hyperprior $\theta_{ref}[g] \sim \mathrm{Normal}(\mu_{\theta_{ref}}, \sigma_{\theta_{ref}})$, analogous to a random intercept. Only one-sided formulas with a single variable name are accepted (for example ~ species); the variable must exist in data. The grouping is rejected at pre-flight if (C7) of Block 6.5 is violated: that is, if a or b contain columns that are constant within every group level or otherwise rank-deficient with the group indicator (perfect aliasing with the per-group anchor).

parametrization
: Character scalar selecting the sampling parametrization for the additive component a and the modulating component W. One of "auto" (default, runs a short pre-flight NCP fit and decides per-component via a sequence of three filters: a divergence-attribution t-test, an E-BFMI threshold, and a chain-aware block-bootstrap z-test on the posterior-to-prior contraction of the effective coefficient, Path B'), "ncp" (forces non-centered parametrization for both components, skipping the pre-flight), "cp" (forces centered parametrization for both components, skipping the pre-flight). The pre-flight adds approximately 30 fit when active. See vignette("vop01_parametrization_toggle", package = "gdpar") for the operational guide.

parametrization_a
: Optional character scalar ("ncp" or "cp") overriding the parametrization for component a. When NULL (default), inherits from parametrization.

parametrization_W
: Optional character scalar ("ncp" or "cp") overriding the parametrization for component W. When NULL (default), inherits from parametrization. If both parametrization_a and parametrization_W are explicit, the pre-flight is skipped regardless of parametrization.

parametrization_aggregation
: Optional character scalar ("any_ncp", "majority", or "per_k") controlling how the per-coordinate CP/NCP decisions of the multivariate (amm$p > 1L) pre-flight are aggregated to a per-component decision. "any_ncp" (default) is conservative: a component is CP only when every coordinate's decision is CP. "majority" selects CP when the strict majority of coordinates votes CP; ties break toward NCP. "per_k" keeps the per-coordinate decisions intact and honors them at the sampling level via per-k segment()-based priors (Phase H.2 of Block 5.2). The uniform branches (all-CP or all-NCP) compile to byte-identical Stan code with the H.1 template, preserving bit-exact paridad. Ignored for the univariate path (amm$p == 1L). The full per-coord report is stored in fit$parametrization$report as an object of class gdpar_preflight_report.

id_check_rigor
: Character scalar, one of "full" (default) or "fast", controlling the strictness of the basis-restricted identifiability check. "full" aborts on any C1-C4 (univariate path) or C4-bis cross-coordinate identifiability violation (multivariate path). "fast" downgrades the C4-bis check to a single consolidated warning at the end of the per-coordinate loop, allowing the fit to proceed. The design pattern for legitimate overlap between the additive and modulating channels is documented in vignette("v01_amm_identifiability", package = "gdpar") Section 6.6.1.7.

"fast" is only meaningful for the multivariate path (amm$p > 1L); for the univariate path the value is forwarded but the C4-bis check is not run.

... Additional arguments forwarded to the underlying sampler.

Details

The function orchestrates the Path 1 fit in five steps: (1) input validation and standardization of the covariates entering the design matrices; (2) the basis-restricted identifiability diagnostic of gdpar_check_identifiability; (3) generation of the Stan model source by substituting the prior placeholders into the static template inst/stan/amm_main.stan; (4) compilation and sampling via cmdstanr; (5) collection of convergence diagnostics from **posterior** and assembly of the returned object.

Covariates entering the additive and multiplicative bases are centered before fitting, enforcing assumption (C1) of Block 1 at the empirical level. The covariates entering the modulating component are additionally scaled to unit standard deviation; this standardization is recorded so that predictions on new data can apply the same transformation consistently.

The anchor value theta_0 enters the parametrization W(theta) - W(theta_anchor) and is therefore a parametrization device, not an inferential statement about the posterior of theta_ref. Choosing a different anchor changes the parametrization but not the data-generating model.

Value

An object of class gdpar_fit with components fit (the underlying cmdstanr fit object), amm, family, prior, design, anchor, identifiability_report, diagnostics, parametrization (a list with the resolved CP/NCP flags per component plus the pre-flight diagnostic statistics when applicable), call and path.

Methodological notes

Path 1 supports finite-dimensional parametric AMM specifications in this version. Non-parametric extensions (Gaussian-process priors on a, b, W; adaptive splines with growing basis dimension) are deferred to a future version; users who need such flexibility are referred to Path 2 (varying-coefficient models via mgcv).

The polynomial restriction on W_basis in Path 1 is also a v0 limitation; B-spline and user-defined W bases require either a Stan-side basis evaluator that the current template does not implement, or a precomputed basis values approach that complicates the data block. Both extensions are planned.

Dependencies

Uses **cmdstanr** (Suggests) for compilation and sampling, and **posterior** (Imports) for convergence diagnostics on the resulting posterior draws. **cmdstanr** is loaded conditionally via requireNamespace; the function aborts with an informative error if the package is not installed.

References

See vignette("v01_amm_identifiability", package = "gdpar") for the canonical form and identifiability conditions; vignette("v04_asymptotics_path1_bayesian", package = "gdpar") for the asymptotic theory of Path 1; and vignette("vop01_parametrization_toggle", package

= "gdpar") for the operational guide to the CP/NCP toggle, including the three-filter pre-flight diagnostic, introspection of fit$parametrization$meta, and known limitations under confounding.

Stan Development Team (2024). Stan User's Guide, version 2.35.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. Journal of Statistical Software, 76(1).

See Also

amm_spec, W_basis, gdpar_family, gdpar_prior, gdpar_check_identifiability, gdpar_bvm_check, gdpar_contraction_diagnostic

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(NULL)
  n <- 200
  df <- data.frame(
    x1 = rnorm(n),
    x2 = rnorm(n)
  )
  df$y <- with(df, 1 + 0.5 * x1 - 0.3 * x2 + rnorm(n, sd = 0.5))
  spec <- amm_spec(a = ~ x1 + x2)
  fit <- gdpar(
    formula       = y ~ x1 + x2,
    family        = gdpar_family("gaussian"),
    amm           = spec,
    data          = df,
    iter_warmup   = 200,
    iter_sampling = 200,
    chains        = 2
  )
  print(fit)
}
```

---

gdpar_adapter_econml *Reference adapter: EconML CausalForestDML for gdpar_compare_meta_learners*

Description

Build a gdpar_meta_learner_adapter that wraps a Python-side EconML estimator (Chernozhukov et al., 2018) via **reticulate** for use with gdpar_compare_meta_learners. The default estimator is econml.dml.CausalForestDML, the orthogonal double-machine-learning causal forest of Athey, Tibshirani, and Wager (2019). The adapter exposes both the mandatory fit_predict_fun and the optional predict_fun; the latter reuses the fitted Python estimator on a fresh evaluation grid without a refit. Native CIs are produced by EconML's effect_interval(X, alpha = 1 - level) method.

Usage

```
gdpar_adapter_econml(
  estimator = "CausalForestDML",
  n_estimators = 1000L,
  model_y = NULL,
  model_t = NULL,
  seed = NULL
)
```

Arguments

estimator	Character scalar identifying the EconML estimator. Default "CausalForestDML". Other identifiers can be added in future sub-phases.
n_estimators	Integer scalar; number of trees in the EconML causal forest. Default 1000L.
model_y	Optional Python model object for the outcome stage (econml's model_y argument). Default NULL delegates to EconML's default (a regression-tree-based model).
model_t	Optional Python model object for the treatment stage (econml's model_t argument). Default NULL delegates to EconML's default.
seed	Optional integer scalar with the seed propagated to the EconML estimator (random_state). Default NULL.

Details

The Python module econml is in Suggests and must be installed by the user outside of the package (e.g. reticulate::py_install("econml") or manual installation in the active Python environment). The adapter aborts cleanly when **reticulate** or the econml module is unavailable.

Cached state caveat: the returned state carries a reference to a Python object. The reference is valid for the duration of the R session in which the bridge was built; serializing the comparison via saveRDS and reloading in a fresh R session invalidates the Python reference and the predict_fun aborts with gdpar_unsupported_feature_error when invoked on a restored state. Rebuild the comparison in such cases.

Value

A gdpar_meta_learner_adapter object with requires_r = "reticulate", requires_py = "econml", native_ci = TRUE, and both fit_predict_fun and predict_fun populated.

References

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1-C68.

Athey, S., Tibshirani, J., and Wager, S. (2019). Generalized random forests. *The Annals of Statistics*, 47(2), 1148-1178.

See Also

gdpar_compare_meta_learners, gdpar_meta_learner_adapter, gdpar_adapter_grf.

Examples

```
if (requireNamespace("reticulate", quietly = TRUE)) {
  adapter <- gdpar_adapter_econml(n_estimators = 200L)
  print(adapter)
}
```

---

gdpar_adapter_grf *Reference adapter: grf::causal_forest for gdpar_compare_meta_learners*

Description

Build a gdpar_meta_learner_adapter that wraps the R-side causal forest of **grf** (Athey, Tibshirani, and Wager, 2019) for use with gdpar_compare_meta_learners. The adapter exposes both the mandatory fit_predict_fun and the optional predict_fun so the comparator's predict method can reuse the fitted causal forest on a fresh evaluation grid without a refit. Native CIs are obtained by the normal approximation $\widehat{\tau}(x) \pm z_{1-\alpha/2} \cdot \sqrt{\widehat{\mathrm{Var}}(\widehat{\tau}(x))}$ using grf's built-in variance estimator (predict(..., estimate.variance = TRUE)).

Usage

```
gdpar_adapter_grf(
  num_trees = 2000L,
  sample_fraction = 0.5,
  mtry = NULL,
  honesty = TRUE,
  seed = NULL
)
```

Arguments

num_trees	Integer scalar; number of trees in the forest. Default 2000L, matching **grf**'s default.
sample_fraction	Numeric scalar in (0, 0.5]; fraction of the training sample drawn for each tree. Default 0.5.
mtry	Optional integer scalar with the number of candidate variables per split; default NULL delegates to **grf**'s own default (min(ceiling(sqrt(p) + 20), p)).
honesty	Logical scalar; whether to use honest splitting. Default TRUE (recommended; the **grf** confidence intervals are valid only under honesty).
seed	Optional integer scalar with the seed propagated to **grf**'s internal RNG when the comparator's seed_run is NULL. Default NULL.

Details

Categorical covariates are not handled by **grf** directly; the adapter coerces character columns to factors and then applies `stats::model.matrix(~ . - 1, ...)` to obtain a fully numeric design matrix. Numeric or factor inputs pass through unchanged.

Value

A `gdpar_meta_learner_adapter` object with `requires_r = "grf"`, `native_ci = TRUE`, and both `fit_predict_fun` and `predict_fun` populated.

References

Athey, S., Tibshirani, J., and Wager, S. (2019). Generalized random forests. *The Annals of Statistics*, 47(2), 1148-1178.

Wager, S., and Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523), 1228-1242.

See Also

`gdpar_compare_meta_learners`, `gdpar_meta_learner_adapter`, `gdpar_adapter_econml`.

Examples

```
if (requireNamespace("grf", quietly = TRUE)) {
  adapter <- gdpar_adapter_grf(num_trees = 500L)
  print(adapter)
}
```

---

gdpar_bf *Construct a formula set with brms-style sugar*

Description

Build a `gdpar_formula_set` from a sequence of two-sided formulas in the style of `brms::bf`. The first formula carries the outcome on its LHS and defaults the first slot name to `"mu"` (the canonical name of the location parameter across built-in families); each subsequent formula must carry the canonical parameter name as its LHS (e.g., `sigma ~ a(x)`). The result is an object of class `gdpar_formula_set` identical to what the explicit constructor produces.

Usage

```
gdpar_bf(...)
```

Arguments

`...`	Two-sided formulas. The first carries the outcome on its LHS; subsequent ones carry the canonical parameter name on their LHS.

Details

This is sugar: `gdpar_bf(y ~ a(x1), sigma ~ a(x2))` is equivalent to `gdpar_formula_set(mu = y ~ a(x1), sigma = ~ a(x2))`. Naming the first argument overrides the default `"mu"`: `gdpar_bf(theta = y ~ a(x))` produces a single-slot set with name `"theta"`, intended for custom families whose location parameter is not canonically called `mu`.

The brms package is not a runtime dependency of `gdpar`; the constructor returns a native `gdpar_formula_set` regardless of whether **brms** is installed.

Value

An object of class `gdpar_formula_set` (see `gdpar_formula_set` for components).

Dependencies

Internally calls `gdpar_formula_set`.

See Also

`gdpar_formula_set`, `gdpar`

Examples

```
fs <- gdpar_bf(y ~ a(x1) + b(z1), sigma ~ a(x2))
print(fs)
```

---

gdpar_bvm_check *Bernstein-von Mises calibration check (opt-in, costly)*

Description

Verify numerically the conclusion of the Bernstein-von Mises theorem (Theorem 4C of Block 4) for a fitted Path 1 model: that the posterior is asymptotically Gaussian around the maximum likelihood estimator with covariance equal to the inverse Fisher information matrix divided by n. The function refits the model by maximum likelihood (MLE) using a derived Stan model in which the prior block is stripped (see `generate_stan_code(mle = TRUE)` and the `// BEGIN PRIORS` / `// END PRIORS` markers in `inst/stan/amm_main.stan`); it computes a Hessian-based covariance estimate via a Laplace approximation around the MLE, both on the unconstrained scale (the Stan optimizer is invoked with `jacobian = FALSE`); and it compares the resulting interval coverage with the Bayesian posterior intervals reported by the fitted model.

Usage

```
gdpar_bvm_check(fit, parameters = NULL, level = 0.95, verbose = TRUE)
```

Arguments

fit	An object of class gdpar_fit produced by gdpar with path = "bayes".
parameters	Optional character vector of parameter names to include in the comparison. Defaults to the user-facing parameters that the prior-stripped likelihood identifies: theta_ref, sigma_y (when present), and phi (when present).
level	Numeric scalar in (0, 1) with the nominal credible / confidence level. Defaults to 0.95.
verbose	Logical scalar; when TRUE, prints an estimated cost message before starting. Defaults to TRUE.

Details

This function is opt-in and computationally expensive: it refits the model in MLE mode and inverts a Hessian matrix. It is intended as a methodological audit, not as part of the standard inference flow. The conclusions of gdpar are not affected by calling this function.

Theorem 4C of Block 4 establishes that, for finite-dimensional parametric AMM specifications under the (LAN) condition with non-singular Fisher information at the true parameter, the posterior distribution converges in total variation to the asymptotic-Gaussian distribution of the maximum likelihood estimator. Empirically, this entails that posterior credible intervals at any nominal level should agree with the Hessian-based asymptotic confidence intervals as the sample size grows.

This function performs the empirical comparison at the observed sample size. Substantial discrepancy between the two interval families at large n signals either (i) the limit has not yet been approached, requiring more data; (ii) the (LAN) condition fails (e.g., singular Fisher information at the true parameter); or (iii) the model is misspecified (Block 7).

Applies only to fits with finite-dimensional parametric AMM specifications. Non-parametric components (planned for a future version) are outside the scope of Theorem 4C; the function will abort if invoked on a non-parametric fit. The hierarchical regime activated by the group argument of gdpar() (Block 6.5) is likewise out of scope, because the classical asymptotic theory that underwrites Theorem 4C assumes a fixed-dimension parameter vector while the hierarchical case introduces an increasing number of random anchors; the function aborts with gdpar_unsupported_feature_error when invoked on a grouped fit.

Value

A list of class gdpar_bvm_report with components table (data frame comparing posterior intervals with Hessian-based intervals per parameter), discrepancy (numeric vector of relative interval-width differences), level and warnings. A print method provides a human-readable summary.

Methodological notes

This function is part of the methodological audit toolkit, not of the standard inference flow. It does not modify the fit object in any way; its output is informational. Users running large simulations are advised to call this function selectively rather than after every fit.

The MLE estimate uses Stan's optimize method with the LBFGS algorithm on the prior-stripped variant of the model produced by generate_stan_code(mle = TRUE); the optimizer is invoked with jacobian = FALSE so the maximum is taken on the constrained (natural) scale of the parameters rather than on the unconstrained scale that would carry the Jacobian of the transformation. The

Hessian-based interval estimate comes from `cmdstanr::laplace` applied around the MLE on the same prior-stripped model and with the same `jacobian = FALSE` convention.

The comparison is restricted to the user-facing parameters that the likelihood identifies in MLE mode without the prior anchoring: the global reference parameter `theta_ref`, the response-level scale `sigma_y` (Gaussian families), and the dispersion `phi` (Negative Binomial). The hierarchical scales `sigma_a`, `sigma_b`, `sigma_W` are not identified by the likelihood alone (they enter only through the random-effect priors that have been stripped) and are excluded from the table.

Dependencies

Uses **cmdstanr** for the optimization run and **posterior** to extract the Bayesian intervals.

References

See `vignette("v04_asymptotics_path1_bayesian", package = "gdpar")`, Section 7 (Theorem 4C).

See Also

`gdpar`, `gdpar_contraction_diagnostic`

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  df <- data.frame(x1 = rnorm(200), y = rnorm(200))
  fit <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df,
               iter_warmup = 200, iter_sampling = 200, chains = 2)
  gdpar_bvm_check(fit)
}
```

---

gdpar_causal_bridge *Causal bridge (T-learner) between two gdpar fits*

Description

Estimate the conditional average treatment effect (CATE) from a pair of independent `gdpar_fit` objects fitted to disjoint arms of a treatment / control design. Implements the T-learner meta-learner of Kuenzel et al. (2019) on the AMM-side: each arm is fitted independently and the CATE is the per-observation difference of the two predictive distributions, evaluated on a common evaluation set.

Usage

```
gdpar_causal_bridge(
  fit_treat,
  fit_ctrl,
  newdata = NULL,
  type = c("response", "theta_i", "linear_predictor"),
  level = 0.95,
  ...
)
```

Arguments

fit_treat	An object of class gdpar_fit fitted to the treatment arm.
fit_ctrl	An object of class gdpar_fit fitted to the control arm. Must share the family, anchor, AMM level, and covariate structure of fit_treat (see Details).
newdata	Optional data frame on which the CATE is evaluated. When NULL (default), the function attempts to recover the training data of each arm by evaluating the captured data argument of each fit's call in the caller's environment. If both recoveries succeed and the two data frames share their column structure, their rbind is used; otherwise the function aborts and requests an explicit newdata.
type	Character scalar selecting the scale on which the CATE is estimated. One of "response" (default; the inverse link is applied per draw, so the CATE is on the response scale), "theta_i" (the linear predictor of the individual parameter), or "linear_predictor" (synonym of "theta_i").
level	Numeric scalar in (0, 1) with the nominal credible level for the per-observation CATE intervals. Defaults to 0.95.
...	Reserved for future arguments; currently unused.

Details

This function does not modify either fit. It assumes the two fits are independent posterior samples from disjoint subsets of the population (treatment arm and control arm). It does not perform any causal adjustment beyond what is encoded in the two AMM specifications: the assumption of no-unmeasured-confounding within each arm is the responsibility of the user (see Section 4 of the bridge vignette).

Structural compatibility. The function aborts with gdpar_unsupported_feature_error when the two fits differ in any of: path (both must be the Path 1 Bayesian fit), family identifier (family$name, or per-slot family identifiers when K > 1), K, p, AMM level (amm$level or the equivalent level inferred from each slot's spec), modulating basis type (amm$W$type), anchor value, or covariate column structure. The function also aborts when either fit was sampled in the hierarchical regime (stan_data$use_groups == 1L); the T-learner bridge for grouped fits is queued for a future sub-phase and would require careful treatment of the per-group anchors which is outside the scope of Sub-phase 8.5.A.

Identifiability per arm. The constructor records the identifiability report of each fit in the id_check slot. (C7) anti-aliasing of Block 6.5 is not invoked because the hierarchical guard above rules out the regime in which (C7) applies; this is documented for the eventual extension to grouped fits.

CATE estimator. For each posterior draw indexed by $s = 1, \ldots, S$ and observation $i = 1, \ldots, n_{\text{new}}$, the bridge computes

$$\widehat{\tau}_i^{(s)} = \hat{\mu}_{\text{treat}}^{(s)}(x_i) - \hat{\mu}_{\text{ctrl}}^{(s)}(x_i),$$

where $\hat{\mu}_{\text{arm}}^{(s)}(x)$ is the posterior prediction of the chosen `type` at x, drawn from the fit's predictive distribution. The marginal posterior of the CATE at each x_i is summarized by the empirical mean and the $(\alpha/2, 1-\alpha/2)$ quantiles with $\alpha = 1-$ `level`.

Independence of draws. The two fits are independent (they were sampled from disjoint data subsets), so the joint posterior of $(\theta_{\text{treat}}, \theta_{\text{ctrl}})$ factorizes and any pairing of marginal draws is a valid sample from the joint. The function trims to $S = \min(S_{\text{treat}}, S_{\text{ctrl}})$ when the two fits differ in number of draws and emits a `gdpar_diagnostic_warning`.

Multi-dimensional and K-individual fits. For `p > 1` (multivariate) and `K > 1` (distributional regression), `predict.gdpar_fit` returns a 3-array of shape `[S, n, dim]`; the CATE is computed elementwise and the per-coordinate or per-slot CATEs are returned as the last dimension of `cate_draws`. For `type = "response"`, the canonical inverse link of each coordinate or slot is applied by `predict.gdpar_fit` before the difference is taken; the resulting CATE is therefore on the natural response scale of each slot, not a uniform link-transformed scale.

Value

An object of class `gdpar_causal_bridge` with components `cate_draws` (matrix `[S, n]` when both fits are scalar, or array `[S, n, dim]` when both fits are multivariate or K-individual), `cate_mean`, `cate_ci`, `newdata`, `id_check`, `fits`, `type`, `level`, `n_draws`, `n_obs`, `call`, `warnings` (character vector recording fallback notifications such as posterior-draw trimming; empty in the happy path), and `meta`. The companion S3 methods `print` and `summary` are documented in `print.gdpar_causal_bridge` and `summary.gdpar_causal_bridge`.

Methodological notes

The T-learner is the most direct meta-learner to map onto the gdpar pipeline: each arm is one `gdpar_fit` and the CATE reuses the posterior machinery of `predict.gdpar_fit`. S-learner and X-learner are queued for Block 9. The T-learner is known to suffer from regularization-induced bias in unbalanced samples (see Kuenzel et al. 2019, Section 3.4); the bridge vignette discusses the trade-off and the alternatives.

Dependencies

Inherits the **posterior** dependency of `predict.gdpar_fit`.

References

Kuenzel, S. R., Sekhon, J. S., Bickel, P. J., and Yu, B. (2019). Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10), 4156-4165.

See Also

`gdpar`, `predict.gdpar_fit`

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  n <- 300L
  df <- data.frame(x1 = rnorm(2L * n))
  df$arm <- rep(c("treat", "ctrl"), each = n)
  df$y <- with(df, ifelse(arm == "treat", 0.5, 0) + 0.8 * x1 + rnorm(2L * n, sd = 0.5))
  df_treat <- subset(df, arm == "treat")
  df_ctrl  <- subset(df, arm == "ctrl")
  fit_t <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df_treat,
                 iter_warmup = 200, iter_sampling = 200, chains = 2)
  fit_c <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df_ctrl,
                 iter_warmup = 200, iter_sampling = 200, chains = 2)
  bridge <- gdpar_causal_bridge(fit_t, fit_c,
                                newdata = data.frame(x1 = seq(-2, 2, length.out = 21L)))
  print(bridge)
  summary(bridge)
}
```

---

gdpar_check_identifiability

Check basis-restricted functional independence via Gram matrix

---

Description

Diagnose, before model fitting, whether the chosen finite parametric representation of the AMM canonical form satisfies the basis-restricted Functional Independence Condition at a candidate value of the population reference. Failure indicates that the model parameters are not identifiable in the chosen basis and that fitting should not proceed without revising the specification.

Usage

```
gdpar_check_identifiability(
  amm,
  data,
  theta_ref_init = NULL,
  formula_rhs = NULL,
  family = NULL,
  tol = 1e-08,
  rigor = c("full", "fast")
)
```

Arguments

amm	An object of class amm_spec produced by amm_spec, defining the bases for the additive, multiplicative and modulating components.

`data` A data frame containing the variables referenced in `amm`. Covariates are centered internally before the Gram matrix is computed, consistent with assumption (C1) of Block 1.

`theta_ref_init` Numeric vector of length `p` (the dimension of the population reference) at which the diagnostic is computed. Defaults to a vector of zeros, which corresponds to the prior mean under the default `gdpar_prior` on the linear-predictor scale.

`formula_rhs` Optional formula or character vector identifying the covariates that enter the modulating component as the linear factor x. Defaults to `amm$x_vars`.

`family` Optional `gdpar_family` or `gdpar_family_multi` object. When supplied, the report includes the parameter-level identifiability (D-ID) pre-fit layer of Block 8 Session 1 decision (3C): for each individual-scope parameter declared by the family's `param_specs` (Block 8 Session 1 decision 1C), the declarative `did_status` is reported, and when the family declares K >= 2 individual parameters the symbolic separability between them is checked under `rigor = "full"`. In Block 8.0 K = 1 in every built-in family, so the layer reduces to a status echo; the slot is exposed for the K > 1 extension scheduled for Block 8.1. Defaults to `NULL` (slot omitted, backward compatible).

`tol` Numeric scalar with the tolerance for the relative condition number criterion. Defaults to `1e-8`. The diagnostic flags failure when the smallest eigenvalue of the normalized Gram matrix is below `tol` times the largest eigenvalue.

`rigor` Character scalar in `c("full", "fast")` controlling the C4-bis cross-component check for multivariate (`p > 1`) specs. Both modes (a) check the per-coordinate rank of $Z_a[k]$ via its normalized Gram condition number, and (b) detect structural overlap between the column names of $Z_a[k]$ and the modulating `X` columns (the `x_vars`). `"full"` (default) FAILS the report when any overlap is detected (conservative: structural overlap is a necessary condition for cross-component non-identifiability and the user should redesign the spec). `"fast"` emits a structured warning on overlap but does not flag failure (use when the user has explicitly accepted the overlap, e.g. via regularization). Note: the pre-fit check cannot detect cross- component non-identifiabilities that arise only through the posterior geometry (e.g. θ_{ref}-mediated coupling between `a` and `W`); those are surfaced post-fit via divergences, low ESS and high R-hat. Ignored when `p == 1`.

Details

Proposition 1C of Block 1 establishes that, in a chosen finite basis B for the AMM components, the basis-restricted Functional Independence Condition at a value theta_ref holds if and only if the population Gram matrix of the extended design matrix Z_n(theta_ref) is non-singular. This function computes the empirical Gram matrix from the observed data and checks its condition number.

The diagnostic is by design local to the supplied `theta_ref_init`: it reports the condition at one point in the parameter space. The basis-restricted Functional Independence Condition is a property of the basis, not of the point; in practice the diagnostic at the prior mean is informative because non-identifiability that is structural to the basis manifests at typical reference points.

The function does not test the abstract Functional Independence Condition, which is a property of the full function classes F_a, F_b, F_W. When these classes are infinite-dimensional and the basis

B is a finite truncation, abstract failure may occur in directions outside B and would not be detected here.

Value

An object of class `gdpar_identifiability_report` with components `passed` (logical), `lambda_min`, `lambda_max`, `condition_number`, `collinear_directions` (a list describing basis-function combinations corresponding to near-zero eigenvectors when the diagnostic fails, and NULL otherwise), `theta_ref_used`, `tol_used` and `column_labels`. A `print` method provides a human-readable summary.

Methodological notes

The threshold criterion is relative (smallest eigenvalue divided by the largest, compared against `tol`), not absolute. The relative criterion coincides with the inverse of the condition number, which is invariant to rescaling of the basis columns. An absolute eigenvalue threshold would depend on the scale of the covariates and would produce false positives when the basis matrix has columns of very different magnitudes. To make the diagnostic robust to scale heterogeneity across basis terms, the columns of the extended design matrix are normalized to unit norm before the Gram matrix is computed. Normalization affects only the diagnostic; model fitting uses the matrices in their natural scale.

When the diagnostic fails, `collinear_directions` is populated by projecting the eigenvectors associated with eigenvalues below `tol` times the largest eigenvalue onto the named basis columns, producing a human-readable description of which combinations of basis functions are linearly dependent.

Dependencies

This function calls `eigen` with `symmetric = TRUE` for the eigendecomposition of the Gram matrix, and `model.matrix` for evaluating the formula-based bases of the additive and multiplicative components.

References

Block 1 of the package theoretical addendum, Section 6.6, Proposition 1C. See `vignette("01_amm_identifiability", package = "gdpar")`.

See Also

`amm_spec`, `W_basis`, `gdpar`

Examples

```
set.seed(1)
df <- data.frame(x1 = rnorm(50), x2 = rnorm(50))
spec <- amm_spec(
  a = ~ x1 + x2,
  b = ~ x1,
  W = W_basis(type = "polynomial", degree = 1),
  x_vars = "x2"
```

