

Package ‘proxymix’

October 8, 2026

Type Package

Title Kullback-Leibler Optimal Gaussian Mixture Proxies for Target Densities

Version 0.16.0

Description Fits multivariate Gaussian-mixture proxies that are Kullback-Leibler optimal to user-supplied target densities on real Euclidean space. Three fitting regimes are unified under one verb: (i) closed-form moment matching for a single component, (ii) classical expectation-maximisation when independent samples are available, and (iii) importance-sampled expectation-maximisation that minimises the Kullback-Leibler divergence when the target can be evaluated point-wise but not (cheaply) sampled. Closed-form Gaussian-mixture operators (density, sampling, marginalisation, conditioning, divergence) round out the toolkit. The conditioning operator drives multiple imputation of data missing at random, covering the multimodal and heteroscedastic cases a single-Gaussian model cannot represent. Implements the regime hierarchy of van der Hoeek and Elliott (2024) [doi:10.1080/07362994.2024.2372605](https://doi.org/10.1080/07362994.2024.2372605).

License MIT + file LICENSE

URL <https://github.com/max578/proxymix>,
<https://max578.github.io/proxymix/>

BugReports <https://github.com/max578/proxymix/issues>

Encoding UTF-8

Language en-AU

Depends R (>= 4.3)

Imports S7, stats, mvnfast, cli, rlang, data.table, withr

Suggests mclust, mice, generics, ggplot2, viridis, patchwork, knitr, rmarkdown, testthat (>= 3.0.0), vdiff, spelling, covr

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true
Config/roxygen2/version 8.1.0
NeedsCompilation no
Author Max Moldovan [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9680-8474>),
 John van der Hoek [aut]
Maintainer Max Moldovan <max.moldovan@gmail.com>
Repository CRAN
Date/Publication 2026-10-08 18:30:14 UTC

Contents

as_mids	4
autoplot.gmm_fit	5
banana_target	6
bic_aic	7
dgmm	8
donut_target	9
epanechnikov_target	10
ess_summary	11
ess_trace	12
fit_em_samples	12
fit_kld_em	14
fit_moment_match	18
fit_proxymix	19
fit_uplift	20
from_kde	22
from_objective	24
glance.gmm_fit	26
gmm	27
gmm_affine	28
gmm_aggregate	29
gmm_anneal_path	30
gmm_canonicalise	32
gmm_cf_mean	33
gmm_cf_tail_prob	33
gmm_cf_variance	34
gmm_complete	35
gmm_conditionalise	36
gmm_conditional_entropy	37
gmm_convolve	38
gmm_counterfactual	39
gmm_counterfactual_law	40
gmm_dim	41
gmm_divergence	42
gmm_entropy	43

gmm_eos_test	44
gmm_evidence	45
gmm_filter	47
gmm_fit	49
gmm_fit_ensemble	50
gmm_fit_quality	51
gmm_imputation	52
gmm_impute	54
gmm_independence_graph	55
gmm_intervene	57
gmm_kld	58
gmm_marginalise	59
gmm_mean	60
gmm_missing	61
gmm_mix	62
gmm_modes	62
gmm_mutual_information	63
gmm_n_components	64
gmm_observe	65
gmm_product	66
gmm_reduce	67
gmm_target	69
gmm_target_from_posterior	71
gmm_target_from_samples	72
gmm_weights	73
hellinger_mc	74
init_kmeans	75
init_moment_seed	75
init_random	76
init_warm_start	77
is_mvn	78
is_mvt	78
is_proposal	79
is_uniform	80
kld_trace	81
maxent_target	81
mechanism	83
mixture_target	84
multi_start_best_of	85
pgmm	86
proposal_uniform	87
proxy_cate	88
proxy_confounding_gap	89
proxy_decide	90
proxy_fmi	92
proxy_functional_ci	92
proxy_identification_report	94
proxy_mnar_sensitivity	95

proxy_overlap	96
proxy_policy_value	97
proxy_pool	98
proxy_predict	99
proxy_regime_segments	100
proxy_retrospective_uplift	101
proxy_uplift	102
rgmm	103
select_N	104
tidy.gmm	105
uplift_identification	105
uplift_model	106
Index	108

as_mids	<i>Convert imputations to a mice multiply-imputed dataset</i>
---------	---

Description

Packages a [gmm_imputation](#) as a `mice::mids` object so that an arbitrary model estimand can be fitted and pooled with the established mice workflow. The joint Gaussian-mixture imputations – including the multimodal and heteroscedastic shapes a univariate imputer cannot produce – flow through unchanged; mice supplies `with()`, `mice::pool()`, and the pooled diagnostics.

Usage

`as_mids(object)`

Arguments

`object` A [gmm_imputation](#).

Value

A `mice::mids` object with `m` imputations.

See Also

[gmm_impute\(\)](#), [proxy_pool\(\)](#) for the closed-form column-mean pooling.
Other imputation: [gmm_complete\(\)](#), [gmm_imputation\(\)](#), [gmm_impute\(\)](#), [mechanism](#), [proxy_fmi\(\)](#), [proxy_mnar_sensitivity\(\)](#), [proxy_pool\(\)](#)

Examples

```
set.seed(1)
x1 <- rnorm(150); x2 <- x1 + rnorm(150)
x2[runif(150) < 0.3] <- NA
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
if (requireNamespace("mice", quietly = TRUE)) {
  fit <- with(as_mids(imp), lm(x2 ~ x1))
  summary(mice::pool(fit))
}
```

autoplot.gmm_fit

Plot a fitted Gaussian-mixture proxy

Description

An autoplot() method for [gmm_fit](#) objects, rendering the fitted mixture with ggplot2. The displayed coordinates are reduced to the requested one or two dimensions through the closed-form marginal [gmm_marginalise\(\)](#), so the method works for a proxy of any ambient dimension p .

A one-dimensional request draws the marginal mixture density, optionally with the per-component densities underneath and a rug of the target's samples. A two-dimensional request draws the marginal density as a viridis raster with white contour lines, optionally overlaying each component's mean and a probability-contour ellipse.

Details

The method is registered against the ggplot2::autoplot() generic only when ggplot2 is installed; call it as ggplot2::autoplot(fit) or load ggplot2 first. It returns the ggplot object, so the usual + layering applies for further customisation.

Value

A ggplot object.

Arguments

Called as ggplot2::autoplot(object, dims = c(1L, 2L), n_grid = 120L, n_sd = 3.5, level = 0.95, show_components = TRUE, show_data = TRUE, ...).

- object: a [gmm_fit](#), typically from [fit_proxymix\(\)](#).
- dims: integer vector of length one or two giving the coordinate(s) to display, in 1:p. Defaults to the first two coordinates (or the only coordinate when $p == 1$).
- n_grid: integer scalar, the number of grid points per axis at which the density is evaluated. A two-dimensional plot evaluates n_grid^2 points.
- n_sd: numeric scalar, how many component standard deviations beyond the extreme component means the plotting window extends.
- level: numeric scalar in $(0, 1)$, the probability level of the per-component ellipse drawn on a two-dimensional plot.

- `show_components`: logical scalar, whether to overlay the per-component densities (one dimension) or mean-and-ellipse glyphs (two dimensions).
- `show_data`: logical scalar, whether to overlay the target's samples, when the fitted target carries any.
- `...`: ignored, present for generic compatibility.

See Also

Other classes: `glance.gmm_fit`, `gmm()`, `gmm_counterfactual_law()`, `gmm_dim()`, `gmm_fit()`, `gmm_n_components()`, `gmm_target()`, `gmm_weights()`, `is_proposal()`, `tidy.gmm`

Examples

```
samples <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(samples)
fit <- fit_proxymix(tgt, N = 2L, regime = "sample", max_iter = 25L)
ggplot2::autoplot(fit)
ggplot2::autoplot(fit, dims = 1L)
```

banana_target

Banana-shaped 2-D target

Description

A 2-D "banana" density obtained by warping an isotropic Gaussian through the map $(z_1, z_2) \mapsto (z_1, z_2 + \frac{1}{2}(z_1^2 - 1))$. The map has unit Jacobian, so the resulting density is exactly normalised.

Usage

```
banana_target(with_samples = FALSE, n = 2000L, seed = 1L)
```

Arguments

<code>with_samples</code>	If TRUE, attach <code>n</code> exact samples drawn by the change-of-variables trick. Default FALSE — the target then exposes only its <code>log_density</code> , which is the regime-(iii) use case.
<code>n</code>	Number of samples to attach when <code>with_samples = TRUE</code> .
<code>seed</code>	Optional integer seed used when drawing the samples.

Value

A `gmm_target` in dimension 2.

See Also

Other targets: `donut_target()`, `epanechnikov_target()`, `gmm_target_from_samples()`, `maxent_target()`, `mixture_target()`

Examples

```
b <- banana_target()
b
b@log_density(matrix(c(0, 0, 1, 0), ncol = 2, byrow = TRUE))
```

bic_aic
Information criteria: BIC, AIC, and ICL

Description

Returns the Bayesian and Akaike information criteria of a regime-(ii) fit, together with the integrated completed likelihood (ICL). All three are computed against the *empirical* log-likelihood of the samples used to fit the model and are reported on the same scale (smaller is better). They are NA for regimes that do not have an empirical likelihood ("moment", "kld").

Usage

```
bic_aic(fit)
```

Arguments

`fit` A [gmm_fit](#).

Details

The ICL of Biernacki, Celeux and Govaert (2000) adds to the BIC twice the entropy of the fitted classification, $ICL = BIC + 2E_N$, where $E_N = -\sum_{i,k} \gamma_{ik} \log \gamma_{ik} \geq 0$ is the entropy of the responsibilities γ_{ik} . It therefore penalises mixtures whose components overlap (uncertain assignments), and favours well-separated clustering solutions over the merely best-fitting ones. Because $E_N \geq 0$, the ICL is never smaller than the BIC, and the two coincide for a single component ($K = 1$), where every responsibility is one. The classification entropy itself is returned as `classification_entropy`.

Value

A list with `bic`, `aic`, `icl`, `classification_entropy`, and `n_params`.

References

Biernacki, C., Celeux, G. and Govaert, G. (2000) Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(7), 719–725. doi:[10.1109/34.865189](https://doi.org/10.1109/34.865189)

See Also

Other diagnostics: [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(x)
fit <- fit_proxymix(tgt, N = 2L, regime = "sample", max_iter = 25L)
bic_aic(fit)
```

dgmm

Density of a Gaussian mixture

Description

Evaluates the density (or log-density) of a Gaussian mixture at one or more points.

Usage

```
dgmm(x, g, log = FALSE)
```

Arguments

x	A numeric matrix with one observation per row, or a length-p numeric vector (treated as a single observation).
g	A gmm (or gmm_fit) object.
log	Logical. If TRUE, return log-densities.

Value

A numeric vector of length `nrow(x)`.

See Also

Other ops: [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
          means = list(c(-1, 0), c(1, 0)),
          covariances = list(diag(2), diag(2)))
dgmm(c(0, 0), g)
dgmm(c(0, 0), g, log = TRUE)
```

donut_target	<i>Donut-shaped 2-D target</i>
--------------	--------------------------------

Description

A rotationally symmetric annulus on $\mathbb{R}^2$, with density

$$f(x) \propto \exp\left(-\frac{(\|x\| - r_0)^2}{2\sigma^2}\right).$$

Numerical integration in polar coordinates fixes the normaliser; the returned target exposes a normalised log_density.

Usage

```
donut_target(r0 = 2.5, sigma = 0.5, with_samples = FALSE, n = 2000L, seed = 1L)
```

Arguments

r0	Centre radius of the annulus.
sigma	Annulus width.
with_samples	If TRUE, attach n exact samples via polar change-of-variables and a one-dimensional rejection step.
n	Number of samples to attach when with_samples = TRUE.
seed	Optional integer seed used when drawing the samples.

Value

A [gmm_target](#) in dimension 2.

See Also

Other targets: [banana_target\(\)](#), [epanechnikov_target\(\)](#), [gmm_target_from_samples\(\)](#), [maxent_target\(\)](#), [mixture_target\(\)](#)

Examples

```
d <- donut_target()
d
```

epanechnikov_target *Compact-support Epanechnikov target*

Description

The canonical compact-support target: a product of one-dimensional Epanechnikov densities $K(u) = \frac{3}{4}(1 - u^2) \mathbf{1}_{|u| \leq 1}$, rescaled to $[\text{center} - \text{half_width}, \text{center} + \text{half_width}]$ in each coordinate. No mixture of full-support Gaussians can have compact support, so this target is the clean case where regime (iii) is the only viable fitting route. It declares its support, which makes [fit_kld_em\(\)](#) (and [fit_proxymix\(\)](#) with `regime = "kld"`) select a support-matched uniform proposal automatically instead of the default multivariate-t, which would place importance mass where the log-density is $-\text{Inf}$.

Usage

```
epanechnikov_target(
  n_dim = 1L,
  center = 0,
  half_width = 1,
  with_samples = FALSE,
  n = 2000L,
  seed = 1L
)
```

Arguments

<code>n_dim</code>	Ambient dimension p .
<code>center</code>	Length-1 (recycled) or length- p numeric centre per coordinate.
<code>half_width</code>	Length-1 (recycled) or length- p positive numeric half-width per coordinate.
<code>with_samples</code>	If TRUE, attach n exact samples drawn by the Devroye (1986) three-uniform method. Default FALSE.
<code>n</code>	Number of samples to attach when <code>with_samples = TRUE</code> .
<code>seed</code>	Optional integer seed used when drawing the samples.

Value

A [gmm_target](#) in dimension `n_dim` with a declared compact support.

See Also

Other targets: [banana_target\(\)](#), [donut_target\(\)](#), [gmm_target_from_samples\(\)](#), [maxent_target\(\)](#), [mixture_target\(\)](#)

Examples

```
e <- epanechnikov_target()
e
e@log_density(matrix(c(0, 0.5, 1.5), ncol = 1L)) # finite, finite, -Inf
```

ess_summary

Summary of importance-sampling diagnostics

Description

Convenience accessor returning the headline IS-quality numbers for a regime-(iii) fit: effective sample size and its ratio to `is_size`, the largest self-normalised weight, the support fraction (proportion of draws that received a finite weight), and the Monte-Carlo standard error of the final KLD estimate. Returns NA fields for regimes that do not use importance sampling.

Usage

```
ess_summary(fit)
```

Arguments

`fit` A [gmm_fit](#).

Details

Validation-side numbers (`validation_*`) are populated only when the fit was called with `validation_size > 0`.

Value

A list of numeric scalars (or NAs where not applicable), plus a logical `support_truncated` flag that is TRUE when the fit's auto-selected proposal replaced an unbounded target-support end with a finite, data-derived working box (so `support_fraction` reflects only that finite region and the target tail beyond it was not sampled). The finite box itself is recorded in `fit@diagnostics$working_bounds`.

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 3L, regime = "kld",
                  is_size = 1500L, max_iter = 20L, seed = 1L,
                  validation_size = 1500L)
ess_summary(fit)
```

ess_trace

Effective sample size of the importance-sampling weights

Description

Returns the effective sample size ($1 / \sum(W^2)$) of the self-normalised importance weights used by a regime-(iii) fit. NA for regimes that do not use importance sampling.

Usage

```
ess_trace(fit)
```

Arguments

fit A [gmm_fit](#).

Value

Numeric scalar (or NA_real_).

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                   is_size = 1000L, max_iter = 15L, seed = 1L)
ess_trace(fit)
```

fit_em_samples

Classical EM fit on samples

Description

Implements regime (ii) of Hoek and Elliott (2024). Runs the textbook expectation-maximisation algorithm for Gaussian mixtures on the supplied samples, with diagonal ridge regularisation for numerical stability, optional multi-start, and monotone-log-likelihood checking.

Usage

```
fit_em_samples(
  target,
  N = 2L,
  init = NULL,
  max_iter = 100L,
  tol = 1e-06,
  ridge_eps = 1e-06,
  n_starts = 5L,
  anneal = FALSE,
  temp_schedule = NULL,
  seed = NULL,
  canonicalise = TRUE
)
```

Arguments

target	A gmm_target carrying an n by p samples matrix.
N	Number of mixture components.
init	A gmm initialisation, or NULL to use init_kmeans() .
max_iter	Maximum number of EM iterations.
tol	Relative-log-likelihood convergence tolerance.
ridge_eps	Relative ridge. At every M-step, ridge_eps times each coordinate's sample variance is added to that coordinate's diagonal entry of each component covariance.
n_starts	Number of multi-start initialisations (only when init is NULL). The best fit by final log-likelihood is returned.
anneal	Logical. If TRUE, a deterministic-annealing warm-start (see gmm_anneal_path()) replaces the multi-start: the components are annealed from a high temperature down to one, and the resulting parameters seed a single final (cold) EM polish. This attacks the local-optima sensitivity of cold EM at the cost of the schedule length. Defaults to FALSE (cold best-of-n_starts).
temp_schedule	Optional numeric vector of descending temperatures for the annealing warm-start. NULL (the default) uses a geometric schedule from 10 down to 1 in covariance-whitened units. Ignored when anneal = FALSE.
seed	Optional integer seed for the annealing perturbations (the warm-start is deterministic given a seed). Ignored when anneal = FALSE.
canonicalise	Logical. If TRUE (the default), the fitted mixture is post-processed by gmm_canonicalise() so that components are sorted by descending weight and (as a tiebreaker) by descending $ \mu $.

Value

A [gmm_fit](#) with regime = "sample". When anneal = TRUE the diagnostics list also carries annealed = TRUE and the temp_schedule used.

See Also

Other fitting: [fit_kld_em\(\)](#), [fit_moment_match\(\)](#), [from_kde\(\)](#), [from_objective\(\)](#), [select_N\(\)](#)

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(x)
fit <- fit_em_samples(tgt, N = 2L, max_iter = 30L, n_starts = 2L)
fit@diagnostics$loglik_final
```

fit_kld_em

Importance-sampled KLD-EM fit (regime iii)

Description

Implements regime (iii) of Hoek and Elliott (2024). Minimises $KL(f || g_{\theta})$ where f is supplied as an evaluable log-density on the target, via expectation-maximisation against importance-sampled draws from a user-chosen proposal q .

Usage

```
fit_kld_em(
  target,
  N = 3L,
  proposal = NULL,
  is_size = 5000L,
  init = NULL,
  max_iter = 100L,
  tol = 1e-05,
  ridge_eps = 1e-06,
  min_ess = 50,
  on_low_ess = c("warn", "abort"),
  seed = NULL,
  validation_size = NULL,
  validation_proposal = NULL,
  validation_seed = NULL,
  support_warn = TRUE,
  adapt = c("none", "pmc"),
  refresh_every = 5L,
  defensive_gamma = 0.15,
  inflate = 1.5,
  anneal = FALSE,
  temp_schedule = NULL,
  canonicalise = TRUE
)
```

Arguments

target	A gmm_target with a non-NULL log_density.
N	Number of mixture components.
proposal	An is_proposal . When NULL (the default) the proposal is chosen automatically: a support-matched is_uniform() when the target declares a bounded or one-sided support, otherwise a multivariate-t with $df = 5$ in <code>target@n_dim</code> dimensions. The automatic choice is announced with a one-line message so it is never silent.
is_size	Number of importance-sampling draws used for fitting.
init	A gmm initialisation, or NULL to use a kmeans pass on the importance-resampled draws.
max_iter	Maximum number of EM iterations.
tol	Convergence tolerance on the relative change in the importance-weighted EM objective $Q(\theta) = \sum_n W_n \log g(x_n)$. Q is invariant to the target's normalising constant, so the stopping rule behaves identically for normalised and unnormalised targets (the importance-sampled KLD estimate carries an additive $\log Z(f)$ offset and is therefore never used for stopping).
ridge_eps	Relative ridge. At every M-step, <code>ridge_eps</code> times each coordinate's variance under the target is added to that coordinate's diagonal entry of each component covariance.
min_ess	Minimum effective sample size below which the fit is flagged as degenerate: a classed warning (<code>proxymix_low_ess</code>) is issued (or, with <code>on_low_ess = "abort"</code> , a classed error <code>proxymix_degenerate_fit</code>), the fit's converged flag is forced to FALSE, and <code>degenerate = TRUE</code> is recorded in the diagnostics and the quality certificate.
on_low_ess	What to do when the effective sample size falls below <code>min_ess</code> : "warn" (the default) flags and continues, "abort" refuses to return a degenerate fit.
seed	Optional integer seed. When supplied, the fit is reproducible end-to-end: the fitting IS draw, the initialisation resample and kmeans pass, and any empty-component reseed draws are all derived from it. When NULL, those draws consume the ambient random-number stream.
validation_size	Number of independent importance-sampling draws for the held-out estimate <code>validation_kld</code> and its standard error <code>validation_mc_se</code> . The standard error is roughly $\sqrt{2 * KL / ESS}$, where ESS is the effective sample size of the validation draws, so a close fit needs many draws before the estimate is precise. The default NULL draws batches of <code>is_size</code> until the standard error is at most a tenth of the estimate (at most 0.001 when the estimate is below 0.01), stopping at $10 * is_size$ draws. When the target's normalising constant is unknown, the goal is a standard error of 0.001. A number draws exactly that many; 0L skips validation.
validation_proposal	Optional is_proposal for the validation sample. Defaults to the same proposal used for fitting.

validation_seed	Optional integer seed used when drawing the validation sample. Defaults to seed + 1L when seed is supplied, NULL otherwise.
support_warn	Logical. If TRUE (the default), issue a warning when more than 5% of IS draws receive non-finite weights (typically because the proposal does not dominate the target's support).
adapt	Proposal adaptation: "none" (the default; one fixed IS draw, the historical behaviour) or "pmc" (population-Monte-Carlo refresh of the proposal from the current iterate; see Details).
refresh_every	With adapt = "pmc", refresh the proposal after this many EM iterations on the current batch. Default 5L.
defensive_gamma	With adapt = "pmc", the mass kept on the original proposal as a heavy-tailed defensive anchor at every refresh (bounds the importance-weight variance). Default 0.15.
inflate	With adapt = "pmc", the factor inflating the current iterate's covariances inside the refreshed proposal. Default 1.5.
anneal	Logical. If TRUE, a deterministic-annealing warm-start (see gmm_anneal_path()) replaces the kmeans initialisation: components are annealed from a high temperature down to one on the importance-weighted draws, and the resulting parameters seed the (unchanged) cold KLD-EM loop. This attacks the local-optima sensitivity of cold EM. Defaults to FALSE.
temp_schedule	Optional numeric vector of descending temperatures for the annealing warm-start. NULL (the default) uses a geometric schedule from 10 down to 1 in covariance-whitened units. Ignored when anneal = FALSE.
canonicalise	Logical. If TRUE (the default), the fitted mixture is post-processed by gmm_canonicalise() .

Details

With adapt = "none" (the default) the Monte Carlo draws from q are computed once at the start and the resulting self-normalised importance-sampling weights are reused at every EM iteration. With adapt = "pmc" the proposal is refreshed every refresh_every iterations with a defensive mixture built from the current iterate – the population-Monte-Carlo scheme: the fitted mixture (covariances inflated by inflate) carries $1 - \text{defensive_gamma}$ of the proposal mass and the original proposal q keeps defensive_gamma as a heavy-tailed anchor, a fresh IS batch is drawn, and EM continues on the refreshed weights. Because the refreshed proposal tracks the target, the effective sample size recovers from a poor initial proposal and the usable dimension range extends well beyond what a fixed proposal reaches; the per-batch ESS trace is reported as `diagnostics$ess_history`. While a batch is degenerate (its effective sample size is below `min_ess`), the refresh fires every iteration with an escalating covariance inflation floored at a growing fraction of the batch's sample covariance, so a collapsed iterate walks back out toward the target instead of freezing; and convergence is only accepted on an adapted batch, so a run that stabilises on the original proposal's draw is refreshed at least once before it is allowed to stop. The scheme is the mixture population-Monte-Carlo idea of Cappé et al. (2008) with the defensive-mixture safeguard of Owen and Zhou (2000); it re-draws rather than recycles batches (compare the adaptive multiple importance sampling of Cornuet et al., 2012).

kld_final is estimated on the same importance draws the fit was tuned to, so it reads low and can come out negative, more so for large N and small is_size. When validation_size > 0 the function also draws an *independent* validation IS sample and reports its KLD estimate (validation_kld), effective sample size, and largest weight share; validation_kld is the estimate to quote as the fit's accuracy.

When the target's normalised property is FALSE or NA, the importance-sampled kld_final, kld_trace and validation_kld estimate $KL(f||g) + \log Z(f)$, where $Z(f)$ is the integral of $\exp(\log_density)$. The fit's diagnostics list records this via kld_is_shifted = TRUE and a kld_shift_explanation string. When the target also supplies a finite log_normalizer, the fit subtracts it and reports the KL divergence as kld_final_absolute and validation_kld_absolute.

Value

A `gmm_fit` with regime = "kld". The diagnostics list contains, among others, kld_trace, kld_final (in-sample), kld_is_shifted, kld_final_absolute (when computable), ess, ess_relative (ess / is_size), max_weight, support_fraction, mc_se_kld, validation_kld (held-out), validation_ess, validation_max_weight, and heldout_kld, the held-out estimate of the KL divergence that `gmm_fit_quality()` describes.

References

- Cappé, O., Douc, R., Guillin, A., Marin, J.-M. and Robert, C. P. (2008) Adaptive importance sampling in general mixture classes. *Statistics and Computing* 18, 447–459. doi:10.1007/s11222008-9059x
- Cornuet, J.-M., Marin, J.-M., Mira, A. and Robert, C. P. (2012) Adaptive multiple importance sampling. *Scandinavian Journal of Statistics* 39, 798–812. doi:10.1111/j.14679469.2011.00756.x
- Owen, A. and Zhou, Y. (2000) Safe and effective importance sampling. *Journal of the American Statistical Association* 95(449), 135–143. doi:10.1080/01621459.2000.10473909

See Also

Other fitting: `fit_em_samples()`, `fit_moment_match()`, `from_kde()`, `from_objective()`, `select_N()`

Examples

```
tgt <- banana_target()
q <- is_mvt(n_dim = 2L, mean = c(0, 0),
            sigma = 4 * diag(2), df = 5)
fit <- fit_kld_em(tgt, N = 3L, proposal = q,
                 is_size = 1500L, max_iter = 25L, seed = 1L,
                 validation_size = 1500L)
fit@diagnostics$kld_final
fit@diagnostics$validation_kld
```

fit_moment_match	<i>Closed-form moment-matching fit</i>
------------------	--

Description

Implements regime (i) of Hoek and Elliott (2024). When $N == 1$, this is the exact moment match: μ is the target mean and Σ is the target covariance. When $N > 1$, the function returns the deterministic moment-seed of `init_moment_seed()` wrapped as a `gmm_fit`, without iterative refinement — useful as a starting point for the iterative regimes.

Usage

```
fit_moment_match(target, N = 1L, ridge_eps = 1e-06, canonicalise = TRUE)
```

Arguments

target	A <code>gmm_target</code> .
N	Number of components. $N \geq 2$ returns a moment-seeded mixture without iterative refinement.
ridge_eps	Ridge added to the empirical covariance for numerical stability.
canonicalise	Logical. If TRUE (the default), the fitted mixture is post-processed by <code>gmm_canonicalise()</code> so that components are sorted by descending weight and (as a tiebreaker) by descending $\ \mu\ $. Set FALSE to retain the raw component order.

Details

Either the target must carry an n by p samples matrix, or its metadata slot must contain pre-computed moments of the form `list(mean = <p-vec>, cov = <p-by-p>)`.

Value

A `gmm_fit` with `regime = "moment"`.

See Also

Other fitting: `fit_em_samples()`, `fit_kld_em()`, `from_kde()`, `from_objective()`, `select_N()`

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(x)
fit_moment_match(tgt, N = 1L)
```

fit_proxymix

*Fit a Gaussian-mixture proxy to a target density***Description**

The unified front door of proxymix. Picks a fitting regime (or honours an explicit choice) and dispatches to the corresponding regime-specific fitter:

Usage

```
fit_proxymix(
  target,
  N = 1L,
  regime = c("auto", "moment", "sample", "kld"),
  ...
)
```

Arguments

target	A gmm_target .
N	Number of components.
regime	One of "auto", "moment", "sample", "kld".
...	Additional arguments forwarded to the regime-specific fitter. The most useful pass-throughs are canonicalise (whether to apply gmm_canonicalise() to the returned fit; default TRUE), validation_size and validation_proposal (held-out IS validation for regime "kld"), max_iter, tol, n_starts, and seed.

Details

- "moment" - closed-form moment matching ([fit_moment_match\(\)](#)).
- "sample" - classical EM on i.i.d. samples ([fit_em_samples\(\)](#)).
- "kld" - importance-sampled KLD-EM ([fit_kld_em\(\)](#)) for a target that can be evaluated but not sampled.

With regime = "auto" the choice is made from the shape of the supplied target:

- N == 1 and the target carries samples or moments: "moment".
- N >= 2 and the target carries samples: "sample".
- The target carries log_density only (no samples): "kld".

Value

A [gmm_fit](#).

Examples

```
## auto: samples + N=2 -> classical EM.
x <- matrix(stats::rnorm(200), ncol = 2)
tgt_s <- gmm_target_from_samples(x)
fit_proxymix(tgt_s, N = 2L, max_iter = 25L)

## explicit "kld" on a log-density-only target.
fit_proxymix(banana_target(), N = 3L, regime = "kld",
             is_size = 1000L, max_iter = 20L, seed = 1L)
```

fit_uplift

Fit an uplift / next-best-action model from a data frame

Description

Assembles a joint Gaussian-mixture proxy over the outcome, the treatment, and the covariates, and returns an [uplift_model](#) that the decision verbs read in closed form. One fit yields prediction, heterogeneous treatment effects, optimal actions, off-line policy value, and an identification audit – see [proxy_cate\(\)](#), [proxy_decide\(\)](#), [proxy_policy_value\(\)](#) and [proxy_identification_report\(\)](#).

Usage

```
fit_uplift(
  data,
  outcome,
  treatment,
  covariates,
  N = "auto",
  regime = "auto",
  assume = c("ignorability", "latent_confounder"),
  outcome_type = c("continuous", "binary", "count"),
  n_grid = 1:4,
  seed = NULL,
  ...
)
```

Arguments

data	A data frame holding the outcome, treatment and covariate columns.
outcome	A single column name – the outcome Y.
treatment	A single column name – the binary treatment T.
covariates	A character vector of one or more column names – the pre-treatment covariates X.
N	Number of mixture components, or "auto" (the default) to select by BIC over n_grid.

regime	One of "auto", "moment", "sample", "kld", forwarded to fit_proxymix() . The default "auto" uses classical EM on the supplied rows.
assume	One of "ignorability" (the default) or "latent_confounder" – the identification regime the effects are read under. See proxy_cate() and proxy_confounding_gap() .
outcome_type	One of "continuous" (the default), "binary" or "count". Effects are reported on the response scale via a discretised predictive for the non-continuous types; see proxy_cate() .
n_grid	Integer vector of candidate component counts used when N = "auto". Default 1:4.
seed	Optional integer. When supplied, the fitting (including any random EM starts) runs under a fixed seed and the global RNG state is restored on exit, so the fit is reproducible without disturbing the caller's stream.
...	Additional arguments forwarded to fit_proxymix() (e.g. max_iter, n_starts, ridge_eps).

Details

The component count N may be fixed or chosen automatically. With N = "auto" the function sweeps n_grid and keeps the K that minimises the joint BIC; the full BIC trace is stored in the model's metadata. The treatment is binary at this version ({t0, t1}); a continuous dose is a future extension.

Value

An [uplift_model](#).

See Also

[proxy_cate\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#)

Other decision: [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 400L
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.5)
y <- 1 + 0.5 * x + (1 + x) * t + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, outcome = "y", treatment = "t", covariates = "x",
               N = 2L, regime = "sample", max_iter = 50L, seed = 1L)
m
```

from_kde

Compile a kernel-density estimate into a Gaussian-mixture proxy

Description

Fits an N-component Gaussian-mixture proxy to a (Gaussian, diagonal- bandwidth) kernel-density estimate over samples, via regime (iii) KLD-EM.

Usage

```
from_kde(
  samples,
  N = 3L,
  bandwidth = "silverman",
  proposal = NULL,
  is_size = NULL,
  max_iter = 100L,
  tol = 1e-05,
  ridge_eps = 1e-06,
  min_ess = 50L,
  seed = NULL,
  validation_size = NULL,
  validation_proposal = NULL,
  validation_seed = NULL,
  support_warn = TRUE,
  canonicalise = TRUE
)
```

Arguments

samples	An n by p numeric matrix of points. $n \geq 5$, $p \leq 10$.
N	Number of mixture components in the proxy.
bandwidth	Either "silverman", "scott", a positive numeric scalar (absolute bandwidth applied to every coordinate), or a length-p positive numeric vector of per-coordinate absolute bandwidths. Default "silverman".
proposal	Optional is_proposal . Default is a multivariate-t centred at <code>colMeans(samples)</code> , <code>scale = ridge(cov(samples)) + diag(h^2)</code> , <code>df = 5</code> .
is_size	Importance-sample size for fitting. The default NULL uses $10000 + 2500 * N$: the fitted components overfit a fixed draw, and larger N needs more draws to keep the held-out divergence close to its large-sample value.
max_iter	Maximum EM iterations. Forwarded to fit_kld_em() .
tol	Convergence tolerance. Forwarded to fit_kld_em() .
ridge_eps	Ridge added to each component covariance at every M-step. Forwarded to fit_kld_em() .

min_ess	Minimum effective sample size below which a warning is issued. Forwarded to fit_kld_em() .
seed	Optional integer seed for the fitting IS draw.
validation_size	Held-out IS sample size. The default NULL adds draws until validation_kld is precise, as described in fit_kld_em() ; 0L disables it. Forwarded to fit_kld_em() .
validation_proposal	Optional is_proposal for the held-out sample. Forwarded to fit_kld_em() .
validation_seed	Optional integer seed for the held-out IS draw. Forwarded to fit_kld_em() .
support_warn	Logical. Forwarded to fit_kld_em() .
canonicalise	Logical. If TRUE, the fitted mixture is post-processed by gmm_canonicalise() . Forwarded to fit_kld_em() .

Details

A Gaussian-kernel KDE on n points is already an n -component Gaussian mixture with equal weights, so it can be marginalised, conditioned and sampled in closed form as it stands. `from_kde()` is a **compression**: it replaces the n components with the closest N -component mixture in the Kullback-Leibler sense, with N much smaller than n . Density evaluation, storage and the operator calculus then cost in proportion to N rather than n (products and convolutions of two mixtures have as many components as the product of their counts). Bias inherited from the KDE is reproduced in the proxy; the bandwidth controls the bias-variance trade-off.

Dimensional scope. The dimensional guard is $p \leq 5$ (recommended), $p \leq 10$ (allowed with warning), $p > 10$ (rejected). Regime-(iii) KLD-EM is driven by importance sampling, whose effective sample size collapses sharply in high dimensions.

Value

A [gmm_fit](#) with `regime = "kld"` and metadata recording the KDE inputs (`kde_samples_n`, `bandwidth`, `bandwidth_method`). Its in-sample `kld_final` reads low, often below zero; the held-out `validation_kld` (see [ess_summary\(\)](#)) measures the compression error.

See Also

Other fitting: [fit_em_samples\(\)](#), [fit_kld_em\(\)](#), [fit_moment_match\(\)](#), [from_objective\(\)](#), [select_N\(\)](#)

Examples

```
set.seed(1L)
x <- rbind(
  mvnfast::rmvn(120L, mu = c(-2, 0), sigma = diag(2)),
  mvnfast::rmvn(120L, mu = c( 2, 0), sigma = diag(2))
)
fit <- from_kde(x, N = 2L, is_size = 2000L, max_iter = 40L, seed = 1L)
fit
ess_summary(fit)
```

from_objective

Map the optima of an objective with a Gaussian-mixture proxy

Description

Fits a Gaussian-mixture proxy to the Gibbs measure $\exp(-f(x)/T)$ of a user-supplied objective f over a bounded box, by cooling a short temperature ladder through regime-(iii) importance-sampled KLD-EM ([fit_kld_em\(\)](#)). As the temperature falls the mixture mass concentrates on the low regions of f , so the fitted mixture is a closed-form *map* over the optima rather than a single point estimate. Pair it with [gmm_modes\(\)](#) to read off the distinct optima.

Usage