```
)
report <- gdpar_check_identifiability(spec, df, theta_ref_init = 0.5)
print(report)
```

`gdpar_compare_eb_fb` *Compare an Empirical-Bayes fit against a Fully-Bayes fit*

Description

Sub-phase 8.6.E (Charter Section 3.5, decision 2.5 Trio of vignettes). Reports the operational comparison promised in v07 Section 11 between a `gdpar_eb_fit` (Empirical Bayes via `gdpar_eb`) and a `gdpar_fit` (Fully Bayes via `gdpar`) fitted on the same dataset: per-component differences in the population anchor θ_{ref}, the empirical total variation distance between the lower-level posteriors of $\xi = (a, b, W, \text{dispersion})$ marginally per parameter, and the operational verification of the higher-order coverage discrepancy of v07 Section 6 (Proposition 7B scalar / 7B* matricial / 7B* tensorial) on the nominal EB and FB credible intervals.

Usage

```
gdpar_compare_eb_fb(eb_fit, fb_fit, level = 0.95, tv_bins = 30L, ...)
```

Arguments

`eb_fit`	An object of class `gdpar_eb_fit` produced by `gdpar_eb`. Covers all four path regimes (K = 1 + p = 1; Path A K = 1 + p > 1; Path B K > 1 + p = 1; Path C K > 1 + p > 1 via the K x p tensor extension of Sub-phase 8.6.D).
`fb_fit`	An object of class `gdpar_fit` produced by `gdpar`. Must have been fitted on the same dataset as `eb_fit` (same outcome, same covariates, same K / p regime). The comparator does not refit either model.
`level`	Numeric scalar in (0, 1); credible-interval level for the coverage discrepancy reporting. Defaults to 0.95.
`tv_bins`	Integer scalar; number of histogram bins used to approximate the marginal TV distance per parameter. Defaults to 30. Larger values give a finer empirical TV but require more draws per parameter for stability.
`...`	Reserved for future arguments; currently unused.

Details

The comparator is descriptive: it does not assert algorithmic equivalence, nor does it test hypotheses across the EB and FB inferential frames. The TV distance is computed marginally parameter by parameter via histogram-based plug-in (relative bin counts at a common breakpoint grid); a finite-sample correction is not applied. Joint TV across the high-dimensional ξ typically requires kernel Stein discrepancy or similar density-free metrics that are out of scope of the initial 8.6.E iteration; the marginal TV reported here is the operational proxy recommended in v07 Section 11.1.

Value

An object of class `gdpar_eb_fb_comparison` with components `theta_diff_table`, `tv_table`, `coverage_table`, `level`, `tv_bins`, `n_common_params`, `path_eb`, `path_fb`, `call`, `warnings` (character vector recording per-helper fallback notifications: silent extraction failures of the FB `theta_ref` draws, missing EB or FB ξ draws, or zero-common-parameter TV inputs; empty in the happy path), and `meta`. See `print.gdpar_eb_fb_comparison` and `summary.gdpar_eb_fb_comparison` for the companion S3 methods.

Path coverage

The comparator handles all four EB regimes uniformly by extracting the per-element anchor estimate (vector / matrix / 3D array) and the corresponding lower-level posterior draws via the canonical `posterior::as_draws_matrix` interface. For Path C (K > 1 + p > 1) the `theta_ref_kp_hat` tensor is flattened to a length-K*p vector keyed by (slot, coord) for per-element comparison; the joint K x p inflation tensor is reported in the coverage_table per cell via its diagonal block entries.

Diagnostic value

Under the standing hypotheses of v07 Section 4 (EB-MARG-ID + PRIOR-FB-WEAK + HIER-COMPLEX), Theorem 7A predicts marginal TV -> 0 in probability as n -> Inf (specializing to Theorems 7A* / 7C* / 7C* compound multi-slot under Path A / Path B / Path C of v07b Sections 4-6). The empirical TV reported here is the operational diagnostic of this theoretical prediction. Persistent large marginal TV across ξ suggests one of the discrepancy conditions of Proposition 7D (multi-modality of the marginal likelihood, near-singular Fisher information, informative prior on theta_ref, deep hierarchy). The coverage_table operationally verifies the $O(n^{-1})$ under-cover claim of Proposition 7B by comparing EB-nominal vs FB-nominal IC widths per anchor cell.

References

Petrone, S., Rousseau, J., and Scricciolo, C. (2014). Bayes and empirical Bayes: do they merge? *Biometrika*, 101(2), 285–302.

Rousseau, J., and Szabo, B. (2017). Asymptotic behaviour of the empirical Bayes posteriors associated to maximum marginal likelihood estimator. *Annals of Statistics*, 45(2), 833–865.

Carlin, B. P., and Gelfand, A. E. (1990). Approaches for empirical Bayes confidence intervals. *JASA*, 85(409), 105–114.

See Also

`gdpar_eb`, `gdpar`, `vignette("v07_eb_vs_fb", package = "gdpar")`, `vignette("v07b_eb_multivariate", package = "gdpar")`, `vignette("vop07_eb_workflow", package = "gdpar")`.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("posterior", quietly = TRUE)) {
  set.seed(20260526L)
  n <- 120L
  df <- data.frame(x = stats::rnorm(n))
  df$y <- 0.5 + 0.4 * df$x + stats::rnorm(n, sd = 0.3)
```

```
  spec <- amm_spec(a = ~ x)
  fit_eb <- gdpar_eb(
    formula = y ~ x, family = gdpar_family("gaussian"),
    amm = spec, data = df,
    iter_warmup = 200L, iter_sampling = 200L, chains = 2L,
    refresh = 0L, verbose = FALSE, seed = 1L
  )
  fit_fb <- gdpar(
    formula = y ~ x, family = gdpar_family("gaussian"),
    amm = spec, data = df,
    iter_warmup = 200L, iter_sampling = 200L, chains = 2L,
    refresh = 0L, verbose = FALSE, seed = 1L
  )
  cmp <- gdpar_compare_eb_fb(fit_eb, fit_fb)
  print(cmp)
}
```

`gdpar_compare_meta_learners` *Compare the AMM-side T-learner against external meta-learners*

Description

Evaluate a fitted `gdpar_causal_bridge` object against a user-supplied set of external meta-learner adapters (e.g. `gdpar_adapter_grf` for **grf** R-side, `gdpar_adapter_econml` for EconML Python-side) on a common evaluation grid. The function does not refit either of the two gdpar fits embedded in `bridge`; it only consumes the bridge's CATE estimates and reconstructs the (X, T, Y) dataset needed by the external adapters from the captured calls of the two fits, or from an explicit `data` argument when the captured calls cannot be resolved.

Usage

```
gdpar_compare_meta_learners(
  bridge,
  methods,
  newdata = NULL,
  data = NULL,
  seed = NULL,
  ...
)
```

Arguments

`bridge`	An object of class `gdpar_causal_bridge` produced by `gdpar_causal_bridge`.
`methods`	A non-empty named or unnamed list of objects of class `gdpar_meta_learner_adapter`; the comparator orchestrates each adapter in turn. When the list is unnamed, the names of the methods are taken from the `name` field of each adapter.

newdata	Optional data frame on which the CATE is evaluated. When NULL (default), the function reuses bridge$newdata.
data	Optional list with components X (data frame of covariates), T (integer 0/1 vector), Y (numeric vector). Used when the captured calls of the two gdpar fits cannot be evaluated in the caller's environment (e.g. when the comparator is invoked from a wrapper that loses the data scope). When NULL (default), the function attempts to recover the training data of each arm from the bridge's stored fits.
seed	Optional integer scalar propagated to each adapter as seed_run; default NULL leaves the RNG state to each adapter's discretion.
...	Reserved for future arguments; currently unused.

Details

The comparator is descriptive: it reports per-method posterior / point CATE estimates together with their native CIs (when the adapter exposes one) and three concordance metrics (RMSE, Pearson correlation, mean absolute discrepancy) between every ordered pair of methods over cate_mean. Tests of hypothesis and claims of algorithmic equivalence are deliberately out of scope (the inferential origin of each method differs); the interpretation of the discrepancy is left to the user.

Value

An object of class gdpar_meta_learner_comparison with components bridge_cate, external, comparison, newdata, level, n_obs, n_methods, call, meta. See print.gdpar_meta_learner_comparison and summary.gdpar_meta_learner_comparison for the companion S3 methods.

Scalar-outcome restriction

The current scope of Sub-phase 8.5.B supports scalar outcomes only. Bridges constructed from fits with K > 1 (distributional regression) or p > 1 (multivariate response) are rejected with gdpar_unsupported_feature_error; multi-output external adapters are queued for Block 9 (see vignette v08c_meta_learner_comparison, section "Limits").

Dataset reconstruction

When data is NULL, the helper .assemble_bridge_dataset recovers the training data of each arm via eval(fit$call$data, eval_env) (same mechanism used by the bridge constructor in gdpar_causal_bridge), identifies the outcome via the LHS of fit$call$formula, and assembles a single (X, T, Y) dataset with T = 1L for the treatment arm and T = 0L for the control arm. When the evaluations fail, the helper aborts with gdpar_input_error and instructs the user to pass data explicitly.

Concordance metrics

For every ordered pair of methods (i, j) (including the bridge as a method indexed by "bridge"), the comparator reports

$$\mathrm{RMSE}_{ij} = \sqrt{\mathrm{mean}((\widehat{\tau}_i - \widehat{\tau}_j)^2)}$$

$$\mathrm{Pearson}_{ij} = \mathrm{cor}(\widehat{\tau}_i, \widehat{\tau}_j)$$

$$\mathrm{MAD}_{ij} = \mathrm{mean}(|\ \widehat{\tau}_i - \widehat{\tau}_j\ |)$$

computed on `cate_mean` only; CIs are not pooled across methods because the inferential origin of each CI is heterogeneous (posterior vs. asymptotic vs. bootstrap; see Appendix B of the bridge vignette).

References

Kuenzel, S. R., Sekhon, J. S., Bickel, P. J., and Yu, B. (2019). Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10), 4156-4165.

Athey, S., and Wager, S. (2019). Estimating treatment effects with causal forests: An application. *Observational Studies*, 5, 37-51.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1-C68.

See Also

`gdpar_causal_bridge`, `gdpar_meta_learner_adapter`, `gdpar_adapter_grf`, `gdpar_adapter_econml`.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("grf", quietly = TRUE)) {
  n <- 300L
  df <- data.frame(x1 = rnorm(2L * n))
  df$arm <- rep(c("treat", "ctrl"), each = n)
  df$y <- with(df, ifelse(arm == "treat", 0.5, 0) + 0.8 * x1 +
                  rnorm(2L * n, sd = 0.5))
  df_t <- subset(df, arm == "treat", select = -arm)
  df_c <- subset(df, arm == "ctrl",  select = -arm)
  fit_t <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df_t,
                 iter_warmup = 200, iter_sampling = 200, chains = 2)
  fit_c <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df_c,
                 iter_warmup = 200, iter_sampling = 200, chains = 2)
  bridge <- gdpar_causal_bridge(fit_t, fit_c,
                                newdata = data.frame(x1 = seq(-2, 2, 0.2)))
  cmp <- gdpar_compare_meta_learners(bridge,
                                     methods = list(gdpar_adapter_grf()))
  print(cmp); summary(cmp)
}
```

gdpar_contraction_diagnostic *Empirical posterior contraction rate diagnostic (opt-in, costly)*

Description

Verify numerically the predicted posterior contraction rate (Theorem 4B of Block 4) for a fitted Path 1 model. Refits the model at multiple subsample sizes, records the median posterior credible interval width across the user-facing parameters, and fits the regression $\log(\text{width}) = \alpha + \beta \log(n)$ across the subsample sizes. The slope estimate is consistent with a parametric $n^{-1/2}$ contraction rate when its value is in the interval (-0.6, -0.4).

Usage

```
gdpar_contraction_diagnostic(
  fit,
  data,
  sizes = NULL,
  replicates = 1L,
  parameters = NULL,
  level = 0.95,
  iter_warmup = 500L,
  iter_sampling = 500L,
  chains = 2L,
  verbose = TRUE,
  ...
)
```

Arguments

`fit`	An object of class `gdpar_fit` produced by `gdpar` with `path = "bayes"`.
`data`	The data frame originally passed to `gdpar` (or another data frame compatible with the AMM specification of `fit`).
`sizes`	Integer vector with the subsample sizes at which to refit. Defaults to a length-five geometric sequence between `ceiling(n / 8)` and `n`.
`replicates`	Integer scalar with the number of independent subsamples per size. Defaults to 1; a higher value reduces Monte Carlo variance of the curve at additional cost.
`parameters`	Optional character vector of parameter names to include in the credible-width calculation. Defaults to the user-facing parameters.
`level`	Numeric scalar in (0, 1) with the nominal credible level used for the width calculation. Defaults to 0.95.
`iter_warmup`	Integer scalar; warmup iterations for each refit. Defaults to 500.
`iter_sampling`	Integer scalar; sampling iterations for each refit. Defaults to 500.
`chains`	Integer scalar; chains per refit. Defaults to 2.

verbose	Logical scalar; when TRUE, prints an estimated cost message before starting. Defaults to TRUE.
...	Additional arguments forwarded to gdpar.

Details

This function is opt-in and computationally expensive: it refits the model `length(sizes) * replicates` times. A cost message is printed at the start. The conclusions of gdpar are not affected by calling this function.

Theorem 4B of Block 4 establishes that, under the conditions of Theorem 4A plus the prior thickness condition (PRIOR-THICK) and the sieve condition (SIEVE), the posterior contracts at rate ε_n with $n\varepsilon_n^2 \to \infty$. For finite-dimensional parametric AMM specifications, the rate is $n^{-1/2}$. This diagnostic checks the empirical slope of the log-width against log-n.

Deviations from the predicted slope can indicate (i) prior misspecification (the prior fails (PRIOR-THICK) at the true parameter), (ii) failure of the homogeneity (HOM) or regularity (REG) conditions of Block 2, or (iii) non-parametric components whose smoothness assumption does not match the truth. The diagnostic flags the discrepancy without diagnosing the cause.

Value

A list of class `gdpar_contraction_report` with components `table` (data frame with columns `n`, `replicate`, `median_width`), `slope_estimate`, `slope_se`, `slope_ci_lower`, `slope_ci_upper`, `verdict` (a character indicating whether the empirical slope is consistent with the parametric $n^{-1/2}$ rate), and `warnings` (character vector recording per-refit fallback notifications; empty when every refit succeeded). A `print` method provides a human-readable summary.

Methodological notes

This function is part of the methodological audit toolkit, not of the standard inference flow. It is computationally expensive: each subsample size requires a full refit. The default settings keep the refits short (500 + 500 iterations, 2 chains) to make the diagnostic affordable on moderate datasets; users with more computational budget should pass higher values via the relevant arguments.

Subsamples are drawn without replacement and stratified by row order; users with structured data (time series, clustered observations) should pass an explicit `sizes` vector that respects the structure if random subsampling is inappropriate.

Dependencies

Uses **cmdstanr** for the refits and **posterior** to extract credible-interval widths.

References

See `vignette("v04_asymptotics_path1_bayesian", package = "gdpar")`, Section 6.2 (numerical verification of contraction).

See Also

gdpar, gdpar_bvm_check

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  df <- data.frame(x1 = rnorm(400), y = rnorm(400))
  fit <- gdpar(y ~ x1, amm = amm_spec(a = ~ x1), data = df,
               iter_warmup = 200, iter_sampling = 200, chains = 2)
  gdpar_contraction_diagnostic(fit, data = df, replicates = 1)
}
```

---

gdpar_dependence_diagnostic

Residual dependence diagnostic for a scalar Empirical-Bayes fit

---

Description

Quantifies serial (temporal) dependence in the residuals of a fitted scalar Path 1 Empirical-Bayes model. gdpar assumes conditional independence; under temporal (or spatial) autocorrelation that assumption is violated and the model-based (posterior / Laplace) uncertainty is too narrow. This diagnostic makes the violation *visible and measurable* before any remedy is applied, and is the natural gate for gdpar_dependence_robust.

Usage

```
gdpar_dependence_diagnostic(
  object,
  index = NULL,
  residual_type = c("quantile", "response", "pearson", "deviance"),
  max_lag = NULL,
  level = 0.95,
  randomize_seed = NULL,
  ...
)
```

Arguments

object	A scalar Path 1 fit (K = 1, p = 1): either a gdpar_eb_fit (Empirical Bayes, from gdpar_eb) or a gdpar_fit (full Bayes, from gdpar).
index	Optional vector of length n giving the temporal (or any one-dimensional) ordering of the observations. Residuals are sorted by order(index) before the autocorrelation statistics are computed. When NULL (default) the natural row order of the training data is used.
residual_type	One of "quantile" (default; randomized quantile / Dunn-Smyth residuals, the gdpar canonical choice), "response", "pearson" or "deviance". Forwarded to the internal residual dispatcher.
max_lag	Integer scalar; the maximum lag for the Ljung-Box test. Defaults to min(floor(10 * log10(n)), n - 1).

`level`	Numeric scalar in (0, 1); the confidence level used to turn the test p-values into the verdict. Defaults to 0.95 (i.e. dependence is flagged when a p-value falls below `1 - level`).
`randomize_seed`	Optional integer seed used by the randomized quantile residuals for discrete families; ignored otherwise. Pass a value for reproducibility.
`...`	Unused; present for signature stability.

Details

The Durbin-Watson statistic is reported descriptively as $DW = \sum_{t=2}^{n}(r_t - r_{t-1})^2 / \sum_{t=1}^{n} r_t^2$ ($DW \approx 2(1 - \hat{\rho}_1)$); values near 2 indicate no first-order autocorrelation. The Ljung-Box test (`stats::Box.test`) provides the omnibus p-value across lags and drives the verdict. The Ljung-Box degrees of freedom are not reduced by the number of estimated AMM coefficients (`fitdf = 0`); for residuals of a fitted model this makes the test mildly optimistic, a caveat stated honestly rather than masked.

Spatial dependence (Moran's I and a spatial weight structure) is handled by the sibling gdpar_spatial_dependence_diagno Both the scalar Empirical-Bayes (`gdpar_eb_fit`) and the scalar full-Bayes (`gdpar_fit`) paths are supported (decision D102); the residuals are the Bayesian Dunn-Smyth residuals of whichever fit is supplied. The K > 1 / p > 1 paths remain deferred.

Value

A list of class `gdpar_dependence_diagnostic` with components `residual_type`, `n`, `max_lag`, `lag1_autocorr`, `lag1_p_value` (normal approximation $\sqrt{n}\,\hat{\rho}_1 \sim N(0,1)$), `durbin_watson`, `ljung_box_statistic`, `ljung_box_df`, `ljung_box_p_value`, `level`, `index_supplied` and `verdict`. A `print` method provides a human-readable summary.

Methodological note

A flagged verdict says the model-based uncertainty is not trustworthy under the detected dependence, not that the point estimates are wrong. The companion remedy gdpar_dependence_robust re-estimates the uncertainty by a temporal block bootstrap.

References

Ljung, G. M. & Box, G. E. P. (1978). On a measure of lack of fit in time series models. *Biometrika* 65(2), 297-303.

Durbin, J. & Watson, G. S. (1950). Testing for serial correlation in least squares regression. I. *Biometrika* 37(3/4), 409-428.

See Also

gdpar_dependence_robust, gdpar_eb

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("posterior", quietly = TRUE)) {
  n <- 100
```

```
  x <- rnorm(n)
  y <- 1 + 0.5 * x + as.numeric(stats::arima.sim(list(ar = 0.6), n))
  df <- data.frame(x = x, y = y, t = seq_len(n))
  fit <- gdpar_eb(y ~ x, amm = amm_spec(a = ~ x), data = df,
                  chains = 2, iter_warmup = 100, iter_sampling = 100)
  gdpar_dependence_diagnostic(fit, index = df$t)
}
```

`gdpar_dependence_robust` *Dependence-robust standard errors via a temporal block bootstrap*

Description

Re-estimates the uncertainty of a scalar Path 1 Empirical-Bayes fit so that it is robust to temporal (serial) dependence in the data, without modelling that dependence. It refits the model on `B` moving (or circular) block bootstrap resamples of the data ordered by `index`, and reports the bootstrap standard deviation and percentile intervals of each AMM coefficient alongside the model-based (Laplace / posterior) standard errors. This is the working-independence + robust-variance stance of Liang & Zeger (1986): the point estimates are unchanged (consistent when the mean structure is correct, not efficient), only the reported uncertainty is made dependence-robust.

Usage

```
gdpar_dependence_robust(
  object,
  data,
  index = NULL,
  block_length = NULL,
  residual_type = c("quantile", "response", "pearson", "deviance"),
  randomize_seed = NULL,
  type = c("moving", "circular"),
  B = 199L,
  level = 0.95,
  seed = NULL,
  iter_warmup = 500L,
  iter_sampling = 500L,
  chains = 2L,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	A scalar Path 1 fit (`K = 1`, `p = 1`): either a `gdpar_eb_fit` (Empirical Bayes, from `gdpar_eb`) or a `gdpar_fit` (full Bayes, from `gdpar`).

data	The data frame originally passed to the fitting function (gdpar_eb or gdpar; the fit object deliberately does not store the data, to stay lightweight). It is resampled by contiguous blocks and the model is refit on each resample; the model specification (formula, AMM, family, prior) is recovered from object$call.
index	Optional vector of length n giving the temporal ordering of the rows of data. The data are sorted by order(index) so that contiguous blocks correspond to contiguous time. When NULL (default) the natural row order is assumed to be the temporal order.
block_length	The block size, one of three forms: NULL (default) uses the rate-optimal max(1, round(n^(1/3))) (Kuensch 1989; Hall, Horowitz & Jing 1995); a positive integer fixes it manually; or the string "auto" selects it data-drivenly by the Politis & White (2004) automatic rule (with the Patton, Politis & White 2009 correction), computed from the fitted residuals (no extra refit), falling back to the rate on a degenerate series. The chosen value and method are reported in the result (block_length, block_length_method).
residual_type	One of "quantile" (default; Dunn-Smyth randomized quantile residuals), "response", "pearson" or "deviance". Used **only** when block_length = "auto", to feed the Politis-White selector; ignored otherwise.
randomize_seed	Optional integer seed for the randomized quantile residuals of discrete families; used only by the "auto" selector, for a reproducible block-length choice. Ignored otherwise.
type	One of "moving" (default) or "circular" block bootstrap.
B	Integer scalar; the number of bootstrap refits. Defaults to 199.
level	Numeric scalar in (0, 1); the percentile-interval level. Defaults to 0.95.
seed	Optional integer seed controlling both the block resampling and (deterministically derived) the per-refit Stan seeds, for full reproducibility.
iter_warmup, iter_sampling, chains	Integer scalars controlling each refit's conditional HMC. Defaults (500, 500, 2) keep the refits short.
verbose	Logical scalar; when TRUE, prints an opt-in cost message.
...	Additional arguments forwarded to gdpar_eb for every refit.

Value

A list of class gdpar_dependence_robust with components table (data frame with one row per coefficient and columns estimate, model_se, robust_se, se_ratio, ci_lower, ci_upper), block_length, block_length_method ("rate", "fixed" or "auto"; "rate" also flags an "auto" request that fell back), type, B, B_ok (successful refits), level, index_supplied, seed, warnings and refit_diagnostics (aggregate per-refit convergence: max_rhat, min_ess_bulk, n_divergent_refits, n_high_rhat_refits). A print method provides a human-readable summary.

Honest scope

The bootstrap delivers robust variance, not better point estimates, and is valid for weak / short-range dependence relative to block_length; it does not rescue long-memory or unit-root processes. gdpar does not model the dependence (that is deferred to a future block); here it only makes its inference robust to it.

Empirical-Bayes and full-Bayes paths

Both the scalar Empirical-Bayes (`gdpar_eb_fit`) and the scalar full-Bayes (`gdpar_fit`) paths are supported (decision D102), through a single shared engine; only the per-fit extraction of the point estimate, the model SE and the residuals is class-dispatched. On the EB path the point estimate / model SE are the Laplace / conditional-posterior mean and SD; on the full-Bayes path they are the **posterior mean and posterior SD** of each AMM coefficient (`theta_ref`, `a_coef`, `b_coef`, `W_raw`, the latter on its raw scale, for parity with the EB extractor). In both cases `robust_se` is the block-bootstrap SD of the per-refit point estimate and `se_ratio` = `robust_se` / `model_se` is a like-for-like SD-vs-SD ratio: it contrasts the dependence-robust sampling variability of the point estimate against the within-model (posterior / Laplace) SD, and exceeds 1 when the latter understates the former. The posterior mean / SD choice (rather than median / IQR) preserves this parity and avoids an undeclared normal-scaling constant. (For a strongly skewed coefficient posterior – `W_raw` under sparse data or strong shrinkage is the usual culprit – the posterior mean can sit off the posterior mode; inspect `object$fit$draws()` directly in that case. The reported point estimate is always the posterior mean.)

Three honest full-Bayes caveats. (1) Each refit re-runs the **full** HMC (markedly more costly than an EB refit). (2) A finite-iteration refit carries Monte-Carlo error in its posterior mean that slightly and *conservatively* inflates `robust_se` (reducible by a larger `iter_sampling`; the aggregate refit ESS is reported in `refit_diagnostics`). (3) The `se_ratio` has a subtly different reading across paths: under an **informative prior** the full-Bayes posterior SD can be smaller than the bootstrap SD *even under correct independent specification*, because the prior concentrates the posterior beyond what the data alone support, giving `se_ratio` < 1. That is benign prior regularization, not the model SE overstating uncertainty; only `se_ratio` clearly **above** 1 signals dependence / misspecification (the analytic conjugate-Gaussian check gives $\mathrm{se\_ratio}^2 = n\tau/(n\tau + \tau_0) < 1$ with prior precision τ_0). Note too that the EB and full-Bayes `theta_ref` point estimates are different estimands – the Laplace **mode** (EB) versus the posterior **mean** (full Bayes) – which coincide asymptotically (Bernstein-von Mises) but may differ in finite samples; their `se_ratio` values are therefore not expected to match to the last digit when the same data are run through both paths. A widened / bagged posterior (BayesBag; Huggins & Miller 2019) is a different object – a re-architected estimator rather than a robust variance for the same one – and is a documented deferred lateral, not adopted here.

Dependencies

Uses **cmdstanr** for the refits and **posterior** to extract the coefficient estimates (Empirical-Bayes or full-Bayes).

References

Liang, K.-Y. & Zeger, S. L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika* 73(1), 13-22.

Kuensch, H. R. (1989). The jackknife and the bootstrap for general stationary observations. *Annals of Statistics* 17(3), 1217-1241.

Politis, D. N. & White, H. (2004). Automatic block-length selection for the dependent bootstrap. *Econometric Reviews* 23(1), 53-70.

Patton, A., Politis, D. N. & White, H. (2009). Correction to "Automatic block-length selection for the dependent bootstrap". *Econometric Reviews* 28(4), 372-375.