```
from_objective(
  objective,
  lower,
  upper,
  N = NULL,
  minimise = TRUE,
  temperature = NULL,
  n_steps = 6L,
  exploration = 0.5,
  inflate = 1.8,
  is_size = 10000L,
  max_iter = 70L,
  ridge_eps = 1e-04,
  seed = NULL,
  scale = c("objective", "loglik")
)
```

Arguments

objective	Function taking a length- p numeric vector and returning a finite numeric scalar – the objective to minimise (or maximise; see <code>minimise</code>).
lower, upper	Numeric vectors of equal length p giving the box over which the optima are sought. Every upper must exceed its lower. <code>objective</code> is only evaluated inside the box; points outside it are treated like points where <code>objective</code> is not finite and receive a strongly unattractive value.
N	Number of mixture components. Default $\max(10L, 5L * p)$. Raise it for objectives with many optima or strong symmetry.
minimise	Logical. If TRUE (the default) lower objective values are better (the proxy concentrates on the minima); if FALSE the proxy concentrates on the maxima.
temperature	Optional control of the cooling ladder. NULL (the default) derives a ladder from the range r of objective over a uniform probe of the box: from $r / 2$ down to $r / 80$ when <code>scale = "objective"</code> , and from $\max(r / 2, 1)$ down to

	1 when <code>scale = "loglik"</code> . A positive scalar sets the final (lowest) temperature; a length-2 numeric <code>c(high, low)</code> sets both ends explicitly. A supplied temperature is used as given, whatever scale is.
<code>n_steps</code>	Number of temperatures in the cooling ladder. Default 6L. When <code>scale = "loglik"</code> , temperature is NULL and <code>n_steps</code> is not supplied, the ladder takes as many steps as it needs to cool by no more than the default factor of $40^{1/5} \approx 2.1$ per step, and never fewer than 6.
<code>exploration</code>	Probability mass the importance proposal places on uniform exploration of the box at each step, in $[0, 1]$. Default 0.5. Larger values explore more (and starve no basin); smaller values exploit the basins found so far.
<code>inflate</code>	Factor by which the current mixture covariances are inflated when used as the exploitation part of the proposal. Default 1.8.
<code>is_size</code>	Importance-sample size per cooling step. Default 1e4L.
<code>max_iter</code>	Maximum EM iterations per cooling step. Default 70L.
<code>ridge_eps</code>	Ridge added to each component covariance at every M-step. Forwarded to <code>fit_kld_em()</code> .
<code>seed</code>	Optional integer seed for reproducibility.
<code>scale</code>	What the objective's values mean, which sets the default temperature ladder. "objective" (the default) is any objective to optimise: the ladder scales with its range, so the result does not depend on the objective's units. "loglik" is a negative log-likelihood (or a log-likelihood with <code>minimise = FALSE</code>): the ladder ends at temperature 1, where the Gibbs measure is the likelihood itself.

Details

The Gibbs measure can be evaluated point-wise but not directly sampled, which is precisely the setting of regime (iii): minimising the Kullback-Leibler divergence from a Gaussian mixture to a peaked target is the rank-weighted Gaussian update at the heart of `fit_kld_em()`. This function is that fit, driven against a sequence of cooling Gibbs targets and warm-started from the previous fit at each step. Because a multimodal `f` produces a multimodal target, the components spread across the basins and recover the optima together.

Recovery is most reliable with component headroom – a number of components `N` comfortably larger than the number of optima you expect, so the defensive proposal can keep a component on each basin. Symmetric landscapes (where several optima are exchangeable) need the most headroom.

Dimensional scope. As with the rest of regime (iii), the importance- sampling effective sample size falls sharply with dimension; the guard is `p <= 5` (recommended), `p <= 10` (allowed with a warning), `p > 10` (rejected).

Effective sample size. Each cooling step records its importance-sampling effective sample size in the `from_objective` metadata. A classed `proxymix_low_ess` warning is raised at most once per call, and only when the final step, the one that produced the returned fit, falls below 50; its message gives the final effective sample size, how many steps were low, and the lowest value with its step.

Likelihoods. With `scale = "loglik"` the objective is read as a negative log-likelihood (or, with `minimise = FALSE`, a log-likelihood) and the ladder ends at temperature 1. The fitted map is then a proxy for the likelihood itself, restricted to the box, which is the posterior under a uniform prior on the box.

Value

A [gmm_fit](#) (the fitted proxy) carrying a `from_objective` metadata record with the box, the temperature ladder, scale, and the effective sample size at each cooling step (`ess`). Pass it to [gmm_modes\(\)](#) to extract the distinct optima.

See Also

[gmm_modes\(\)](#) to resolve the fitted map into distinct optima.

Other fitting: [fit_em_samples\(\)](#), [fit_kld_em\(\)](#), [fit_moment_match\(\)](#), [from_kde\(\)](#), [select_N\(\)](#)

Examples

```
## A bimodal 1-D objective with minima at +/- 2.
f <- function(v) (v[1]^2 - 4)^2
fit <- from_objective(f, lower = -5, upper = 5, N = 6L,
                    is_size = 2000L, n_steps = 5L, seed = 1L)
gmm_modes(fit)$modes

## The likelihood surface of a logistic-regression slope.
x <- c(-1.2, -0.6, -0.1, 0.3, 0.8, 1.4)
y <- c(0, 0, 1, 0, 1, 1)
nll <- function(b) sum(log1p(exp(b[1] * x))) - sum(y * b[1] * x)
lik <- from_objective(nll, lower = -5, upper = 10, N = 4L,
                    is_size = 2000L, seed = 1L, scale = "loglik")
lik@metadata$from_objective$temperatures
```

glance.gmm_fit

Glance at a fitted Gaussian-mixture proxy

Description

A broom-style `glance()` method: a one-row summary of a [gmm_fit](#) with the regime, the component count and dimension, convergence, iteration count, and the regime's headline fit statistics. Available as `generics::glance(fit)` when the `generics` package is installed.

Details

For a regime-(iii) fit, `kld_final` is estimated on the same importance draws the fit was tuned to, so it reads low and can be negative. `validation_kld` is the estimate on an independent draw and is the one to quote as the fit's accuracy; it is NA when the fit was made with `validation_size = 0`.

Value

A one-row data frame with columns `regime`, `n_components`, `dim`, `converged`, `iterations`, `ess`, `kld_final`, `validation_kld`, `loglik_final` and `bic` (NA where not applicable to the regime).

Arguments

Called as `generics::glance(x, ...)`, where `x` is a [gmm_fit](#) and `...` is ignored.

See Also

Other classes: `autoplot.gmm_fit`, `gmm()`, `gmm_counterfactual_law()`, `gmm_dim()`, `gmm_fit()`, `gmm_n_components()`, `gmm_target()`, `gmm_weights()`, `is_proposal()`, `tidy.gmm`

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                  is_size = 1000L, max_iter = 10L, seed = 1L)
generics::glance(fit)
```

gmm

*A Gaussian mixture***Description**

Lightweight S7 class representing an N-component multivariate Gaussian mixture on $\mathbb{R}^p$. Use `gmm()` to construct, `dgmm()` / `rgmm()` to evaluate or sample, and `gmm_marginalise()` / `gmm_conditionalise()` for closed-form operations.

Usage

```
gmm(
  weights = numeric(0),
  means = list(),
  covariances = list(),
  name = "gmm",
  metadata = list()
)
```

Arguments

<code>weights</code>	Numeric vector of length K, non-negative, summing to one.
<code>means</code>	List of length K, each element a length-p numeric vector.
<code>covariances</code>	List of length K, each element a p-by-p symmetric positive-definite numeric matrix.
<code>name</code>	Optional human-readable name.
<code>metadata</code>	Optional list of arbitrary metadata (regime tags, diagnostic snapshots, etc.).

Value

An S7 object inheriting from `gmm`.

See Also

Other classes: `autoplot.gmm_fit`, `glance.gmm_fit`, `gmm_counterfactual_law()`, `gmm_dim()`, `gmm_fit()`, `gmm_n_components()`, `gmm_target()`, `gmm_weights()`, `is_proposal()`, `tidy.gmm`

Examples

```
g <- gmm(
  weights = c(0.4, 0.6),
  means = list(c(-1, 0), c(1, 0)),
  covariances = list(diag(2), diag(2))
)
g
```

gmm_affine

*Affine pushforward of a Gaussian mixture***Description**

Returns the (closed-form) distribution of $Y = AX + b + \epsilon$ when $X \sim g$ is a Gaussian mixture and $\epsilon \sim \mathcal{N}(0, R)$ is independent additive Gaussian noise.

Usage

```
gmm_affine(g, A, b = 0, noise_cov = NULL, ridge_eps = 1e-06)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) in $\mathbb{R}^p$.
<code>A</code>	An m by p numeric matrix.
<code>b</code>	Numeric scalar or length- m vector. Default 0 .
<code>noise_cov</code>	m by m SPD numeric matrix, or <code>NULL</code> (treated as the zero matrix — a deterministic channel).
<code>ridge_eps</code>	Tiny ridge added to the output covariances for numerical hygiene. Set to 0 to disable.

Details

For each component k , the pushed-forward parameters are

$$\mu'_k = A\mu_k + b, \quad \Sigma'_k = A\Sigma_k A^\top + R,$$

and the mixture weights are unchanged. This is the finite-mixture analogue of a Kalman-style predict step.

The channel is required to be **affine** in x and the noise is required to be Gaussian. Non-linear channels are not closed form and are not silently approximated: push samples through the map instead (`rgmm()` then the transform) and refit with [fit_em_samples\(\)](#) when a mixture of the image is needed.

Value

A [gmm](#) in $\mathbb{R}^m$ with the same number of components and the same weights as `g`.

See Also

Other operators: [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
        means = list(c(-1, 0), c(1, 0)),
        covariances = list(diag(2), diag(2)))
A <- matrix(c(1, 0, 0, 1, 1, -1), nrow = 3L, byrow = TRUE)
gmm_affine(g, A, b = c(0, 0, 0), noise_cov = 0.01 * diag(3))
```

gmm_aggregate

*Aggregation pushforward of a Gaussian mixture***Description**

A named alias for [gmm_affine\(\)](#) when A is a (row-wise) aggregation matrix — e.g. a block-sum, block-average, or unequal-weight aggregation used in downscaling pipelines. The mathematics is identical to [gmm_affine\(\)](#); the alias gives the public API a clearer hook for aggregation-specific diagnostics in later releases.

Usage

```
gmm_aggregate(g, A, noise_cov = NULL, ridge_eps = 1e-06)
```

Arguments

g	A gmm (or gmm_fit) in R^p .
A	An m by p numeric matrix.
noise_cov	Optional m by m SPD numeric matrix. Default NULL (deterministic aggregation).
ridge_eps	Tiny ridge added to the output covariances for numerical hygiene.

Value

A [gmm](#) in R^m .

See Also

Other operators: [gmm_affine\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
          means = list(c(-1, 0, 1), c(1, 0, -1)),
          covariances = list(diag(3), diag(3)))
# Sum coordinates 1 and 2 into a single aggregate; pass coord 3 through.
A <- matrix(c(1, 1, 0,
              0, 0, 1), nrow = 2L, byrow = TRUE)
gmm_aggregate(g, A)
```

gmm_anneal_path

Phase-transition component discovery by deterministic annealing

Description

Tracks the number of distinct mixture centroids as a function of temperature under mass-constrained deterministic annealing (Rose, Gurewitz and Fox 1990), a physics-derived alternative to information-criterion model selection. The system starts at a high temperature where all `k_max` centroids collapse to the data centroid (a single effective component) and is cooled along a geometric schedule; at each critical temperature a centroid bifurcates, so the number of distinct centroids grows in steps. The temperatures at which it grows are the phase transitions, and the count occupying the widest temperature range is the discovered component number.

Usage

```
gmm_anneal_path(
  x,
  k_max = 8L,
  sigma = NULL,
  t_high = NULL,
  t_low = NULL,
  n_steps = 80L,
  n_inner = 30L,
  w = NULL,
  perturb = 0.02,
  merge_tol = 0.1,
  ridge_eps = 1e-06,
  seed = 1L
)
```

Arguments

<code>x</code>	A numeric <code>n</code> by <code>p</code> matrix of samples, or a gmm_target carrying a samples matrix. For regime (iii) targets, pass an importance-resampled draw.
<code>k_max</code>	Maximum number of centroids tracked (the discovered count is at most <code>k_max</code>).
<code>sigma</code>	Reference scale: the shared covariance is $\sigma^2 \cdot I$. When <code>NULL</code> (the default) <code>sigma</code> is 1, so the first critical temperature is the largest eigenvalue of the data covariance.

t_high, t_low	Top and bottom of the temperature schedule. When NULL they default to $3 * t_critical_analytic$ and $0.05 * t_critical_analytic$, bracketing the bifurcation cascade.
n_steps	Number of temperatures on the geometric schedule.
n_inner	Fixed-point iterations run at each temperature.
w	Optional length-n vector of non-negative observation weights (e.g. importance weights). Defaults to uniform.
perturb	Symmetry-breaking perturbation, as a fraction of the data scale, applied to the centroids at each temperature.
merge_tol	Two centroids count as distinct when their distance exceeds merge_tol times the data scale.
ridge_eps	Ridge added to the reference covariance for stability.
seed	Optional integer seed for the perturbations (the result is deterministic given a seed).

Details

The first bifurcation has a closed-form critical temperature $T_c = \lambda_{\max}(\Sigma^{-1}C)$, where C is the (weighted) data covariance and $\Sigma = \sigma^2 I$ the shared reference covariance. This value is returned as `t_critical_analytic` and serves as an independent analytic check on the empirically detected first transition. Subsequent transitions have no comparably simple closed form, and the count is a diagnostic rather than a guarantee.

Annealing fixes the component covariance to the reference Σ so the temperature is the only scale; this is the clean isotropic regime in which the critical temperature is exact. For robust *fitting* under free covariances, use `anneal = TRUE` on `fit_em_samples()` or `fit_kld_em()` instead.

Value

A list with elements `path` (a data frame of temperature, `n_effective` and `free_energy`), `critical_temperatures` (the temperatures at which the count increased), `first_critical_temperature` (the first such, or NA if none was detected), `t_critical_analytic` ($\lambda_{\max}(\Sigma^{-1}C)$), `k_selected` (the widest-plateau component count), `lambda_max` and `sigma`.

References

Rose, K., Gurewitz, E. and Fox, G. C. (1990) Statistical mechanics and phase transitions in clustering. *Physical Review Letters* 65(8), 945–948. doi:[10.1103/PhysRevLett.65.945](https://doi.org/10.1103/PhysRevLett.65.945)

See Also

Other diagnostics: `bic_aic()`, `ess_summary()`, `ess_trace()`, `gmm_conditional_entropy()`, `gmm_entropy()`, `gmm_evidence()`, `gmm_fit_ensemble()`, `gmm_fit_quality()`, `gmm_independence_graph()`, `gmm_mutual_information()`, `hellinger_mc()`, `kld_trace()`, `proxy_functional_ci()`

Examples

```
set.seed(1)
x <- rbind(
  matrix(stats::rnorm(120, mean = -4), ncol = 2),
  matrix(stats::rnorm(120, mean = 4), ncol = 2)
)
path <- gmm_anneal_path(x, k_max = 4L, n_steps = 40L)
path$k_selected
path$first_critical_temperature
```

gmm_canonicalise

Canonicalise the component ordering of a Gaussian mixture

Description

Returns a new [gmm](#) (or [gmm_fit](#)) with the components permuted into a canonical order: weight descending, then $||\mu||$ descending as a tiebreaker. The mixture distribution is unchanged — only the bookkeeping order is — but the canonical ordering removes the EM label-switching nuisance from snapshot tests, cross-run comparisons, and printed summaries.

Usage

```
gmm_canonicalise(g)
```

Arguments

g A [gmm](#) (or [gmm_fit](#)) object.

Details

Applied automatically by the regime-specific fitters ([fit_moment_match\(\)](#), [fit_em_samples\(\)](#), [fit_kld_em\(\)](#)) and by the top-level dispatcher [fit_proxymix\(\)](#) when `canonicalise = TRUE` (the default).

Value

A [gmm](#) (or [gmm_fit](#)) of the same subclass as `g`, with the components permuted into canonical order.

See Also

Other ops: [dgmm\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.1, 0.6, 0.3),
  means = list(c(0, 0), c(3, 0), c(-1, 1)),
  covariances = list(diag(2), diag(2), diag(2)))
gmm_canonicalise(g)
```

gmm_cf_mean

*The identified counterfactual mean***Description**

Returns the one identified summary of a [gmm_counterfactual_law](#) – its mean.

Usage

```
gmm_cf_mean(x)
```

Arguments

`x` A [gmm_counterfactual_law](#).

Value

Numeric scalar.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
g <- gmm(weights = 1, means = list(c(0, 0, 0)),
          covariances = list(diag(3)))
cf <- gmm_counterfactual(g, evidence = c(1, 0, 0.2),
                        do = c(NA, 1, NA), query = 1L)
gmm_cf_mean(cf)
```

gmm_cf_tail_prob

*Refused: a tail probability of an individual counterfactual law***Description**

Like [gmm_cf_variance\(\)](#), a per-unit counterfactual tail probability $P(Y_{t'} > c \mid y, t, x)$ is not identified – it is a functional of the unidentified counterfactual law, not of its identified mean. This accessor refuses rather than mislead.

Usage

```
gmm_cf_tail_prob(x, threshold)
```

Arguments

`x` A [gmm_counterfactual_law](#).
`threshold` Numeric scalar `c`.

Value

Never returns; always raises a `proxymix_not_identified` error.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
g <- gmm(weights = 1, means = list(c(0, 0, 0)),
          covariances = list(diag(3)))
cf <- gmm_counterfactual(g, evidence = c(1, 0, 0.2),
                        do = c(NA, 1, NA), query = 1L)
try(gmm_cf_tail_prob(cf, threshold = 2))
```

<code>gmm_cf_variance</code>	<i>Refused: the variance of an individual counterfactual law</i>
------------------------------	--

Description

The per-unit counterfactual *variance* is not identified from the joint density: it depends on the cross-world coupling of the structural residuals under the factual and counterfactual treatments, which no goodness-of-fit can certify. This accessor therefore raises an error rather than return the (misleading) spread of the abduction atoms.

Usage

```
gmm_cf_variance(x)
```

Arguments

`x` A [gmm_counterfactual_law](#).

Value

Never returns; always raises a `proxymix_not_identified` error.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
g <- gmm(weights = 1, means = list(c(0, 0, 0)),
         covariances = list(diag(3)))
cf <- gmm_counterfactual(g, evidence = c(1, 0, 0.2),
                        do = c(NA, 1, NA), query = 1L)
try(gmm_cf_variance(cf))
```

gmm_complete	<i>Extract completed datasets from a gmm_imputation</i>
--------------	---

Description

Extract completed datasets from a `gmm_imputation`

Usage

```
gmm_complete(object, which = 1L)
```

Arguments

object	A gmm_imputation .
which	Either an integer vector of imputation indices, or "all" for every completion. Default 1L.

Value

When `which` selects one completion, a single completed dataset (matrix, or data frame if the input was one); otherwise a list of them.

See Also

Other imputation: [as_mids\(\)](#), [gmm_imputation\(\)](#), [gmm_impute\(\)](#), [mechanism](#), [proxy_fmi\(\)](#), [proxy_mnar_sensitivity\(\)](#), [proxy_pool\(\)](#)

Examples

```
x1 <- rnorm(60)
x2 <- x1 + rnorm(60, sd = 0.4)
x2[sample(60, 10)] <- NA
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
first <- gmm_complete(imp, which = 1L) # one completed dataset
anyNA(first)
all_completions <- gmm_complete(imp, which = "all") # a list of ten
length(all_completions)
```

gmm_conditionalise *Conditional of a Gaussian mixture*

Description

Computes the conditional distribution of a Gaussian mixture given fixed values of a subset of coordinates, by the Schur-complement formula applied component-wise and re-weighted by the marginal evidence $p(x_b)$ of each component.

Usage

```
gmm_conditionalise(g, given)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) object.
<code>given</code>	A length-p numeric vector. Coordinates to <i>condition on</i> take their numeric value; coordinates left <i>free</i> are NA.