See Also

gdpar_dependence_diagnostic, gdpar_eb

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("posterior", quietly = TRUE)) {
  n <- 100
  x <- rnorm(n)
  y <- 1 + 0.5 * x + as.numeric(stats::arima.sim(list(ar = 0.6), n))
  df <- data.frame(x = x, y = y, t = seq_len(n))
  fit <- gdpar_eb(y ~ x, amm = amm_spec(a = ~ x), data = df,
                  chains = 2, iter_warmup = 100, iter_sampling = 100)
  # B is kept small here for a fast example; use B >= 199 in practice.
  gdpar_dependence_robust(fit, data = df, index = df$t, B = 10,
                          seed = 1, iter_warmup = 100,
                          iter_sampling = 100, chains = 2)
  # Data-driven block length (Politis-White), opt-in:
  gdpar_dependence_robust(fit, data = df, index = df$t,
                          block_length = "auto", B = 10, seed = 1,
                          iter_warmup = 100, iter_sampling = 100, chains = 2)
}
```

---

gdpar_dharma_object *Build a DHARMa simulation object from a fitted gdpar model*

Description

If **DHARMa** is available (Suggests), returns a DHARMa::DHARMa object built from the posterior predictive draws of y_pred and the observed response y. The user can then call DHARMa::testResiduals(), DHARMa::testZeroInflation(), DHARMa::plotResiduals() and other DHARMa tests on the returned object.

Usage

```
gdpar_dharma_object(object, coord = NULL)
```

Arguments

object	An object of class gdpar_fit.
coord	Integer scalar between 1 and p; required for multivariate fits. Ignored for scalar / K-individual paths.

Value

A DHARMa simulation object.

Dependencies

Requires **DHARMa** (Suggests). If the package is not installed the function raises `gdpar_input_error` pointing to `residuals(., type = "quantile")` as the built-in fallback.

`gdpar_eb` *Fit an AMM canonical model via Empirical Bayes (EB)*

Description

Path 1 Empirical-Bayes counterpart of `gdpar`: estimates the population reference θ_{ref} by maximizing the marginal likelihood (Type II ML) and samples the lower-level parameters $\xi = (a, b, W, \sigma_*, \phi)$ from the conditional posterior given $\widehat{\theta}_{ref}^{EB}$. See `vignette("v07_eb_vs_fb", package = "gdpar")` for the Fully-Bayes versus Empirical-Bayes asymptotic comparison ($p = 1$, $K = 1$) and `vignette("v07b_eb_multivariat` `package = "gdpar")` for the multivariate extension that motivates the family of dedicated templates introduced in this sub-phase.

Usage

```
gdpar_eb(
  formula,
  family = gdpar_family("gaussian"),
  amm = amm_spec(),
  W = NULL,
  data,
  prior = NULL,
  anchor = "prior_mean",
  skip_id_check = FALSE,
  chains = 4L,
  iter_warmup = 1000L,
  iter_sampling = 1000L,
  adapt_delta = 0.95,
  max_treedepth = 12L,
  refresh = 100L,
  verbose = TRUE,
  seed = NULL,
  group = NULL,
  parametrization = c("auto", "ncp", "cp"),
  id_check_rigor = c("full", "fast"),
  eb_correction = TRUE,
  laplace_control = list(),
  ...
)
```

Arguments

`formula`	A two-sided formula `y ~ ...`; same semantics as `gdpar`'s `formula` argument.

`family` A `gdpar_family` object. Sub-phase 8.6.B exclusively supports the K = 1 univariate families whose canonical Stan dispatch lives in `inst/stan/amm_main.stan`, namely the four families with `stan_id` in `c(1, 2, 3, 4)` (Gaussian, Poisson, neg-binomial-2, Bernoulli). Canonical K >= 2 families (Beta, Gamma, Student-t, Tweedie, lognormal-loc-scale) and mixtures (ZIP, ZINB, Hurdle-Poisson, Hurdle-NB) are rejected with `gdpar_unsupported_feature_error`; their EB habilitation is wired into Sub-phase 8.6.C jointly with the K > 1 relax.

`amm` An `amm_spec` object with `amm$p == 1L`. Multivariate specifications are rejected.

`W` Optional `W_basis` object. Polynomial and B-spline modulating bases are supported identically to `gdpar`.

`data` Data frame containing the variables referenced by `formula` and `amm`.

`prior` Optional `gdpar_prior` object. When `NULL`, the package defaults are used.

`anchor` Either a numeric scalar or one of `"prior_mean"` (default) and `"empirical_y"`.

`skip_id_check` Logical scalar; identical semantics to `gdpar`.

`chains` Integer scalar; number of HMC chains for Step (iii) conditional sampling. Defaults to 4.

`iter_warmup`, `iter_sampling`
Integer scalars; HMC warmup and sampling iterations per chain. Defaults to 1000.

`adapt_delta`, `max_treedepth`, `refresh`, `verbose`, `seed`
Identical semantics to `gdpar`.

`group` Optional one-sided formula identifying the grouping variable in `data`. Same semantics as `gdpar`.

`parametrization`
Character scalar selecting the CP/NCP sampling parametrization for the additive and modulating components in Step (iii). One of `"auto"` (default; runs the pre-flight diagnostic), `"ncp"`, `"cp"`.

`id_check_rigor` Character scalar, one of `"full"` or `"fast"`.

`eb_correction` Logical scalar; when `TRUE` (default), apply the scalar Proposition 7B coverage-discrepancy inflation factor to the conditional credible intervals (see v07 Section 6 and v07b Section 5.2). When `FALSE`, the credible intervals are nominal and a `gdpar_diagnostic_warning` is issued advising of the expected $O(n^{-1})$ undercover.

`laplace_control`
Named list controlling the Step (i) Laplace approximation and the anti-fragility strategy of Charter Section 2.8. Recognized entries (all optional, with documented defaults):

- `multi_start_M` (integer; default 5): number of independent random inits.
- `kappa_threshold` (numeric; default 1e10): maximum condition number of the marginal Hessian (after adaptive ridge) before `gdpar_eb_numerical_error` is raised.
- `ridge_init` (numeric; default 1e-6): initial L-M ridge value used by the adaptive perturbation helper.

- epsilon_lm (numeric; default sqrt(.Machine$double.eps), approximately 1.5e-8): the adaptive Levenberg-Marquardt ridge triggers when either the posterior covariance has a non-positive or non-finite eigenvalue, or its determinant is strictly smaller than this threshold (canon *|det(H)| < epsilon_LM* of Charter Section 2.8). Canonized in Sub-bloque 9.3.b (Sesion B9.2, 2026-05-27).
- ridge_max_iter (integer; default 10): maximum iterations of the adaptive geometric ridge loop. Each iteration multiplies the current lambda by ridge_grow_factor; the loop terminates with status "converged" once the post-ridge condition number is at or below kappa_threshold, or with status "exhausted" after ridge_max_iter attempts.
- ridge_grow_factor (numeric; default 10.0, must be > 1): geometric growth factor for the L-M ridge across iterations.
- laplace_draws (integer; default 1000): number of Gaussian draws produced per Laplace call (used to estimate the marginal covariance of θ_{ref}).
- optim_algorithm (character; default "lbfgs"): algorithm forwarded to cmdstanr::optimize().

... Additional arguments forwarded to the underlying HMC sampler of Step (iii).

Value

An object of class gdpar_eb_fit with components:

- theta_ref_hat: numeric vector of length J_groups with the EB point estimates.
- theta_ref_se: numeric vector of length J_groups with the marginal standard errors derived from the Laplace covariance.
- conditional_fit: the underlying cmdstanr fit object of Step (iii).
- correction_applied: logical scalar.
- eb_correction_constant: numeric scalar; the scalar Proposition 7B inflation constant when eb_correction = TRUE, NA_real_ otherwise.
- diagnostics_numerical: list with kappa, lm_perturbation, lm_n_iter, lm_status (one of "not_needed", "converged", "exhausted"), kappa_post_ridge, multi_start_dispersion, marginal_log_lik_history. For Path C (K > 1 and p > 1) the slot-vectorized counterparts replace the scalars: kappa_per_slot, lm_lambda_per_slot, lm_n_iter_per_slot, lm_status_per_slot.
- diagnostics: gdpar_diagnostics object from the conditional HMC fit (same shape as in gdpar_fit).
- amm, family, prior, design, anchor, stan_data, group_info, identifiability_report, parametrization, call, path ("eb").

Scope of Sub-phase 8.6.B

This version implements the base regime $K = 1$, $p = 1$ (single distributional slot per group, scalar θ_{ref}). Multivariate $p > 1$ and multi-slot $K > 1$ are explicitly rejected by the input guard with gdpar_unsupported_feature_error; their habilitation is canonized in Charter Sub-phases 8.6.C and 8.6.D and exercises the K_slots / p_dim fields declared in the two dedicated templates inst/stan/amm_eb_marginal.stan and inst/stan/amm_eb_conditional.stan.

Dependencies

Uses **cmdstanr** (Suggests) for Laplace approximation (`cmdstanr::laplace()`) in Step (i) and HMC sampling in Step (iii); **posterior** (Imports) for diagnostics. The Laplace approximation requires the `laplace()` method of `cmdstanr`, available since cmdstanr 0.7.0.

References

Carlin, B. P., and Gelfand, A. E. (1990). Approaches for empirical Bayes confidence intervals. JASA 85(409), 105–114.

Petrone, S., Rousseau, J., and Scricciolo, C. (2014). Bayes and empirical Bayes: do they merge? Biometrika 101(2), 285–302.

Rousseau, J., and Szabo, B. (2017). Asymptotic behaviour of the empirical Bayes posteriors associated to maximum marginal likelihood estimator. Annals of Statistics 45(2), 833–865.

See Also

`gdpar`, `amm_spec`, `gdpar_family`, `gdpar_prior`.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(NULL)
  n <- 200
  df <- data.frame(
    x1 = stats::rnorm(n),
    x2 = stats::rnorm(n)
  )
  df$y <- with(df, 1 + 0.5 * x1 - 0.3 * x2 + stats::rnorm(n, sd = 0.5))
  spec <- amm_spec(a = ~ x1 + x2)
  fit_eb <- gdpar_eb(
    formula       = y ~ x1 + x2,
    family        = gdpar_family("gaussian"),
    amm           = spec,
    data          = df,
    iter_warmup   = 200,
    iter_sampling = 200,
    chains        = 2
  )
  print(fit_eb)
}
```

`gdpar_family` *Construct a family object for AMM fitting*

Description

Define the response distribution that links the individual parameter theta_i to the observed outcome y_i. The family object carries the link function, the inverse link, the metadata for the parameter identifiability condition (D-ID) of Lemma 1B in Block 1, and the family identifier consumed by the Stan code generator.

Usage

```
gdpar_family(
  name = c("gaussian", "poisson", "neg_binomial_2", "bernoulli", "beta", "gamma",
   "student_t", "tweedie", "zip", "zinb", "hurdle_poisson", "hurdle_neg_binomial_2"),
  link = NULL,
  did_override = NULL
)
```

Arguments

`name`	Character scalar identifying the family. One of `"gaussian"`, `"poisson"`, `"neg_binomial_2"`, `"bernoulli"`, `"beta"`, `"gamma"`, `"student_t"`, `"tweedie"`, `"zip"`, `"zinb"`, `"hurdle_poisson"` or `"hurdle_neg_binomial_2"` for built-in families. For user-defined families, see `gdpar_family_custom`.
`link`	Character scalar identifying the link function. The default is the canonical link for each family.
`did_override`	Optional named list to override the canonical identifiability descriptors of one or more slots without changing the likelihood or links of the family. Keys are slot names of the family's `param_specs` (e.g., `"mu"`, `"sigma"`, `"nu"` for Student-t); each value is a list with optional fields `did_status`, `did_condition`, `did_reference`. `did_status` must be one of `"holds"`, `"holds_under_condition"`, `"user_responsible"`. Defaults to `NULL` (canonical D-ID stays in place). Use this when your design violates the canonical D-ID assumption of the family and you want the pre-fit identifiability report to reflect that, without forking the likelihood through the custom-family surface.

Details

The Path 1 implementation in this version of the package fits the AMM canonical form on the linear-predictor scale of the family. The inverse link is applied at the likelihood block in Stan; the centering, anchoring and prior specifications all live on the linear-predictor scale, consistent with assumptions (C1)-(C6) of Block 1 and the prior conditions (PRIOR-KL), (PRIOR-THICK) of Block 4.

Built-in families and their D-ID status:

`"gaussian"` Identity link by default. D-ID holds unconditionally for the location parameter when the variance is identifiable from the data.

`"poisson"` Log link by default. D-ID holds unconditionally on the rate parameter.

`"neg_binomial_2"` Log link by default; Stan's neg_binomial_2 parametrization with mean mu and dispersion phi such that variance equals mu + mu^2 / phi. D-ID holds unconditionally for mu when phi is identifiable from the data.

"bernoulli" Logit link by default. D-ID holds unconditionally on the success probability.

"beta" Logit link by default for the mean mu on (0, 1); precision phi on log link. Stan parametrization beta_proportion(mu, phi) with variance mu*(1-mu)/(1+phi). D-ID holds for mu unconditionally; for phi when the empirical dispersion identifies it (sub-phase 8.3.4 of Block 8).

"gamma" Log link by default for the mean mu on positive reals; shape on log link. Stan parametrization gamma(shape, shape/mu) with variance mu^2/shape. D-ID holds for mu unconditionally; for shape when the empirical dispersion identifies it (sub-phase 8.3.4 of Block 8).

"student_t" Identity link by default for the location mu on the real line; sigma on log link; nu (degrees of freedom) on log link. Stan parametrization student_t(nu, mu, sigma) with mean mu (for nu > 1) and variance sigma^2 * nu / (nu - 2) (for nu > 2). D-ID holds for mu unconditionally; for sigma when the empirical residual dispersion identifies it; for nu when the empirical tail thickness identifies it. Sub-phase 8.3.5a of Block 8 wires the family exclusively under the K = 3 distributional regression path of amm_distrib_K.stan; routings with K < 3 (population-scoped sigma and / or nu) are deferred.

"tweedie" Log link by default for the mean mu on positive reals; dispersion phi on log link; power p on identity link with support on the open interval (1.01, 1.99) (canonical compound Poisson-gamma regime, with point mass at zero and continuous positive body). Stan parametrization tweedie(mu, phi, p) with mean mu and variance phi * mu^p. The log-pdf is not native in Stan and is implemented in the model's functions block as a hybrid: the Dunn–Smyth (2005) infinite-series expansion in the central region |p - 1.5| < tau (tau = 0.4 by default) and the saddlepoint approximation elsewhere, with the same dispatch applied to the random-number generator. D-ID holds for mu unconditionally; for phi when the empirical dispersion identifies it; for p when the empirical balance between zero mass and continuous body identifies it. Sub-phase 8.3.5b of Block 8 wires the family exclusively under the K = 3 distributional regression path of amm_distrib_K.stan.

"zip" Zero-inflated Poisson. Log link by default for the count mean mu on positive reals; mixture probability pi on logit link with support on (0, 1). Likelihood: with probability pi the outcome is a structural zero, otherwise it is drawn from Poisson(mu). Stan implementation via log_sum_exp(bernoulli_logit_lpmf(1 | eta_pi), bernoulli_logit_lpmf(0 | eta_pi) + poisson_log_lpmf(0 | eta_mu)) for y = 0 and bernoulli_logit_lpmf(0 | eta_pi) + poisson_log_lpmf(y | eta_mu) for y > 0. D-ID holds for mu unconditionally; for pi when the empirical proportion of structural zeros, distinct from the sampling zeros of the count component, is sufficient to identify the zero-inflation probability. Sub-phase 8.3.6 of Block 8 wires the family exclusively under the K = 2 distributional regression path of amm_distrib_K.stan; routings with K < 2 are deferred.

"zinb" Zero-inflated negative binomial. Log link by default for the count mean mu on positive reals; dispersion phi on log link; mixture probability pi on logit link. Likelihood: with probability pi the outcome is a structural zero, otherwise it is drawn from NegBinomial2(mu, phi) with variance mu + mu^2 / phi. D-ID holds for mu unconditionally; for phi when the empirical overdispersion identifies it; for pi as in the zero-inflated Poisson case. Sub-phase 8.3.6 wires the family exclusively under the K = 3 distributional regression path; routings with K < 3 are deferred.

"hurdle_poisson" Hurdle Poisson. Log link by default for the truncated-count mean mu on positive reals; hurdle probability pi on logit link. Likelihood: a Bernoulli draw with probability pi decides whether the outcome equals zero or is strictly positive; the positive branch is drawn from a Poisson truncated at one. Stan implementation via bernoulli_logit_lpmf(1 |

eta_pi) for y = 0 and bernoulli_logit_lpmf(0 | eta_pi) + poisson_log_lpmf(y | eta_mu) - log1m_exp(-exp(eta_mu)) for y > 0. Distinct from the zero-inflated Poisson in that the zero mass is a structural decision, not a mixture between a structural zero and a Poisson sampling zero. D-ID holds for mu unconditionally; for pi when the empirical proportion of zeros is sufficient to identify the hurdle probability. Sub-phase 8.3.6 wires the family exclusively under the K = 2 distributional regression path.

"hurdle_neg_binomial_2" Hurdle negative binomial. Log link by default for the truncated-count mean mu; dispersion phi on log link; hurdle probability pi on logit link. Likelihood: Bernoulli draw decides zero vs. positive; the positive branch is drawn from a NegBinomial2(mu, phi) truncated at one. D-ID holds for mu unconditionally; for phi as in negative binomial; for pi as in hurdle Poisson. Sub-phase 8.3.6 wires the family exclusively under the K = 3 distributional regression path.

Value

An object of class gdpar_family with components name, link, inv_link, linkfun, stan_id, has_dispersion, did_status, did_condition and did_reference. A print method provides a human-readable summary.

Methodological notes

Identifiability of the response family in its parameter is a structural property of the model, not a property that can be tested from finite data. The package therefore documents this property through the did_status field but does not attempt to verify it at fitting time. When the family carries a conditional D-ID status, the fitting routine emits an informative message reminding the user of the relevant condition. See Lemma 1B and the commentary in Block 1, Section 6.4.

The returned family declares every structural parameter the distribution admits as *eligible* for an individual specification (e.g., mu and sigma for Gaussian; mu and phi for negative binomial; mu for Poisson and Bernoulli). Each eligible parameter carries a canonical scope in its gdpar_param_spec: the location parameter is per_observation (it is modeled through the AMM canonical form) and auxiliary parameters default to population. K-individual membership (which auxiliaries are promoted to per_observation) is declared exclusively at the entry of gdpar via a gdpar_formula_set (for the high-level formula path) or via a named list of amm_spec objects (for the low-level path). Parameters that are not named in that entry keep their canonical scope and are estimated as population-level constants. The family is the registry of eligibles, and the entry to gdpar() is the single source of truth for K.

Dependencies

This function uses **stats** family objects (gaussian, poisson, binomial) for the link metadata.

References

See vignette("01_amm_identifiability", package = "gdpar"), Section 6.4 (Lemma 1B) for D-ID; Lambert (1992), Greene (1994) and Mullahy (1986) for identifiability of zero-inflated and hurdle counts (sub-phase 8.3.6 of Block 8).

See Also

gdpar_family_custom, gdpar

Examples

```
fam <- gdpar_family("poisson")
print(fam)
```

---

gdpar_family_custom *Construct a custom family object for AMM fitting*

Description

Build a user-defined family for use with gdpar when the built-in families of gdpar_family do not cover the application. The user is responsible for declaring whether the identifiability condition (D-ID) of Lemma 1B in Block 1 holds for the family.

Usage

```
gdpar_family_custom(
  name,
  link,
  did_holds,
  did_condition,
  stan_loglik_block,
  stan_log_lik_block,
  stan_y_pred_block,
  y_type,
  did_reference
)
```

Arguments

name	Character scalar identifying the custom family. Must not coincide with any built-in family name.
link	Character scalar identifying the link function. One of "identity", "log", "logit".
did_holds	Logical scalar. The user must explicitly declare whether the family is identifiable in its parameter; a missing declaration raises an error.
did_condition	Character scalar describing any condition under which D-ID holds when did_holds = TRUE but identifiability is conditional. Use NA_character_ if D-ID holds unconditionally.

`stan_loglik_block`
: Character scalar with a Stan code snippet for the `model` block: declares `target += custom_lpdf | custom_lpmf` for one observation. The snippet must reference the linear predictor `eta[i]` and either `y_real[i]` (when `y_type = "real"`) or `y_int[i]` (when `y_type = "integer"`); the legacy placeholder `y[i]` is rejected.

`stan_log_lik_block`
: Character scalar with the `generated quantities` snippet that assigns to `log_lik[i]` for one observation, e.g. `log_lik[i] = normal_lpdf(log(y_real[i]) | eta[i], sigma);`. Used by `gdpar_loo` downstream.

`stan_y_pred_block`
: Character scalar with the `generated quantities` snippet that assigns to `y_pred[i]` for one observation, e.g. `y_pred[i] = exp(normal_rng(eta[i], sigma));`. Used by posterior-predictive utilities.

`y_type`
: Character scalar, one of `"real"` or `"integer"`, declaring whether the outcome is real-valued (Stan template references `y_real[i]`) or integer-valued (Stan template references `y_int[i]`).

`did_reference`
: Character scalar with a citation supporting the D-ID declaration.

Details

Building a custom family is an advanced use of the package. The user assumes the responsibility of ensuring (i) that the Stan likelihood block is mathematically correct and (ii) that the family is identifiable in its parameter. The package emits an informative message when the custom family is created, restating these responsibilities.

Value

An object of class `gdpar_family`.

Methodological notes

The package never attempts to test identifiability from data; it only registers the user's declaration. See Lemma 1B in Block 1, Section 6.4.

Dependencies

None beyond the **base** R installation.

See Also

`gdpar_family`

Examples

```
my_family <- gdpar_family_custom(
  name              = "my_log_normal",
  link              = "log",
  did_holds         = TRUE,
```

```
  did_condition      = NA_character_,
  stan_loglik_block  =
    "target += normal_lpdf(log(y_real[i]) | eta[i], sigma_y[1]);",
  stan_log_lik_block =
    "log_lik[i] = normal_lpdf(log(y_real[i]) | eta[i], sigma_y[1]);",
  stan_y_pred_block  =
    "y_pred[i] = exp(normal_rng(eta[i], sigma_y[1]));",
  y_type             = "real",
  did_reference      = "User declaration"
)
print(my_family)
```

`gdpar_family_custom_K` *Construct a K = 2 custom family from a canonical lpdf pattern*

Description

Build a custom distributional regression family by selecting a canonical bi-parametric likelihood pattern from the registry returned by `.gdpar_K_custom_patterns`. The constructor is the descriptor-based wiring agreed in sub-phase 8.3.4 of Block 8 (D-A3.B; option (b) of the scoping): the user does *not* supply free-form Stan code; they choose a `stan_lpdf_id` from the whitelist and the family routes through the same `amm_distrib_K.stan` branch that the built-in distributional regression families use.

Usage

```
gdpar_family_custom_K(
  name,
  stan_lpdf_id,
  did_holds = TRUE,
  did_condition = NULL,
  did_reference = NULL
)
```

Arguments

`name`	Character scalar identifying the custom family. Must not coincide with any built-in family name or another registered custom-K family in the calling session.
`stan_lpdf_id`	Character scalar selecting one of the canonical patterns in `.gdpar_K_custom_patterns`. Sub-phase 8.3.4 opens the registry with `"lognormal_loc_scale"`.
`did_holds`	Logical scalar declaring whether the D-ID condition of Lemma 1B holds for the family. Default `TRUE` (the registry's pattern-level declaration is used; see Details).
`did_condition`	Optional character scalar to override the pattern-level D-ID condition. Default `NULL` (use registry).
`did_reference`	Optional character scalar with the user's citation supporting the D-ID declaration. Default `NULL` (use registry).

Details

This is the K = 2 sibling of gdpar_family_custom (the K = 1 free-form custom path). The two coexist with distinct contracts: gdpar_family_custom() accepts user-authored Stan code blocks at K = 1 (user-validated identifiability and correctness); gdpar_family_custom_K() accepts only registered canonical patterns at K = 2 (descriptor-validated, no Stan injection surface).

The descriptor path enforces structural validation: name must be unique, stan_lpdf_id must be in the registry, and the slot configuration (links, supports, prior kinds) is fixed by the registry entry. Users who need to deviate from the canonical parametrization must request a new entry in the registry; this keeps the Stan-side surface auditable and bit-exact across versions of the package.

Sub-phase 8.3.4 (D-A3.B option (b)) decided this descriptor approach over (i) plug-in of free-form Stan code per family slot and (ii) a hybrid descriptor + escape-hatch. The decision applies [[feedback-max-robustness-priority]] (structural validation over surface flexibility) at the cost of forcing the user to contribute upstream when a new K = 2 family is needed.

Value

An object of class gdpar_family with two param_specs (slot 1 = location, slot 2 = scale), the stan_id of the pattern (e.g. 7 for lognormal_loc_scale), and is_custom = TRUE. The family routes through amm_distrib_K.stan via the family_id_k dispatcher.

Methodological notes

The K = 2 custom family inherits the canonical likelihood, links, and priors of the chosen registry pattern. The user only chooses the family name and (optionally) overrides the D-ID metadata. The AMM canonical form, the per-slot scope, and the dispatch in amm_distrib_K.stan are the same as for the built-in gdpar_family families wired in 8.3.4.

See Also

gdpar_family, gdpar_family_custom

Examples

```
my_lognorm <- gdpar_family_custom_K(
  name          = "my_lognormal_K2",
  stan_lpdf_id  = "lognormal_loc_scale",
  did_holds     = TRUE,
  did_reference = "User declaration"
)
print(my_lognorm)
```

gdpar_family_multi *Construct a multivariate family for AMM fitting with p > 1*

Description

Build a per-coordinate family object for use with gdpar when the AMM specification has dimension p > 1L. The resulting object declares one univariate family per coordinate of the individual parameter vector $\theta_i \in \mathbb{R}^p$; the likelihood factorizes across coordinates as

$$p(y_i \mid \theta_i) = \prod_{k=1}^{p} D_k(y_{ik} \mid \theta_i[k]),$$

with cross-dimensional coupling carried exclusively by the modulating component $W(\theta_{\text{ref}})$ of the AMM canonical form.

Usage

```
gdpar_family_multi(family, p, link = NULL)
```

Arguments

family	Either a character scalar with the name of a built-in family (one of "gaussian", "poisson", "neg_binomial_2", "bernoulli"), an object of class gdpar_family produced by gdpar_family, or a list of length p where each entry is itself a gdpar_family (heterogeneous coordinates). In this version the list form is restricted to the homogeneous case (all entries must share the same stan_id); heterogeneous families per coordinate are deferred to a later sub-phase.
p	Positive integer giving the dimension of θ_i. Must match the p of the amm_spec passed to gdpar.
link	Character scalar identifying the link function when family is supplied as a name. Ignored when family is already a gdpar_family object or a list. Defaults to the canonical link of the named family.

Details

The factorization is the canonization of architectural Option B of the Phase F decision (handoff 10 continuation, 2026-05-11): $y_i \in \mathbb{R}^p$ is multivariate with marginals independent conditional on θ_i; cross-dimensional dependence enters the model through the coupling of $\theta_i[k]$ via $W(\theta_{\text{ref}})$. Multiparametric families (a single univariate outcome parametrized by the whole vector $\theta_i \in \mathbb{R}^p$, e.g., gaussian with $\theta_i = (\mu_i, \log \sigma_i)$ in the distributional regression sense) are deferred to a dedicated post-validation block; see the project memory entry project_gdpar_multiparametric_extension_postvalidation.

Value

An object of class gdpar_family_multi with components families (a list of p gdpar_family objects), p, homogeneous, stan_id (the common Stan family identifier when homogeneous), has_dispersion (the common dispersion flag), name (the common family name) and did_status (the common identifiability status). A print method provides a human-readable summary.

Methodological notes

The identifiability condition (D-ID) of Lemma 1B in Block 1 applies coordinate-wise under this factorization: each univariate marginal D_k identifies $\theta_i[k]$ from y_{ik} independently. The cross-dimensional identifiability condition (C4-bis), which guards against aliasing between coordinates of θ_{ref} that share basis structure, is checked by `gdpar_check_identifiability` (Phase H pending).

Dependencies

Calls `gdpar_family` when `family` is supplied as a name.

See Also

`gdpar_family`, `amm_spec`

Examples

```
fam_mv <- gdpar_family_multi("gaussian", p = 2L)
print(fam_mv)

fam_mv2 <- gdpar_family_multi(gdpar_family("poisson"), p = 3L)
print(fam_mv2)
```

---

gdpar_formula_set *Construct a canonical formula set for multi-parameter AMM modeling*

Description

Build the canonical internal representation that `gdpar` consumes when more than one structural parameter of the family is modeled with an AMM design (multi-parametric distributional regression in the sense of decision D-F1 of Block 8 Session 1, materialized in sub-phase 8.3.3 of the package). One slot per individual parameter, named with the canonical parameter name as declared by the family's `param_specs`; the first slot carries the outcome variable on its left-hand side, while subsequent slots are one-sided formulas that describe the AMM design of an auxiliary parameter (e.g., the Gaussian `sigma` or the negative-binomial `phi`).

Usage

```
gdpar_formula_set(...)
```

Arguments

`...`	Named formulas. The first must be two-sided (`mu = y ~ a(x)`). Subsequent ones must be one-sided (`sigma = ~ a(x)`).

Details

This constructor is the canonical low-cost entry point; the brms-style sugar `gdpar_bf` produces an equivalent object from a sequence of two-sided formulas.

Value

An object of class `gdpar_formula_set` with components `outcome` (character scalar with the outcome variable name), `formulas` (named list of formula objects; the first is two-sided, subsequent ones are one-sided), `param_names` (character vector of slot names, identical to `names(formulas)`) and `env` (the environment of the first formula, used downstream for evaluation).

Validation contract

- All formulas must be named with the canonical parameter name (no positional formulas).
- The first slot must be a two-sided formula whose left-hand side is a single symbol naming the outcome variable (e.g., `mu = y ~ a(x)`). Function calls on the LHS such as `log(y)` are rejected to keep the contract explicit; pre-transform the outcome in the data frame instead.
- Subsequent slots must be one-sided formulas (e.g., `sigma = ~ a(x)`).
- Slot names must be unique.
- No formula may suppress the intercept via `-1` or `+0`: the AMM population anchor θ_{ref} is a structural term that cannot be removed. The constructor aborts with an informative `gdpar_input_error` when this pattern is detected.

Set membership against the family's eligible `param_specs` (slot names must be a subset of the eligible parameters of the family passed to `gdpar`) is enforced downstream by `gdpar` once both the formula set and the family are known.

Methodological notes

Block 8 Session 1 decision 1C established that a family is a list of parameter specifications. Block 8.3.3 closes the high-level API: the formula set is the single source of truth for the K-individual parameter set; any parameter named in this object is promoted to `scope = "per_observation"` in the family copy that `gdpar` uses internally. Parameters of the family that are not named in the formula set remain with their canonical `scope` (typically `population`). See memory entries `project_gdpar_block_8_3_extended_plan` and `project_gdpar_block_8_session_1_decisions` (2026-05-20).

Dependencies

Uses `terms` to detect intercept suppression.

See Also

`gdpar_bf`, `gdpar`, `gdpar_family`

Examples

```
fs <- gdpar_formula_set(
  mu    = y ~ a(x1) + b(z1),
  sigma = ~ a(x2)
)
print(fs)
fs[["mu"]]
names(fs)
```

gdpar_geometry_diagnostic *Forensic diagnostic of posterior geometry (opt-in)*

Description

Probe the geometry of a posterior with cheap Hamiltonian pilots and classify the sampling pathology, localise the culprit parameter(s), estimate the difficulty-vs-n behaviour, and recommend the geometry level that remedies it. Built for the Block RG capability: when the no-U-turn sampler stalls (the eBird count / Tweedie case), this tells you *why* and *what* to escalate to, rather than leaving an unexplained `rhat = Inf`.

Usage

```
gdpar_geometry_diagnostic(
  target,
  n_grid = NULL,
  difficulty = NULL,
  pilot_warmup = 150L,
  pilot_sampling = 150L,
  chains = 4L,
  adapt_delta = 0.8,
  max_treedepth = 10L,
  seed = 20260602L,
  thresholds = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`target`	The posterior to probe. One of three forms (the three-way adapter): (i) a `gdpar_geometry_target` from `gdpar_geometry_suite` (carries Stan code, a size knob and a ground-truth label); (ii) a list `list(stan_code, stan_data)` or a compiled **cmdstanr** model wrapped as `list(model = , data = ,data_n_fn = )`, where the optional `data_n_fn(n)` returns the data list at size `n`; (iii) a list `list(type = "gdpar", formula = ,amm = , data = , ...)` probing a real gdpar specification (data is subsampled to realise the size knob).
`n_grid`	Optional numeric vector of size-knob values at which to run pilots. Defaults to the target's own `n_grid` (suite targets) or to a single pilot (other forms without a size knob). At least two values are required to estimate the difficulty-vs-n curve.
`difficulty`	Optional pathology-intensity knob forwarded to a suite target's `make()`. Defaults to the target's `default_difficulty`.
`pilot_warmup, pilot_sampling`	Integer warmup / sampling iterations per pilot. Kept small on purpose (defaults 150 / 150).

chains	Integer number of chains per pilot. Defaults to 4 (multiple chains are needed for the multimodality signal).
adapt_delta	Numeric target acceptance for the pilots. Defaults to 0.8 (the diagnostic measures the geometry as the default sampler sees it).
max_treedepth	Integer maximum tree depth for the pilots. Defaults to 10.
seed	Integer base seed. Pilot i uses seed + i for reproducibility.
thresholds	Named list of classifier thresholds; see gdpar_geometry_thresholds. Defaults to that function's output.
verbose	Logical; print an opt-in cost message before sampling. Defaults to TRUE.
...	Additional arguments forwarded to the underlying sampler / fit.

Details

The diagnostic uses only size-invariant sampler signals – the divergence rate, the minimum energy Bayesian fraction of missing information ("E-BFMI"), the tree-depth saturation rate, the posterior condition number, and the adapted-step-to-scale ratio. It deliberately does *not* use R-hat or the effective sample size as decision signals: on the short pilots used here those are unreliable and were the false positives of sessions B9.20/B9.21.

Value

A list of class gdpar_geometry_diagnostic with components pathology (classified class), confidence, recommended_geometry (the remedy level), signals (data frame of per-n size-invariant signals), difficulty_curve (list with slope, grows_with_n), culprit (ranked data frame), cost (list with the cost extrapolation and tractability verdict), reproducibility (seed and pilot configuration), and, when the target carries a ground truth, ground_truth and correct. A print method summarises the verdict.

Pathology taxonomy and remedies

The classifier returns one of: isotropic (Euclidean diagonal; default is fine), anisotropic (dense Euclidean metric), funnel (Riemannian metric), heavy_tails (Finsler / relativistic kinetic energy), quasi_deterministic (sub-Riemannian), multimodal (tempering), boundary (boundary reparametrisation), or flat_direction (reparametrise / eliminate; the Option A case). The rules are documented in the package source and are calibrated against gdpar_geometry_suite.

Dependencies

Uses **cmdstanr** for the pilots and **posterior** to read draws and sampler diagnostics.

See Also

gdpar_geometry_suite, gdpar_geometry_thresholds.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  suite <- gdpar_geometry_suite()
  diag <- gdpar_geometry_diagnostic(suite$G2_funnel,
                                    pilot_warmup = 150, pilot_sampling = 150)
  print(diag)
}
```

`gdpar_geometry_suite` *Catalogue of synthetic posterior geometries of known difficulty*

Description

Build the Block RG calibration suite: a registry of synthetic target distributions, each engineered to exhibit one row of the posterior-geometry pathology taxonomy with a *known* ground truth. The suite is the falsifiable backbone against which `gdpar_geometry_diagnostic` is calibrated and its error rates are measured.

Usage

```
gdpar_geometry_suite(which = NULL)
```

Arguments

`which`	Optional character vector selecting a subset of target ids. Defaults to all eight targets.

Details

Each target is returned in dual representation so that the same geometry can be exercised by the cmdstan NUTS pilots of RG.1 and by the R-native geometric engine of RG.2: a Stan program string plus an R log-density closure carrying an analytic gradient on the unconstrained scale.

The eight targets cover the taxonomy of the Block RG charter:

`G0_isotropic` Standard normal. Easy negative control; remedy is the Euclidean diagonal metric (the current default).

`G1_anisotropic` Diagonal normal with a fixed, large condition number that does *not* grow with `n`. A straight canyon; remedy is a constant (dense Euclidean) metric.

`G2_funnel` Neal's funnel: a log-scale variable `v` governs the spread of the remaining coordinates, so curvature varies with position. Remedy is a position-dependent (Riemannian) metric.

`G3_heavy_tails` Independent Student-t with a small number of degrees of freedom. Directional heaviness; remedy is a non inner-product (Finsler / relativistic) kinetic energy.

`G4_quasi_deterministic` A near-degenerate canyon in which `d - 1` directions are pinned with variance `1 / n` while one direction stays free. The condition number *grows* with `n`: the posterior collapses to a lower-dimensional manifold. This is the eBird count case; remedy is a sub-Riemannian (distribution-of-directions) treatment.

`G5_multimodal` Equal-weight mixture of two separated normals. Remedy is tempering / a general metric space.

`G6_boundary` One parameter bounded on (0, 1) with mass pinned against the lower bound, plus free nuisance coordinates. Singular curvature at the edge; the analogue of the Tweedie shape parameter `p` hugging its bound.

`G7_flat_direction` A reparametrisation redundancy (`a + b` identified, the contrast `a - b` flat) with a wide prior, yielding a near-zero Hessian eigenvalue. The exact analogue of the declared-but-unused `sigma_a_k` scales of session B9.21; remedy is to reparametrise or eliminate (Option A).

Each registry entry is a list of class `gdpar_geometry_target` with the static fields `id`, `label`, `pathology`, `geometry_remedy`, `culprit` (parameter names or `NA`), `difficulty_scales_with_n` (the ground-truth answer the difficulty-vs-n curve must recover), `bounds` (named list of constrained-scale bounds, or `NULL`), `default_n`, `default_difficulty`, `n_grid` (a suggested size sweep), and a constructor `make(n,difficulty)` returning an *instance*: a list with `stan_code`, `stan_data`, `log_prob(theta)`, `grad_log_prob(theta)`, `dim`, and `param_names`.

The R closures operate on the unconstrained scale (matching the scale on which Hamiltonian samplers act). Absolute log-density values may differ from Stan's by an additive constant; the gradients agree exactly and are what the geometric machinery uses.

Value

A named list of `gdpar_geometry_target` objects.

See Also

`gdpar_geometry_diagnostic`.

Examples

```
suite <- gdpar_geometry_suite()
names(suite)
funnel <- suite$G2_funnel$make(n = 1, difficulty = 3)
funnel$dim
funnel$log_prob(c(v = 0, x = rep(0, funnel$dim - 1)))
```

`gdpar_geometry_thresholds` *Default classifier thresholds for the posterior-geometry diagnostic*

Description

The decision thresholds of gdpar_geometry_diagnostic's rule-based classifier. They are exposed as data (not hard-coded) so the calibration of RG.1 can tune them against gdpar_geometry_suite and so that a user can re-calibrate them for their own setting.

Usage

```
gdpar_geometry_thresholds()
```

Details

These defaults were calibrated in sub-phase RG.1.c (session B9.23) against the synthetic suite over a difficulty x pilot-budget x replica grid, with the thresholds proposed by a data-driven Youden cut, regularised to interpretable values on a calibration fold and validated out-of-sample on a held-out fold (held-out balanced accuracy rose from 0.60 to 0.89). The chief change versus the initial hand-set values reflects that the SHORT pilots used by the diagnostic attenuate the signals relative to their asymptotic values (the empirical condition number underestimates the true one, sample kurtosis is damped, boundary pile-up is subtle), so several cuts moved to catch the attenuated signal: condition_high 50 -> 12, heavy_kurtosis_high 3 -> 1.8, boundary_prox_high 0.10 -> 0.02, nslope_grows 0.30 -> 0.80, funnel_ebfmi_low 0.25 -> 0.35, heavy_cond_max 8 -> 25, flat_var_high 1000 -> 600, multimodal_high 2 -> 2.5. The calibration is against an idealised synthetic suite and is not a claim of optimality on real posteriors; the funnel and heavy-tail classes remain mutually confusable (per-class recall ~0.6-0.7), which is reported honestly in inst/benchmarks/results/block_rg_calibration.md. Re-calibrate for a specific application if needed.

Value

A named list of numeric thresholds.

See Also

gdpar_geometry_diagnostic.

Examples

```
str(gdpar_geometry_thresholds())
```

---

gdpar_geom_bridge *Bridge a fitted gdpar model to the geometry-adaptive orchestrator*

Description

Turns an already-fitted gdpar object into the inputs the geometry-adaptive controller gdpar_geom_orchestrate consumes, *without touching the fit path*. This is the durable, path-agnostic core of the Block RG integration (RG.6 part ii): it reads the compiled cmdstan model and the Stan data carried by the fit, exposes the standalone log_prob / grad_log_prob / hessian methods, derives the unconstrained dimension and a posterior-mean warm-start, and packages a target (a re-samplable cmdstan model for the size-invariant diagnostic) together with a geom_target (the engine sampling target on the unconstrained scale).

Usage

```
gdpar_geom_bridge(
  object,
  fisher = NULL,
  reference = NULL,
  hessian = TRUE,
  methods_seed = 1L,
  ...
)
```

Arguments

object	A fitted gdpar_fit (the result of gdpar).
fisher	Optional function of the unconstrained theta returning the expected Fisher information; required by the sub-Riemannian level (the Tweedie remedy). Use gdpar_geom_fisher_simulator when there is no closed form.
reference	Optional unconstrained reference position (warm-start for the position-dependent levels). Defaults to the posterior mean read from the fit.
hessian	Logical; whether to compile the standalone Hessian method (needed by the Riemannian SoftAbs level). Defaults to TRUE.
methods_seed	Integer seed forwarded to init_model_methods() (the standalone methods are deterministic; this only seeds any internal RNG). Defaults to 1L.
...	Reserved for future extension; currently unused.

Details

The orchestrator needs two things: a target it can *re-sample* for the diagnostic pilots, and an engine target exposing the unconstrained log-density and its gradient (and Hessian, for the Riemannian / SoftAbs level). A fitted gdpar object carries the posterior draws but its CmdStanMCMC object cannot be re-sampled. The bridge therefore recompiles a fresh CmdStanModel from the fit's own Stan source ($code()) with compile_model_methods = TRUE (cmdstanr's content-hash cache makes this a cache hit when the methods variant already exists), which serves both consumers, and reads the unconstrained dimension and posterior mean from the fit's draws.

The bridge is **path-agnostic**: it works for any gdpar_fit that carries $fit (a CmdStanMCMC) and $stan_data – the K-individual, multi-coordinate and single-coordinate paths alike. It is the tool RG.7 points at the real Tweedie count of benchmark 9.2.O.

Nothing here modifies gdpar; the returned object is plain data plus closures, so the default fit branch stays bit-identical and the goldens are untouched.

Value

An object of class gdpar_geom_bridge: a list with target (the cmdstan-model diagnostic target), geom_target (the engine target), fisher, reference, dim (the unconstrained dimension), model (the methods-enabled CmdStanModel) and stan_data. Feed these to gdpar_geom_orchestrate.

See Also

gdpar_geom_orchestrate, gdpar_geom_fit, gdpar_geom_target, gdpar_geom_fisher_simulator.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(1)
  n <- 80
  x <- rnorm(n); z <- rnorm(n)
  y <- rnorm(n, 0.5 + 0.8 * (x - mean(x)), exp(-0.2 + 0.4 * (z - mean(z))))
  d <- data.frame(y = y, x = x, z = z)
  fit <- gdpar(gdpar_bf(y ~ a(x), sigma ~ a(z)), data = d,
               family = gdpar_family("gaussian"), chains = 1,
               iter_warmup = 200, iter_sampling = 200, refresh = 0,
               seed = 1, skip_id_check = TRUE, verbose = FALSE)
  bridge <- gdpar_geom_bridge(fit)
  bridge$dim
  # res <- gdpar_geom_orchestrate(bridge$target, bridge$geom_target,
  #                               reference = bridge$reference)
}
```

---

gdpar_geom_fisher_simulator

Simulation-based estimator of the expected Fisher information

---

Description

Build the fisher(theta) function that feeds the learned metric gdpar_geom_metric_gp_fisher where the expected Fisher information has no closed form – the general realisation completing the Riemannian level of the Block RG hierarchy. The expected Fisher is $I(\theta) = \mathbb{E}_{y \sim p(\cdot|\theta)}\left[s(\theta, y)\, s(\theta, y)^\top\right]$ with s the score (the gradient of the log-likelihood). The estimator is the average outer product of the scores over n_sim data sets simulated from the model at θ: it is **positive semi-definite by construction** (a sum of rank-one outer products) and unbiased for the expected Fisher.

Usage

```
gdpar_geom_fisher_simulator(target, n_sim = 64L, seed = 1L, floor = 1e-08)
```

Arguments

target	A generative gdpar_geom_target carrying both simulate(theta) (one synthetic data set drawn from the model at theta) and score(theta, y) (the gradient of the log-likelihood of y at theta, on the unconstrained scale).
n_sim	Integer number of simulated data sets averaged per evaluation.
seed	Integer base seed combined with the position key.
floor	Minimum eigenvalue imposed on the returned matrix so it is strictly positive-definite (needed by the log-Cholesky map of the surrogate).

Details

The estimate is a *deterministic* function of θ: each evaluation reseeds the RNG from a key derived from θ and the base seed (and the RNG state is saved and restored around the call), so the surrogate trained on it is reproducible and independent of call order. The simulation noise is then absorbed by the Gaussian-process surrogate: its SoftAbs mean carries the bulk of the curvature deterministically and the process learns only the smooth residual, so the SoftAbs acts as a structural control variate and few replicates per site suffice (the deferred half of decision D91, now closed). Antithetic simulation is deliberately *not* used: for location families the score is odd in the centred data, so the antithetic partner has score $-s$ and $(-s)(-s)^\top = ss^\top$ leaves the outer product unchanged – no variance reduction.

For a well-conditioned estimate take n_sim comfortably larger than the dimension (the average of fewer than dim rank-one terms is singular and is only made strictly positive-definite by the eigenvalue floor).

Value

A function fisher(theta) returning the dim x dim estimated expected Fisher matrix (symmetric positive-definite), with the number of simulations recorded in its "n_sim" attribute. Pass it as the fisher argument of gdpar_geom_metric_gp_fisher.

See Also

gdpar_geom_metric_gp_fisher, gdpar_geom_target, gdpar_geom_rmhmc_adaptive.

Examples

```
# A bivariate normal location model y ~ N(theta, Sigma0): the expected Fisher
# is the constant precision Sigma0^{-1}, which the estimator recovers.
Sigma0 <- matrix(c(1, 0.3, 0.3, 2), 2, 2)
P0 <- solve(Sigma0)
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta, dim = 2,
  simulate = function(theta)
    as.numeric(theta + t(chol(Sigma0)) %*% stats::rnorm(2)),
  score = function(theta, y) as.numeric(P0 %*% (y - theta)))
fisher <- gdpar_geom_fisher_simulator(tgt, n_sim = 4000, seed = 1)
round(fisher(c(0, 0)), 2)   # close to solve(Sigma0)
```

gdpar_geom_fit *One-call geometry-adaptive fit (K-individual path)*

Description

The ergonomic single-call entry of the Block RG integration: a *sister* of gdpar (not an internal branch of it) that builds and compiles a K-individual model, then runs the opt-in geometry-adaptive controller gdpar_geom_orchestrate on it instead of the default NUTS fit. It diagnoses the posterior geometry, selects a level of the sampler hierarchy (Euclidean diagonal / dense, Riemannian, relativistic, sub-Riemannian), samples, re-diagnoses and either resolves or emits a certified limit.

Usage

```
gdpar_geom_fit(
  formula,
  family = gdpar_family("gaussian"),
  amm = amm_spec(),
  W = NULL,
  data,
  prior = NULL,
  anchor = "prior_mean",
  skip_id_check = FALSE,
  parametrization = c("auto", "ncp", "cp"),
  parametrization_a = NULL,
  parametrization_W = NULL,
  id_check_rigor = c("full", "fast"),
  group = NULL,
  fisher = NULL,
  budget = NULL,
  criteria = NULL,
  entry_level = NULL,
  level_map = NULL,
  reference = NULL,
  speed = 10,
  rest_mass = 1,
  laplace_fallback = FALSE,
  laplace_draws = 0L,
  n_grid = NULL,
  seed = 20260603L,
  verbose = TRUE,
  ...
)
```

Arguments

formula	A gdpar_bf() formula set, a classic formula with AMM wrapper calls on the RHS, or (with amm) a two-sided y ~ . . . .

family	A gdpar_family (broadcast across the K slots) or a named list of gdpar_family (heterogeneous families per slot).
amm	A named list of amm_spec (one per slot) for the named-list path; otherwise left at its default.
W	Optional modulating basis for the K-individual paths.
data	A data frame.
prior	A gdpar_prior; defaults to gdpar_prior().
anchor	The slot anchor(s); see gdpar.
skip_id_check	Logical; skip the identifiability check.
parametrization, parametrization_a, parametrization_W	The CP/NCP toggles forwarded to the model build.
id_check_rigor	One of "full" / "fast".
group	Optional grouping variable for grouped anchors.
fisher	Optional expected-Fisher function (sub-Riemannian level). Use gdpar_geom_fisher_simulator when there is no closed form.
budget, criteria	The orchestrator budget / success gate; see gdpar_geom_orchestrate_budget and gdpar_geom_orchestrate_criteria.
entry_level, level_map	Optional overrides of the entry level and the pathology-to-level map.
reference	Optional unconstrained warm-start position.
speed, rest_mass	The relativistic level's speed and rest mass.
laplace_fallback	Logical; forwarded to gdpar_geom_orchestrate. When TRUE and the run ends in a certified limit, a gdpar_geom_laplace approximation is attached ($laplace) and the status becomes "certified_limit_laplace". Defaults to FALSE (bit-identical output).
laplace_draws	Number of iid Laplace draws carried on the fallback when laplace_fallback = TRUE (default 0L).
n_grid	Optional diagnostic size grid (forwarded).
seed	Integer base seed for the (deterministic) adaptive trajectory.
verbose	Logical; opt-in progress trace.
...	Forwarded to .gdpar_K_build() / the diagnostic.

Details

gdpar_geom_fit() shares the model-building seam .gdpar_K_build() with gdpar's internal K path: the model and data are assembled and compiled exactly once, through a single source, with no duplication and no throwaway computation. The model is compiled with the standalone gradient and Hessian methods exposed (compile_model_methods = TRUE) so the geometry engine can integrate on the unconstrained scale; the default gdpar branch, which compiles without those methods, is byte for byte unchanged and its goldens stay bit-identical.

This entry is scoped to the K-individual (distributional) regime where the Tweedie count lives. For an already-fitted model, or for the single- and multi-coordinate paths, use gdpar_geom_bridge on the fit.

Correctness versus efficiency (the honesty convention of ORPHEUS-PIMC section 16.3 the package follows): every sampler level is Metropolis-exact, so *which* geometry is selected only governs efficiency, never the validity of the returned draws.

Value

An object of class gdpar_geom_fit: a list carrying the orchestration (a gdpar_geom_orchestrate result), the bridge, the status, and, when resolved, the winning level, metric and draws; when the budget is exhausted, the certificate (and, under laplace_fallback = TRUE, the laplace approximation with status "certified_limit_laplace"). It also carries stan_data, family, K, slot_names and the call.

See Also

gdpar_geom_bridge, gdpar_geom_orchestrate, gdpar.

Examples

```
b <- gdpar_geom_orchestrate_budget()
b$max_rounds

if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(1)
  n <- 80
  x <- rnorm(n); z <- rnorm(n)
  y <- rnorm(n, 0.5 + 0.8 * (x - mean(x)), exp(-0.2 + 0.4 * (z - mean(z))))
  d <- data.frame(y = y, x = x, z = z)
  b$tune_epsilon <- FALSE
  b$probe_iter <- 60L; b$full_iter <- 80L; b$full_warmup <- 80L
  res <- gdpar_geom_fit(gdpar_bf(y ~ a(x), sigma ~ a(z)), data = d,
                        family = gdpar_family("gaussian"),
                        skip_id_check = TRUE, budget = b, n_grid = 1,
                        verbose = FALSE)
  res$status
}
```

---

gdpar_geom_hmc *Static Hamiltonian Monte Carlo with a pluggable geometry (Block RG engine)*

Description

Sample a target with the R-native geometric engine of decision A: a fixed step-size, fixed trajectory-length Hamiltonian Monte Carlo with a Metropolis correction, over a pluggable metric. With the default Euclidean metric this is textbook HMC; the higher levels of the Block RG hierarchy (Riemannian, Finsler / relativistic, sub-Riemannian) reuse the same loop with a richer metric and integrator. This is the validated Euclidean scaffolding for those levels, not a replacement for the package's cmdstan fit path.

Usage

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

Arguments

`target`	A `gdpar_geom_target`, or a list/closure accepted by it.
`metric`	A `gdpar_geom_metric_euclidean` (or compatible) metric. Defaults to the identity Euclidean metric of the target dimension.
`epsilon`	Numeric leapfrog step size.
`L`	Integer number of leapfrog steps per proposal.
`n_iter`	Integer number of retained iterations.
`n_warmup`	Integer number of warmup iterations discarded from the returned draws (no adaptation is performed; warmup only burns in).
`init`	Optional numeric vector of length `target$dim` giving the initial position. Defaults to zeros.
`seed`	Optional integer seed. If supplied, the RNG state is set and restored around the run.
`fp_tol`, `fp_max`	Convergence tolerance and maximum iteration count for the fixed-point solves of the implicit generalised leapfrog (used only when the metric is position-dependent). A proposal whose solves do not converge is counted as divergent and rejected.

Value

A list of class `gdpar_geom_hmc` with `draws` (an `n_iter` x `dim` matrix), `accept_rate`, `n_divergent` (proposals with a non-finite Hamiltonian, a large energy error, or a non-converged implicit solve),

`energy` (the per-iteration Hamiltonian energy trace), `ebfmi` (the energy Bayesian fraction of missing information; higher is better, low values flag a metric ill-matched to the geometry), `epsilon`, `L` and `metric_type`.

See Also

`gdpar_geom_target`, `gdpar_geom_metric_euclidean`, `gdpar_geometry_diagnostic`.

Examples

```
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta, dim = 2)
fit <- gdpar_geom_hmc(tgt, epsilon = 0.3, L = 12, n_iter = 200,
                      n_warmup = 100, seed = 1)
colMeans(fit$draws)
```

---

gdpar_geom_laplace *Laplace approximation of a posterior at its mode*

Description

Compute the Laplace (mode + curvature) Gaussian approximation $N(\hat{\theta}, M^{-1})$ of a posterior exposed as a `gdpar_geom_target`, on the unconstrained scale: climb to the mode $\hat{\theta}$, form the precision $M = -\nabla^2 \log p(\hat{\theta})$ (the observed information), and report its covariance, optional draws, and a **fidelity diagnostic** of the Gaussian against the true posterior. This is the first-class, honest endpoint for a posterior the geometry-adaptive orchestrator certifies as a non-sampleable, genuinely non-Gaussian canyon (Block RG, RG.7): exactly the regime of **mgcv**/REML and **INLA**/Laplace competitors, accompanied by a measurement of how good the Gaussian is.

Usage

```
gdpar_geom_laplace(
  geom_target,
  reference = NULL,
  draws = 0L,
  climb = TRUE,
  seed = NULL,
  fit_quality_draws = 256L,
  eigen_floor_rel = 1e-10,
  climb_steps = 300L,
  cond_warn = 1e+12
)
```

Arguments

geom_target	A gdpar_geom_target (or an object coercible to one) exposing log_prob, grad_log_prob, dim and, ideally, hessian.
reference	Optional unconstrained warm-start position for the mode climb; defaults to the origin. When called by the orchestrator this is the best on-ridge position found during sampling.
draws	Number of iid Laplace draws $\hat{\theta} + M^{-1/2}z$ to return (default 0L: the mode plus precision Gaussian *is* the approximation; the caller asks for draws explicitly when it needs them for a downstream predictive).
climb	Logical; whether to climb to the mode from reference (default TRUE). FALSE treats reference as the mode and only reads the curvature there.
seed	Integer seed for the (local, stream-preserving) draw RNG; defaults to a fixed value so the result is reproducible.
fit_quality_draws	Number of internal iid draws used to assess fidelity when draws is small (default 256L); the fidelity label is always computed, independently of how many user-facing draws are requested.
eigen_floor_rel	Relative eigenvalue floor applied to M for positive-definiteness (default 1e-10 of the largest eigenvalue).
climb_steps	Maximum modified-Newton steps in the mode climb (default 300L).
cond_warn	Condition-number threshold above which the un-floored curvature triggers an ill-conditioning warning (default 1e12).

Details

The mode is reached by an L-BFGS-B warm start on $-\log p$ followed, when the target exposes an exact Hessian, by a modified-Newton polish; both stages read the *same* target and gradient the sampler uses. The precision M is the symmetrised $-$Hessian (the exact cmdstan Hessian when available, central finite differences otherwise), eigen-floored to be positive-definite so the draw machinery stays numerically alive; the **un-floored** condition number is reported separately and a warning is raised when the floor could be masking ill-conditioning. A non-positive-definite raw curvature (a saddle, not a maximum) is flagged loudly.

Fidelity, never blind trust. The Gaussian q is scored against the true posterior p over the same draws: the self-normalised importance-sampling effective sample size with log-weights $\log p - \log q$, its PSIS Pareto-k tail index (when **loo** is installed), and the mean / max log-density drop $\log p(\hat{\theta}) - \log p(\theta)$ against its Gaussian expectation $d/2$. These are distilled into a single scalar label, "good" / "poor" / "very_poor", so the approximation is never mistaken for exact MCMC. On a curved canyon (the real 9.2.O Tweedie count) the label is "very_poor" – the scientific finding, not a defect.

Value

An object of class gdpar_geom_laplace: a list with mode, the precision M, the covariance cov (M^{-1}), the symmetric square root Lhalf ($M^{-1/2}$), logdet (of M), the eigenvalues eig, the

floored and un-floored condition numbers `cond` / `cond_unfloored`, the count `n_floored` of eigen-floored directions and the `floor_value` used, `all_pos` (was the raw curvature positive-definite), `mode_offset_sd` (the Newton-decrement bound on the mode offset in posterior SDs), `grad_norm`, `logp`, `converged`, `method` (the Hessian route), `dim`, `draws` (a `draws` $\times d$ matrix, possibly zero rows, carrying `attr(., "approximation") = "laplace"` so it is never mistaken for exact MCMC draws downstream), `fit_quality` (the fidelity diagnostics) and `fit_quality_label`.

See Also

`gdpar_geom_orchestrate` (the `laplace_fallback` opt-in), `gdpar_geom_fit`, `gdpar_geom_target`.

Examples

```
# A correlated Gaussian: the Laplace approximation is exact, so the fidelity
# label is "good". The target carries an analytic (constant) Hessian.
A <- matrix(c(2, 0.8, 0.8, 1), 2, 2)
mu <- c(1, -0.5)
tgt <- gdpar_geom_target(
  log_prob = function(th) -0.5 * as.numeric(t(th - mu) %*% A %*% (th - mu)),
  grad_log_prob = function(th) -as.numeric(A %*% (th - mu)),
  hessian = function(th) -A, dim = 2L)
lap <- gdpar_geom_laplace(tgt, draws = 200L, seed = 1L)
lap$fit_quality_label
max(abs(lap$mode - mu)) < 1e-6
```

---

gdpar_geom_metric_euclidean

Euclidean (constant) metric for the geometric sampling engine

---

Description

Build the level-0/1 metric of the Block RG geometry hierarchy: a position-independent mass matrix. With the identity it is the default diagonal Euclidean metric; with a supplied symmetric positive-definite matrix (or a positive vector of variances) it is the dense Euclidean metric (a constant linear preconditioner), the remedy for a straight anisotropic canyon. The Riemannian level of RG.3 replaces this with a position-dependent metric implementing the same interface.

Usage

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

Arguments

`dim`	Integer dimension. Required when `M` is `NULL`.
`M`	Optional mass matrix: a `dim` x `dim` symmetric positive-definite matrix, or a length-`dim` positive vector interpreted as its diagonal. Defaults to the identity.

Value

A list of class `gdpar_geom_metric` with `position_dependent = FALSE` and functions `mass(theta)`, `inv_mass(theta)`, `chol_mass(theta)` (lower Cholesky factor, for drawing momenta) and `logdet(theta)`, each ignoring `theta`.

See Also

`gdpar_geom_hmc`.

Examples