Value

A [gmm](#) object in dimension equal to the number of free coordinates.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
          means = list(c(-1, 0), c(1, 0)),
          covariances = list(diag(2), diag(2)))
gmm_conditionalise(g, given = c(NA, 0.5))
```

gmm_conditional_entropy

Renyi-2 or Shannon entropy of a conditional Gaussian mixture

Description

Returns the order-2 Renyi entropy (the default) or the Shannon differential entropy of the conditional mixture $g_{Y|X=x}$ obtained from [gmm_conditionalise\(\)](#) – the predictive uncertainty of the target coordinates given the conditioned ones. The Renyi-2 entropy is closed-form; order = "shannon" is a Monte Carlo estimate. Multiple conditioning configurations are evaluated row-by-row.

Usage

```
gmm_conditional_entropy(
  g,
  given,
  order = c("renyi2", "shannon"),
  n_mc = 5000L,
  seed = NULL
)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) joint mixture.
<code>given</code>	Either a numeric vector with one entry per coordinate, or a matrix whose rows are such vectors. NA marks a target (kept) coordinate; a numeric value conditions on that coordinate (the gmm_conditionalise() convention).
<code>order</code>	"renyi2" (closed-form, the default) or "shannon".
<code>n_mc, seed</code>	Passed to gmm_entropy() for order = "shannon".

Value

The entropy of the chosen order in nats: a numeric scalar for a single configuration, or a numeric vector with one entropy per row of given.

See Also

[gmm_entropy\(\)](#), [gmm_conditionalise\(\)](#)

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
## Joint over (Y, X); predictive entropy of Y at several X values.
s <- matrix(c(2, 0.8, 0.8, 1), 2, 2)
g <- gmm(weights = 1, means = list(c(0, 0)), covariances = list(s))
gmm_conditional_entropy(g, given = rbind(c(NA, 0), c(NA, 1)))
```

gmm_convolve

*Convolution of two independent Gaussian mixtures***Description**

The exact distribution of $X + Y$ for independent $X \sim g_1$ and $Y \sim g_2$: a Gaussian mixture with $K_1 K_2$ components,

$$g_1 * g_2 = \sum_{ij} w_i v_j \mathcal{N}(\mu_i + m_j, \Sigma_i + S_j).$$

Usage

```
gmm_convolve(g1, g2)
```

Arguments

`g1, g2` Two [gmm](#) (or [gmm_fit](#)) objects of the same ambient dimension.

Details

For the affine special case $X + c$ with a constant c , or $AX + \epsilon$ with Gaussian ϵ , use [gmm_affine\(\)](#); the convolution operator is the general mixture-plus-mixture case that [gmm_affine\(\)](#) cannot express.

Value

A [gmm](#) with $K_1 * K_2$ components.

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g1 <- gmm(weights = c(0.5, 0.5), means = list(-1, 1),
          covariances = list(matrix(0.5), matrix(0.5)))
g2 <- gmm(weights = 1, means = list(2), covariances = list(matrix(1)))
gmm_convolve(g1, g2)
```

gmm_counterfactual	<i>Counterfactual law of one unit (abduction, action, prediction)</i>
--------------------	---

Description

Computes the per-unit counterfactual of a query coordinate (the outcome) for an observed unit, under a `do()` intervention – Pearl’s third rung on the latent-class structural causal model read off a fitted Gaussian mixture.

Usage

```
gmm_counterfactual(g, evidence, do, query, ridge_eps = 1e-06)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) in R^p .
<code>evidence</code>	A length- p numeric vector of the observed unit. Unobserved coordinates are NA; the query coordinate must be observed.
<code>do</code>	A length- p numeric vector. Intervened coordinates take their counterfactual value; the rest are NA. Intervened coordinates must be observed in evidence.
<code>query</code>	A single integer coordinate index in <code>seq_len(p)</code> – the outcome whose counterfactual is sought.
<code>ridge_eps</code>	Tiny ridge added to the conditioning covariances for numerical hygiene.

Details

The three steps are closed form. *Abduction* recovers the regime posterior $\pi_k(\text{evidence})$ of the observed unit and, within each component, its structural residual. *Action* sets the `do` coordinates. *Prediction* re-evaluates the query coordinate. For a binary treatment the result is a discrete law on K atoms,

$$Y_{t'} \mid (y, t, x) = \sum_k \pi_k(y, t, x) \delta(y + \beta_k^T(t' - t)),$$

where β_k^T is component k ’s within-class treatment slope. Only the **mean** of this law is identified; its spread reflects regime uncertainty, not the (unidentified) cross-world coupling, so the variance and tail accessors refuse to answer (see [gmm_cf_variance\(\)](#)).

Value

A `gmm_counterfactual_law` object carrying the K atoms, their abduction weights, and the identified counterfactual mean.

See Also

[gmm_intervene\(\)](#), [gmm_cf_variance\(\)](#), [proxy_retrospective_uplift\(\)](#)

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
## Observed unit (y = 1.2, t = 0, x = 0.5); imagine t = 1.
g <- gmm(weights = c(0.6, 0.4),
         means = list(c(0, 0, 0), c(2, 1, 1)),
         covariances = list(diag(3), diag(3)))
cf <- gmm_counterfactual(g, evidence = c(1.2, 0, 0.5),
                       do = c(NA, 1, NA), query = 1L)
cf@mean
```

`gmm_counterfactual_law`

A per-unit counterfactual law

Description

The return type of [gmm_counterfactual\(\)](#): a discrete law on K atoms (the per-component counterfactual outcomes) with abduction weights, plus the one identified summary – the counterfactual mean. The atom spread is regime (epistemic) uncertainty about which structural component generated the unit; it is **not** the counterfactual outcome’s dispersion, which is unidentified. Accordingly [gmm_cf_variance\(\)](#) and [gmm_cf_tail_prob\(\)](#) refuse to answer.

Usage

```
gmm_counterfactual_law(
  atoms = numeric(0),
  weights = numeric(0),
  mean = numeric(0),
  query = integer(0),
  evidence = numeric(0),
  do = numeric(0),
  name = "gmm_counterfactual_law",
  metadata = list()
)
```

Arguments

atoms	Numeric vector of length K – the per-component counterfactual outcomes.
weights	Numeric vector of length K – the abduction weights $\pi_k(\text{evidence})$.
mean	Numeric scalar – the identified counterfactual mean.
query	Integer scalar – the queried coordinate index.
evidence	Numeric vector – the observed unit.
do	Numeric vector – the intervention.
name	Human-readable name.
metadata	Optional list of descriptors.

Value

An S7 object of class `gmm_counterfactual_law`.

See Also

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_dim\(\)](#), [gmm_fit\(\)](#), [gmm_n_components\(\)](#), [gmm_target\(\)](#), [gmm_weights\(\)](#), [is_proposal\(\)](#), [tidy.gmm](#)

<code>gmm_dim</code>	<i>Dimension of a Gaussian mixture</i>
----------------------	--

Description

Convenience accessor returning the ambient dimension p .

Usage

```
gmm_dim(x)
```

Arguments

`x` A [gmm](#) (or [gmm_fit](#)) object.

Value

Integer scalar.

See Also

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_counterfactual_law\(\)](#), [gmm_fit\(\)](#), [gmm_n_components\(\)](#), [gmm_target\(\)](#), [gmm_weights\(\)](#), [is_proposal\(\)](#), [tidy.gmm](#)

Examples

```
g <- gmm(weights = 1, means = list(c(0, 0)),
         covariances = list(diag(2)))
gmm_dim(g)
```

gmm_divergence

*Divergence between two Gaussian mixtures***Description**

Computes a divergence between two Gaussian mixtures of the same ambient dimension. The Cauchy-Schwarz divergence

$$D_{CS}(p, q) = \frac{1}{2} \log V(p, p) + \frac{1}{2} \log V(q, q) - \log V(p, q),$$

with $V(p, q) = \int p(x)q(x) dx$, is closed-form, symmetric, non-negative, and zero exactly when $p \propto q$. The "kl" option delegates to [gmm_kld\(\)](#), a Monte-Carlo estimate of the asymmetric Kullback-Leibler divergence $KL(p||q)$.

Usage

```
gmm_divergence(p, q, type = c("cs", "kl"), n_mc = 5000L)
```

Arguments

p, q	Two gmm (or gmm_fit) objects of the same ambient dimension.
type	"cs" (closed-form symmetric Cauchy-Schwarz divergence, the default) or "kl" (delegates to gmm_kld()).
n_mc	Number of Monte Carlo samples used when type = "kl".

Value

For type = "cs", a non-negative numeric scalar. For type = "kl", the list returned by [gmm_kld\(\)](#).

See Also

[gmm_entropy\(\)](#), [gmm_kld\(\)](#)

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
p <- gmm(weights = c(0.5, 0.5),
  means = list(c(-1, 0), c(1, 0)),
  covariances = list(diag(2), diag(2)))
q <- gmm(weights = 1, means = list(c(0, 0)),
  covariances = list(diag(2) * 2))
gmm_divergence(p, q)
gmm_divergence(p, p)
```

gmm_entropy

*Renyi-2 or Shannon entropy of a Gaussian mixture***Description**

Computes the order-2 Renyi entropy (the default) or the Shannon differential entropy of a Gaussian mixture. The quadratic (order-2) Renyi entropy $H_2(g) = -\log \int g(x)^2 dx$ is available in closed form, because $\int g^2$ is a finite sum of Gaussian-density evaluations. Shannon entropy has no closed form for a mixture (the integrand carries the logarithm of a sum) and is estimated by Monte Carlo, reported with its standard error and an analytic upper bound that brackets it from above.

Usage

```
gmm_entropy(g, order = c("renyi2", "shannon"), n_mc = 5000L, seed = NULL)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) object.
<code>order</code>	"renyi2" (closed-form quadratic Renyi entropy, the default) or "shannon" (Monte-Carlo estimate with an analytic upper bound).
<code>n_mc</code>	Number of Monte Carlo samples for order = "shannon".
<code>seed</code>	Optional integer seed for the Monte Carlo draw.

Value

For order = "renyi2", the Renyi-2 entropy in nats, a numeric scalar. For order = "shannon", a list with components `mc` (the estimate), `mc_se` (its standard error), `upper_bound` (the analytic upper bound), and `n_mc`.

See Also

[gmm_divergence\(\)](#), [gmm_kld\(\)](#)

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
         means = list(c(-2, 0), c(2, 0)),
         covariances = list(diag(2), diag(2)))
gmm_entropy(g)
gmm_entropy(g, order = "shannon", n_mc = 2000L, seed = 1L)
```

gmm_eos_test

*End-of-sample instability test on a Gaussian state-space filter***Description**

Tests whether the last m observations of a series are consistent with a linear-Gaussian state-space model fitted on the rest, in the regime where m is small (even $m = 1$) and smaller than the parameter count – where ordinary structural-break tests (Chow, sup-Wald) are undefined because the post-break parameters cannot be estimated. The statistic is the sum of the last m squared standardised one-step innovations from the filter; a break inflates it.

Usage

```
gmm_eos_test(
  prior,
  dynamics,
  measurement,
  y,
  m = 1L,
  method = c("chisq", "andrews"),
  alpha = 0.05
)
```

Arguments

prior	A single-component gmm giving the state prior (the test is defined for a fitted linear-Gaussian model; multi-component priors are not yet supported).
dynamics	A list with A (the state-transition matrix), Q (the process-noise covariance) and an optional offset b, or a function <code>function(t)</code> returning such a list, exactly as for gmm_filter() . Gaussian-sum (mixture) process noise is rejected: the calibrations below are defined for Gaussian innovations only.
measurement	A list with C (the observation matrix), R (the observation-noise covariance) and an optional offset d, or a function <code>function(t)</code> returning such a list, exactly as for gmm_filter() . Gaussian-sum measurement noise is rejected for the same reason.
y	A numeric vector or an $n \times d$ matrix of observations.
m	Integer; the number of end-of-sample observations to test, $1 \leq m < \text{nrow}(y)$. The tiny- m regime ($m = 1, 2, 3$) is the point of the test.
method	Either "chisq" (parametric) or "andrews" (subsampling calibration after Andrews' P-test, asymptotic in n).
alpha	The nominal level used to set reject.

Details

Two calibrations are offered. `method = "chisq"` refers the statistic to a chi-square distribution on $m * ncol(y)$ degrees of freedom, which is exact when the standardised innovations are Gaussian. `method = "andrews"` is a subsampling calibration after the P-test of Andrews (2003): the statistic's rank among the $n - 2m + 1$ overlapping m -blocks of the earlier innovations gives the p-value $(1+k)/(n-2m+2)$, where k is the number of blocks at least as large as the statistic. It differs from Andrews' test in two ways: the p-value counts the tested block itself, and the block statistics are computed at the supplied model instead of at estimates that leave out each block. It does not assume Gaussian innovations, so it suits heavy-tailed observation noise, but its validity is asymptotic: n large with m fixed, and innovations that are stationary and ergodic before the tested block. Its size equals the nominal level only when the end block is exchangeable with the earlier blocks, and in short series it can over-reject. The model is supplied exactly as for `gmm_filter()`.

Two finite-sample cautions. The chi-square calibration is exact when the model is given; with parameters estimated on a short series it over-rejects (the end-of-sample vignette reports its size by series length). The subsampling calibration can also over-reject in short series, so neither method holds its nominal size when the model is estimated on little data. The subsampling p-value has a floor of $1 / (n - 2m + 2)$, so it can reject at level 0.05 only when $n > 2m + 18$.

Value

An object of class `gmm_eos_test`: a list with the statistic, the `p_value`, the logical reject, the `method`, `m`, `alpha`, and the in-sample block statistics used for the subsampling calibration.

References

Andrews, D. W. K. (2003). End-of-Sample Instability Tests. *Econometrica*, 71(6), 1661–1694.

See Also

`gmm_filter()`

Examples

```
prior <- gmm(weights = 1, means = list(0), covariances = list(matrix(10)))
dyn    <- list(A = matrix(1), Q = matrix(0.04))
meas   <- list(C = matrix(1), R = matrix(1))
set.seed(1)
y <- c(rnorm(119), 6) # a stable series with a jump at the end
gmm_eos_test(prior, dyn, meas, y, m = 1L, method = "andrews")
```

Description

Importance-sampling estimate of $Z = \int f(x) dx$ for the fit's target f , using the fitted mixture $\hat{g}$ as the proposal:

$$\hat{Z} = \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{\hat{g}(x_i)}, \quad x_i \sim \hat{g},$$

computed in the log domain. A draw at which the target density is zero contributes a zero term and stays in the average. For a Bayesian posterior handed over as likelihood $\times$ prior, $\log Z$ is the log marginal likelihood, so a fitted proxy doubles as a model-comparison device.

Usage

```
gmm_evidence(fit, n = 4000L, seed = NULL)
```

Arguments

fit	A gmm_fit whose target carries a log_density.
n	Number of evidence draws from the fitted proxy.
seed	Optional integer seed for the evidence draw.

Details

The estimator is exact in expectation for any proposal that dominates f , and its Monte Carlo error is driven by how well $\hat{g}$ matches f – which is precisely what the fit optimised. The variance is finite only when $\hat{g}$ has tails at least as heavy as f ; a right-tail diagnostic is returned (the effective sample size and the share of the estimate carried by the largest ten percent of weights, which sits near 0.10 for a well-matched proxy), and a classed warning (`proxymix_heavy_tail`) is raised when it indicates an untrustworthy tail. Results also report the delta-method standard error of $\log \hat{Z}$.

When the target declares itself normalised (`normalised = TRUE`), the true value is $\log Z = 0$ and the function still estimates it – a useful end-to-end diagnostic of the fit.

Value

A list of class `proxymix_evidence` with elements `log_z`, `se_log_z`, `n`, `ess`, `max_weight_share`, `top_decile_share`, and `flagged` (the heavy-tail indicator).

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
## An unnormalised target with a known constant: log f = log N(., 0, I) + 3.
tgt <- gmm_target(
  n_dim = 2L,
  log_density = function(x) {
    if (is.null(dim(x))) x <- matrix(x, ncol = 2L)
```

```

      -0.5 * rowSums(x^2) - log(2 * pi) + 3
    },
    normalised = FALSE, name = "shifted_gaussian"
  )
fit <- fit_kld_em(tgt, N = 1L, is_size = 2000L, max_iter = 40L, seed = 1L)
ev <- gmm_evidence(fit, n = 2000L, seed = 2L)
ev$log_z    # close to 3

```

gmm_filter

Bounded Gaussian-sum filtering over an observation series

Description

Runs a Gaussian-sum filter over a series of n observations by alternating the predict operator [gmm_affine\(\)](#), the update operator [gmm_observe\(\)](#) and an optional reduction [gmm_reduce\(\)](#). At one component this is the classical Kalman filter; at several components, with Gaussian-mixture process or measurement noise, it is the Gaussian-sum filter of Alspach and Sorenson (1972). The filter is a pure composition of the existing closed-form operators; nothing here leaves the affine-Gaussian world.

Usage

```

gmm_filter(
  prior,
  dynamics,
  measurement,
  y,
  k_max = NULL,
  reduce = c("merge", "anneal"),
  ridge_eps = 1e-08
)

```

Arguments

prior	A gmm (or gmm_fit) in $\mathbb{R}^p$: the initial belief.
dynamics	A list <code>list(A =, b =, Q =)</code> describing the predict step, or a function <code>function(t)</code> returning such a list for time-varying dynamics. A is the p -by- p state-transition matrix; b is an optional length- p offset (default $\emptyset$); Q is the process noise – a p -by- p covariance matrix, a gmm on $\mathbb{R}^p$ (a Gaussian-sum process noise), or NULL (a deterministic predict).
measurement	A list <code>list(C =, R =, d =)</code> describing the update step, or a function <code>function(t)</code> returning such a list. C is the m -by- p observation matrix; R is the measurement noise – an m -by- m covariance matrix or a gmm on $\mathbb{R}^m$ (a Gaussian-sum measurement noise); d is an optional length- m offset (default $\emptyset$).
y	The observations: an n -by- m numeric matrix, a length- n list of length- m numeric vectors, or (when $m = 1$) a numeric vector of length n .

k_max	Optional component cap. NULL (the default) disables reduction and runs the exact Kalman / Gaussian-sum filter; a positive integer caps the component count after each step via <code>gmm_reduce()</code> .
reduce	The reduction method passed to <code>gmm_reduce()</code> when k_max is set: "merge" (the default, a deterministic moment-preserving merge) or "anneal".
ridge_eps	Tiny ridge passed to <code>gmm_affine()</code> and <code>gmm_observe()</code> for numerical hygiene. Set to 0 for an exact (ridge-free) recursion.