```
m <- gdpar_geom_metric_euclidean(dim = 3)
m$mass(c(0, 0, 0))
```

---

gdpar_geom_metric_gp_fisher

Learned Gaussian-process Riemannian metric (expected Fisher surrogate)

Description

Build the general, learned realisation of the level-3 Riemannian metric of the Block RG hierarchy: a position-dependent mass matrix $M(\theta) = L(\theta)L(\theta)^\top$ whose log-Cholesky factor is a Gaussian-process surrogate of the expected Fisher information (the natural Rao–Amari metric), for use where the Fisher has no closed form. Where the Fisher *is* closed (the funnel, a Gaussian, generalised-linear-model slots) `gdpar_geom_metric_riemannian` is exact and preferable; this surrogate covers the general case and is validated against those closed forms.

Usage

```
gdpar_geom_metric_gp_fisher(
  target,
  fisher,
  sites,
  weights = NULL,
  lengthscale = NULL,
  nugget = 1e-06,
  alpha = 1e+06,
  floor = 1e-08,
  fd_step = 1e-04
)
```

Arguments

`target`	A `gdpar_geom_target` (or an object accepted by it), used for the SoftAbs mean (its Hessian / gradient) and for the dimension.
`fisher`	A function `fisher(theta)` returning the expected Fisher information at `theta` as a `dim` x `dim` symmetric positive-definite matrix. The training targets at the reservoir sites are evaluated through it; a simulation-based estimator (the score outer product over simulated data sets) plugs into this same slot.
`sites`	A numeric `m` x `dim` matrix of reservoir positions (one row per site) at which the Fisher is evaluated to train the surrogate. See `gdpar_geom_reservoir` for collecting sites from a warmup run.
`weights`	Optional positive vector of length `m` of importance weights (e.g. $1/Q$) reweighting a reservoir biased towards rare positions back to the typical set; entered as per-site noise scaling. Defaults to equal weights.
`lengthscale`	Optional positive radial-basis-function length-scale on the standardised inputs. Defaults to the median pairwise-distance heuristic.
`nugget`	Non-negative kernel nugget (observation-noise variance). Small values interpolate the Fisher at the sites; larger values smooth it.
`alpha`, `floor`, `fd_step`	SoftAbs softening, eigenvalue floor and finite-difference step governing the SoftAbs mean function (passed to the Capa 1 machinery).

Details

The surrogate's *mean function* is the SoftAbs curvature of the observed Hessian (the cold-start metric, always positive-definite); the Gaussian process learns only the smooth residual to the expected Fisher at the reservoir sites, in the log-Cholesky parametrisation. Three properties follow by construction:

- **Positive-definite always**: $M = LL^\top$ with $L_{ii} = \exp(\cdot) > 0$.
- **Graceful degradation**: far from the reservoir the kernel decays to zero and the posterior mean returns to the SoftAbs mean, so the metric degrades *continuously* to the always-available SoftAbs – there is no hard metric switch that would break the reversibility of the implicit generalised leapfrog of `gdpar_geom_hmc`.
- **Exactness independent of surrogate quality**: the metric is a preconditioner, not part of the target, so the Metropolis correction with the exact density keeps the sampler exact for any surrogate; delayed acceptance is unnecessary.

The spatial derivative `dmass(theta)` is closed form: the analytic kernel derivative for the learned residual plus the Daleckii–Krein derivative of the SoftAbs mean (reused from Capa 1), pushed through the log-Cholesky map. The predictive standard deviation is exposed as `novelty(theta)`, the epistemic-uncertainty extrapolation detector.

Value

A list of class `gdpar_geom_metric` with `position_dependent = TRUE`, `metric_kind = "gp_fisher"`, the functions `mass`, `inv_mass`, `chol_mass`, `logdet` and `dmass` of the metric interface, plus `novelty(theta)` (the predictive standard deviation, higher = more out-of-distribution) and the fields `n_sites` and `lengthscale`.

See Also

gdpar_geom_metric_riemannian, gdpar_geom_reservoir, gdpar_geom_hmc.

Examples

```
# A two-dimensional target whose expected Fisher is the identity; the learned
# metric recovers it from a small reservoir.
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta, dim = 2)
sites <- matrix(stats::rnorm(40), ncol = 2)
m <- gdpar_geom_metric_gp_fisher(tgt, fisher = function(theta) diag(2),
                                 sites = sites)
round(m$mass(c(0, 0)), 3)
```

---

gdpar_geom_metric_relativistic

Finsler / relativistic metric for the geometric sampling engine

---

Description

Build the level-4 geometry of the Block RG hierarchy: a bounded, non-Gaussian (relativistic) kinetic energy coupled to the position-dependent Riemannian metric of gdpar_geom_metric_riemannian, the remedy for heavy tails and directional anisotropy (the G3_heavy_tails target). A Gaussian kinetic energy lets the velocity grow without bound, so a large momentum in a heavy tail overshoots and the integrator must be tuned to the stiffest region; the relativistic kinetic energy caps the velocity at a finite speed, taming the tails and the ill-conditioning while staying exact.

Usage

```
gdpar_geom_metric_relativistic(
  target,
  curvature = c("fisher", "softabs"),
  fisher = NULL,
  dfisher = NULL,
  speed = 10,
  rest_mass = 1,
  alpha = 1e+06,
  floor = 1e-08,
  fd_step = 1e-04
)
```

Arguments

target	A gdpar_geom_target (or an object accepted by it). Supplies the dimension and, for curvature = "softabs", the Hessian / gradient of the underlying Riemannian metric.

`curvature` Curvature source of the underlying Riemannian mass: `"fisher"` (expected Fisher, supplied through `fisher`) or `"softabs"` (SoftAbs of the observed Hessian). See `gdpar_geom_metric_riemannian`.

`fisher, dfisher` For `curvature = "fisher"`: the expected Fisher matrix function `fisher(theta)` and optionally its derivative `dfisher(theta)` (a length-`dim` list of partials); finite differenced from `fisher` when `dfisher` is `NULL`.

`speed` Positive speed of light c bounding the velocity. Larger values approach the Gaussian Riemannian kinetic (less tail-taming); smaller values cap the velocity sooner (more robust). Tune to the momentum scale.

`rest_mass` Positive rest mass m; with `speed` it sets the bulk curvature of the kinetic energy (the non-relativistic limit is `speed` or `rest_mass` to infinity).

`alpha, floor, fd_step` SoftAbs softening, eigenvalue floor and finite-difference step of the underlying Riemannian metric.

Details

The kinetic energy is the relativistic energy of a particle of rest mass m on the statistical manifold with local metric $M(\theta)$ (the expected Fisher or the SoftAbs of the observed Hessian, supplied through the RG.3 machinery):

$$K(\theta, p) = c\sqrt{p^\top M(\theta)^{-1} p + m^2 c^2} + \tfrac{1}{2}\log\det M(\theta).$$

The velocity is $\nabla_p K = c\, M^{-1}p / \sqrt{p^\top M^{-1} p + m^2 c^2}$, whose M-norm is strictly below the speed c for every momentum – the bounded-velocity property that tames heavy tails. Three properties hold by construction:

- **Exactness independent of c and m**: the $\frac{1}{2}\log\det M$ term is the same normaliser as the Gaussian Riemannian kinetic, so the θ-marginal of the joint e^{-H} is exactly $e^{\log p(\theta)}$; the kinetic energy is a preconditioner, not part of the target, and the Metropolis correction with the exact density keeps the sampler exact for any c and m (they govern only efficiency).
- **Non-relativistic limit**: as $c \to \infty$ the kinetic energy reduces to the Gaussian Riemannian kinetic of `gdpar_geom_metric_riemannian`, so this level strictly generalises the Riemannian one. Larger `speed` approaches that limit (less tail-taming, faster bulk mixing); smaller `speed` caps the velocity sooner (more robust tails, slower bulk).
- **Finsler structure** (the user's document section 12.3): a relativistic kinetic energy is the Legendre dual of a Finsler norm on velocities, a norm not induced by an inner product, so the cost of motion depends on direction as well as magnitude (the speed c is the Finsler unit ball). The asymmetric Randers extension $F = \sqrt{g(v,v)} + \beta(v)$ (an irreversible drift) is deliberately not included: it makes the kinetic energy odd in p and would break the reversibility the Metropolis correction relies on; it models irreversible dynamics rather than exact sampling of a fixed target.

Because K depends on both θ (through M) and p, the Hamiltonian is non-separable. Sampling therefore uses a **dedicated** generalised implicit leapfrog (Girolami & Calderhead 2011) carried in the metric's `integrator` slot – three reversible, volume-preserving sub-steps with the relativistic velocity – which `gdpar_geom_hmc` runs in place of the default leapfrog, leaving the default branch bit-identical. The momentum is refreshed from the exact relativistic momentum law (an inverse-CDF radial sampler under $p = Lq$), so every momentum draw is from $e^{-K(\theta,\cdot)}$.

Value

A list of class `gdpar_geom_metric` with `position_dependent = TRUE`, `metric_kind = "relativistic"`, the metric interface (`mass`, `inv_mass`, `chol_mass`, `logdet`, `dmass`) of the underlying Riemannian mass, a `kinetic` object (the relativistic `value`, `grad_p`, `grad_theta` and `draw_momentum`) and an `integrator` closure, both consumed by `gdpar_geom_hmc`, plus the fields `speed`, `rest_mass` and `curvature`.

References

Lu, X., Perrone, V., Hasenclever, L., Teh, Y. W. and Vollmer, S. (2017) Relativistic Monte Carlo. *AISTATS* 54, 1236–1245.

Livingstone, S., Faulkner, M. F. and Roberts, G. O. (2019) Kinetic energy choice in Hamiltonian/hybrid Monte Carlo. *Biometrika* 106, 303–319.

Girolami, M. and Calderhead, B. (2011) Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *JRSS-B* 73, 123–214.

Randers, G. (1941) On an asymmetrical metric in the four-space of general relativity. *Physical Review* 59, 195–199.

See Also

`gdpar_geom_metric_riemannian`, `gdpar_geom_metric_subriemannian`, `gdpar_geom_hmc`.

Examples

```
# A two-dimensional Student-t (heavy tails); the expected Fisher of the
# independent t is diagonal. The bounded kinetic energy samples the tails
# without the overshoot of a Gaussian kinetic.
nu <- 2
tgt <- gdpar_geom_target(
  log_prob = function(theta) -((nu + 1) / 2) * sum(log1p(theta^2 / nu)),
  grad_log_prob = function(theta) -(nu + 1) * theta / (nu + theta^2), dim = 2)
fisher <- function(theta) diag((nu + 1) / (nu + 3), 2)   # expected Fisher of t.
metric <- gdpar_geom_metric_relativistic(tgt, fisher = fisher, speed = 5)
fit <- gdpar_geom_hmc(tgt, metric = metric, epsilon = 0.4, L = 10,
                      n_iter = 200, n_warmup = 100, seed = 1)
fit$ebfmi
```

`gdpar_geom_metric_riemannian` *Riemannian (position-dependent) metric for the geometric sampling engine*

Description

Build the level-3 metric of the Block RG geometry hierarchy: a position-dependent mass matrix $M(\theta)$ that adapts the sampler's local notion of distance to the curvature of the log-posterior, the remedy for a funnel (variable curvature). Two curvature sources are offered, matching the two ways to obtain the local bending of the density.

Usage

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

Arguments

`target`	A `gdpar_geom_target` (or an object accepted by it). Used for the Hessian / gradient of the `"softabs"` source and for the dimension.
`curvature`	Curvature source: `"fisher"` (expected Fisher, supplied) or `"softabs"` (SoftAbs of the observed Hessian).
`fisher`, `dfisher`	For `curvature = "fisher"`: a function `fisher(theta)` returning the `dim` x `dim` expected Fisher matrix, and optionally `dfisher(theta)` returning a length-`dim` list of its partial derivatives. When `dfisher` is `NULL` the derivatives are finite-differenced from `fisher`.
`alpha`	SoftAbs softening parameter (`curvature = "softabs"`). Larger values track the true curvature more faithfully ($\to |\lambda|$) but make the metric vary faster; the default `1e6` is the near-absolute limit.
`floor`	Minimum eigenvalue imposed on $M(\theta)$ to keep it strictly positive-definite at genuinely flat directions (non-identification).
`fd_step`	Finite-difference step for the Hessian (softabs) or for the metric derivative when an analytic one is unavailable.

Details

`curvature = "fisher"` The expected Fisher information (the natural metric of the statistical manifold; Rao–Amari). It is positive-definite by construction wherever the model is identifiable, so no eigenvalue surgery is needed. It is model-specific and must be supplied as a function `fisher(theta)` returning a symmetric positive-definite matrix (optionally with its derivative `dfisher`). This is the primary, maximally robust choice; its fully general, learned amortisation across families is a separate Block RG sub-phase.

`curvature = "softabs"` The SoftAbs regularisation of the observed Hessian (Betancourt 2013): the eigenvalues λ of the Hessian of $-\log\pi$ are mapped to $\lambda\coth(\alpha\lambda)$, turning any bending into a sensible positive mass and flooring nearly flat directions at $1/\alpha$. It needs only the

Hessian (taken from the target's `$hessian` when available, otherwise finite-differenced from the gradient), so it is fully general and serves as the cold-start and extrapolation fallback for the Fisher metric.

The metric exposes, beyond the Euclidean interface, the spatial derivatives `dmass(theta)` (a length-`dim` list of $\partial M/\partial\theta_k$) that the generalised implicit leapfrog of `gdpar_geom_hmc` requires. The Riemannian sampler stays exact regardless of how crude the metric is: the metric is a preconditioner, not part of the target, so the Metropolis correction with the exact log-density is the corrector.

Value

A list of class `gdpar_geom_metric` with `position_dependent = TRUE` and functions `mass(theta)`, `inv_mass(theta)`, `chol_mass(theta)`, `logdet(theta)` and `dmass(theta)`.

See Also

`gdpar_geom_hmc`, `gdpar_geom_metric_euclidean`.

Examples

```
# SoftAbs Riemannian metric for a two-dimensional standard normal.
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta, dim = 2)
m <- gdpar_geom_metric_riemannian(tgt, curvature = "softabs")
m$mass(c(0, 0))
```

---

gdpar_geom_metric_subriemannian

Sub-Riemannian metric and integrator for the geometric sampling engine

---

Description

Build the level-5 geometry of the Block RG hierarchy: the remedy for a quasi-deterministic posterior (the eBird count / tweedie case), where the typical set contracts onto a lower-dimensional manifold and the expected Fisher information grows without bound along the stiff "wall" directions while the soft "floor" directions still carry genuine variation. A sub-Riemannian structure equips only a distribution $D_\theta \subsetneq T_\theta M$ of accessible directions with an inner product (Montgomery 2002): the sampler glides along the floor instead of fighting the walls with a vanishing step size.

Usage

```
gdpar_geom_metric_subriemannian(
  target,
  fisher,
  reference = NULL,
  tau = NULL,
```

```
  softness = 1,
  floor = 1e-08
)
```

Arguments

target	A gdpar_geom_target (or an object accepted by it).
fisher	A function fisher(theta) returning the dim x dim expected Fisher information matrix (symmetric positive semi-definite); for models without a closed form pass gdpar_geom_fisher_simulator. Evaluated once at reference.
reference	Optional numeric vector at which the Fisher and the reference quadratic are evaluated; for a real model use a warmup mode or mean. Defaults to zeros.
tau	Eigenvalue threshold of the verticality filter (directions with $\lambda \gg \tau$ are walls). Defaults to the smallest Fisher eigenvalue (the floor scale): with the rational filter the leapfrog residual curvature is then capped at the floor, so the step is never limited by the stiff walls.
softness	Positive logistic width of the filter in log-eigenvalue. Small values approach a hard floor/wall split; the default 1 keeps the blend smooth.
floor	Minimum eigenvalue imposed on the Fisher before the filter, so a genuinely flat direction stays well defined.

Details

The accessible distribution is read from the **near-null space of the expected Fisher**. At a reference position the Fisher is eigendecomposed, $I = U\Lambda U^\top$, and a **continuous** verticality filter $w_i = \sigma((\log\lambda_i - \log\tau)/s)$ assigns each eigendirection a weight in $(0,1)$ – one for a wall (large λ), zero for the floor (small λ), with no hard cut, so a borderline direction is blended smoothly. The wall curvature $A = U\,\mathrm{diag}(w_i\lambda_i)\,U^\top$ (frequencies $\omega_i = \sqrt{w_i\lambda_i}$) defines a fixed reference quadratic.

Sampling uses a Strang splitting with a Euclidean (identity) kinetic energy: each step is a half momentum kick by $\nabla U_{\mathrm{rest}} = -\nabla\log p(\theta) - A(\theta-\theta^*)$, an **exact** harmonic flow of the reference quadratic (.gdpar_geom_subriemann_flow; a closed-form symplectic rotation per mode), and a second half kick. The stiff walls are integrated exactly regardless of the step size, so the step is limited only by the gentle floor; the soft directions follow the free-drift limit. The scheme is symplectic and time-reversible by construction (split HMC with a Gaussian part; Shahbaba et al. 2014). Because the reference quadratic is a preconditioner inside the integrator and **not** part of the target, the Metropolis correction with the exact log-density of gdpar_geom_hmc keeps the sampler exact however coarse the Gaussian approximation of the walls is: only efficiency, never correctness, depends on it.

Value

A list of class gdpar_geom_metric with position_dependent = FALSE, the identity kinetic interface (mass, inv_mass, chol_mass, logdet), an integrator closure consumed by gdpar_geom_hmc, and the diagnostic fields metric_kind = "sub_riemannian", reference, eigenvalues, verticality (the w_i), frequencies (the ω_i), n_walls, tau, softness and suggested_epsilon (a leapfrog step matched to the floor scale, since the exact walls impose no step-size limit – the source of the speed-up over a Euclidean sampler, whose step is bottlenecked by the stiffest wall).

References

Montgomery, R. (2002) *A Tour of Subriemannian Geometries, Their Geodesics and Applications*. AMS.

Shahbaba, B., Lan, S., Johnson, W. O. and Neal, R. M. (2014) Split Hamiltonian Monte Carlo. *Statistics and Computing* 24, 339–349.

See Also

gdpar_geom_hmc, gdpar_geom_fisher_simulator, gdpar_geom_metric_riemannian.

Examples

```
# A two-dimensional quasi-deterministic canyon: a soft floor (variance one)
# and a stiff wall (variance one hundredth). The expected Fisher is constant.
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * (theta[1]^2 + 100 * theta[2]^2),
  grad_log_prob = function(theta) -c(theta[1], 100 * theta[2]), dim = 2)
metric <- gdpar_geom_metric_subriemannian(
  tgt, fisher = function(theta) diag(c(1, 100)))
fit <- gdpar_geom_hmc(tgt, metric = metric, epsilon = 0.5, L = 10,
                      n_iter = 200, n_warmup = 100, seed = 1)
fit$ebfmi
```

---

gdpar_geom_orchestrate

Geometry-adaptive sampling orchestrator (opt-in)

Description

Diagnose the geometry of a posterior, select a level of the Block RG sampler hierarchy that remedies it, sample, re-diagnose, and either escalate the level or emit a **certified limit**. This is the closed loop of the Block RG charter: a single opt-in entry point over the diagnostic (gdpar_geometry_diagnostic) and the five geometry levels (gdpar_geom_metric_euclidean, gdpar_geom_metric_riemannian, gdpar_geom_metric_relativistic, gdpar_geom_metric_subriemannian), all sampled by gdpar_geom_hmc. It is standalone and does not touch the package's fit path; the default branch is bit-identical.

Usage

```
gdpar_geom_orchestrate(
  target,
  geom_target = NULL,
  fisher = NULL,
  reference = NULL,
  level_map = NULL,
  entry_level = NULL,
  budget = NULL,
```

```
  criteria = NULL,
  speed = 10,
  rest_mass = 1,
  n_grid = NULL,
  checkpoint_dir = NULL,
  laplace_fallback = FALSE,
  laplace_draws = 0L,
  seed = 20260603L,
  verbose = TRUE,
  ...
)
```

Arguments

target	The posterior the diagnostic probes: a gdpar_geometry_target from gdpar_geometry_suite, a list(model = , data = , dim =) wrapping a compiled **cmdstanr** model, a list(stan_code = , stan_data = , dim =), or a list(type = "gdpar", formula = , data = , ...) (see gdpar_geometry_diagnostic).
geom_target	The gdpar_geom_target sampled by the engine. When NULL it is derived from target for a suite target (via its make()) or a cmdstan model (via the compiled methods); supply it explicitly for other forms.
fisher	Optional fisher(theta) returning the expected Fisher information at theta. The sub-Riemannian level *requires* it (the level is otherwise skipped and named in the prescription); the Riemannian and relativistic levels use it when given and fall back to the SoftAbs curvature (always available) otherwise. Pass gdpar_geom_fisher_simulator for models with no closed form.
reference	Optional reference position for the position-dependent levels (dense, sub-Riemannian); defaults to zeros and is warm-started to the best position found as the loop proceeds.
level_map	Optional named list overriding the pathology-to-level map (names are pathology classes, values are ladder keys).
entry_level	Optional ladder key forcing the entry level (one of "euclidean_diagonal", "euclidean_dense", "riemannian", "relativistic", "sub_riemannian").
budget	A budget / stopping-rule list; see gdpar_geom_orchestrate_budget.
criteria	A success-gate list; see gdpar_geom_orchestrate_criteria.
speed, rest_mass	The relativistic level's speed and rest mass.
n_grid	Optional size grid for the diagnostic pilots (forwarded).
checkpoint_dir	Optional directory; when given, the running ledger is written atomically after each round for durability.
laplace_fallback	Logical; when TRUE and the run ends in a certified limit (the genuinely non-Gaussian canyon the ladder cannot sample), attach a gdpar_geom_laplace approximation to the result and relabel the status "certified_limit_laplace" – the mgcv/REML and INLA/Laplace competitor-parity regime (Block RG, RG.7). On the out-of-scope path it is attached only when the curvature at

	the mode is positive-definite (status "out_of_scope_laplace"). Defaults to FALSE, which leaves the output **bit-identical** (no $laplace, the original status set).
laplace_draws	Number of iid Laplace draws to carry on the attached approximation when laplace_fallback = TRUE (default 0L: the mode plus precision Gaussian is the approximation).
seed	Integer base seed; the whole adaptive trajectory is a deterministic function of it.
verbose	Logical; print an opt-in progress trace. Defaults to TRUE.
...	Forwarded to gdpar_geometry_diagnostic.

Details

Level selection (the combined map). The diagnostic's transparent, calibrated rule-based classifier is the primary selector; a continuous proximity score over the size-invariant signals is a second, independent estimator. When they agree and confidence is high, the discrete level is used directly; when they conflict or confidence is low, the controller starts at the lower of the two candidate levels and lets the escalation climb – robust at the funnel/heavy-tail border the calibration found mutually confusable. A user level_map or entry_level overrides this.

Escalation (closed-loop, armoured). On a failed level the controller re-diagnoses with a fresh pilot and lets the new signature pick the next level, under a monotone ratchet (never below a level already tried), a visited-state memo (an acyclic walk), a global round cap, per-level caps, a no-progress detector, hierarchical budget accounting with cost-aware admission and cheap-probe-then-full tiering, a per-fit wall-time watchdog, graceful degradation (the best result so far is always returned), deterministic seeding (the adaptive trajectory is bit-reproducible, so a re-run retraces it – the resumability guarantee), a multi-signal success gate with hysteresis, and a frozen-level final sampling phase. Each level is Metropolis-exact, so a wrong jump wastes only a bounded amount of budget, never correctness; the monotone ladder is the proven-generalisation backstop (level L+1 generalises level L).

The certified limit. When the budget is exhausted without success the controller returns a gdpar_geom_certificate: the demonstrated evidence in three rigour layers (algebraic = the geometry; statistical = the per-level sampler diagnostics; numerical = the budget and fits) plus a *conjectured* prescription – the smallest change predicted to break the limit, tagged as a conjecture and made falsifiable by a re-run test. A diagnosis pointing outside the geometry ladder (multimodality, a boundary, a flat direction) short-circuits to such a certificate naming the proper remedy (tempering, reparametrisation, Option A), without overreaching by pretending a geometry metric fixes it.

Value

A list of class gdpar_geom_orchestration with status ("resolved", "certified_limit" or "out_of_scope"; or, under laplace_fallback = TRUE, "certified_limit_laplace" / "out_of_scope_laplace" with an attached $laplace). A resolved run carries draws, the winning level and metric, the final fit and its gate; a limit or out-of-scope run carries a gdpar_geom_certificate. Every run carries the ledger (the per-round decision and sampling record), the initial diagnosis, the best result so far, budget_spent and reproducibility.

See Also

gdpar_geometry_diagnostic, gdpar_geom_hmc, gdpar_geom_orchestrate_budget, gdpar_geom_orchestrate_criter

Examples

```
# The budget and criteria are plain data and can be inspected / re-tuned.
b <- gdpar_geom_orchestrate_budget()
b$max_rounds

if (requireNamespace("cmdstanr", quietly = TRUE)) {
  suite <- gdpar_geometry_suite()
  # A minimal run on the isotropic control: it resolves at the cheapest
  # (Euclidean) level. Heavier targets and the certified-limit path are shown
  # in the gated tests and the Block RG vignette.
  b$tune_epsilon <- FALSE
  b$probe_iter <- 60L; b$full_iter <- 80L; b$full_warmup <- 80L
  res <- gdpar_geom_orchestrate(suite$G0_isotropic, n_grid = 1, budget = b,
                                pilot_warmup = 80L, pilot_sampling = 80L,
                                verbose = FALSE)
  res$status
}
```

`gdpar_geom_orchestrate_budget` *Budget and stopping rule for the geometry-adaptive orchestrator*

Description

The budget that bounds `gdpar_geom_orchestrate`'s closed loop and the knobs of its armour. The loop spends fits and wall-clock time against these caps, checks the remaining budget *before* starting each fit (no half-fits), and emits a certified limit rather than overrunning.

Usage

```
gdpar_geom_orchestrate_budget()
```

Details

The fields are: `max_rounds` (hard ceiling on diagnose–sample cycles), `max_levels` (cap on distinct ladder levels tried), `probe_warmup` / `probe_iter` (the cheap probe budget every level gets first) and `full_warmup` / `full_iter` (the full budget a level earns only by passing its probe – the successive-halving spirit), `epsilon` / `L` (the base leapfrog step and the trajectory length), `tune_epsilon` (whether to run a short coarse step-size search per level before the probe, targeting a healthy acceptance band, so a level is not failed merely because the base step was ill-matched to its geometry; the sub-Riemannian level searches around its own `suggested_epsilon`) and `tune_iter` (iterations per tuning probe), `max_seconds` (global wall-time cap), `max_seconds_per_fit` (the per-fit watchdog), `max_fits` (global fit cap), `n_rediagnose` (pilots per re-diagnosis; ≥ 2 enables the stability check that falls back to the safe ladder step when the re-diagnosis itself is unstable), `stall_limit` (consecutive no-progress rounds before stopping) and `hysteresis` (the margin tightening the decisive success gate).

Value

A named list of budget and stopping-rule settings.

See Also

gdpar_geom_orchestrate, gdpar_geom_orchestrate_criteria.

Examples

```
str(gdpar_geom_orchestrate_budget())
```

---

gdpar_geom_orchestrate_criteria

Success criteria for the geometry-adaptive orchestrator

Description

The multi-signal success gate of gdpar_geom_orchestrate: a level is accepted only when a *conjunction* of size-invariant sampler signals holds, never a single number. Using a conjunction (and never R-hat or the effective sample size on short runs) is the generalisation of the `rhat = Inf` false-positive lesson of sessions B9.20/B9.21: a sampler that accepts every proposal but never moves must not be mistaken for success.

Usage

```
gdpar_geom_orchestrate_criteria()
```

Details

The gate requires the acceptance rate inside a healthy band, the divergence rate below a ceiling, and the energy Bayesian fraction of missing information ("E-BFMI") above a floor. The decisive (full-budget) gate is applied with a hysteresis margin (the thresholds tightened by `1 + margin`) so a level must *clearly* pass to be declared a success, preventing chattering around the boundary.

Value

A named list of numeric criteria: `accept_low`, `accept_high` (the healthy acceptance band), `divergent_rate_high` (the divergence ceiling) and `ebfmi_low` (the E-BFMI floor).

See Also

gdpar_geom_orchestrate, gdpar_geom_orchestrate_budget.

Examples