Details

At step t the belief is propagated through the linear-Gaussian dynamics $x_t = Ax_{t-1} + b + w$, then conditioned on the observation $y_t = Cx_t + d + v$, then (if k_max is set) reduced back to at most k_max components. With Gaussian noise the component count is constant and the recursion is the Kalman filter. Non-Gaussian noise is supplied as a Gaussian sum – a `gmm` in place of the covariance matrix Q or R. Each such mixture multiplies the component count by its own component count every step, so a long horizon is only runnable with reduction enabled. Reduction is moment-preserving (see `gmm_reduce()`), so the filtered mean and covariance are unaffected to the moment-matching order while the count stays bounded.

Value

A list with elements

filtered a length-n list of the filtered `gmm` beliefs, one per step (after predict, update and any reduction).

mean an n-by-p matrix of the filtered mixture means.

cov a length-n list of the p-by-p filtered mixture covariances.

summary a data frame with one row per step: the step index, the per-coordinate filtered mean (mean_1, ...) and standard deviation (sd_1, ...), the component count n_components, and the step log marginal evidence log_evidence (whose sum over steps is the log likelihood of the series).

References

Alspach, D. L. and Sorenson, H. W. (1972) Nonlinear Bayesian estimation using Gaussian sum approximations. *IEEE Transactions on Automatic Control* 17(4), 439–448. doi:10.1109/TAC.1972.1100034

See Also

Other operators: `gmm_affine()`, `gmm_aggregate()`, `gmm_convolve()`, `gmm_counterfactual()`, `gmm_intervene()`, `gmm_missing()`, `gmm_mix()`, `gmm_observe()`, `gmm_product()`, `gmm_reduce()`

Examples

```
## A one-dimensional local-level filter (random walk observed in noise).
prior <- gmm(weights = 1, means = list(0), covariances = list(matrix(1)))
truth <- cumsum(stats::rnorm(20, sd = 0.3))
y <- truth + stats::rnorm(20, sd = 0.5)
out <- gmm_filter(
```

```

    prior,
    dynamics    = list(A = matrix(1), Q = matrix(0.09)),
    measurement = list(C = matrix(1), R = matrix(0.25)),
    y = y
  )
  head(out$summary)

  ## Heavy-tailed process noise as a two-component Gaussian sum, capped at
  ## four components per step.
  q_noise <- gmm(weights = c(0.9, 0.1),
                 means = list(0, 0),
                 covariances = list(matrix(0.05), matrix(1)))
  out2 <- gmm_filter(
    prior,
    dynamics    = list(A = matrix(1), Q = q_noise),
    measurement = list(C = matrix(1), R = matrix(0.25)),
    y = y, k_max = 4L
  )
  max(out2$summary$n_components)

```

gmm_fit

A fitted Gaussian-mixture proxy

Description

A `gmm_fit` is the result of `fit_proxymix()` (or one of the regime-specific fitters). It inherits the mixture parameters of `gmm` and adds a record of the target it was fitted to, the regime used, and the iteration diagnostics.

Usage

```

gmm_fit(
  weights = numeric(0),
  means = list(),
  covariances = list(),
  name = "gmm",
  metadata = list(),
  target = NULL,
  regime = NA_character_,
  diagnostics = list(),
  converged = NA,
  iterations = NA_integer_,
  call = NULL
)

```

Arguments

`weights`, `means`, `covariances`, `name`, `metadata`

See `gmm`.

target	The gmm_target the mixture was fitted to.
regime	One of "moment", "sample", "kld".
diagnostics	A list of regime-specific diagnostics (see kld_trace() , ess_trace()).
converged	Logical scalar.
iterations	Integer scalar.
call	The matched call.

Value

An S7 object inheriting from `gmm_fit` (and `gmm`).

See Also

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_counterfactual_law\(\)](#), [gmm_dim\(\)](#), [gmm_n_components\(\)](#), [gmm_target\(\)](#), [gmm_weights\(\)](#), [is_proposal\(\)](#), [tidy.gmm](#)

Examples

```
samples <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(samples)
fit <- fit_proxymix(tgt, N = 2L, regime = "sample", max_iter = 25L)
inherits(fit, "proxymix::gmm_fit")
```

`gmm_fit_ensemble`

Bootstrap ensemble of a fitted proxy

Description

Quantifies the sampling variability of the fitted mixture itself by a Bayesian (weighted) bootstrap: each replicate re-weights the fit's own observations with $\text{Dirichlet}(1, \dots, 1)$ weights and refits by a warm-started weighted EM. In regime "kld" the observations are the fit's cached importance draws with their self-normalised weights, so a replicate costs **zero new target evaluations**; in regimes "sample" and "moment" the observations are the target's samples. Summaries of any functional of the proxy are then read off the ensemble with [proxy_functional_ci\(\)](#) – functional-space intervals sidestep the label-switching that makes parameter-space intervals incoherent.

Usage

```
gmm_fit_ensemble(fit, B = 200L, max_iter = 25L, tol = 1e-05, seed = NULL)
```

Arguments

fit	A gmm_fit whose fitting inputs are recoverable (a regime-"kld" fit carries its importance draws; regimes "sample" and "moment" require the target to carry its samples).
B	Number of bootstrap replicates.
max_iter, tol	Convergence controls for the per-replicate warm-started weighted EM.
seed	Optional integer seed for the replicate weights.

Value

A list of class `gmm_ensemble`: `fit` (the base fit), `members` (a length-`B` list of [gmm](#)), `B`, and `regime`.

References

Rubin, D. B. (1981) The Bayesian bootstrap. *The Annals of Statistics* 9(1), 130–134.

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                  is_size = 1500L, max_iter = 20L, seed = 1L)
ens <- gmm_fit_ensemble(fit, B = 30L, seed = 2L)
proxy_functional_ci(ens, gmm_mean)
```

gmm_fit_quality

The quality certificate of a fit or derived mixture

Description

Returns the fit-quality certificate: a small list recording the fitting regime, convergence, degeneracy, the effective-sample-size profile of the importance weights, and (when a validation split was drawn) the held-out validation gap. The certificate is stamped into the object's metadata at fit time and carried through every closed-form operator: a unary operator (marginal, conditional, affine map, Bayes update) preserves it unchanged, and an operator with several operands (product, convolution, mixing) combines their certificates conservatively – worst-case of each field, with the individual per-operand certificates retained under `quality_sources` and `regime` set to "composite". It can therefore be read off a marginal, a conditional, a product, a filtered belief, or any other derived mixture – alongside the provenance vector recording the chain of operations that produced it.

Usage

```
gmm_fit_quality(g)
```

Arguments

`g` A [gmm](#) or [gmm_fit](#).

Details

Downstream verbs read the same certificate and raise a one-shot advisory (class `proxymix_low_quality`) when the source fit is flagged. A fit is flagged when it did not converge, when it is degenerate (fewer effective importance draws than `min_ess`), or when its held-out KL divergence `heldout_kld` exceeds 0.3 nats. A fit made with `validation_size = 0` has no held-out estimate, and is flagged when `kld_approx` exceeds 0.3. That fallback misses a proxy wider than the target, because such a proxy can keep `kld_approx` below 0.3 while its KL divergence is large. Relative ESS is not used: with a fixed proposal it describes the proposal, not the fit, and stays the same however good the fit is.

Value

A list with elements `regime`, `converged`, `degenerate`, `ess`, `ess_relative`, `min_component_ess`, `max_weight`, `support_fraction`, `kld_final`, `kld_approx`, `validation_gap`, and `heldout_kld` (fields not applicable to the regime are NA), or NULL for a mixture that was never fitted (e.g. built directly with `gmm()`). A certificate produced by an operator over several operands has `regime = "composite"`, the conservative worst-case value of each numeric field across the operands, and an additional `quality_sources` element holding the operands' own certificates. `kld_final` is estimated on the importance draws the fit was tuned to, so it reads low; `validation_gap` is the held-out `validation_kld` (see `ess_summary()`) minus `kld_final`, and is NA when the fit was made with `validation_size = 0`. `heldout_kld` is the KL divergence from the target to the proxy estimated on the validation draws. For a target with an unknown normalising constant it is `validation_kld` minus a log normalising constant estimated from the same draws. `kld_approx` is half the importance-weighted variance of $\log f - \log g$ over the fitting draws. It approximates the KL divergence only when the proxy is close to the target, and it does not depend on the target's normalising constant. Both are NA outside regime "kld".

See Also

Other diagnostics: `bic_aic()`, `ess_summary()`, `ess_trace()`, `gmm_anneal_path()`, `gmm_conditional_entropy()`, `gmm_entropy()`, `gmm_evidence()`, `gmm_fit_ensemble()`, `gmm_independence_graph()`, `gmm_mutual_information()`, `hellinger_mc()`, `kld_trace()`, `proxy_functional_ci()`

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                  is_size = 1500L, max_iter = 15L, seed = 1L)
gmm_fit_quality(fit)
## The certificate survives the operator calculus.
gmm_fit_quality(gmm_marginalise(fit, keep = 1L))
```

gmm_imputation

A Gaussian-mixture multiple-imputation result

Description

The object returned by `gmm_impute()`. It carries the `m` completed data matrices, the bootstrap-fitted mixtures behind them (used by the analytic pooling in `proxy_pool()`), the mixture fitted to the full data, and a record of the missingness. Pass it to `gmm_complete()` to extract the completed datasets and to `proxy_pool()` / `proxy_fmi()` for inference.

Usage

```
gmm_imputation(
  data = NULL,
  completions = list(),
  fits = list(),
  point_fit = NULL,
  n_components = integer(0),
  m = integer(0),
  mechanism = "mar",
  observed = NULL,
  var_names = character(0),
  is_data_frame = FALSE,
  diagnostics = list(),
  call = NULL
)
```

Arguments

<code>data</code>	The numeric data matrix supplied to <code>gmm_impute()</code> , with NA for missing entries.
<code>completions</code>	List of <code>m</code> completed data matrices.
<code>fits</code>	List of <code>m</code> bootstrap-fitted <code>gmm</code> objects behind the completions.
<code>point_fit</code>	The <code>gmm</code> fitted to the full data.
<code>n_components</code>	Integer number of mixture components.
<code>m</code>	Integer number of completions.
<code>mechanism</code>	Missingness mechanism (currently "mar").
<code>observed</code>	Logical matrix marking the observed entries.
<code>var_names</code>	Character vector of column names.
<code>is_data_frame</code>	Logical; whether the input was a data frame.
<code>diagnostics</code>	List of fit diagnostics (per-column missing rates, convergence, iterations).
<code>call</code>	The matched call.

Value

An S7 object of class `gmm_imputation`.

See Also

Other imputation: `as_mids()`, `gmm_complete()`, `gmm_impute()`, `mechanism`, `proxy_fmi()`, `proxy_mnar_sensitivity()`, `proxy_pool()`

Examples

```
x1 <- rnorm(60)
x2 <- x1 + rnorm(60, sd = 0.4)
x2[sample(60, 10)] <- NA
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
imp # an S7 object of class gmm_imputation
```

gmm_impute

*Multiple imputation by Gaussian-mixture conditioning***Description**

Fits a Gaussian mixture to a numeric dataset that contains missing values and draws m completed datasets from the mixture conditional $p(x_{\text{missing}} \mid x_{\text{observed}})$. Because the mixture can be multi-modal and heteroscedastic, the imputations follow the shape of the joint distribution rather than a single Gaussian, which keeps downstream inference valid on data that a single-Gaussian or linear-Gaussian imputer mis-specifies.

Usage

```
gmm_impute(
  data,
  N = NULL,
  m = 20L,
  mechanism = mar(),
  seed = NULL,
  max_iter = 100L,
  tol = 1e-06,
  ridge_eps = 1e-06
)
```

Arguments

data	A numeric matrix or data frame with NA for missing entries.
N	Number of mixture components. NULL (the default) selects it by the Bayesian information criterion over 1:6.
m	Number of completed datasets to draw. Default 20L.
mechanism	A missingness mechanism: <code>mar()</code> , <code>censored()</code> , or <code>mna()</code> . The string "mar" is also accepted. Default <code>mar()</code> .
seed	Optional integer seed. When supplied the result is reproducible and the ambient random-number state is restored on exit.
max_iter	Maximum EM iterations per fit. Default 100L.
tol	Relative log-likelihood tolerance for EM convergence. Default 1e-6.
ridge_eps	Ridge added to each component covariance at every M-step. Default 1e-6.

Details

Imputation is conditioning. For a row with observed coordinates the missing coordinates follow the closed-form mixture conditional (the same Schur-complement algebra as `gmm_conditionalise()`). The mixture is fitted to the incomplete data by expectation-maximisation whose E-step uses each row's observed margin and whose M-step restores the conditional covariance of the filled entries, so component variances are not under-estimated.

Proper multiple imputation requires the fitting parameters themselves to carry uncertainty, otherwise the pooled intervals are too narrow. Each of the m imputations is therefore drawn under a mixture fitted to an independent bootstrap resample of the rows, so `proxy_pool()` reflects both imputation and parameter uncertainty.

The mechanism says how an entry came to be missing, which sets the conditional the missing value is drawn from: `mar()` (the default) for missing at random, `censored()` for a known interval such as a detection limit, or `mnr()` for a value-dependent selection model. The interval and value-dependent gates act on a single coordinate, and a row missing that coordinate must have its other coordinates observed. Numeric data only; categorical variables are out of scope.

Value

A `gmm_imputation` object.

See Also

`gmm_complete()` to extract completions, `proxy_pool()` to pool an estimand across them, `gmm_conditionalise()` for the conditioning algebra.

Other imputation: `as_mids()`, `gmm_complete()`, `gmm_imputation()`, `mechanism`, `proxy_fmi()`, `proxy_mnr_sensitivity()`, `proxy_pool()`

Examples

```
set.seed(1)
x1 <- rnorm(200)
x2 <- x1 + rnorm(200)
x2[runif(200) < plogis(x1)] <- NA      # missing at random on x1
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
proxy_pool(imp, "x2")$estimate        # pooled mean of x2
```

`gmm_independence_graph`

Conditional-independence (Gaussian graphical model) structure of a mixture

Description

Returns the undirected second-order conditional-independence graph of a fitted Gaussian mixture: the partial-correlation (Gaussian graphical model) structure of the mixture's overall covariance. An edge $i - j$ is present when the partial correlation r_{ij} of coordinates i and j given all the others is judged nonzero, and absent otherwise – the latter is the Markov statement $x_i \perp x_j \mid x_{\text{rest}}$ at second order. The overall covariance

$$\text{Cov}(X) = \sum_k w_k (\Sigma_k + \mu_k \mu_k^\top) - \bar{\mu} \bar{\mu}^\top, \quad \bar{\mu} = \sum_k w_k \mu_k,$$

is closed-form in the mixture parameters, so no sampling is required.

Usage

```
gmm_independence_graph(g, threshold = NULL, alpha = 0.05, n = NULL)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) mixture.
<code>threshold</code>	Non-negative partial-correlation magnitude above which an edge is drawn. NULL (the default) uses the Fisher-z test when a sample size is available and 0.05 otherwise. Cannot be combined with <code>n</code> .
<code>alpha</code>	Per-edge level of the Fisher-z test, in $(0, 1)$. Used only when a sample size is available and <code>threshold</code> is not given.
<code>n</code>	Number of data points the mixture was fitted to. NULL (the default) takes it from a regime "sample" or "moment" fit and otherwise leaves it unknown.

Details

When the mixture was fitted to n data points, each partial correlation is tested with Fisher's z : an edge is drawn when

$$|\operatorname{atanh}(r_{ij})|\sqrt{n-p-1} > \Phi^{-1}(1-\alpha/2).$$

A [gmm_fit](#) from regime "sample" or "moment" supplies n as the number of rows of its target's samples; n can also be given directly. The test is per edge, and a graph on p coordinates carries $p(p-1)/2$ of them: `alpha = 0.05 / choose(p, 2)` keeps the chance of any false edge below 0.05. Without a sample size (a regime "kld" fit, or a plain [gmm](#)), an edge is drawn when $|r_{ij}|$ exceeds `threshold`, which defaults to 0.05 there. Giving `threshold` always selects the magnitude rule.

This is a graphical-model (dependency-structure) diagnostic, not a causal discovery method: it recovers the undirected Markov skeleton, not edge directions. Its distinctive use is **regime (iii)**: composed with [fit_kld_em\(\)](#), it recovers the dependency structure of a target you can only *evaluate* (an unnormalised energy / Gibbs density), where no sample exists to hand a sampling-based estimator. Being second order, it sees dependencies that enter the covariance; a purely higher-order coupling (zero correlation, nonzero dependence) is not detected – raise the mixture's component count, or read the coordinate-block dependence with [gmm_mutual_information\(\)](#) instead.

Value

A symmetric integer adjacency matrix (1 = edge, 0 = none) with the coordinate names of `g`, carrying the partial-correlation matrix as the "pcor" attribute.

See Also

[gmm_mutual_information\(\)](#), [fit_kld_em\(\)](#), [gmm_conditionalise\(\)](#)

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
## Regime (iii): the Markov structure of an evaluable-but-unsampleable density.
## A continuous chain field x1 - x2 - x3 (couplings only between neighbours).
energy <- function(X) {
  X <- matrix(X, ncol = 3)
  rowSums((X^2 - 1)^2) - 0.7 * (X[, 1] * X[, 2] + X[, 2] * X[, 3])
}
target <- gmm_target(n_dim = 3L, log_density = function(X) -energy(X))
g <- fit_kld_em(target, N = 8L, proposal = is_uniform(3L, -3, 3),
  is_size = 8000L, anneal = TRUE, seed = 1L, support_warn = FALSE)
gmm_independence_graph(g)          # recovers x1 - x2 - x3 (no x1 - x3 edge)
```

gmm_intervene

Interventional law of a Gaussian mixture (the do-operator)

Description

Reads a fitted Gaussian mixture as a latent-class structural causal model and returns the interventional distribution of the free coordinates under `do()` of some coordinates, optionally conditioning on others.