```
str(gdpar_geom_orchestrate_criteria())
```

gdpar_geom_reservoir *Collect a reservoir of positions for the learned Gaussian-process metric*

Description

Phase one of the two-phase, decoupled-archivist design (ORPHEUS-PIMC-A section 16): run a short warmup of the geometric sampler and return the retained positions as a reservoir of sites at which the expected Fisher is later evaluated to train gdpar_geom_metric_gp_fisher. For an easy or moderately curved target the default Euclidean warmup suffices; for a funnel-like target pass a SoftAbs metric (or a hand-designed sweep of the curvature axis) instead.

Usage

```
gdpar_geom_reservoir(
  target,
  n_sites = 50L,
  metric = NULL,
  epsilon = 0.25,
  L = 15L,
  n_warmup = 200L,
  init = NULL,
  seed = NULL
)
```

Arguments

target	A gdpar_geom_target (or an object accepted by it).
n_sites	Integer number of reservoir sites to return (the retained warmup draws).
metric	Optional metric for the warmup run (defaults to the Euclidean identity); a gdpar_geom_metric_riemanni SoftAbs metric is a robust choice for curved targets.
epsilon, L, n_warmup	Leapfrog step, trajectory length and discarded warmup length for the collecting run.
init	Optional initial position (defaults to zeros).
seed	Optional integer seed for reproducibility.

Value

A numeric n_sites x dim matrix of reservoir positions.

See Also

gdpar_geom_metric_gp_fisher, gdpar_geom_hmc.

Examples

```
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta, dim = 2)
sites <- gdpar_geom_reservoir(tgt, n_sites = 30, n_warmup = 100, seed = 1)
dim(sites)
```

---

gdpar_geom_rmhmc_adaptive

Adaptive Riemannian HMC with online novelty-driven active learning

---

Description

Drive the geometric sampler with the learned expected-Fisher metric (gdpar_geom_metric_gp_fisher) while refining that metric online between trajectories – the full novelty / active-learning loop that completes the Riemannian level of the Block RG hierarchy. It realises the two-phase, decoupled-archivist design of ORPHEUS-PIMC-A (section 16) with a correctness improvement specific to Riemannian sampling.

Usage

```
gdpar_geom_rmhmc_adaptive(
  target,
  fisher = NULL,
  n_sim = 64L,
  n_sites_init = 30L,
  max_rounds = 5L,
  batch = 60L,
  n_add = 20L,
  decay = 0.5,
  novelty_tol = 0.2,
  warmup_metric = NULL,
  epsilon = 0.05,
  L = 25L,
  n_iter = 500L,
  n_warmup = 200L,
  init = NULL,
  seed = NULL,
  alpha = 1e+06,
  nugget = 1e-04,
  lengthscale = NULL
)
```

Arguments

target	A generative gdpar_geom_target carrying simulate and score.
fisher	Optional fisher(theta) function; defaults to the simulation-based estimator gdpar_geom_fisher_simulator built from target with n_sim simulations.
n_sim	Simulations per Fisher evaluation for the default estimator.
n_sites_init	Initial reservoir size (cold-start warmup draws).
max_rounds	Maximum number of adaptation rounds.
batch	Trajectories sampled per adaptation round.
n_add	Maximum new sites admitted in the first round.
decay	Geometric decay in $(0, 1]$ of admitted sites per round (the decreasing-adaptation schedule).
novelty_tol	Novelty (predictive standard deviation) threshold, used both to admit new sites and to stop the loop.
warmup_metric	Optional metric for the cold-start reservoir run; defaults to a SoftAbs Riemannian metric (good neck coverage on curved targets). Pass gdpar_geom_metric_euclidean(dim = target$dim) for a cheaper warmup.
epsilon, L	Leapfrog step and trajectory length for all phases.
n_iter, n_warmup	Retained and discarded iterations of the final phase.
init	Optional initial position (defaults to zeros).
seed	Optional integer seed for the whole loop.
alpha, nugget, lengthscale	Surrogate hyperparameters forwarded to gdpar_geom_metric_gp_fisher.

Details

The metric is held **fixed within each trajectory** (preserving the reversibility of the implicit generalised leapfrog) and is re-learned only **between rounds** from a growing reservoir; the number of new sites admitted shrinks geometrically per round (a decreasing, Robbins–Monro-style schedule) so the metric sequence settles. A final sampling phase with the **frozen** metric produces the returned draws.

One adaptation round:

1. sample a batch of trajectories with the current frozen metric (continuing the chain from the previous round);
2. score the kept positions by the surrogate's epistemic novelty (predictive standard deviation); positions above novelty_tol are candidate new reservoir sites;
3. admit at most ceiling(n_add * decay^(round - 1)) of the most novel candidates and re-learn the surrogate on the augmented reservoir.

The loop stops when the batch's maximum novelty falls below novelty_tol (the reservoir covers the typical set) or max_rounds is reached.

Correctness vs efficiency (the honesty convention of ORPHEUS-PIMC-A section 16.3): the sampler is *exact* in every phase regardless of the metric – the metric is a preconditioner, not part of

the target, so the Metropolis correction with the exact density is the corrector, and no delayed acceptance is needed (the improvement over ORPHEUS, where the surrogate enters the acceptance). What the active learning buys is *efficiency*, which is measured (E-BFMI, acceptance, novelty trace), not asserted.

Value

A list of class `gdpar_geom_rmhmc_adaptive` with `draws` (final phase), the final `metric`, the `reservoir` sites, `n_sites_trace`, `novelty_trace`, `accept_rate`, `ebfmi`, `n_divergent`, `n_rounds`, `epsilon` and `L`.

See Also

`gdpar_geom_fisher_simulator`, `gdpar_geom_metric_gp_fisher`, `gdpar_geom_hmc`.

Examples

```
# Bivariate normal location model; the loop learns the constant expected
# Fisher and samples with the frozen metric.
Sigma0 <- diag(c(1, 4))
P0 <- solve(Sigma0)
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum((theta^2) / diag(Sigma0)),
  grad_log_prob = function(theta) -theta / diag(Sigma0),
  hessian = function(theta) -P0, dim = 2,
  simulate = function(theta)
    as.numeric(theta + sqrt(diag(Sigma0)) * stats::rnorm(2)),
  score = function(theta, y) as.numeric(P0 %*% (y - theta)))
fit <- gdpar_geom_rmhmc_adaptive(tgt, n_sim = 80, n_sites_init = 8,
                                 max_rounds = 1, batch = 10, L = 8,
                                 n_iter = 20, n_warmup = 10, seed = 1)
fit$n_rounds
```

---

gdpar_geom_target *Build a target for the geometric sampling engine*

Description

Wrap a posterior so the Block RG geometric engine (`gdpar_geom_hmc`) can evaluate its log-density and gradient on the unconstrained scale. This is the three-way adapter of decision A: the integrator runs in R while the density work is delegated to a compiled cmdstan model, to an R closure, or to a suite target.

Usage

```
gdpar_geom_target(
  object = NULL,
  log_prob = NULL,
  grad_log_prob = NULL,
  dim = NULL,
  hessian = NULL,
  param_names = NULL,
  data = NULL,
  simulate = NULL,
  score = NULL
)
```

Arguments

`object`	Optional. Either a **cmdstanr** fit/model compiled with `compile_model_methods = TRUE` (exposing `$log_prob`, `$grad_log_prob` and optionally `$hessian`), or a suite instance from `gdpar_geometry_suite`'s `make()` (a list carrying `log_prob`, `grad_log_prob` and `dim`). If supplied, the closure arguments are ignored.
`log_prob, grad_log_prob`	Functions of an unconstrained parameter vector returning the log-density and its gradient. Used when `object` is `NULL`.
`dim`	Integer dimension of the unconstrained parameter vector. Required for the closure form and for a cmdstan `object` (whose unconstrained dimension cannot always be inferred).
`hessian`	Optional function returning the Hessian of `log_prob` (used by the Riemannian level of RG.3). For a cmdstan `object` the built-in `$hessian` is used when available.
`param_names`	Optional character vector of parameter names.
`data`	Optional data list when `object` is an uninstantiated cmdstan model rather than a fit (forwarded to a one-iteration fit used only to expose the methods).
`simulate`	Optional generative function `simulate(theta)` returning one synthetic data set drawn from the model at `theta` (any object the companion `score` understands). Supplying `simulate` and `score` turns the target into a *generative* target, the input the simulation-based expected-Fisher estimator `gdpar_geom_fisher_simulator` consumes.
`score`	Optional function `score(theta, y)` returning the gradient of the model's log-*likelihood* of a data set `y` at `theta`, on the unconstrained scale (the per-data-set score whose outer product is the expected Fisher). Distinct from `grad_log_prob`, which is the gradient of the log-*posterior* at the fixed observed data.

Value

A list of class `gdpar_geom_target` with elements `log_prob`, `grad_log_prob`, `hessian` (or `NULL`), `dim`, `param_names`, `backend` (`"closure"` or `"cmdstan"`), and the optional generative pieces `simulate` and `score`.

See Also

gdpar_geom_hmc, gdpar_geom_metric_euclidean.

Examples

```
# A two-dimensional standard normal via an R closure.
tgt <- gdpar_geom_target(
  log_prob = function(theta) -0.5 * sum(theta^2),
  grad_log_prob = function(theta) -theta,
  dim = 2
)
tgt$log_prob(c(0, 0))
```

---

gdpar_golden_compare *Compare a fit snapshot against a golden reference (experimental)*

Description

Four-layer comparator that locks the posterior summaries and sampler diagnostics of a fitted gdpar model against a persisted reference snapshot. Designed for regression testing of MCMC outputs: any future deviation that exceeds the principled tolerance is flagged as a failure together with the layer, item, expected and observed values, and the diagnostic delta. Layer A (structural) compares class signatures, slot shapes and column names; layer B (discrete) enforces bit-exact agreement on integer sampler diagnostics; layer C (continuous) uses Monte Carlo standard error from the golden as the principled tolerance, `|obs - exp| <= k_sigma * MC_SE_golden`; layer D (sanity) checks absolute floors that must hold regardless of the golden (R-hat, ESS, divergent percentage, E-BFMI). All four layers are evaluated; failures are aggregated rather than short-circuited so that the caller obtains a comprehensive report.

Usage

```
gdpar_golden_compare(observed, golden, k_sigma = 3, sanity_floor = NULL)
```

Arguments

observed	A list with the snapshot schema produced by gdpar_snapshot_fit: fields structural, discrete, continuous, sanity, and parametrization_resolved.
golden	A list of identical schema, typically loaded via readRDS() from an archived reference snapshot.
k_sigma	Numeric scalar; tolerance multiplier for layer C (continuous). Default 3. Lower values increase sensitivity to posterior mean drift; higher values relax it.
sanity_floor	Optional named list overriding the default absolute thresholds for layer D. Defaults to list(rhat_max = 1.05, ess_bulk_min = 100, ess_tail_min = 100,divergent_pct = 0.01, ebfmi_min = 0.3).

Details

This function is the comparator counterpart of gdpar_snapshot_fit. The typical workflow is documented in vignette("vop03_regression_testing", package = "gdpar").

Value

A list with components passed (logical scalar; TRUE iff every layer is clean), failures (data.frame with one row per failure; columns layer, item, expected, observed, delta, threshold, severity), and by_layer (named integer vector of failure counts per layer).

Status

This function is flagged **experimental**. The snapshot schema is versioned at schema_version = 1L; future Blocks 6-9 of the development roadmap may add fields and bump the schema. The tolerance contract (k_sigma, sanity floors) is also subject to refinement until the first stable release.

See Also

gdpar_snapshot_fit, gdpar.

Examples

```
make_snapshot <- function(mean_val) list(
  structural = NULL,
  discrete   = NULL,
  continuous = list(theta_ref = list(
    "theta_ref[1]" = list(mean = mean_val, sd = 0.01,
                          ess_bulk = 1000, ess_tail = 1000,
                          rhat = 1.001, mc_se = 0.001)
  )),
  sanity     = list(rhat_max = 1.0, ess_bulk_min = 1000,
                    ess_tail_min = 1000, divergent_pct = 0,
                    ebfmi_min = 1.0)
)
golden   <- make_snapshot(0.5)
observed <- make_snapshot(0.5005)
cmp <- gdpar_golden_compare(observed, golden, k_sigma = 3)
cmp$passed
```

---

gdpar_ksd_joint *Joint kernel Stein discrepancy between EB and FB posteriors*

Description

Sub-bloque 9.3.c (Block 9, Session B9.4, 2026-05-27) under canonized decision H.iv lateral. Operationalizes the open question recorded in the Roxygen of gdpar_compare_eb_fb that the marginal TV reported by the latter is only a coarse proxy of the distributional discrepancy between the EB and FB posteriors, and that the joint discrepancy over $\xi = (a, b, W, \text{dispersion})$ deserves a density-free spectral metric. The canonical choice is the kernel Stein discrepancy (KSD) of Gorham and Mackey (2017) "Measuring Sample Quality with Kernels", JMLR 18(196):1-72; Liu, Lee, and Jordan (2016) "A Kernelized Stein Discrepancy for Goodness-of-Fit Tests", ICML.

Usage

```
gdpar_ksd_joint(
  eb_fit,
  fb_fit,
  kernel = c("imq", "rbf"),
  bandwidth = c("median", "fixed"),
  bandwidth_value = NULL,
  beta = -0.5,
  ess_weighted = FALSE,
  seed = NULL,
  ...
)
```

Arguments

eb_fit	An object of class gdpar_eb_fit produced by gdpar_eb.
fb_fit	An object of class gdpar_fit produced by gdpar. Must have been fitted on the same dataset as eb_fit (same outcome, same covariates, same K/p regime).
kernel	Character scalar; one of "imq" (default) or "rbf".
bandwidth	Character scalar; one of "median" (default, classic median heuristic on FB squared pairwise distances) or "fixed" (use bandwidth_value).
bandwidth_value	Numeric scalar; bandwidth value (squared units of x) when bandwidth = "fixed". Defaults to 1.0 if NULL.
beta	Numeric scalar in (-1, 0); IMQ exponent. Defaults to -0.5 (Gorham-Mackey canonical choice). Ignored when kernel = "rbf".
ess_weighted	Logical scalar; if TRUE, thins both EB and FB draws to $\min(\widehat{\text{ESS}}_{EB}, \widehat{\text{ESS}}_{FB})$ common rows. Defaults to FALSE.
seed	Integer scalar; RNG seed for ess_weighted thinning (no effect if ess_weighted = FALSE). Defaults to NULL.
...	Reserved for future arguments; currently unused.

Details

Target choice in this iteration. The KSD requires the target distribution's score function $s_p(x) = \nabla \log p(x)$. For tractability and atomicity within Session B9.4, the implementation uses an empirical Gaussian target derived from the FB posterior draws via the empirical mean $\hat{\mu}$ and covariance

$\hat{\Sigma}$ (a Laplace approximation of the FB target): $s(x) = -\hat{\Sigma}^{-1}(x - \hat{\mu})$. The full-KSD variant that uses the FB Stan model's exact gradient via `cmdstanr`'s `grad_log_prob()` (true one-sample KSD of EB samples against the actual FB target) is a documented extension for B9.x.

Base kernel. Inverse multi-quadric (IMQ) of Gorham-Mackey $k(x, y) = (h + \|x - y\|^2)^{\beta}$, $\beta \in (-1, 0)$, default $\beta = -1/2$; the bandwidth h (in squared units of x) defaults to the *median heuristic*: the median of squared pairwise distances between FB draws, which is dimension-adaptive. The RBF (Gaussian) kernel $k(x, y) = \exp(-\|x - y\|^2/(2h))$ is provided as a textbook alternative (Liu, Lee, Jordan 2016).

Stein kernel under a Gaussian target. With $s(x) = -\hat{\Sigma}^{-1}(x - \hat{\mu})$, the canonical Stein kernel is

$$k_p(x, y) = \langle s(x), s(y)\rangle k(x, y) + \langle s(x), \nabla_y k(x, y)\rangle + \langle s(y), \nabla_x k(x, y)\rangle + \mathrm{tr}(\nabla_x \nabla_y^{\top} k(x, y)),$$

and the KSD V-statistic is

$$\mathrm{KSD}(Q, P) = \sqrt{\max\{0,\, n^{-2} \sum_{i,j=1}^{n} k_p(x_i, x_j)\}}, \qquad x_i \sim Q.$$

A value close to zero indicates that the EB posterior Q matches the empirical Gaussian fit of the FB target P; a positive value indicates joint distributional deviation. The marginal TV reported by `gdpar_compare_eb_fb` may be small even when the joint KSD detects a deviation in the joint dependence structure; the two diagnostics are complementary.

ESS-weighted variant. When `ess_weighted = TRUE`, both posteriors are thinned to a common count $\min\{\widehat{\mathrm{ESS}}_{EB}, \widehat{\mathrm{ESS}}_{FB}\}$ via uniform subsampling. This mitigates the autocorrelation bias on the V-statistic when the MCMC chains are short relative to the integrated autocorrelation time; per-variable basic ESS is computed via `posterior::ess_basic` and the minimum across retained ξ-variables is used.

Cross-reference. Pedagogical interpretation of the Stein operator and of the asymmetry between the score-based KSD and the density-free TV is documented in the vignette `vignette("v07b_eb_multivariate", package = "gdpar")`, Section 11.

Value

An object of class `gdpar_ksd_joint` with components:

`ksd_value` Numeric scalar; the KSD V-statistic (square root, clamped to ≥ 0).

`ksd_squared` Numeric scalar; the raw V-statistic before clamping (may be slightly negative under numerical noise; a negative value of small magnitude is consistent with a true KSD of zero).

`kernel` Character scalar; `"imq"` or `"rbf"`.

`bandwidth` Character scalar; `"median"` or `"fixed"`.

`bandwidth_value` Numeric scalar; the bandwidth used (median heuristic result or supplied fixed value).

`beta` Numeric scalar; IMQ exponent (or `NA` for RBF).

`n_eb_draws`, `n_fb_draws` Integer scalars; row counts after optional thinning.

`n_dim` Integer scalar; dimension of the common ξ vector.

`target_mu`, `target_Sigma` Empirical mean and covariance of the FB draws over the common ξ variables; the Gaussian target.

`ess_weighted`, `thinned_to` Logical and integer; thinning configuration.

`vars` Character vector; common parameter names between EB and FB used for the computation.

`call` The matched call.

See `print.gdpar_ksd_joint` and `summary.gdpar_ksd_joint`.

References

Gorham, J., Mackey, L. (2017). Measuring Sample Quality with Kernels. JMLR 18(196):1-72.

Liu, Q., Lee, J., Jordan, M. (2016). A Kernelized Stein Discrepancy for Goodness-of-Fit Tests. ICML.

See Also

`gdpar_compare_eb_fb` (marginal TV comparator); `gdpar_eb`; `gdpar`.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(NULL)
  n <- 200
  df <- data.frame(
    x1 = stats::rnorm(n),
    x2 = stats::rnorm(n)
  )
  df$y <- with(df, 1 + 0.5 * x1 - 0.3 * x2 + stats::rnorm(n, sd = 0.5))
  spec <- amm_spec(a = ~ x1 + x2)

  # Fit the full-Bayes (FB) and empirical-Bayes (EB) models on the same data
  fit_fb <- gdpar(
    formula       = y ~ x1 + x2,
    family        = gdpar_family("gaussian"),
    amm           = spec,
    data          = df,
    iter_warmup   = 200,
    iter_sampling = 200,
    chains        = 2
  )
  fit_eb <- gdpar_eb(
    formula       = y ~ x1 + x2,
    family        = gdpar_family("gaussian"),
    amm           = spec,
    data          = df,
    iter_warmup   = 200,
    iter_sampling = 200,
    chains        = 2
  )

  # Joint KSD on the xi posterior under an empirical Gaussian target
  ksd <- gdpar_ksd_joint(fit_eb, fit_fb)
  print(ksd)
```

```
  # ESS-weighted variant
  ksd_ess <- gdpar_ksd_joint(fit_eb, fit_fb,
                             ess_weighted = TRUE, seed = 1L)
  summary(ksd_ess)
}
```

---

gdpar_loo *Approximate leave-one-out cross-validation for a gdpar fit*

Description

Computes PSIS-LOO ("Pareto smoothed importance sampling leave-one-out") approximate cross-validation via the **loo** package, using the per-observation log-likelihood persisted by the Stan model in the generated quantity `log_lik`.

Usage

```
gdpar_loo(
  fit,
  aggregation = c("subject", "cell"),
  r_eff = NULL,
  cores = 1L,
  ...
)
```

Arguments

`fit`	A `gdpar_fit` object.
`aggregation`	Character scalar, one of `"subject"` (default) or `"cell"`. Ignored for univariate fits. See Details.
`r_eff`	Optional numeric vector of relative effective sample sizes per observation. If `NULL` (default) it is computed from the draws via `loo::relative_eff()` with the actual chain identifiers extracted from the fit. Supplying it explicitly is useful when reusing the result across multiple LOO calls on the same fit.
`cores`	Number of cores for the LOO computation; passed to `loo::loo()`. Defaults to 1 (sequential) to avoid non-determinism.
`...`	Additional arguments forwarded to `loo::loo()`.

Details

For univariate fits (`p == 1L`) the Stan model emits `log_lik` as a vector of length `n`; observations are the `n` rows of the input data.

For multivariate fits (`p > 1L`) the Stan model emits `log_lik` as an `n` by `p` matrix with `log_lik[i, k]` equal to $\log p(y_{ik} \mid \theta_i[k])$. Two aggregations are available, selected by `aggregation`:

`"subject"` **(default)** The natural observational unit is the row (subject) of the input data. Following the coord-wise factorization $p(y_i \mid \theta_i) = \prod_{k=1}^{p} D_k(y_{ik} \mid \theta_i[k])$, the per-subject log-likelihood is the sum over coordinates, $\log p(y_i \mid \theta_i) = \sum_k \log p(y_{ik} \mid \theta_i[k])$. This aggregation matches the convention used by **brms** multivariate fits with `set_rescor(FALSE)` and yields ELPD values directly comparable to per-coordinate competitors aggregated identically.

`"cell"` Each pair (i, k) is treated as an independent observation, yielding PSIS-LOO over $n \cdot p$ cells. This is useful for per-coordinate diagnostics (Pareto-k mass concentrated in a specific coordinate is a signal of a marginally identified component for that dimension), but breaks the leave-one-subject-out interpretation of classical ELPD: the implicit assumption is that the cells are exchangeable given θ_i, which is technically true under the coord-wise factorization but conflates subject-level and coordinate-level cross-validation. Use for diagnostics, not for reporting comparable ELPD values across methods.

Value

A `loo` object (S3 class `"psis_loo"`) with the ELPD estimate, the elpd_loo standard error, the Pareto-k diagnostics, and pointwise contributions.

Status

This function is flagged `@keywords experimental`. The aggregation rule (sum over k for multivariate, default `"subject"`) is stable and documented above; the signature may gain additional arguments in future versions (e.g. `integrand` for non-pointwise predictive quantities). Pareto-k diagnostics with `k > 0.7` signal that the PSIS approximation is unreliable for the affected observations; consider `loo::loo_moment_match()` or `loo::reloo()` as refinements.

See Also

gdpar, loo.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("loo", quietly = TRUE)) {
  n <- 200
  set.seed(42)
  x1 <- rnorm(n); x2 <- rnorm(n)
  y <- 0.5 + 0.7 * x1 - 0.3 * x2 + rnorm(n, sd = 0.5)
  dat <- data.frame(y = y, x1 = x1, x2 = x2)
  fit <- gdpar(y ~ x1 + x2, data = dat,
               family = gdpar_family("gaussian"),
               chains = 2, iter_warmup = 500,
               iter_sampling = 500, refresh = 0)
  lo <- gdpar_loo(fit)
  print(lo)
}
```

gdpar_meta_learner_adapter *Adapter for external meta-learners used by gdpar_compare_meta_learners*

Description

Constructor of the pluggable contract through which the comparator gdpar_compare_meta_learners dispatches to external meta-learner implementations such as **grf** or EconML. The constructor returns an object of class gdpar_meta_learner_adapter that the comparator orchestrates; two reference adapters distributed with the package (gdpar_adapter_grf and gdpar_adapter_econml) are produced by calling this constructor with the appropriate closures. Users can build additional adapters by passing their own fitting and (optionally) prediction closures, which makes the comparator open to integration with other meta-learner ecosystems (e.g. **DoubleML**, custom doubly-robust estimators) without modification of the package code.

Usage

```
gdpar_meta_learner_adapter(
  name,
  fit_predict_fun,
  predict_fun = NULL,
  requires_r = character(0L),
  requires_py = character(0L),
  native_ci = FALSE,
  description = NULL
)
```

Arguments

name	Character scalar with the adapter name. Must be non-empty and unique within a single call to gdpar_compare_meta_learners.
fit_predict_fun	Function with signature function(X, Y, T, X_newdata, level, seed_run) that fits the meta-learner on (X, Y, T) and predicts the CATE on X_newdata. X and X_newdata are data frames of covariates (the adapter is responsible for any internal conversion to matrices or to language-native arrays such as numpy.ndarray); Y is a numeric vector and T a 0/1 integer vector. level is the nominal credible level inherited from the bridge (default 0.95); seed_run is the per-method seed propagated by the comparator. The function must return a list with components cate_mean (numeric vector), cate_ci (numeric matrix [n_newdata, 2] with columns lower, upper, or NULL when the adapter does not expose a native CI), state (opaque object cached for the optional predict_fun; may be NULL), and notes (character vector with method-specific diagnostics or warnings emitted during the fit).

`predict_fun` Optional function with signature `function(state, X_newdata, level)` that re-evaluates the meta-learner on `X_newdata` using the cached `state` produced by `fit_predict_fun`. Must return a list with components `cate_mean` and `cate_ci` (same shape as in `fit_predict_fun`). `NULL` (default) signals that the adapter does not support re-prediction; the comparator falls back to `fit_predict_fun` with a diagnostic warning.

`requires_r` Character vector of R packages that the adapter needs (checked via `requireNamespace` before the fit). Default `character(0)`.

`requires_py` Character vector of Python modules that the adapter needs (checked via `reticulate::py_module_avail` before the fit, which itself requires the R package **reticulate** to be present). Default `character(0)`.

`native_ci` Logical scalar. `TRUE` when the adapter returns a native CI in `cate_ci`, `FALSE` when the adapter does not produce one (the comparator does not synthesize intervals in that case; the slot is left at `NULL`).

`description` Optional character scalar with a one-line human-readable description of the adapter, used by the `print` method.

Details

The contract is intentionally two-layered. The mandatory `fit_predict_fun` carries the full fit-and-predict cycle and is the only function required to build a valid adapter. The optional `predict_fun` re-evaluates the meta-learner on a new evaluation grid by reusing the fitted state returned by `fit_predict_fun`; when an adapter exposes `predict_fun`, the comparator method `predict.gdpar_meta_learner_comparison` dispatches to it and avoids a costly re-fit. When `predict_fun` is `NULL`, the comparator falls back to `fit_predict_fun` and emits a `gdpar_diagnostic_warning` announcing the re-fit.

Value

An object of class `gdpar_meta_learner_adapter` with components `name`, `fit_predict_fun`, `predict_fun`, `requires_r`, `requires_py`, `native_ci`, `description`.

See Also

`gdpar_compare_meta_learners`, `gdpar_adapter_grf`, `gdpar_adapter_econml`.

Examples

```
fit_pred <- function(X, Y, T, X_newdata, level, seed_run) {
  m_t <- stats::lm(Y[T == 1L] ~ ., data = X[T == 1L, , drop = FALSE])
  m_c <- stats::lm(Y[T == 0L] ~ ., data = X[T == 0L, , drop = FALSE])
  p_t <- stats::predict(m_t, newdata = X_newdata)
  p_c <- stats::predict(m_c, newdata = X_newdata)
  list(cate_mean = as.numeric(p_t - p_c), cate_ci = NULL,
       state = list(m_t = m_t, m_c = m_c), notes = character(0))
}
lin_adapter <- gdpar_meta_learner_adapter(
  name = "lm_t_learner", fit_predict_fun = fit_pred,
  native_ci = FALSE,
  description = "Linear T-learner via stats::lm (example only)"
```

```
)
print(lin_adapter)
```

`gdpar_posterior_predict` *Posterior predictive draws for a fitted gdpar model*

Description

Extracts the posterior predictive draws `y_pred` emitted by the Stan templates. Returns a matrix of dimensions S by n (scalar and K-individual paths) or an array of dimensions S by n by p (multivariate path).

Usage

```
gdpar_posterior_predict(object, ndraws = NULL, ...)
```

Arguments

`object`	An object of class `gdpar_fit`.
`ndraws`	Optional integer scalar. When given, subsamples the first `ndraws` posterior draws. When NULL (default), all draws are returned.
`...`	Unused; present for S3 generic compatibility.

Details

This function is independent of the **rstantools** generic; it is exported under the name `gdpar_posterior_predict` to avoid a hard dependency on **rstantools**. Users that load **rstantools** or **brms** will see `posterior_predict()` route to `gdpar_posterior_predict()` via the S3 method registered in NAMESPACE.

Value

Numeric matrix S by n (scalar/K-individual paths) or numeric array S by n by p (multivariate path).

gdpar_prior *Specify the priors for the AMM hierarchical Bayesian model*

Description

Build a prior specification consumed by gdpar when path = "bayes". All defaults are weakly informative on the linear-predictor scale of the family, calibrated for covariates standardized to unit variance and outcomes on the natural scale of the family. Each component can be overridden individually.

Usage