Usage

```
gmm_intervene(g, do, given = NULL, ridge_eps = 1e-06)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) in $\mathbb{R}^p$.
<code>do</code>	A length- p numeric vector. Coordinates to intervene on take their do-value; coordinates not intervened on are NA.
<code>given</code>	A length- p numeric vector, or NULL (the default, meaning no conditioning). Coordinates to condition on take their value; coordinates not conditioned on are NA. A coordinate may not appear in both do and given.
<code>ridge_eps</code>	Tiny ridge added to the conditional covariances for numerical hygiene. Set to 0 to disable.

Details

Intervened (do) coordinates are *set* inside every component but do not re-weight the regime gate – this is the graph surgery that distinguishes $p(\cdot \mid \text{do}(T = t))$ from $p(\cdot \mid T = t)$. Conditioned (given) coordinates re-weight the gate in the usual Bayesian way. Writing the component prior as π_k , the within-component conditional mean as μ_k , and the given-coordinate evidence as e_k , the returned mixture has weights $\pi_k(\text{given}) \propto \pi_k e_k$ and per-component parameters from the Schur conditional on the union of the do and given coordinates.

For a joint fit over (Y, T, X) , `gmm_intervene(fit, do = T = 1, given = X = x)` is the do-response $p(Y \mid \text{do}(T = 1), X = x)$; its mean is $\sum_k \pi_k(x) \mu_k^{y|1,x}$, the latent-confounder-mode interventional mean of [proxy_cate](#).

Value

A [gmm](#) over the free coordinates (those NA in both do and given), with weights re-gated by the given evidence only.

See Also

[gmm_counterfactual\(\)](#), [proxy_cate\(\)](#)

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
## Joint (Y, T, X): set T = 1 while conditioning on X = 0.3.
g <- gmm(weights = c(0.5, 0.5),
         means = list(c(0, 0, 0), c(2, 1, 1)),
         covariances = list(diag(3), diag(3)))
gmm_intervene(g, do = c(NA, 1, NA), given = c(NA, NA, 0.3))
```

gmm_kld

Kullback-Leibler divergence between two Gaussian mixtures

Description

Estimates $KL(p || q)$ between two Gaussian mixtures by Monte Carlo and, optionally, evaluates the Hershey–Olsen variational approximation as a deterministic sanity check.

Usage

```
gmm_kld(p, q, n_mc = 5000L, variational = TRUE)
```

Arguments

p, q	Two gmm (or gmm_fit) objects of the same ambient dimension.
n_mc	Number of Monte Carlo samples drawn from p.
variational	If TRUE, also return the Hershey–Olsen variational approximation.

Details

The Monte Carlo estimator draws `n_mc` samples from `p` and returns the empirical mean of $\log p(x) - \log q(x)$, together with a Monte Carlo standard error.

The variational approximation is

$$\hat{D}_{\text{var}}(p||q) = \sum_a \pi_a \log \left(\frac{\sum_{a'} \pi_{a'} e^{-\text{KL}(p_a || p_{a'})}}{\sum_b \omega_b e^{-\text{KL}(p_a || q_b)}} \right),$$

which is exact when `p == q` and tends to be a usable lower bound when the components of `p` and `q` are well-separated. The closed-form Gaussian–Gaussian KL $\text{KL}(p_a || q_b)$ is used internally.

Value

A list with components

- `mc` - the Monte Carlo estimate of $\text{KL}(p || q)$,
- `mc_se` - its Monte Carlo standard error,
- `variational` - the variational approximation (NA if `variational = FALSE`),
- `n_mc` - the number of Monte Carlo samples used.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
p <- gmm(weights = c(0.5, 0.5),
  means = list(c(-1, 0), c(1, 0)),
  covariances = list(diag(2), diag(2)))
q <- gmm(weights = 1,
  means = list(c(0, 0)),
  covariances = list(diag(2) * 2))
gmm_kld(p, q, n_mc = 500L)
```

<code>gmm_marginalise</code>	<i>Marginal of a Gaussian mixture</i>
------------------------------	---------------------------------------

Description

Computes the marginal distribution of a Gaussian mixture over a subset of coordinates. The marginal of a Gaussian mixture is itself a Gaussian mixture with the same weights.

Usage

```
gmm_marginalise(g, keep)
```

Arguments

`g` A [gmm](#) (or [gmm_fit](#)) object.

`keep` Integer vector of coordinate indices to retain (in 1 . . p).

Value

A [gmm](#) object in dimension `length(keep)`.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
         means = list(c(-1, 0, 2), c(1, 0, -2)),
         covariances = list(diag(3), diag(3)))
gmm_marginalise(g, keep = c(1L, 3L))
```

gmm_mean

*Mean and covariance of a Gaussian mixture***Description**

The exact first two moments of a mixture:

$$\mu = \sum_k w_k \mu_k, \quad \Sigma = \sum_k w_k (\Sigma_k + \mu_k \mu_k^\top) - \mu \mu^\top.$$

Usage

```
gmm_mean(g)
```

```
gmm_cov(g)
```

Arguments

`g` A [gmm](#) (or [gmm_fit](#)).

Value

`gmm_mean()` returns a length-p numeric vector; `gmm_cov()` returns a p by p numeric matrix.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.3, 0.7), means = list(c(-1, 0), c(1, 2)),
         covariances = list(diag(2), 0.5 * diag(2)))
gmm_mean(g)
gmm_cov(g)
```

gmm_missing

*Condition a Gaussian mixture on the exact values of some coordinates***Description**

A structured wrapper around [gmm_conditionalise\(\)](#) for the common case where the observed coordinates are specified by integer index rather than NA-padded vector. Equivalent to [gmm_observe\(\)](#) with a selection matrix A and zero noise covariance, but routed through the Schur-complement path for efficiency.

Usage

```
gmm_missing(g, observed, values)
```

Arguments

g	A gmm (or gmm_fit) in R^p .
observed	Integer vector of indices in <code>seq_len(p)</code> . The coordinates to condition on (fully observed).
values	Numeric vector of length <code>length(observed)</code> . The observed values, in the same order as observed.

Value

A [gmm](#) in $R^{(p - \text{length}(\text{observed}))}$.

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g <- gmm(weights = c(0.4, 0.6),
         means = list(c(-1, 0), c(1, 0)),
         covariances = list(diag(2), diag(2)))
# Condition coord 2 on the value 0.5; keep coord 1.
gmm_missing(g, observed = 2L, values = 0.5)
```

gmm_mix

*Mix Gaussian mixtures into one mixture***Description**

Flattens a list of Gaussian mixtures into a single mixture whose density is the weighted average $\sum_b \alpha_b g_b(x)$ – model averaging, prior pooling, or the mixture-of-mixtures construction.

Usage

```
gmm_mix(gmms, weights = NULL)
```

Arguments

gmms A non-empty list of [gmm](#) (or [gmm_fit](#)) objects sharing one ambient dimension.

weights Optional non-negative mixing weights, one per element of gmms (normalised internally). Default: equal weights.

Value

A [gmm](#) with $\text{sum}(K_b)$ components.

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g1 <- gmm(weights = 1, means = list(-1), covariances = list(matrix(1)))
g2 <- gmm(weights = 1, means = list(2), covariances = list(matrix(0.5)))
gmm_mix(list(g1, g2), weights = c(0.7, 0.3))
```

gmm_modes

*Modes of a Gaussian mixture***Description**

Returns the distinct local modes of a Gaussian-mixture density by Gaussian mean-shift (the fixed-point hill-climb of Carreira-Perpinan 2000) started from each component mean, together with the mixture density at each mode. Nearby converged points are merged so that each genuine mode is reported once.

Usage

```
gmm_modes(object, starts = NULL, tol = 1e-05, dedup = NULL, max_iter = 200L)
```

Arguments

object	A gmm or gmm_fit .
starts	Optional m-by-p matrix of starting points for the mean-shift. Default NULL uses the component means.
tol	Convergence tolerance on the mean-shift step length. Default 1e-5.
dedup	Distance below which two converged points are treated as the same mode. Default NULL derives a small fraction of the spread of the component means.
max_iter	Maximum mean-shift iterations per start. Default 200L.

Details

A mixture of K components has at most K modes, and fewer when components overlap; mean-shift from every component mean finds them robustly without a grid. The companion of [from_objective\(\)](#): applied to the fitted map it returns the recovered optima (ordered by density, which for a Gibbs proxy ranks the deepest optima first).

Value

A list with modes (an n-by-p matrix of distinct modes, ordered by descending mixture density), density (the mixture density at each mode), and n (the number of modes).

See Also

[from_objective\(\)](#), whose fitted map this resolves into optima.

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [pgmm\(\)](#), [rgmm\(\)](#)

Examples

```
g <- gmm(
  weights = rep(1 / 3, 3),
  means = list(c(-3, 0), c(3, 0), c(0, 4)),
  covariances = rep(list(0.3 * diag(2)), 3)
)
gmm_modes(g)$modes
```

Description

Measures the dependence between two disjoint coordinate blocks of a fitted joint Gaussian mixture as the Cauchy-Schwarz divergence between the joint over the two blocks and the product of their marginals,

$$I_{CS}(A; B) = D_{CS}(p_{AB}, p_A p_B).$$

The product of the marginals is itself a Gaussian mixture, so the quantity is closed-form. It is non-negative and zero exactly when the two blocks are independent. (The naive combination $H_2(A) + H_2(B) - H_2(A, B)$ is **not** a valid mutual information: order-2 Renyi entropies are not additive over independent blocks and that difference can be negative.)

Usage

```
gmm_mutual_information(g, block_a, block_b)
```

Arguments

`g` A [gmm](#) (or [gmm_fit](#)) joint mixture.
`block_a, block_b` Disjoint integer vectors of coordinate indices (in 1..p) naming the two blocks.

Value

A non-negative numeric scalar.

See Also

[gmm_entropy\(\)](#), [gmm_divergence\(\)](#)

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
## A correlated bivariate Gaussian: mutual information grows with |rho|.
s <- matrix(c(1, 0.7, 0.7, 1), 2, 2)
g <- gmm(weights = 1, means = list(c(0, 0)), covariances = list(s))
gmm_mutual_information(g, 1L, 2L)
```

`gmm_n_components`

Number of components in a Gaussian mixture

Description

Number of components in a Gaussian mixture

Usage

```
gmm_n_components(x)
```

Arguments

`x` A [gmm](#) (or [gmm_fit](#)) object.

Value

Integer scalar.

See Also

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_counterfactual_law\(\)](#), [gmm_dim\(\)](#), [gmm_fit\(\)](#), [gmm_target\(\)](#), [gmm_weights\(\)](#), [is_proposal\(\)](#), [tidy.gmm](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5), means = list(c(0, 0), c(1, 1)),
         covariances = list(diag(2), diag(2)))
gmm_n_components(g)
```

gmm_observe

Bayesian update of a Gaussian mixture on a noisy linear observation

Description

Conditions a Gaussian mixture `g` on a single noisy linear observation $y = Ax + b + \epsilon$, $\epsilon \sim \mathcal{N}(0, R)$. Per component, applies the Kalman gain

$$K_k = \Sigma_k A^\top S_k^{-1}, \quad S_k = A \Sigma_k A^\top + R,$$

and updates

$$\mu'_k = \mu_k + K_k(y - A\mu_k - b), \quad \Sigma'_k = (I - K_k A) \Sigma_k.$$

Component weights are multiplied by the marginal evidence $\pi_k \mathcal{N}(y; A\mu_k + b, S_k)$ and renormalised. This is the finite-mixture analogue of a Kalman update step.

Usage

```
gmm_observe(g, A, y, noise_cov, b = 0, ridge_eps = 1e-06)
```

Arguments

<code>g</code>	A gmm (or gmm_fit) in $\mathbb{R}^p$.
<code>A</code>	An m by p numeric matrix.
<code>y</code>	A length- m numeric vector (the observation).
<code>noise_cov</code>	An m by m SPD numeric matrix (the observation noise covariance R). Required.
<code>b</code>	Numeric scalar or length- m vector. Default $\emptyset$.
<code>ridge_eps</code>	Tiny ridge added to updated covariances for numerical hygiene. Set to $\emptyset$ to disable.

Details

If the marginal evidence vanishes at every component (e.g. y is many standard deviations from every component), the function issues a warning and returns g unchanged with `metadata$gmm_observe_no_update = TRUE`.

Value

A [gmm](#) in $\mathbb{R}^p$ with the same number of components and the reweighted component weights.

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_product\(\)](#), [gmm_reduce\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
  means = list(c(-1, 0), c(1, 0)),
  covariances = list(diag(2), diag(2)))
A <- matrix(c(1, 0), nrow = 1L)
gmm_observe(g, A = A, y = 0.8, noise_cov = matrix(0.25, 1, 1))
```

gmm_product

Pointwise product of two Gaussian mixtures

Description

The normalised pointwise product of two mixture densities – the conjugate Bayes update when one mixture plays the prior and the other the likelihood. The product of two Gaussian mixtures is again a Gaussian mixture with $K_1 K_2$ components:

$$p(x) q(x) \propto \sum_{ij} w_i v_j z_{ij} \mathcal{N}(x; \mu_{ij}, \Sigma_{ij}),$$

with $\Sigma_{ij} = (\Sigma_i^{-1} + S_j^{-1})^{-1}$, $\mu_{ij} = \Sigma_{ij}(\Sigma_i^{-1}\mu_i + S_j^{-1}m_j)$, and $z_{ij} = \mathcal{N}(\mu_i; m_j, \Sigma_i + S_j)$ evaluated in the log domain. The overall normaliser $\int p q = \sum_{ij} w_i v_j z_{ij}$ is returned in the result's metadata as `log_integral` (it is the marginal evidence of the update).

Usage

```
gmm_product(g1, g2, reduce = NULL)
```

Arguments

`g1, g2` Two [gmm](#) (or [gmm_fit](#)) objects of the same ambient dimension.

`reduce` Optional positive integer: reduce the product to at most this many components via [gmm_reduce\(\)](#). Default NULL (no reduction).

Details

Because the component count multiplies, chained products grow geometrically; pass `reduce` to cap the count via [gmm_reduce\(\)](#) after the product (the standard assumed-density trick), or manage the budget yourself.

Value

A [gmm](#) with $K1 * K2$ components (or at most `reduce`), with `metadata$log_integral` recording $\log \int p q$.

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_reduce\(\)](#)

Examples

```
prior <- gmm(weights = c(0.5, 0.5), means = list(-2, 2),
             covariances = list(matrix(1), matrix(1)))
lik <- gmm(weights = 1, means = list(0.5), covariances = list(matrix(0.5)))
post <- gmm_product(prior, lik)
post@weights
```

gmm_reduce

Reduce a Gaussian mixture to fewer components

Description

Collapses a Gaussian mixture to at most `k_max` components. The default method = "merge" is a greedy, moment-preserving pairwise merge: at each step the cheapest pair of components is replaced by the single Gaussian that preserves their combined weight, mean and covariance, until the component budget is met. Because every merge is moment-preserving, the reduced mixture has the **same global mean and covariance** as the original, and reducing all the way to a single component returns the moment-matched Gaussian.

Usage

```
gmm_reduce(
  g,
  k_max,
  method = c("merge", "anneal"),
  cost = c("kl", "cs"),
  draws = 5000L,
  seed = NULL,
  ridge_eps = 0
)
```

Arguments

<code>g</code>	A gmm (or gmm_fit).
<code>k_max</code>	Maximum number of components to keep (a positive integer). When <code>k_max</code> is at least the current component count the mixture is returned unchanged.
<code>method</code>	"merge" (the default, moment-preserving greedy merge) or "anneal" (the merge refined by an annealed re-fit; never worse than the merge).
<code>cost</code>	The pairwise merge cost: "kl" (the Runnalls Kullback-Leibler bound, the default) or "cs" (the Cauchy-Schwarz divergence).
<code>draws</code>	Number of draws used by the "anneal" re-fit. Ignored when <code>method = "merge"</code> .
<code>seed</code>	Optional integer seed for the "anneal" re-fit (the result is deterministic given a seed). Ignored when <code>method = "merge"</code> .
<code>ridge_eps</code>	Ridge added to each merged covariance. The moment-preserving merge is positive-definite by construction, so the default 0 keeps the global moments exact; set a small positive value for extra numerical headroom in very long reduction chains (at the cost of a tiny moment drift).

Details

The merge cost decides which pair is collapsed first. `cost = "kl"` is the Kullback-Leibler upper bound of Runnalls (2007), the standard mixture-reduction criterion; `cost = "cs"` is the closed-form Cauchy-Schwarz divergence between the two-component sub-mixture and its merge (the same Gaussian-product identity that underlies [gmm_divergence\(\)](#)), scaled by the merged mass. Both are non-negative and zero for identical components.

`method = "anneal"` additionally draws a sample from the mixture and refits a `k_max`-component proxy by annealed EM (see [fit_em_samples\(\)](#)), then returns whichever of the merge and the re-fit has the smaller Cauchy-Schwarz divergence from the original. The re-fit can improve on the greedy merge for smooth, over-parameterised mixtures, where a globally fitted proxy beats any sequence of pairwise merges; the merge is returned when it is at least as good, so the result is never worse than `method = "merge"`. Unlike the merge, the re-fit is a Monte Carlo fit and does not preserve the global moments exactly; raise `draws` for a closer re-fit.

Reduction is the closing operation of a Gaussian-sum filter: repeated [gmm_observe\(\)](#) / [gmm_affine\(\)](#) steps under a Gaussian-mixture noise or dynamics model multiply the component count, and [gmm_reduce\(\)](#) returns it to a fixed budget.

Value

A [gmm](#) with at most `k_max` components.