```
gdpar_prior(
  theta_ref = "normal(0, 2.5)",
  sigma_theta_ref = "student_t(3, 0, 1)",
  sigma_a = "student_t(3, 0, 1)",
  sigma_b = "student_t(3, 0, 1)",
  sigma_W = "student_t(3, 0, 1)",
  sigma_y = "student_t(3, 0, 2.5)",
  phi = "gamma(2, 0.1)",
  priors_by_kind = NULL
)
```

Arguments

theta_ref	Character scalar with the prior on the population reference theta_ref on the linear-predictor scale, in Stan syntax. Default is "normal(0, 2.5)". When grouping is active (the group argument of gdpar is not NULL), this prior is applied to the hyperparameter mu_theta_ref (the population mean of theta_ref[g]); when grouping is inactive it is applied directly to the single theta_ref, preserving the Block 6 semantics.
sigma_theta_ref	Character scalar with the prior on the hierarchical scale of theta_ref[g] across groups, in Stan syntax. Default is "student_t(3, 0, 1)" (truncated to positive values). Used only when grouping is active; ignored in the single-anchor regime.
sigma_a	Character scalar with the prior on the hierarchical scale of the additive component coefficients, in Stan syntax. Default is "student_t(3, 0, 1)" (truncated to positive values inside the Stan code).
sigma_b	Character scalar with the prior on the hierarchical scale of the multiplicative contribution to the linear predictor, in Stan syntax. Default is "student_t(3, 0, 1)" (truncated to positive values). Internally the model samples c_b = theta_ref * b_coef (the linearly identified quantity in eta = theta_ref + Z_a * a + Z_b * c_b + ruido); the prior sigma_b is applied as the scale of c_b, which coincides with the scale of b_coef only when |theta_ref| is close to 1. For weakly informative priors on covariates standardized to unit variance and outcomes on the linear-predictor scale, student_t(3, 0, 1) remains a reasonable default. See the Methodological notes section.

`sigma_W` Character scalar with the prior on the hierarchical scale of the modulating component coefficients, in Stan syntax. Default is `"student_t(3, 0, 1)"` (truncated to positive values).

`sigma_y` Character scalar with the prior on the residual standard deviation for Gaussian families, in Stan syntax. Default is `"student_t(3, 0, 2.5)"` (truncated to positive values).

`phi` Character scalar with the prior on the negative-binomial dispersion phi (Stan parametrization neg_binomial_2: variance = mu + mu^2 / phi). Default is `"gamma(2, 0.1)"`.

`priors_by_kind` Optional named list of Stan-syntax prior strings, indexed by `prior_canonical_kind` (decision 2b-iii of Block 8 Session 1). Each entry overrides the canonical default for that kind, used by the codegen when K > 1 (multi-parametric extension) introduces individual-scope parameters beyond the primary one. Recognized kinds: `mu`, `log_sigma`, `log_phi`, `logit_p`, `log_shape`, `log_nu`, `logit_pi`, `power_p`. Unrecognized kinds raise an error. Defaults to `NULL` (empty override list; canonical priors apply when K > 1). In Block 8.0 K = 1 in every built-in family, so the slot is inert; it becomes active under the multi-parametric extension queued for Block 8.1+.

Details

The defaults follow the standard weakly informative recommendations of the Stan team and are calibrated for problems in which (i) the covariates entering the additive and multiplicative bases are centered and scaled to unit variance and (ii) theta_ref is on the linear-predictor scale of the family. The package standardizes the covariates internally and reports posterior summaries on both the standardized scale (used during sampling) and the user's original scale (after back-transformation).

All scale parameters are declared on the positive real line in Stan; the prior strings supplied here are interpreted as positive-truncated when the parameter has a lower bound of zero in the Stan model.

Value

An object of class `gdpar_prior` containing the seven legacy character snippets above plus the `priors_by_kind` override list.

Methodological notes

For finite-dimensional parametric AMM specifications the conditions (PRIOR-KL) and (PRIOR-THICK) of Block 4 are satisfied automatically when the prior on the parameter is absolutely continuous with positive density at the true parameter. Each of the defaults above satisfies this property unconditionally on the relevant parameter space. For non-parametric extensions the matching of prior smoothness becomes a substantive question (see Block 4, Sections 6.1 and 7); such extensions are deferred to a future version of the package.

Internal sampling parametrization for the multiplicative component: the AMM canonical form `theta_i = theta_ref + a(x_i) + b(x_i) * theta_ref + W_term` is preserved at the user-facing level, but the Stan model samples `c_b = theta_ref * b_coef` as the free parameter and reports `b_coef = c_b / theta_ref` as a derived quantity. Centering condition (C3) is enforced empirically by column-wise centering of Z_b in the AMM design constructor, not by any constraint on c_b. This linear reparametrization yields a strictly log-concave (Gaussian-conditional) posterior in

`(theta_ref, a, c_b)`, eliminating the bimodality that the non-linear parametrization `(theta_ref, a, b_coef)` admits as an artefact of the term `theta_ref * (Z_b * b_coef)`. The prior `sigma_b` is the hierarchical scale of `c_b`; users supplying custom priors on `sigma_b` should interpret it accordingly.

The user is free to supply any Stan-syntax prior string. The package performs a syntactic check at generation time but does not attempt to verify the prior's mathematical properties.

Dependencies

None at construction time. The prior strings are inserted into the Stan model by the code generator and parsed by Stan at compile time.

References

See `vignette("v04_asymptotics_path1_bayesian", package = "gdpar")`, Section 10.1, for the role of (PRIOR-KL) and (PRIOR-THICK) in posterior consistency and contraction.

Stan Development Team (2024). Stan User's Guide, version 2.35, Section "Prior Choice Recommendations".

See Also

`gdpar`, `amm_spec`

Examples

```
pr <- gdpar_prior()
print(pr)

pr2 <- gdpar_prior(theta_ref = "normal(0, 5)")
print(pr2)
```

`gdpar_snapshot_fit` *Build a reproducibility snapshot of a fitted gdpar model (experimental)*

Description

Extract the four-layer snapshot consumed by `gdpar_golden_compare`: posterior summaries with Monte Carlo standard error per variable, integer sampler diagnostics, aggregated sanity floors, and the structural class signature of the fit. The snapshot is the canonical reference object for regression testing of MCMC outputs across package upgrades, cmdstan upgrades, or refactors that touch the sampling side of the package.

Usage

```
gdpar_snapshot_fit(fit)
```

Arguments

fit	A gdpar_fit object returned by gdpar.

Details

Persist the returned list via saveRDS() to lock the fit, and compare against future fits via gdpar_golden_compare.

Value

A list with fields structural (class signatures, slot shapes, column names), discrete (integer sampler diagnostics: n_divergent, treedepth_max_n, treedepth_max_value, n_leapfrog_total_per_chain, ebfmi_min), continuous (per-variable posterior mean, sd, ESS bulk / tail, R-hat, Monte Carlo standard error), sanity (aggregated convergence floors), and parametrization_resolved (resolved CP/NCP flags and the aggregation method used by the pre-flight).

Status

This function is flagged **experimental**. The schema is at schema_version = 1L; future Blocks 6-9 of the development roadmap may add fields and bump the schema with documented migration.

See Also

gdpar_golden_compare, gdpar.

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE)) {
  set.seed(1L)
  df <- data.frame(x1 = rnorm(50), y = rnorm(50))
  fit <- gdpar(
    y ~ x1, amm = amm_spec(a = ~ x1), data = df,
    chains = 1L, iter_warmup = 100L, iter_sampling = 100L,
    refresh = 0L, verbose = FALSE, seed = 1L
  )
  snap <- gdpar_snapshot_fit(fit)
  names(snap)
}
```

---

gdpar_spatial_dependence_diagnostic

Spatial residual dependence diagnostic for a scalar Empirical-Bayes fit

---

Description

Quantifies spatial autocorrelation in the residuals of a fitted scalar Path 1 Empirical-Bayes model via Moran's I over a spatial weight structure. gdpar assumes conditional independence; under spatial dependence that assumption is violated and the model-based (posterior / Laplace) uncertainty is too narrow. This is the spatial sibling of gdpar_dependence_diagnostic and the natural gate for gdpar_spatial_dependence_robust.

Usage

```
gdpar_spatial_dependence_diagnostic(
  object,
  coords,
  W = NULL,
  weights = c("knn", "distance"),
  k = NULL,
  residual_type = c("quantile", "response", "pearson", "deviance"),
  test = c("permutation", "analytic"),
  n_perm = 999L,
  level = 0.95,
  randomize_seed = NULL,
  seed = NULL,
  ...
)
```

Arguments

object	A scalar Path 1 fit (K = 1, p = 1): either a gdpar_eb_fit (Empirical Bayes, from gdpar_eb) or a gdpar_fit (full Bayes, from gdpar).
coords	A numeric n x 2 matrix or data frame of spatial coordinates, row-aligned with the training data.
W	Optional user-supplied n x n spatial weight matrix. When given it overrides weights/k and is row-standardized internally (its diagonal is zeroed). Supplying W is the right choice when domain knowledge (adjacency, flow, network connectivity) defines the neighbourhood structure.
weights	One of "knn" (default; k-nearest-neighbour adjacency, robust to irregular spacing) or "distance" (a distance-band whose threshold is the smallest that isolates no location). Ignored when W is supplied. Both are row-standardized.
k	Integer number of neighbours for weights = "knn". When NULL (default) the declared heuristic max(4, min(round(log n), n - 1)) is used.
residual_type	One of "quantile" (default; randomized quantile / Dunn-Smyth residuals), "response", "pearson" or "deviance".
test	One of "permutation" (default; n_perm location-relabelling permutations, two-sided via $\lvert I - E[I] \rvert$, robust to non-normal residuals and asymmetric W) or "analytic" (the Cliff-Ord normal approximation, cheaper but assuming a symmetric W – a warning is emitted otherwise).
n_perm	Integer number of permutations for the permutation test (default 999; capped below n! for tiny n).

`level`	Numeric scalar in (0, 1); the confidence level used to turn the p-value into the verdict. Defaults to 0.95.
`randomize_seed`	Optional integer seed for the randomized quantile residuals of discrete families; ignored otherwise.
`seed`	Optional integer seed for the permutation test, for reproducibility.
`...`	Unused; present for signature stability.

Details

Moran's I is hand-rolled in base R (no **spdep** / **sf** dependency). With row-standardized weights $S_0 = n$ and $E[I] = -1/(n-1)$ under the null of spatial exchangeability.

Guards and caveats. Locations with zero total weight (isolated under a supplied `W` or a too-small `k`) make Moran's I undefined: a warning is emitted and `morans_i` is returned as `NA` (kNN with $k \geq 1$ never isolates a point). Duplicate coordinates are permitted (kNN ties broken by index; spatially degenerate but well-defined). For `n < 20` a hard and for `n < 50` a soft small-sample warning is issued. Coordinates are treated as **Euclidean**: lon/lat data should be projected first (e.g. UTM), or the neighbour graph is distorted, severely so at high latitudes – great-circle distance is deliberately not supported to avoid a heavy **geosphere/sf** dependency. Finally, a significant Moran's I may reflect **either** true spatial dependence **or** model misspecification (e.g. an omitted nonlinear covariate effect); the diagnostic tests residual spatial exchangeability, not its cause.

Value

A list of class `gdpar_spatial_dependence_diagnostic` with components `residual_type`, `n`, `weights`, `k`, `style`, `n_zero_weight`, `morans_i`, `expected_i` ($-1/(n-1)$), `var_i` (analytic, else `NA`), `z` (analytic, else `NA`), `p_value`, `test`, `n_perm`, `level` and `verdict`. A `print` method is provided.

Honest scope

gdpar does **not** model the spatial dependence (that is Axis 1 / a future block, deferred and evidence-gated). This diagnostic only makes the violation visible; the companion remedy `gdpar_spatial_dependence_robust` re-estimates the uncertainty by a spatial block bootstrap.

References

Moran, P. A. P. (1950). Notes on continuous stochastic phenomena. *Biometrika* 37(1/2), 17-23.

Cliff, A. D. & Ord, J. K. (1981). *Spatial Processes: Models and Applications*. Pion, London.

See Also

`gdpar_spatial_dependence_robust`, `gdpar_dependence_diagnostic`, `gdpar_eb`

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("posterior", quietly = TRUE)) {
  n <- 100
  gx <- runif(n); gy <- runif(n)
```

```
  x <- rnorm(n)
  y <- 1 + 0.5 * x + (gx + gy) + rnorm(n)  # smooth spatial trend in residuals
  df <- data.frame(x = x, y = y)
  fit <- gdpar_eb(y ~ x, amm = amm_spec(a = ~ x), data = df,
                  chains = 2, iter_warmup = 100, iter_sampling = 100)
  gdpar_spatial_dependence_diagnostic(fit, coords = cbind(gx, gy), seed = 1)
}
```

`gdpar_spatial_dependence_robust` *Dependence-robust standard errors via a spatial block bootstrap*

Description

Re-estimates the uncertainty of a scalar Path 1 Empirical-Bayes fit so that it is robust to spatial dependence in the data, without modelling that dependence. It refits the model on `B` spatial block-bootstrap resamples (tiled or moving blocks over `coords`) and reports the bootstrap standard deviation and percentile intervals of each AMM coefficient alongside the model-based (Laplace / posterior) standard errors. As in `gdpar_dependence_robust` (its temporal sibling, with which it shares one refit engine), this is the working-independence + robust-variance stance of Liang & Zeger (1986): the point estimates are unchanged, only the reported uncertainty is made dependence-robust.

Usage

```
gdpar_spatial_dependence_robust(
  object,
  data,
  coords,
  block_size = NULL,
  residual_type = c("quantile", "response", "pearson", "deviance"),
  randomize_seed = NULL,
  scheme = c("tiled", "moving"),
  random_origin = TRUE,
  B = 199L,
  level = 0.95,
  seed = NULL,
  iter_warmup = 500L,
  iter_sampling = 500L,
  chains = 2L,
  verbose = TRUE,
  ...
)
```

Arguments

object	A scalar Path 1 fit (K = 1, p = 1): either a gdpar_eb_fit (Empirical Bayes, from gdpar_eb) or a gdpar_fit (full Bayes, from gdpar).
data	The data frame originally passed to the fitting function (gdpar_eb or gdpar), row-aligned with coords. It is resampled by spatial blocks and the model is refit on each resample; the specification is recovered from object$call.
coords	A numeric n x 2 matrix or data frame of spatial coordinates, row-aligned with data.
block_size	The number of grid cells per axis g (the block is a square of side range/g), one of three forms: NULL (default) uses the variance-optimal rate max(2, round(n^(1/4))) (see Details); a positive integer fixes it manually; or the string "auto" selects it data-drivenly by a calibration over a grid of g (no spatial plug-in exists; decision D101), computed from the fitted residuals (no extra refit), falling back to the rate when the calibration degenerates. The chosen value and method are reported (block_size, block_size_method).
residual_type	One of "quantile" (default; Dunn-Smyth randomized quantile residuals), "response", "pearson" or "deviance". Used **only** when block_size = "auto", to feed the data-driven selector; ignored otherwise.
randomize_seed	Optional integer seed for the randomized quantile residuals of discrete families; used only by the "auto" selector, for a reproducible block-size choice. Ignored otherwise.
scheme	One of "tiled" (default; non-overlapping cells) or "moving" (overlapping square blocks anchored on sampled points).
random_origin	Logical; when TRUE (default) and scheme = "tiled", the grid origin is randomized per replicate (Politis-Romano-Lahiri), breaking the deterministic boundary artifact at the cost of one extra random draw per refit.
B	Integer number of bootstrap refits. Defaults to 199.
level	Numeric scalar in (0, 1); the percentile-interval level. Defaults to 0.95.
seed	Optional integer seed controlling the block resampling and the per-refit Stan seeds, for reproducibility.
iter_warmup, iter_sampling, chains	Integer scalars controlling each refit's conditional HMC. Defaults (500, 500, 2) keep the refits short.
verbose	Logical scalar; when TRUE, prints an opt-in cost message.
...	Additional arguments (currently absorbed; reserved for forward compatibility, matching the temporal sibling).

Details

Default block-size rate (decision D100). The block side per axis is g = max(2, round(n^(1/4))). This is the $d = 2$ case of the rate that minimises the mean-squared error of the block-bootstrap *variance* estimator. Writing M for the number of points per block (linear extent $M^{1/d}$ per axis), the first-order bias from dependence broken at block edges is $O(M^{-1/d})$ (Kuensch 1989; Hall, Horowitz & Jing 1995) and the estimator variance is $O(M/n)$, so $\mathrm{MSE}(M) \sim M^{-2/d} + M/n$ is

minimised at $M \sim n^{d/(d+2)}$. At $d = 1$ this gives $M \sim n^{1/3}$ points per block, **exactly** the $n^{1/3}$ block length of the temporal default (gdpar_dependence_robust); at $d = 2$ it gives $M \sim n^{1/2}$ points per block, i.e. $g^2 = n/M \sim n^{1/2}$ cells, hence $g \sim n^{1/4}$ cells per axis. The exponent is therefore the variance-optimal rate that reduces correctly to the canonical temporal rate; block_size is user-overridable, and the data-driven *constant* (a spatial analogue of Politis & White 2004, which has no established plug-in form) is available opt-in via block_size = "auto" (the calibration over g described below; decision D101). A decorrelating cross-lineage review argued for the $n^{1/(d+4)}$ rate ($n^{1/6}$ at $d = 2$); that rate governs a different estimand – the second-order bias / two-sided distribution-function coverage, which gives $n^{1/5}$ at $d = 1$ and so does *not* reduce to the variance default's $n^{1/3}$ – and is recorded here as a registered dissent rather than adopted.

Resampling. "tiled" samples non-empty cells with replacement and truncates to n (negative bias $O(1/n)$, negligible); random_origin draws a fresh sub-cell grid shift per replicate. "moving" draws overlapping square blocks anchored to cover a sampled observation (never empty).

Guards. Collinear coordinates (zero range on an axis) abort. If all locations fall in one cell, a warning is emitted and the bootstrap SE collapses toward zero. Coordinates are Euclidean (project lon/lat first).

Data-driven block size (decision D101). With block_size = "auto" the cells-per-axis g is chosen by a calibration over a grid of g, because Politis & White (2004) has **no** established spatial plug-in. For each candidate g, B_0 cheap (no-refit) spatial block resamples give the bootstrap variance $V(g)$ of the design-weighted residual functionals $(1/n)\,[1, \tilde{gx}, \tilde{gy}]^\top z$ (the influence directions of the coefficient, so their MSE-optimal g matches the coefficient's, not merely the residual mean's); g is then chosen to minimise an empirical mean-squared error, the squared bias (anchored at the *largest* blocks, which are the least biased because the dependence-breaking bias grows like $g/\sqrt{n}$) plus a leave-one-out jackknife variance, with the $n^{1/4}$ rate as the fallback. B_0 is a declared calibration constant. A single isotropic g is used; strongly anisotropic residual dependence is a documented limitation (the minimal fix, two independent coordinate-wise calibrations, is deferred). The bias anchor is the corrected form of a cross-lineage proposal that anchored at the smallest blocks, which would have biased the selector toward anticonservative standard errors.

Value

A list of class gdpar_spatial_dependence_robust with components table (one row per coefficient with estimate, model_se, robust_se, se_ratio, ci_lower, ci_upper), block_size, block_size_method ("rate", "fixed" or "auto"; "rate" also flags an "auto" request that fell back), scheme, random_origin, n_tiles, B, B_ok, level, seed, warnings and refit_diagnostics (aggregate per-refit convergence, as in gdpar_dependence_robust). A print method is provided.

Honest scope

The bootstrap delivers robust variance, not better point estimates, and is valid for weak / short-range spatial dependence relative to the block size; it does not rescue strong long-range dependence. gdpar does not model the dependence; here it only makes its inference robust to it.

Empirical-Bayes and full-Bayes paths

Like its temporal sibling, this function accepts both a scalar gdpar_eb_fit and a scalar gdpar_fit (decision D102); see the identically named section of gdpar_dependence_robust for the full-

Bayes point-estimate / model-SE convention (posterior mean / SD), the `se_ratio` interpretation and the full-Bayes cost / Monte-Carlo caveats.

Dependencies

Uses **cmdstanr** for the refits and **posterior** to extract the coefficient estimates (Empirical-Bayes or full-Bayes). The spatial weights and blocks are hand-rolled in base R (no **spdep / sf / geosphere**).

References

Liang, K.-Y. & Zeger, S. L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika* 73(1), 13-22.

Kuensch, H. R. (1989). The jackknife and the bootstrap for general stationary observations. *Annals of Statistics* 17(3), 1217-1241.

Hall, P., Horowitz, J. L. & Jing, B.-Y. (1995). On blocking rules for the bootstrap with dependent data. *Biometrika* 82(3), 561-574.

Politis, D. N. & Romano, J. P. (1992). A circular block resampling procedure for stationary data. In *Exploring the Limits of Bootstrap*, 263-270. Wiley, New York.

Lahiri, S. N. (2003). *Resampling Methods for Dependent Data*. Springer, New York.

Nordman, D. J. & Lahiri, S. N. (2004). On optimal spatial subsample size for variance estimation. *Annals of Statistics* 32(5), 1981-2027.

Politis, D. N. & White, H. (2004). Automatic block-length selection for the dependent bootstrap. *Econometric Reviews* 23(1), 53-70 (with the Patton, Politis & White 2009 correction; the temporal plug-in whose spatial analogue is the `"auto"` calibration).

See Also

`gdpar_spatial_dependence_diagnostic`, `gdpar_dependence_robust`, `gdpar_eb`

Examples

```
if (requireNamespace("cmdstanr", quietly = TRUE) &&
    requireNamespace("posterior", quietly = TRUE)) {
  n <- 100
  gx <- runif(n); gy <- runif(n)
  x <- rnorm(n)
  y <- 1 + 0.5 * x + (gx + gy) + rnorm(n)
  df <- data.frame(x = x, y = y)
  fit <- gdpar_eb(y ~ x, amm = amm_spec(a = ~ x), data = df,
                  chains = 2, iter_warmup = 100, iter_sampling = 100)
  # B kept small for a fast example; use B >= 199 in practice.
  gdpar_spatial_dependence_robust(fit, data = df, coords = cbind(gx, gy),
                                  B = 10, seed = 1, iter_warmup = 100,
                                  iter_sampling = 100, chains = 2)
  # Data-driven block size (calibration over the cell grid), opt-in:
  gdpar_spatial_dependence_robust(fit, data = df, coords = cbind(gx, gy),
                                  block_size = "auto", B = 10, seed = 1,
                                  iter_warmup = 100, iter_sampling = 100,
                                  chains = 2)
```

```
}
```

`is_gdpar_meta_learner_adapter` *Test whether an object is a gdpar_meta_learner_adapter*

Description

Test whether an object is a gdpar_meta_learner_adapter

Usage

```
is_gdpar_meta_learner_adapter(x)
```

Arguments

`x`	Object to test.

Value

`TRUE` when `x` inherits from class `gdpar_meta_learner_adapter`, `FALSE` otherwise.

See Also

`gdpar_meta_learner_adapter`.

Examples

```
a <- gdpar_meta_learner_adapter(
  name = "dummy",
  fit_predict_fun = function(X, Y, T, X_newdata, level, seed_run) {
    list(cate_mean = rep(0, nrow(X_newdata)), cate_ci = NULL,
         state = NULL, notes = character(0))
  }
)
is_gdpar_meta_learner_adapter(a)
is_gdpar_meta_learner_adapter(list())
```

length.gdpar_formula_set *Number of K-individual parameters in a gdpar_formula_set*

Description

Number of K-individual parameters in a gdpar_formula_set

Usage

```
## S3 method for class 'gdpar_formula_set'
length(x)
```

Arguments

x	An object of class gdpar_formula_set.

Value

Integer scalar equal to the number of slots.

names.gdpar_formula_set *Slot names of a gdpar_formula_set*

Description

Slot names of a gdpar_formula_set

Usage

```
## S3 method for class 'gdpar_formula_set'
names(x)
```

Arguments

x	An object of class gdpar_formula_set.

Value

Character vector with the slot names in declaration order.

override *Layer a per-dimension override on a dims_spec*

Description

Given a dims_spec produced by dimwise, attach a per-dimension override that replaces the additive and/or multiplicative formula for a specific dimension index k.

Usage

```
override(dims, k, a, b)
```

Arguments

dims	A dims_spec object produced by dimwise.
k	Integer scalar with the dimension index to override. Must be a positive integer. Coherence with the global dimension p is checked at the point of consumption by amm_spec.
a	Optional one-sided formula replacing the additive basis for dimension k. Pass NULL explicitly to disable the additive component for that dimension while keeping it active elsewhere. Omit to leave it unchanged from the base.
b	Optional one-sided formula replacing the multiplicative basis for dimension k. Same semantics as a.

Details

Overrides are recorded by integer index, not by position in a list, so the order of override calls does not matter beyond the overwrite semantics noted above.

The semantics of "unchanged" versus "disabled" requires distinguishing between an argument that is missing from the call and an argument that is explicitly NULL. The function uses missing for this distinction: omit the argument to inherit from the base; pass NULL to disable for this dimension only.

At least one of a or b must be supplied; calling override without any change is treated as a user error and aborts with an informative message.

Value

A new dims_spec with the override registered. Multiple calls to override compose; calling override twice with the same k replaces the previous override for that index.

See Also

dimwise, amm_spec

Examples

```
base <- dimwise(a = ~ x1 + x2, b = ~ x1)
v1 <- override(base, k = 2L, a = ~ x1)
v2 <- override(v1, k = 3L, b = NULL)
print(v2)
```

`pp_check.gdpar_fit` *Posterior predictive check for a fitted gdpar model*

Description

S3 method off the **bayesplot** generic. Forwards `y_pred` draws and the observed response `y` to one of the `bayesplot::ppc_*` family of functions selected by `type`.

Usage

```
## S3 method for class 'gdpar_fit'
pp_check(
  object,
  type = c("dens_overlay", "hist", "ecdf_overlay", "stat", "intervals"),
  coord = NULL,
  ndraws = 50L,
  ...
)
```

Arguments

`object`	An object of class `gdpar_fit`.
`type`	Character scalar selecting the bayesplot ppc plot: `"dens_overlay"` (default), `"hist"`, `"ecdf_overlay"`, `"stat"` or `"intervals"`.
`coord`	Integer scalar between 1 and p; required for multivariate fits.
`ndraws`	Integer scalar; subsamples the first `ndraws` posterior draws before plotting. Defaults to 50.
`...`	Additional arguments forwarded to the bayesplot function.

Value

A `ggplot` object produced by bayesplot.

Dependencies

Requires **bayesplot** (Suggests). If not installed, raises a clear error.

`predict.gdpar_causal_bridge` *Predict method for gdpar_causal_bridge objects*

Description

Recompute the per-observation CATE on a new evaluation grid using the two underlying fits stored in the bridge. The structural compatibility of the two fits was validated when the bridge was constructed and is not re-checked.

Usage

```
## S3 method for class 'gdpar_causal_bridge'
predict(
  object,
  newdata,
  level = NULL,
  summary = c("all", "draws", "mean_ci"),
  ...
)
```

Arguments

`object`	An object of class `gdpar_causal_bridge`.
`newdata`	Data frame on which to evaluate the CATE. Required.
`level`	Numeric scalar in (0, 1) with the credible level for the new intervals. Defaults to the level recorded on `object`.
`summary`	Character scalar selecting the output form: `"all"` (default; returns the same structure as `gdpar_causal_bridge`'s `cate_*` slots), `"draws"` (returns the raw `cate_draws` object), `"mean_ci"` (returns a list with `cate_mean` and `cate_ci`).
`...`	Unused; present for S3 generic compatibility.

Value

Depends on `summary`: a list with components `cate_draws`, `cate_mean`, `cate_ci`, `n_draws`, `n_obs` for `summary = "all"`; the draws array for `summary = "draws"`; or a list with the two summary slots for `summary = "mean_ci"`.

predict.gdpar_eb_fit *Prediction for gdpar_eb_fit*

Description

Computes posterior predictions from the conditional HMC draws at the plug-in EB estimate $\widehat{\theta}_{ref}^{EB}$. Supports both in-sample and out-of-sample prediction via the `newdata` argument; the multivariate `newdata` path is deferred to Sub-phase 8.6.C together with the rest of `p > 1`.

Usage

```
## S3 method for class 'gdpar_eb_fit'
predict(
  object,
  newdata = NULL,
  type = c("response", "linear_predictor"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `gdpar_eb_fit` object.
`newdata`	Optional data frame with the same variables as the training data. When `NULL` (default), in-sample predictions are returned.
`type`	One of `"response"` (default; on the y scale via the family's inverse link) or `"linear_predictor"` (on the eta scale).
`level`	Numeric scalar in (0, 1); credible-interval level. Defaults to 0.95.
`...`	Unused.

Value

A list with components `mean`, `lower`, `upper`, `draws` (matrix S x n).

predict.gdpar_fit *Posterior draws of theta_i for a fitted gdpar model*

Description

Returns the posterior draws of the individual parameters $\theta_i = \theta_{\text{ref}} + \Delta(x_i, \theta_{\text{ref}})$ on the linear-predictor scale by default, or on the response scale after applying the family's inverse link.

Usage

```
## S3 method for class 'gdpar_fit'
predict(
  object,
  newdata = NULL,
  type = c("theta_i", "linear_predictor", "response"),
  summary = c("draws", "mean_se", "quantiles"),
  ...
)
```

Arguments

object	An object of class gdpar_fit.
newdata	Optional data frame on which to compute predictions. When NULL (default), predictions are computed on the training data and returned from the Stan-side theta_i draws, without re-evaluation.
type	Character scalar: "theta_i" (default; the linear predictor of the individual parameter), "linear_predictor" (synonym), or "response" (the inverse link of the linear predictor).
summary	Character scalar: "draws" (default; full draws as a numeric matrix of shape draws-by-observations), "mean_se" (posterior mean and standard error per observation), or "quantiles" (posterior 5%, 50% and 95% quantiles per observation).
...	Unused; present for S3 generic compatibility.

Details

Predictions on newdata require evaluating the centered bases at the new covariate values. The package uses the centering parameters (column means of Z_a and Z_b, column means and standard deviations of X) recorded at fit time so that the transformation is identical to that applied during training.

For type = "response", the inverse link is applied at the draw level. The reported quantiles are therefore the quantiles of the response-scale distribution and are not the inverse link of the linear-predictor quantiles unless the link is the identity.

Value

The shape depends on summary: a numeric matrix when summary = "draws"; a data frame when summary is "mean_se" or "quantiles".

Dependencies

Uses **posterior** to extract draws.

predict.gdpar_meta_learner_comparison
Predict method for gdpar_meta_learner_comparison objects

Description

Re-evaluate the CATE on a new grid for every method in the comparison. Adapters that expose `predict_fun` reuse the cached fitted state without a refit; adapters that do not are invoked through `fit_predict_fun` (full refit) and a `gdpar_diagnostic_warning` is emitted. The bridge component is re-evaluated via `predict.gdpar_causal_bridge`.

Usage

```
## S3 method for class 'gdpar_meta_learner_comparison'
predict(object, newdata, level = NULL, bridge = NULL, data = NULL, ...)
```

Arguments

`object`	A `gdpar_meta_learner_comparison` object.
`newdata`	Data frame with the new evaluation grid. Required.
`level`	Optional numeric scalar in `(0, 1)` overriding the credible level used when constructing the original comparison. Default `NULL` reuses `object$level`.
`bridge`	Optional `gdpar_causal_bridge` object to use instead of the bridge cached inside `object`. Default `NULL` reuses `object$bridge`; supply this argument when the cached bridge was stripped (e.g. after a `saveRDS` round-trip that lost the two fits).
`data`	Optional list with components `X`, `T`, `Y` (and optionally `X_newdata`) for the case of a forced re-fit. Default `NULL` attempts to recover the training data of each arm from the bridge's stored fits (same convention as `gdpar_compare_meta_learners`).
`...`	Reserved for future arguments; currently unused.

Value

A list of class `predict.gdpar_meta_learner_comparison` with components `bridge`, `external`, `comparison`, and the new `newdata`. The structure mirrors a comparison object but without the cached state.

preflight_global_decision *Accessor: aggregated per-component preflight decisions*

Description

Returns the per-component summary (one row per component) with the aggregated decision, the agreement share, and the aggregation method used.

Usage

```
preflight_global_decision(report)
```

Arguments

report	An object of class gdpar_preflight_report.

Value

Data frame with one row per component.

preflight_per_dim *Accessor: per-dimension preflight decisions*

Description

Returns the per-coordinate, per-component data frame that drives the report. Use this for dplyr/ggplot2 analysis.

Usage

```
preflight_per_dim(report)
```

Arguments

report	An object of class gdpar_preflight_report.

Value

Data frame with one row per (component, dim) pair.

print.amm_builder *Print method for amm_builder objects*

Description

Print method for amm_builder objects

Usage

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

Arguments

x	An object of class amm_builder.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.amm_spec *Print method for amm_spec objects*

Description

Print method for amm_spec objects

Usage

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

Arguments

x	An object of class amm_spec.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.dims_spec *Print method for dims_spec objects*

Description

Print method for dims_spec objects

Usage

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

Arguments

x	A dims_spec object.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_bvm_report *Print method for gdpar_bvm_report objects*

Description

Print method for gdpar_bvm_report objects

Usage

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

Arguments

x	An object of class gdpar_bvm_report.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_causal_bridge *Print method for gdpar_causal_bridge objects*

Description

Concise summary of the bridge: structural compatibility (family, AMM level, anchor, K, p), number of posterior draws, number of evaluation observations, and the credible level used for the intervals.

Usage

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

Arguments

x	An object of class gdpar_causal_bridge.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_coef *Print method for gdpar_coef objects*

Description

Three verbosity levels: "global" (default) shows the theta_ref summary plus active-component counts; "coord" appends per-coordinate component means; "full" appends every per-coordinate data.frame with all summary statistics.

Usage

```
## S3 method for class 'gdpar_coef'
print(x, level = c("global", "coord", "full"), digits = 4L, ...)
```

Arguments

x	Object of class gdpar_coef.
level	One of "global", "coord", "full".
digits	Integer scalar passed to format() for numeric columns.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

`print.gdpar_contraction_report` *Print method for gdpar_contraction_report objects*

Description

Print method for gdpar_contraction_report objects

Usage

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

Arguments

`x`	An object of class `gdpar_contraction_report`.
`...`	Unused; present for S3 generic compatibility.

Value

Invisibly returns `x`.

`print.gdpar_dependence_diagnostic` *Print method for gdpar_dependence_diagnostic objects*

Description

Print method for gdpar_dependence_diagnostic objects

Usage

```
## S3 method for class 'gdpar_dependence_diagnostic'
print(x, digits = 3L, ...)
```

Arguments

`x`	A `gdpar_dependence_diagnostic` object.
`digits`	Integer; significant digits for the printed statistics.
`...`	Unused; present for S3 generic compatibility.

Value

Invisibly returns `x`.

print.gdpar_dependence_robust

Print method for gdpar_dependence_robust objects

Description

Print method for gdpar_dependence_robust objects

Usage

```
## S3 method for class 'gdpar_dependence_robust'
print(x, digits = 3L, ...)
```

Arguments

x	A gdpar_dependence_robust object.
digits	Integer; significant digits for the printed table.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_diagnostics

Print method for gdpar_diagnostics objects

Description

Print method for gdpar_diagnostics objects

Usage

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

Arguments

x	An object of class gdpar_diagnostics.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_eb_fb_comparison *Print method for gdpar_eb_fb_comparison*

Description

Concise console summary of an Empirical-Bayes vs Fully-Bayes comparison: paths involved, number of common xi parameters with TV values, summary statistics of the TV distribution and width-ratio distribution, and the first six rows of the per-anchor diff table.

Usage

```
## S3 method for class 'gdpar_eb_fb_comparison'
print(x, digits = 3L, ...)
```

Arguments

x	A gdpar_eb_fb_comparison object.
digits	Integer scalar passed to format(); defaults to 3.
...	Unused.

Value

The object x invisibly.

print.gdpar_eb_fit *Print method for gdpar_eb_fit*

Description

Concise console summary of an Empirical-Bayes fit: family, link, AMM level, EB point estimate(s) of theta_ref, numerical diagnostics of the Step (i) Laplace approximation, and a one-line summary of the conditional HMC fit.

Usage

```
## S3 method for class 'gdpar_eb_fit'
print(x, digits = 3L, ...)
```

Arguments

x	A gdpar_eb_fit object.
digits	Integer scalar passed to format(); defaults to 3.
...	Unused.

Value

The object x invisibly.

---

print.gdpar_family *Print method for gdpar_family objects*

Description

Print method for gdpar_family objects

Usage

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

Arguments

x	An object of class gdpar_family.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

---

print.gdpar_family_multi *Print method for gdpar_family_multi objects*

Description

Print method for gdpar_family_multi objects

Usage

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

Arguments

x	An object of class gdpar_family_multi.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_fit *Print method for gdpar_fit objects*

Description

Concise summary of the fitted model: AMM specification, family, anchor, sampler dimensions and convergence verdict.

Usage

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

Arguments

x	An object of class gdpar_fit.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_formula_set *Print method for gdpar_formula_set objects*

Description

Print method for gdpar_formula_set objects

Usage

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

Arguments

x	An object of class gdpar_formula_set.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_geometry_diagnostic

Print method for gdpar_geometry_diagnostic objects

Description

Print method for gdpar_geometry_diagnostic objects

Usage

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

Arguments

x	An object of class gdpar_geometry_diagnostic.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_geometry_target

Print method for gdpar_geometry_target objects

Description

Print method for gdpar_geometry_target objects

Usage

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

Arguments

x	An object of class gdpar_geometry_target.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

`print.gdpar_geom_bridge` *Print method for gdpar_geom_bridge objects*

Description

Print method for gdpar_geom_bridge objects

Usage

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

Arguments

`x`	A `gdpar_geom_bridge`.
`...`	Unused.

Value

Invisibly returns `x`.

`print.gdpar_geom_certificate` *The certified-limit object of the geometry-adaptive orchestrator*

Description

The first-class certificate emitted by `gdpar_geom_orchestrate` when the budget is exhausted without a level meeting the success gate, or when the diagnosis points outside the geometry sampler ladder. It separates the demonstrated evidence in three rigour layers (`algebraic` = the geometry, `statistical` = the per-level sampler diagnostics, `numerical` = the budget and fits) from a `prescription` of conjectured, falsifiable fixes, and carries a `reproducibility` block. This print method summarises it.

Usage

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

Arguments

`x`	A `gdpar_geom_certificate`.
`...`	Unused.

Value

Invisibly returns `x`.

See Also

`gdpar_geom_orchestrate`.

`print.gdpar_geom_fit` *Print method for gdpar_geom_fit objects*

Description

Print method for gdpar_geom_fit objects

Usage

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

Arguments

`x`	A `gdpar_geom_fit`.
`...`	Unused.

Value

Invisibly returns `x`.

`print.gdpar_geom_hmc` *Print method for gdpar_geom_hmc objects*

Description

Print method for gdpar_geom_hmc objects

Usage

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

Arguments

`x`	A `gdpar_geom_hmc`.
`...`	Unused.

Value

Invisibly returns `x`.

`print.gdpar_geom_laplace` *Print method for gdpar_geom_laplace objects*

Description

Print method for gdpar_geom_laplace objects

Usage

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

Arguments

`x`	A `gdpar_geom_laplace`.
`...`	Unused.

Value

Invisibly returns `x`.

`print.gdpar_geom_orchestration` *Print method for gdpar_geom_orchestration objects*

Description

Print method for gdpar_geom_orchestration objects

Usage

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

Arguments

`x`	A `gdpar_geom_orchestration`.
`...`	Unused.

Value

Invisibly returns `x`.

print.gdpar_geom_rmhmc_adaptive *Print method for gdpar_geom_rmhmc_adaptive objects*

Description

Print method for gdpar_geom_rmhmc_adaptive objects

Usage

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

Arguments

x	A gdpar_geom_rmhmc_adaptive.
...	Unused.

Value

Invisibly returns x.

print.gdpar_geom_target *Print method for gdpar_geom_target objects*

Description

Print method for gdpar_geom_target objects

Usage

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

Arguments

x	A gdpar_geom_target.
...	Unused.

Value

Invisibly returns x.

print.gdpar_identifiability_report *Print method for gdpar_identifiability_report objects*

Description

Print method for gdpar_identifiability_report objects

Usage

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

Arguments

x	An object of class gdpar_identifiability_report.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_ksd_joint *Print method for gdpar_ksd_joint*

Description

Print method for gdpar_ksd_joint

Usage

```
## S3 method for class 'gdpar_ksd_joint'
print(x, digits = 4L, ...)
```

Arguments

x	A gdpar_ksd_joint object.
digits	Integer scalar; significant digits for the KSD value. Defaults to 4.
...	Reserved.

Value

Invisibly returns x.

print.gdpar_meta_learner_adapter

Print method for gdpar_meta_learner_adapter

Description

Print method for gdpar_meta_learner_adapter

Usage

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

Arguments

x	A gdpar_meta_learner_adapter object.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_meta_learner_comparison

Print method for gdpar_meta_learner_comparison objects

Description

Concise summary of the comparison: bridge identifier, number of observations and methods, per-method timing and CI availability, and a head of the three concordance matrices.

Usage

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

Arguments

x	A gdpar_meta_learner_comparison object.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_preflight_report *Print method for gdpar_preflight_report*

Description

Prints a compact summary by default (`level = "global"`). Use `level = "dim"` to print only the per-coordinate table, or `level = "both"` to print both.

Usage

```
## S3 method for class 'gdpar_preflight_report'
print(x, level = c("global", "dim", "both"), ...)
```

Arguments

`x`	An object of class `gdpar_preflight_report`.
`level`	Character scalar: `"global"` (default), `"dim"`, or `"both"`.
`...`	Unused; present for S3 generic compatibility.

Value

Invisibly returns `x`.

print.gdpar_prior *Print method for gdpar_prior objects*

Description

Print method for gdpar_prior objects

Usage

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

Arguments

`x`	An object of class `gdpar_prior`.
`...`	Unused; present for S3 generic compatibility.

Value

Invisibly returns `x`.

print.gdpar_spatial_dependence_diagnostic

Print method for gdpar_spatial_dependence_diagnostic objects

Description

Print method for gdpar_spatial_dependence_diagnostic objects

Usage

```
## S3 method for class 'gdpar_spatial_dependence_diagnostic'
print(x, digits = 3L, ...)
```

Arguments

x	A gdpar_spatial_dependence_diagnostic object.
digits	Integer; significant digits for the printed statistics.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.gdpar_spatial_dependence_robust

Print method for gdpar_spatial_dependence_robust objects

Description

Print method for gdpar_spatial_dependence_robust objects

Usage

```
## S3 method for class 'gdpar_spatial_dependence_robust'
print(x, digits = 3L, ...)
```

Arguments

x	A gdpar_spatial_dependence_robust object.
digits	Integer; significant digits for the printed table.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

`print.summary.gdpar_causal_bridge` *Print method for summary.gdpar_causal_bridge objects*

Description

Print method for summary.gdpar_causal_bridge objects

Usage

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

Arguments

`x`	An object of class `summary.gdpar_causal_bridge`.
`...`	Unused; present for S3 generic compatibility.

Value

Invisibly returns `x`.

`print.summary.gdpar_eb_fb_comparison` *Print method for summary.gdpar_eb_fb_comparison*

Description

Print method for summary.gdpar_eb_fb_comparison

Usage

```
## S3 method for class 'summary.gdpar_eb_fb_comparison'
print(x, digits = 3L, ...)
```

Arguments

`x`	A `summary.gdpar_eb_fb_comparison` object.
`digits`	Integer scalar passed to `format()`; defaults to 3.
`...`	Unused.

Value

The object `x` invisibly.

print.summary.gdpar_eb_fit

Print method for summary.gdpar_eb_fit

Description

Print method for summary.gdpar_eb_fit

Usage

```
## S3 method for class 'summary.gdpar_eb_fit'
print(x, digits = 3L, ...)
```

Arguments

x	A summary.gdpar_eb_fit object.
digits	Integer scalar passed to format(); defaults to 3.
...	Unused.

Value

The object x invisibly.

print.summary.gdpar_meta_learner_comparison

Print method for summary.gdpar_meta_learner_comparison objects

Description

Print method for summary.gdpar_meta_learner_comparison objects

Usage

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

Arguments

x	A summary.gdpar_meta_learner_comparison object.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

print.W_basis *Print method for W_basis objects*

Description

Print method for W_basis objects

Usage

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

Arguments

x	An object of class W_basis.
...	Unused; present for S3 generic compatibility.

Value

Invisibly returns x.

residuals.gdpar_fit *Residuals for a fitted gdpar model*

Description

Posterior-predictive residuals for a gdpar_fit object, covering response, Pearson, deviance and randomized quantile (Dunn-Smyth 1996) residual types. The residuals are computed from the y_pred draws emitted by the Stan templates and are returned as a numeric vector (scalar and K-individual paths) or as a numeric matrix n by p (multivariate path).

Usage

```
## S3 method for class 'gdpar_fit'
residuals(
  object,
  type = c("quantile", "response", "pearson", "deviance"),
  coord = NULL,
  randomize_seed = NULL,
  ...
)
```

Arguments

`object`	An object of class `gdpar_fit`.
`type`	Character scalar in `"response"` (raw residual $y_i - \bar{y}_{i,\text{pred}}$), `"pearson"` (response divided by the posterior-predictive standard deviation per observation), `"deviance"` (family-specific deviance contribution; falls back to a Pearson-like surrogate for mixtures and Hurdle), or `"quantile"` (Bayesian randomized quantile residuals; the canonical residual under the model). Default is `"quantile"`.
`coord`	Integer scalar between 1 and p; only used for multivariate fits. When `NULL` (default) the function returns a matrix `n` by `p` with one column per coordinate.
`randomize_seed`	Optional integer scalar to set the RNG seed used by the randomized quantile residual under discrete families. When `NULL`, the residuals use the global RNG state.
`...`	Unused; present for S3 generic compatibility.

Value

A numeric vector for scalar / K-individual paths; a numeric matrix `n` by `p` (or numeric vector of length `n` if `coord` is specified) for multivariate paths.

Methodology

The Bayesian randomized quantile residual averages the nonparametric ECDF of `y_pred` draws at y_i across posterior draws, adding a uniform jitter on the equality mass when the family is discrete (Dunn and Smyth 1996). Under a correctly specified model the residuals are marginally $\mathcal{N}(0, 1)$ regardless of the family. Deviance and Pearson residuals are provided for parity with frequentist diagnostics; their distribution under the null model is approximate for non-Gaussian families. For mixtures (ZIP/ZINB) and Hurdle families the deviance residual is approximated by a Pearson-like surrogate; the quantile residual remains canonical and is recommended.

References

Dunn, P. K. and Smyth, G. K. (1996). Randomized Quantile Residuals. *Journal of Computational and Graphical Statistics*, 5(3), 236-244.

`summary.gdpar_causal_bridge` *Summary method for gdpar_causal_bridge objects*

Description

Returns a structured summary object with a per-observation table of the posterior CATE (mean, lower and upper credible bounds), the marginal average treatment effect (ATE) computed as the mean of the per-observation CATE, and the credible level used.

Usage

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

Arguments

object	An object of class gdpar_causal_bridge.
...	Unused; present for S3 generic compatibility.

Details

For scalar bridges (K = 1, p = 1) the table has one row per observation in newdata. For multivariate (p > 1) or K-individual (K > 1) bridges, the table has one row per (observation, dim/slot) pair and includes a slot column.

Value

A list of class summary.gdpar_causal_bridge with components table (data frame), ate (named vector of marginal ATE per slot), ate_ci (matrix of marginal ATE credible bounds per slot), level, type, n_draws, n_obs. The companion print method formats the object.

---

summary.gdpar_coef *Summary method for gdpar_coef objects*

Description

Returns a compact list of aggregated statistics: number of coordinates, count of active components per type, and average of the posterior means across coordinates for theta_ref.

Usage

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

Arguments

object	Object of class gdpar_coef.
...	Unused.

Value

A list with elements p, n_active (named integer with components a/b/W), theta_ref_mean (mean of theta_ref posterior means across coords), summary_stats.

summary.gdpar_eb_fb_comparison *Summary method for gdpar_eb_fb_comparison*

Description

Returns a structured summary suitable for programmatic access and for the canonical `print.summary.gdpar_eb_fb_compar` method. Aggregates the TV table (mean / median / max / quartiles) and the coverage table (mean width_ratio per slot under Path C, or overall under the other regimes).

Usage

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

Arguments

`object`	A `gdpar_eb_fb_comparison` object.
`...`	Unused.

Value

An object of class `summary.gdpar_eb_fb_comparison`.

summary.gdpar_eb_fit *Summary method for gdpar_eb_fit*

Description

Returns a structured summary suitable for programmatic access and for the canonical `print.summary.gdpar_eb_fit` method. Inflates the conditional credible intervals by the Proposition 7B scalar correction when `eb_correction = TRUE` was requested at fit time.

Usage

```
## S3 method for class 'gdpar_eb_fit'
summary(object, level = 0.95, ...)
```

Arguments

`object`	A `gdpar_eb_fit` object.
`level`	Numeric scalar in (0, 1); credible-interval level. Defaults to 0.95.
`...`	Unused.

Value

An object of class `summary.gdpar_eb_fit`.

summary.gdpar_fit *Summary method for gdpar_fit objects*

Description

Returns the posterior summary table for the user-facing parameters (theta_ref, hierarchical scales, family-specific dispersion).

Usage

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

Arguments

object	An object of class gdpar_fit.
...	Unused; present for S3 generic compatibility.

Value

A data frame of posterior summaries (mean, median, standard deviation, 5% and 95% quantiles, R-hat, ESS bulk and tail).

Dependencies

Posterior summaries are computed by **posterior**.

summary.gdpar_ksd_joint *Summary method for gdpar_ksd_joint*

Description

Summary method for gdpar_ksd_joint

Usage

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

Arguments

object	A gdpar_ksd_joint object.
...	Reserved.

Value

An object of class `summary.gdpar_ksd_joint` with components `ksd_value`, `ksd_squared`, `kernel`, `bandwidth_value`, `n_dim`, `vars`, and `interpretation` (character scalar). Use `print()` on the returned object for a formatted display.

---

summary.gdpar_meta_learner_comparison

Summary method for gdpar_meta_learner_comparison objects

Description

Returns a structured summary object with the three concordance matrices in long format, per-method ATE (mean of `cate_mean`), per-method ATE CI bounds (when the adapter exposes native per-observation CIs, the bounds are the mean of the per-observation bounds; otherwise `NA`), and per-method timing.

Usage

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

Arguments

`object`	A `gdpar_meta_learner_comparison` object.
`...`	Unused; present for S3 generic compatibility.

Value

A list of class `summary.gdpar_meta_learner_comparison`.

---

summary.gdpar_preflight_report

Summary method for gdpar_preflight_report

Description

Returns a list with the per-component aggregated table, the overall agreement (mean of the per-component agreement, NA values dropped), the number of components, the number of coordinates, and a count of per-dim CP/NCP/absent decisions.

Usage

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

Arguments

object	An object of class gdpar_preflight_report.
...	Unused; present for S3 generic compatibility.

Value

Named list as described above.

W_basis *Construct a functional basis for the modulating component W*

Description

Define the finite-dimensional space of functions $W : \Theta \to \mathbb{R}^{p \times d}$ from which the modulating component of the AMM canonical form is drawn. The basis is evaluated at the anchor θ_0 and at the working values of the population reference θ_{ref} during fitting.

Usage

```
W_basis(
  type = c("polynomial", "bspline", "user"),
  degree = NULL,
  knots = NULL,
  df = NULL,
  boundary_knots = NULL,
  basis_fn = NULL,
  dim = NULL,
  p = NULL
)
```

Arguments

type	Character scalar identifying the type of basis. One of "polynomial", "bspline", "user".
degree	Positive integer with the polynomial degree (type = "polynomial") or the B-spline degree (type = "bspline"). Ignored for type = "user". When NULL (default), resolves to 1L for type = "polynomial" and 3L for type = "bspline" (the conventional cubic-spline default).
knots	Numeric vector of interior knots used by splines::bs when type = "bspline". Required for type = "bspline" when df is not supplied.
df	Positive integer with the number of basis functions used by splines::bs when type = "bspline". Required when knots is not supplied. Mutually exclusive with knots.

`boundary_knots` Numeric vector of length two giving the univariate boundary knots of the B-spline expansion. Required for `type = "bspline"` when the basis will be wired through the Stan-side fit path (Path 1, sub-phase 8.3.8): the Cox–de Boor recursion in Stan needs a fixed boundary, while `splines::bs` would otherwise infer them from each evaluation point and produce knot vectors that drift between Hamiltonian Monte Carlo steps. Must satisfy `boundary_knots[1] < min(knots)` and `boundary_knots[2] > max(knots)` when `knots` is given. Ignored for `type != "bspline"`.

`basis_fn` A function that takes a numeric vector of length `p` (the value of theta_ref) and returns a numeric vector of length equal to the dimension of the basis. Required for `type = "user"`; ignored otherwise. The function must be pure (no side effects) and must return the same dimension for any input of length `p`.

`dim` Positive integer giving the dimension of the basis. Required for `type = "user"`; computed automatically for `type = "polynomial"` and `type = "bspline"`.

`p` Optional positive integer giving the dimension of `theta_ref`. When supplied, the basis is eagerly materialized at construction time: the package probes the evaluator, records `dim`, and computes per-coordinate block indices (`block_indices`) for separable types. When `NULL` (default), materialization is deferred to `materialize_W_basis()`.

Details

For $p > 1$, the package-provided basis types (`"polynomial"` and `"bspline"`) implement the *separable* case: each coordinate of θ_{ref} contributes an independent block of basis functions, concatenated in dimension order. Non-separable (cross-coupling) bases are planned for a future release together with the Gaussian-process extension; the current API is forward-compatible.

Unlike the additive component $a(x)$ and the multiplicative component $b(x)$, which are defined on the covariate space $\mathcal{X}$ and are declared via standard R formulas, the modulating component $W(\theta)$ is defined on the parameter space Θ. The user therefore declares its basis functionally rather than via `model.matrix`: the basis describes how W depends on the value of theta_ref, not on the observed covariates.

Given a value of theta_ref of length p, the basis evaluator returns a numeric vector of length `dim` encoding the basis functions at that point. The Stan code generator combines these values with the basis coefficients (one matrix of size $p \times d$ per basis function) to assemble $W(\theta)$ at every Hamiltonian Monte Carlo step.

For `type = "polynomial"`, the basis includes monomials of degrees 1 through `degree` of every coordinate of theta_ref, arranged **block-by-coordinate**: for $p > 1$ the output is $(\theta_1, \theta_1^2, \ldots, \theta_1^{\text{degree}}, \theta_2, \theta_2^2, \ldots, \theta_p^{\text{degree}})$. Cross-terms between coordinates are excluded by default to keep the basis dimension manageable; they can be added by supplying a user-defined basis with `type = "user"`.

For `type = "bspline"`, the basis is a B-spline expansion of a single coordinate of theta_ref via `splines::bs`. When theta_ref has more than one coordinate, the B-spline basis is applied to each coordinate independently and **concatenated block-by-coordinate**, matching the polynomial convention.

Value

An object of class `W_basis` with components `type`, `degree`, `knots`, `df`, `boundary_knots`, `dim`, `evaluator`, `p`, and `block_indices`. `evaluator` is a function that maps a numeric value of theta_ref

(length p) to a numeric vector of length dim. block_indices, when populated, is a list of length p whose k-th entry contains the row indices of the basis output corresponding to coordinate k of theta_ref; it is NULL for type = "user" (separability of user bases cannot be inferred automatically).

Methodological notes

The anchor point θ_0 at which $W(\theta_0) = 0$ is a parametrization device, not an inferential statement about the posterior of theta_ref. Choosing a different anchor changes the parametrization but not the data-generating model, provided the basis spans the same function space. See assumption (C4) of Block 1 and the anchoring discussion in Section 6.7 (Theorem 1E).

Identifiability of W as a function on Θ requires the prior on theta_ref to assign positive measure to a connected open subset of Θ (assumption (BAY-1) of Theorem 1E). The basis declared here is the finite-dimensional ambient space; the prior over the basis coefficients is configured via gdpar_prior.

Dependencies

This function uses bs from the **splines** package (a base R package) when type = "bspline".

References

See vignette("v01_amm_identifiability", package = "gdpar"), Section 6.7 (Theorem 1E) for the identifiability of W as a function on the prior support.

See Also

amm_spec, gdpar_prior, as_per_k

Examples

```
wb_poly <- W_basis(type = "polynomial", degree = 2)
print(wb_poly)
wb_poly$evaluator(0.5)

wb_poly_mv <- W_basis(type = "polynomial", degree = 2, p = 2L)
print(wb_poly_mv)
wb_poly_mv$block_indices

wb_user <- W_basis(
  type     = "user",
  basis_fn = function(theta) c(theta, theta^2, sin(theta)),
  dim      = 3
)
wb_user$evaluator(0.5)
```

[.gdpar_formula_set *Subset a gdpar_formula_set by slot name or position*

Description

Subset a gdpar_formula_set by slot name or position

Usage

```
## S3 method for class 'gdpar_formula_set'
x[i]
```

Arguments

x	An object of class gdpar_formula_set.
i	A character vector of slot names or an integer vector of positions.

Value

A named list of formulas (not a gdpar_formula_set; downstream re-validation requires reconstruction via gdpar_formula_set).

[[.gdpar_formula_set *Extract a single formula from a gdpar_formula_set*

Description

Extract a single formula from a gdpar_formula_set

Usage

```
## S3 method for class 'gdpar_formula_set'
x[[i]]
```

Arguments

x	An object of class gdpar_formula_set.
i	A character scalar with the slot name or an integer index.

Value

The formula stored at slot i.

Index