References

Runnalls, A. R. (2007) Kullback-Leibler approach to Gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems* 43(3), 989–999. doi:[10.1109/TAES.2007.4383588](https://doi.org/10.1109/TAES.2007.4383588)

See Also

Other operators: [gmm_affine\(\)](#), [gmm_aggregate\(\)](#), [gmm_convolve\(\)](#), [gmm_counterfactual\(\)](#), [gmm_filter\(\)](#), [gmm_intervene\(\)](#), [gmm_missing\(\)](#), [gmm_mix\(\)](#), [gmm_observe\(\)](#), [gmm_product\(\)](#)

Examples

```
## A six-component mixture with three near-duplicate pairs.
g <- gmm(
  weights = rep(1 / 6, 6),
  means = list(c(-4, 0), c(-4, 0.1), c(4, 0), c(4.1, 0), c(0, 5), c(0, 5.1)),
  covariances = rep(list(diag(2)), 6)
)
gmm_reduce(g, k_max = 3L)
```

gmm_target

A target density on R^p

Description

An S7 representation of a target density that proxymix is asked to approximate. A target may carry an evaluable `log_density`, a matrix of i.i.d. samples, or both. Each of the three fitting regimes consumes a different subset:

Usage

```
gmm_target(
  n_dim = integer(0),
  log_density = NULL,
  samples = NULL,
  support = NULL,
  normalised = NA,
  log_normalizer = NA_real_,
  name = "gmm_target",
  metadata = list()
)
```

Arguments

n_dim	Integer scalar — the ambient dimension p (the property is called <code>n_dim</code> rather than <code>dim</code> because <code>S7</code> reserves <code>dim</code> as an attribute name).
log_density	Optional function: <code>function(x)</code> taking a numeric matrix n by p and returning a length- n numeric vector of $\log f(x)$.
samples	Optional n by p numeric matrix of i.i.d. samples from the target.
support	Optional declaration of the target's support. <code>NULL</code> (the default) means the full $\mathbb{R}^p$. Otherwise a list <code>list(lower = , upper =)</code> of per-coordinate bounds, each of length 1 (recycled) or <code>n_dim</code> , with <code>-Inf / Inf</code> permitted for unbounded coordinates. A bounded or one-sided support drives automatic support-matched importance proposal selection in regime (iii); see <code>fit_kld_em()</code> .
normalised	Logical scalar declaring whether <code>log_density</code> integrates to one. <code>TRUE</code> , <code>FALSE</code> , or <code>NA</code> (unknown). Defaults to <code>NA</code> . Downstream diagnostics treat <code>NA</code> and <code>FALSE</code> identically and label any KLD estimate as shifted.
log_normalizer	Numeric scalar $\log Z(f)$, the log of the integral of $\exp(\log_density)$, if known. Default <code>NA_real_</code> . When <code>normalised = FALSE</code> and <code>log_normalizer</code> is finite, <code>fit_kld_em()</code> subtracts it from the shifted KLD estimates.
name	Human-readable name.
metadata	Optional list of additional descriptors.

Details

- regime "moment" needs samples (or both moments via metadata);
- regime "sample" needs samples;
- regime "kld" needs the log-density.

Use `gmm_target()` or `gmm_target_from_samples()` to construct.

Importance-sampled KLD-EM (regime "kld") only requires `log_density` to be specified up to an unknown additive constant — the self-normalised weights are invariant to scaling. The package's *diagnostics* downstream, however, do depend on normalisation. An importance-sampled KLD estimate against an unnormalised log-density equals $KL(f||g) + \log Z(f)$, where $Z(f)$ is the integral of $\exp(\log_density)$. A squared-Hellinger Monte Carlo estimate is only meaningful when both densities integrate to one. Declare the target's normalisation explicitly via `normalised` (and, where possible, supply `log_normalizer`) so that the package can label shifted KLDs as shifted and refuse misleading Hellinger reports.

Value

An `S7` object of class `gmm_target`.

See Also

Other classes: `autoplot.gmm_fit`, `glance.gmm_fit`, `gmm()`, `gmm_counterfactual_law()`, `gmm_dim()`, `gmm_fit()`, `gmm_n_components()`, `gmm_weights()`, `is_proposal()`, `tidy.gmm`

Examples

```
tgt <- banana_target()
tgt
```

```
gmm_target_from_posterior
```

Compile an unnormalised Bayesian posterior into a gmm_target

Description

Generic S3 constructor that turns a Bayesian posterior — represented either by a fitted model object (e.g. from brms or Stan) or by a bare callable — into a `gmm_target` suitable for regime (iii) of [fit_proxymix\(\)](#) / [fit_kld_em\(\)](#).

Usage

```
gmm_target_from_posterior(model, ...)

## Default S3 method:
gmm_target_from_posterior(model, ...)

## S3 method for class ``function``
gmm_target_from_posterior(
  model,
  ...,
  parameter_names = NULL,
  log_normalizer = NA_real_,
  name = NULL
)
```

Arguments

<code>model</code>	One of: - a function — a bare callable satisfying the contract above; - a fitted model object whose class registers a <code>gmm_target_from_posterior.<class></code> method in its own package.
<code>...</code>	Forwarded to method-specific implementations.
<code>parameter_names</code>	Character vector of parameter names. Required for the function method (or attached as <code>attr(model, "parameter_names")</code>). The length determines <code>n_dim</code> .
<code>log_normalizer</code>	Numeric scalar $\log Z$, the log of the integral of <code>exp()</code> of the log-posterior, if known. With <code>NA_real_</code> (the default), downstream diagnostics label any KLD estimate as shifted.
<code>name</code>	Optional human-readable target name. Defaults to "posterior".

Details

The contract for the underlying callable is:

- **Vectorised:** accepts a numeric matrix with rows indexing independent parameter draws and columns indexing parameters; returns a `length-nrow(theta)` numeric vector of $\log p(\theta \mid \text{data}) + \text{const.}$
- **Unnormalised is fine:** the marginal likelihood $\log Z$ is not required. Where the source package can supply it, pass `log_normalizer`.
- **Side-effect free:** no plotting, no mutable state. Pure function.
- **Domain-safe:** returns $-\text{Inf}$ outside support rather than raising an error.

The default method errors with a hint pointing the user at either (a) a Bayesian package that registers a method, or (b) the function method below.

Value

A `gmm_target` with `normalised = FALSE` and the user-supplied `log_normalizer` (or `NA_real_`).

Examples

```
# A trivial unnormalised log-posterior: a 2D banana centred near (1, 0).
log_post <- function(theta) {
  x <- theta[, 1L]
  y <- theta[, 2L]
  -0.5 * (x^2 + (y - 0.1 * x^2 + 1)^2)
}
tgt <- gmm_target_from_posterior(
  log_post,
  parameter_names = c("x", "y")
)
tgt
```

`gmm_target_from_samples`

Build a target from samples alone

Description

Wraps a numeric matrix of i.i.d. samples as a `gmm_target`. The resulting target carries no `log_density`, so it can only feed regimes "moment" (via empirical moments) and "sample" (classical EM).

Usage

```
gmm_target_from_samples(samples, name = "target_from_samples")
```

Arguments

<code>samples</code>	An <code>n</code> by <code>p</code> numeric matrix.
<code>name</code>	Optional human-readable name. Defaults to "target_from_samples".

Value

A [gmm_target](#) object.

See Also

Other targets: [banana_target\(\)](#), [donut_target\(\)](#), [epanechnikov_target\(\)](#), [maxent_target\(\)](#), [mixture_target\(\)](#)

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(x)
tgt
```

`gmm_weights`

Component parameters of a Gaussian mixture

Description

Read-only accessors for the component weights, means, and covariances.

Usage

```
gmm_weights(g)
```

```
gmm_means(g)
```

```
gmm_covariances(g)
```

Arguments

`g` A [gmm](#) (or [gmm_fit](#)).

Value

`gmm_weights()` returns a numeric vector of length K ; `gmm_means()` and `gmm_covariances()` return length- K lists of length- p numeric vectors and p by p matrices respectively.

See Also

[gmm_mean\(\)](#) / [gmm_cov\(\)](#) for the moments of the mixture as a whole, [gmm_fit_quality\(\)](#) for the fit-quality certificate.

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_counterfactual_law\(\)](#), [gmm_dim\(\)](#), [gmm_fit\(\)](#), [gmm_n_components\(\)](#), [gmm_target\(\)](#), [is_proposal\(\)](#), [tidy.gmm](#)

Examples

```
g <- gmm(weights = c(0.3, 0.7), means = list(-1, 2),
          covariances = list(matrix(1), matrix(0.5)))
gmm_weights(g)
gmm_means(g)
```

hellinger_mc

Monte-Carlo Hellinger distance between a fit and its target

Description

Estimates the squared Hellinger distance $H^2(f, g) = 1 - \int \sqrt{f(x) g(x)} dx$ as $1 - \text{mean}(\sqrt{f(x) / g(x)})$ over n_{mc} fresh draws x from the fit g , for every regime. The draws are independent of any sample used to fit, so the estimate is unbiased, with standard error se . The target's log_density must be supplied **and normalised**; otherwise the Monte Carlo integral is biased by the missing $\sqrt{Z(f)}$. When the target's normalised property is not TRUE, a warning is issued and the returned value is flagged.

Usage

```
hellinger_mc(fit, n_mc = 5000L, seed = NULL)
```

Arguments

fit	A gmm_fit whose target carries a log_density.
n_mc	Number of Monte Carlo draws from the fit.
seed	Optional integer seed for the draws.

Value

A list with components

- h2 - estimate of $H^2(f, g)$,
- se - Monte Carlo standard error,
- n_mc - number of draws with a finite density ratio,
- trustworthy - TRUE when the target is declared normalised.

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [kld_trace\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 3L, regime = "kld",
                  is_size = 2000L, max_iter = 25L, seed = 1L)
hellinger_mc(fit, n_mc = 1000L, seed = 1L)
```

init_kmeans	<i>k-means initialisation</i>
-------------	-------------------------------

Description

Runs `stats::kmeans()` on the supplied samples and uses the resulting cluster centres and within-cluster covariances to seed an EM-style fitter.

Usage

```
init_kmeans(samples, N = 2L, ridge_eps = 1e-06, nstart = 10L)
```

Arguments

<code>samples</code>	An n by p numeric matrix of samples from (or close to) the target.
<code>N</code>	Number of components.
<code>ridge_eps</code>	Ridge added to each cluster covariance for numerical stability when a cluster has fewer than two points.
<code>nstart</code>	<code>stats::kmeans</code> <code>nstart</code> argument.

Value

A [gmm](#) of N components in dimension `ncol(samples)`.

See Also

Other init: [init_moment_seed\(\)](#), [init_random\(\)](#), [init_warm_start\(\)](#), [multi_start_best_of\(\)](#)

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
init_kmeans(x, N = 3L)
```

init_moment_seed	<i>Moment-seed initialisation</i>
------------------	-----------------------------------

Description

Computes the global mean and covariance of the supplied samples and spreads N components along the leading principal direction. Useful as a deterministic starting point that survives multi-modal targets better than a single-Gaussian fit.

Usage

```
init_moment_seed(samples, N = 2L, spread = 1.5)
```

Arguments

samples	An n by p numeric matrix.
N	Number of components.
spread	Multiplier on the principal-direction standard deviation used to place the component means symmetrically about the global mean.

Value

A *gmm* of N components in dimension `ncol(samples)`.

See Also

Other init: `init_kmeans()`, `init_random()`, `init_warm_start()`, `multi_start_best_of()`

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
init_moment_seed(x, N = 3L)
```

init_random	<i>Random initialisation</i>
-------------	------------------------------

Description

Generates an N-component random initialisation by perturbing isotropic means around a centre. Useful as one starting point in a multi-start best-of strategy.

Usage

```
init_random(
  N = 1L,
  p = 2L,
  centre = rep(0, p),
  scale = 1,
  sigma_diag = 1,
  seed = NULL
)
```

Arguments

N	Number of components.
p	Ambient dimension.
centre	Length-p numeric vector — the location around which means are drawn.
scale	Standard deviation of the mean perturbation.
sigma_diag	Diagonal value used for the initial component covariances.
seed	Optional integer seed.

Value

A [gmm](#) of N components in dimension p.

See Also

Other init: [init_kmeans\(\)](#), [init_moment_seed\(\)](#), [init_warm_start\(\)](#), [multi_start_best_of\(\)](#)

Examples

```
init_random(N = 3L, p = 2L, seed = 1L)
```

init_warm_start	<i>Warm-start initialisation from an existing fit</i>
-----------------	---

Description

Returns the input as-is. Provided as a name so that the multi-start driver can include "warm starts" by symbolic name.

Usage

```
init_warm_start(g)
```

Arguments

g A [gmm](#) (or [gmm_fit](#)).

Value

The input g, validated.

See Also

Other init: [init_kmeans\(\)](#), [init_moment_seed\(\)](#), [init_random\(\)](#), [multi_start_best_of\(\)](#)

Examples

```
g <- gmm(weights = 1, means = list(c(0, 0)),
          covariances = list(diag(2)))
init_warm_start(g)
```

is_mvn	<i>Multivariate-normal proposal</i>
--------	-------------------------------------

Description

Builds an [is_proposal](#) using a multivariate-normal $N(\text{mean}, \text{cov})$ density and sampler.

Usage

```
is_mvn(n_dim, mean = rep(0, n_dim), cov = diag(n_dim))
```

Arguments

n_dim	Ambient dimension p .
mean	Length- p numeric mean vector. Defaults to the zero vector.
cov	A p -by- p symmetric positive-definite covariance matrix. Defaults to the identity.

Value

An [is_proposal](#) object.

See Also

Other proposals: [is_mvt\(\)](#), [is_uniform\(\)](#), [proposal_uniform\(\)](#)

Examples

```
q <- is_mvn(n_dim = 2L, mean = c(0, 0), cov = 4 * diag(2))
q
```

is_mvt	<i>Multivariate-t proposal</i>
--------	--------------------------------

Description

Builds an [is_proposal](#) using a multivariate-Student- t density and sampler with df degrees of freedom, location mean, and scale matrix σ . Heavier tails than [is_mvn\(\)](#), so often a safer importance proposal at moderate dimensions.

Usage

```
is_mvt(n_dim, mean = rep(0, n_dim), sigma = diag(n_dim), df = 5)
```

Arguments

n_dim	Ambient dimension p .
mean	Length- p numeric location vector. Defaults to the zero vector.
sigma	A p -by- p symmetric positive-definite scale matrix. Defaults to the identity.
df	Degrees of freedom ($df > 2$ recommended for finite variance).

Value

An [is_proposal](#) object.

See Also

Other proposals: [is_mvn\(\)](#), [is_uniform\(\)](#), [proposal_uniform\(\)](#)

Examples

```
q <- is_mvt(n_dim = 2L, df = 5)
q
```

is_proposal	<i>An importance-sampling proposal</i>
-------------	--

Description

An `is_proposal` packages a sampler with its corresponding log-density. Pass one to [fit_kld_em\(\)](#) (or [fit_proxymix\(\)](#) with `regime = "kld"`) to plug an alternative proposal into the regime-(iii) loop.

Usage

```
is_proposal(
  n_dim = integer(0),
  sample = NULL,
  log_density = NULL,
  name = "is_proposal",
  metadata = list()
)
```

Arguments

n_dim	Integer scalar — the ambient dimension p (the property is called <code>n_dim</code> rather than <code>dim</code> because <code>S7</code> reserves <code>dim</code> as an attribute name).
sample	Function: <code>function(n)</code> returning an n by p numeric matrix of independent draws.
log_density	Function: <code>function(x)</code> taking a numeric matrix n by p and returning a length- n numeric vector of $\log q(x)$.
name	Human-readable name.
metadata	Optional list of additional descriptors.

Value

An S7 object of class `is_proposal`.

See Also

Other classes: `autoplot.gmm_fit`, `glance.gmm_fit`, `gmm()`, `gmm_counterfactual_law()`, `gmm_dim()`, `gmm_fit()`, `gmm_n_components()`, `gmm_target()`, `gmm_weights()`, `tidy.gmm`

Examples

```
q <- is_mvn(n_dim = 2L, mean = c(0, 0), cov = diag(2))
q
```

is_uniform	<i>Uniform-on-a-box proposal</i>
------------	----------------------------------

Description

Builds an [is_proposal](#) that samples uniformly on the hyperrectangle $[\text{lower}_1, \text{upper}_1] \times \dots \times [\text{lower}_p, \text{upper}_p]$ and reports the (constant) log-density on that box. Outside the box the log-density is $-\text{Inf}$.

Usage

```
is_uniform(n_dim, lower = -1, upper = 1)
```

Arguments

n_dim	Ambient dimension p .
lower	Length- p numeric vector of lower bounds (recycled from a single value).
upper	Length- p numeric vector of upper bounds (recycled from a single value).

Value

An [is_proposal](#) object.

See Also

Other proposals: `is_mvn()`, `is_mvt()`, `proposal_uniform()`

Examples

```
q <- is_uniform(n_dim = 2L, lower = -5, upper = 5)
q
q@sample(3L)
```

kld_trace	<i>Per-iteration KLD trace of a fit</i>
-----------	---

Description

Returns the per-iteration estimate of $KL(f \parallel g_{\theta})$ produced during a regime-(iii) fit, or NA for regimes that do not estimate the KLD internally.

Usage

```
kld_trace(fit)
```

Arguments

fit A [gmm_fit](#).

Value

Numeric vector (or NA_real_).

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [proxy_functional_ci\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                   is_size = 1000L, max_iter = 15L, seed = 1L)
kld_trace(fit)
```

maxent_target	<i>Maximum-entropy target under moment and support constraints</i>
---------------	--

Description

Builds the maximum-entropy [gmm_target](#) consistent with the supplied constraints, the least-committal density given what is known. The maximum- entropy density under linear constraints is an exponential family $p(x) \propto \exp(\eta^\top T(x))$ on the support, and three cases admit an exact closed form:

Usage

```
maxent_target(moments = NULL, support = NULL, name = "maxent_target")
```

Arguments

moments	Either NULL (only a support constraint, giving the uniform) or a list with mean (length-p numeric) and cov (a p-by-p symmetric positive-definite matrix). Supplying mean without cov (or vice versa) is an error.
support	NULL for the full $\mathbb{R}^p$ (only valid with second-moment moments, giving the Gaussian), or a list <code>list(lower = , upper =)</code> of per-coordinate box bounds (each length 1, recycled, or length p). The uniform case requires finite bounds.
name	Human-readable name.

Details

- **First- and second-moment constraints on full support** – the Gaussian $\mathcal{N}(\text{mean}, \text{cov})$. The moment constraints are realised exactly, so a regime-(i) moment match recovers the target, and the target carries its moments in metadata for that purpose.
- **First- and second-moment constraints on a box support** – the same canonical Gaussian form restricted to the box, a *truncated Gaussian*. The normaliser is closed-form when cov is diagonal (a product of univariate Gaussian box probabilities) and the target is then exactly normalised; otherwise the target is declared unnormalised and regime (iii) fits it up to the unknown constant. Truncation shifts the realised moments inward, so the truncated density's mean and covariance are not mean and cov; it is the canonical-form maximum-entropy density on the box, not the moment-matched one.
- **A support constraint alone (no moments) on a finite box** – the uniform density, the maximum-entropy density on a compact support. Its differential entropy is exactly $\log \text{vol}(\text{box})$, the largest attainable on that support.

The bounded-support cases declare their support, so `fit_kld_em()` (and `fit_proxymix()` with `regime = "kld"`) selects a support-matched uniform importance proposal automatically. Together with the Gaussian as the least-committal full-support density and the `epanechnikov_target()` as a compact-support obstruction, these complete a family of principled test targets.

Value

A `gmm_target`.

References

Jaynes, E. T. (1957) Information theory and statistical mechanics. *Physical Review* 106(4), 620–630. doi:10.1103/PhysRev.106.620

See Also

Other targets: `banana_target()`, `donut_target()`, `epanechnikov_target()`, `gmm_target_from_samples()`, `mixture_target()`

Examples

```
## Full-support second-moment maximum entropy is the Gaussian.
g <- maxent_target(moments = list(mean = c(0, 0), cov = diag(2)))
g@log_density(matrix(c(0, 0), nrow = 1L))
```

```
## Support alone on a box is the uniform.
u <- maxent_target(support = list(lower = 0, upper = 1))
exp(u@log_density(matrix(c(0.5), nrow = 1L)))

## Second moments on a box is a truncated Gaussian, fit via regime (iii).
tg <- maxent_target(moments = list(mean = 0, cov = matrix(1)),
                    support = list(lower = -2, upper = 2))
tg
```

mechanism	<i>Missingness mechanisms for multiple imputation</i>
-----------	---

Description

Specifications passed to the mechanism argument of `gmm_impute()`. They describe how an entry came to be missing, which sets the conditional the missing value is drawn from.

Usage

```
mar()

mnar(coord, beta, link = c("logit", "probit"))

censored(coord, lower = -Inf, upper = Inf)
```

Arguments

coord	Name or index of the single coordinate the mechanism acts on.
beta	Sensitivity slope of the log-odds (or probit score) of being missing in the missing value itself. Positive beta makes larger values more likely to be missing.
link	Selection link, "logit" (the default) or "probit".
lower, upper	Bounds of the interval a censored missing entry is known to lie in. At least one must be finite.

Details

`mar()` is missing at random: the probability that an entry is missing may depend on the observed entries but not on the missing value, so the imputation conditional is the plain mixture conditional. This is the default.

`censored()` is a known interval. A missing entry of `coord` is known only to lie in `[lower, upper]` – a detection limit (`upper = LOD`), a ceiling (`lower = cap`), or interval censoring. The imputation conditional is the mixture conditional truncated to that interval, which `proxymix` evaluates in closed form, so the imputations respect the bound instead of substituting a constant such as half the detection limit.

`mnar()` is missing not at random through a selection model: an entry of `coord` is missing with probability $g(\alpha + \beta y)$ in its own unobserved value y , where g is the logistic or normal link. The

slope β is the sensitivity parameter and is supplied, not estimated. The observed data inform the slope only through the assumed shape of the outcome distribution, so the appropriate use is to posit β , propagate it, and report how conclusions move with it (see [proxy_mnar_sensitivity\(\)](#), which also reports the observed-data log-likelihood). The intercept is calibrated to the observed missingness rate. $\beta = 0$ is missing at random.

Value

A `proxymix_gate` object for [gmm_impute\(\)](#).

See Also

[gmm_impute\(\)](#), [proxy_mnar_sensitivity\(\)](#).

Other imputation: [as_mids\(\)](#), [gmm_complete\(\)](#), [gmm_imputation\(\)](#), [gmm_impute\(\)](#), [proxy_fmi\(\)](#), [proxy_mnar_sensitivity\(\)](#), [proxy_pool\(\)](#)

Examples

```
mar()
censored("y", upper = 0.5)      # a lower detection limit at 0.5
mnar("y", beta = 0.8)          # larger y more likely missing
```

mixture_target	<i>Three-component Gaussian-mixture target</i>
----------------	--

Description

A toy three-component planar mixture target where everything is known exactly: the `log_density` matches the GMM density formula and the attached samples are drawn from that same mixture. Useful for sanity- checking the three fitting regimes against ground truth.

Usage

```
mixture_target(with_samples = FALSE, n = 2000L, seed = 1L)
```

Arguments

<code>with_samples</code>	If TRUE, attach <code>n</code> exact mixture draws.
<code>n</code>	Number of samples to attach when <code>with_samples = TRUE</code> .
<code>seed</code>	Optional integer seed used when drawing the samples.

Value

A [gmm_target](#) in dimension 2.

See Also

Other targets: [banana_target\(\)](#), [donut_target\(\)](#), [epanechnikov_target\(\)](#), [gmm_target_from_samples\(\)](#), [maxent_target\(\)](#)

Examples

```
m <- mixture_target(with_samples = TRUE, n = 100L)
m
```

multi_start_best_of	<i>Multi-start best-of wrapper</i>
---------------------	------------------------------------

Description

Runs the supplied fitter from each of several initialisations and returns the fit with the best score, following Karlis and Xekalaki (2003)'s recommendation.

Usage

```
multi_start_best_of(fit_fn, inits, score_fn, ...)
```

Arguments

fit_fn	A function with signature <code>function(init, ...)</code> returning a gmm_fit .
inits	A list of gmm initialisations.
score_fn	A function <code>function(fit)</code> returning a numeric score — <i>larger is better</i> (typically the final log-target evidence).
...	Additional arguments forwarded to <code>fit_fn</code> .

Value

The [gmm_fit](#) with the largest `score_fn(fit)`.

See Also

Other init: [init_kmeans\(\)](#), [init_moment_seed\(\)](#), [init_random\(\)](#), [init_warm_start\(\)](#)

Examples

```
x <- matrix(stats::rnorm(200), ncol = 2)
tgt <- gmm_target_from_samples(x)
inits <- list(init_random(2L, 2L, seed = 1L),
             init_moment_seed(x, N = 2L))
best <- multi_start_best_of(
  fit_fn = function(init, ...) fit_em_samples(tgt, init = init, ...),
  inits = inits,
  score_fn = function(fit) fit@diagnostics$loglik_final,
```

```

    max_iter = 25L
  )
  best@diagnostics$loglik_final

```

pgmm

Distribution and quantile functions of a one-dimensional mixture

Description

The exact cumulative distribution function $F(x) = \sum_k w_k \Phi((x - \mu_k)/\sigma_k)$ of a one-dimensional Gaussian mixture, and its inverse by monotone root-finding. Together with [dgmm\(\)](#) and [rgmm\(\)](#) these complete the usual d/p/q/r quartet for the one-dimensional case; for tail probabilities of a multivariate mixture, marginalise first ([gmm_marginalise\(\)](#)) or push through the relevant linear functional ([gmm_affine\(\)](#)).

Usage

```

pgmm(q, g, lower.tail = TRUE)

qgmm(p, g)

```

Arguments

q	Numeric vector of quantiles.
g	A one-dimensional gmm (or gmm_fit).
lower.tail	Logical; if TRUE (default), probabilities are $P(X \leq x)$.
p	Numeric vector of probabilities in $(0, 1)$.

Value

A numeric vector the length of the first argument.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [rgmm\(\)](#)

Examples

```

g <- gmm(weights = c(0.4, 0.6), means = list(-2, 1),
         covariances = list(matrix(0.5), matrix(1)))
pgmm(c(-2, 0, 2), g)
qgmm(c(0.1, 0.5, 0.9), g)

```

proposal_uniform	<i>Preferred names for the importance-proposal constructors</i>
------------------	---

Description

proposal_uniform(), proposal_mvn(), and proposal_mvt() are the preferred names of [is_uniform\(\)](#), [is_mvn\(\)](#), and [is_mvt\(\)](#): the historical is_* prefix reads as a logical predicate, which these constructors are not. The is_* names remain available as aliases and are not scheduled for removal.

Usage

```
proposal_uniform(n_dim, lower = -1, upper = 1)

proposal_mvn(n_dim, mean = rep(0, n_dim), cov = diag(n_dim))

proposal_mvt(n_dim, mean = rep(0, n_dim), sigma = diag(n_dim), df = 5)
```

Arguments

n_dim	Ambient dimension p.
lower	Length-p numeric vector of lower bounds (recycled from a single value).
upper	Length-p numeric vector of upper bounds (recycled from a single value).
mean, cov	Forwarded to is_mvn() .
sigma, df	Forwarded to is_mvt() .

Value

An [is_proposal](#) object.

See Also

Other proposals: [is_mvn\(\)](#), [is_mvt\(\)](#), [is_uniform\(\)](#)

Examples

```
proposal_uniform(n_dim = 2L, lower = -5, upper = 5)
proposal_mvn(n_dim = 2L, mean = c(0, 0), cov = diag(2))
proposal_mvt(n_dim = 2L, df = 5) # heavier tails; a robust default
```

proxy_cate	<i>Heterogeneous treatment effects (CATE / uplift)</i>
------------	--

Description

Per-unit conditional average treatment effect $\tau(x) = E[Y \mid do(T = t_1), X = x] - E[Y \mid do(T = t_0), X = x]$, read in closed form off the fitted joint mixture. Under the model's default "ignorability" assumption this is the contrast of two component-gated conditional means; under "latent_confounder" it is the regime-gated within-class slope (the do-operator). The two coincide when treatment carries no information about the regime beyond X ; their difference is [proxy_confounding_gap\(\)](#).

Usage

```
proxy_cate(
  model,
  newdata,
  t1 = NULL,
  t0 = NULL,
  se = TRUE,
  se_method = c("delta", "mc"),
  level = 0.95,
  B = 200L,
  scale = c("link", "response"),
  threshold = 0.5,
  ...
)
```

Arguments

model	An uplift_model .
newdata	A data frame carrying the covariate columns.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (<code>model@treatment_levels</code>); a value matching neither observed level aborts.
se	Logical – compute standard errors and confidence intervals.
se_method	One of "delta" (closed form, the default) or "mc" (resampling).
level	Confidence level for the interval. Default 0.95.
B	Number of bootstrap refits when <code>se_method = "mc"</code> . Default 200.
scale	One of "link" (the latent / continuous scale, the default) or "response". For a binary outcome the response scale reports the effect on the discretised predictive probability $P(Y > \text{threshold})$; for continuous and count outcomes the two scales coincide.
threshold	Decision threshold for the binary discretised predictive. Default 0.5.
...	Forwarded to fit_proxymix() inside the "mc" refits.

Details

The default delta-method standard error is the within-component prediction variance, holding the regime gate fixed; it reduces to the ordinary least-squares standard error of the treatment effect at $K = 1$. Set `se_method = "mc"` for a resampling standard error that also reflects gate uncertainty.

Value

A `data.table::data.table` with columns `id`, `tau`, `se`, `ci_lo`, `ci_hi`, `overlap_flag`.

See Also

`proxy_decide()`, `proxy_confounding_gap()`, `proxy_overlap()`

Other decision: `fit_uplift()`, `gmm_cf_mean()`, `gmm_cf_tail_prob()`, `gmm_cf_variance()`, `gmm_counterfactual()`, `gmm_intervene()`, `proxy_confounding_gap()`, `proxy_decide()`, `proxy_identification_report()`, `proxy_overlap()`, `proxy_policy_value()`, `proxy_predict()`, `proxy_regime_segments()`, `proxy_retrospective_uplift()`, `proxy_uplift()`, `uplift_identification()`, `uplift_model()`

Examples

```
set.seed(1)
n <- 400L
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.5)
y <- 1 + 0.5 * x + (1 + x) * t + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 50L, seed = 1L)
proxy_cate(m, newdata = data.frame(x = c(-1, 0, 1)))
```

`proxy_confounding_gap` *Confounding gap: the sensitivity of the effect to the latent regime*

Description

Per-unit difference between the ignorability-mode and do-mode effects, $\Delta(x) = \tau_{\text{obs}}(x) - \tau_{\text{do}}(x)$. Under ignorability the two coincide and $\Delta \equiv 0$; a non-zero gap is a **sensitivity signal** – how much the estimated effect would move if a fitted regime confounded treatment and outcome beyond X – not a correction the data licenses.

Usage

```
proxy_confounding_gap(model, newdata, t1 = NULL, t0 = NULL)
```

Arguments

model	An uplift_model .
newdata	A data frame carrying the covariate columns.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (model@treatment_levels); a value matching neither observed level aborts.

Details

When each regime holds a single treatment arm, as often happens with a binary treatment, the within-regime effect is zero, the gap equals the whole estimated effect, and a warning of class `proxymix_constant_treatment` is raised.

Value

A [data.table::data.table](#) with columns `id`, `tau_obs`, `tau_do`, `gap`, `overlap_flag`.

See Also

[proxy_cate\(\)](#), [proxy_identification_report\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 600L
u <- stats::rbinom(n, 1L, 0.5) # unobserved group
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.3 + 0.4 * u)
y <- 1 + 5 * u + 0.5 * t + x + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 80L, seed = 1L)
proxy_confounding_gap(m, data.frame(x = c(-1, 0, 1)))
```

proxy_decide

Optimal action and expected incremental value per unit

Description

Turns the per-unit treatment effect into a next-best action under a linear value model: treat when the value of the effect exceeds the cost, i.e. $d^*(x) = 1\{\text{value} \cdot \tau(x) > \text{cost}\}$, with expected incremental value $\text{value} \cdot \tau(x) - \text{cost}$. The standard error of tau is propagated to an action-flip probability – the chance the recommended action would reverse under sampling noise.

Usage

```
proxy_decide(
  model,
  newdata,
  value,
  cost = 0,
  t1 = NULL,
  t0 = NULL,
  se_method = c("delta", "mc"),
  ...
)
```

Arguments

model	An uplift_model .
newdata	A data frame carrying the covariate columns.
value	Numeric scalar – the value of one unit of outcome.
cost	Numeric scalar – the cost of treating one unit. Default 0.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (model@treatment_levels); a value matching neither observed level aborts.
se_method	One of "delta" (default) or "mc".
...	Forwarded to proxy_cate() .

Value

A [data.table::data.table](#) with columns id, action, expected_value, tau, se, flip_prob, overlap_flag.

See Also

[proxy_cate\(\)](#), [proxy_policy_value\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 400L
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.5)
y <- 1 + (x > 0) * t + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 50L, seed = 1L)
proxy_decide(m, data.frame(x = c(-1, 1)), value = 1, cost = 0.2)
```

proxy_fmi	<i>Fraction of missing information for a column mean</i>
-----------	--

Description

The share of a column mean's total variance attributable to the missing data, read from `proxy_pool()`.

Usage

```
proxy_fmi(object, column, method = c("analytic", "rubin"))
```

Arguments

object	A <code>gmm_imputation</code> .
column	Name of a single numeric column whose mean is pooled.
method	"analytic" (the default) for the closed-form pooling, or "rubin" for Rubin's rules over the drawn completions.

Value

A named numeric scalar.

See Also

`proxy_pool()`.

Other imputation: `as_mids()`, `gmm_complete()`, `gmm_imputation()`, `gmm_impute()`, `mechanism`, `proxy_mnar_sensitivity()`, `proxy_pool()`

Examples

```
set.seed(1)
x1 <- rnorm(150); x2 <- x1 + rnorm(150); x2[runif(150) < 0.3] <- NA
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
proxy_fmi(imp, "x2")
```

proxy_functional_ci	<i>Percentile interval for any functional of a fitted proxy</i>
---------------------	---

Description

Applies a functional to every member of a bootstrap ensemble and returns the base fit's point estimate with percentile confidence limits. The functional may return a scalar or a fixed-length numeric vector – the moments (`gmm_mean()`, `gmm_cov()`), a tail probability via `pgmm()` on a marginal, an entropy, a conditional mean, or any composition of the operator calculus.

Usage

```
proxy_functional_ci(ensemble, fn, level = 0.9, ..., tail_warn = 0.001)
```

Arguments

ensemble	A gmm_ensemble from gmm_fit_ensemble() .
fn	A function mapping a gmm to a numeric scalar or vector.
level	Confidence level. Default 0.9.
...	Forwarded to fn.
tail_warn	Threshold for the tail-probability warning described in Details. Default 1e-3; NULL or 0 turns the warning off.

Details

The interval measures how much the functional moves when the fit is repeated on re-weighted draws. It does not measure how far the Gaussian-mixture family itself is from the target, so it can exclude the target's true value when the mixture has too few components. The gap is widest far in a tail: a mixture of Gaussians has Gaussian tails and can misstate a small tail probability by orders of magnitude while the interval stays narrow. A warning of class `proxymix_tail_functional` is raised when every value of `fn` (at the base fit and at every member) lies in $[0, 1]$ and some element of the base fit's estimate, or its complement, is below `tail_warn`. Check such a probability against numerical integration of the target or a method built for tails.

Value

A data frame with one row per element of `fn`'s value: `term`, `estimate` (the base fit's value), `conf.low`, `conf.high`.

See Also

Other diagnostics: [bic_aic\(\)](#), [ess_summary\(\)](#), [ess_trace\(\)](#), [gmm_anneal_path\(\)](#), [gmm_conditional_entropy\(\)](#), [gmm_entropy\(\)](#), [gmm_evidence\(\)](#), [gmm_fit_ensemble\(\)](#), [gmm_fit_quality\(\)](#), [gmm_independence_graph\(\)](#), [gmm_mutual_information\(\)](#), [hellinger_mc\(\)](#), [kld_trace\(\)](#)

Examples

```
fit <- fit_proxymix(banana_target(), N = 2L, regime = "kld",
                  is_size = 1500L, max_iter = 20L, seed = 1L)
ens <- gmm_fit_ensemble(fit, B = 30L, seed = 2L)
proxy_functional_ci(ens, function(g) gmm_mean(g)[1L])
```

proxy_identification_report

The identification report (an executive one-pager)

Description

The differentiator: a structured audit of what the decision model identifies, what it assumes, and what it cannot answer. Carries the estimand, the identification regime and its requirement, the overlap rate on the supplied population, the confounding-gap magnitude (the value at risk from unobserved confounding), and the explicit non-identification of the individual counterfactual law.

Usage

```
proxy_identification_report(model, newdata, t1 = NULL, t0 = NULL)
```

Arguments

model	An uplift_model .
newdata	A data frame carrying the covariate columns – the population the report is computed over.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (model@treatment_levels); a value matching neither observed level aborts.

Value

An S7 object of class uplift_identification with a print method.

See Also

[proxy_confounding_gap\(\)](#), [proxy_overlap\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 600L
u <- stats::rbinom(n, 1L, 0.5) # unobserved group
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.3 + 0.4 * u)
y <- 1 + 5 * u + 0.5 * t + x + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 80L, seed = 1L)
proxy_identification_report(m, data.frame(x = stats::rnorm(100)))
```

proxy_mnar_sensitivity

Missing-not-at-random sensitivity analysis for a coordinate mean

Description

Sweeps the missing-not-at-random sensitivity slope β over a grid and, at each value, multiply-imputes `coord` under the selection model $P(\text{missing} \mid y) = g(\alpha + \beta y)$ and pools its mean by Rubin's rules. The result traces how the estimate and its confidence interval move as the assumed dependence of missingness on the unobserved value strengthens, so an analyst can read off the value of β at which a conclusion would change. $\beta = 0$ is the missing-at-random anchor.

Usage

```
proxy_mnar_sensitivity(
  data,
  coord,
  beta_grid = seq(0, 1, by = 0.25),
  link = c("logit", "probit"),
  N = NULL,
  m = 20L,
  seed = NULL,
  max_iter = 500L,
  ...
)
```

Arguments

<code>data</code>	A numeric matrix or data frame with NA in <code>coord</code> only (its other columns must be fully observed).
<code>coord</code>	Name or index of the coordinate the mechanism acts on.
<code>beta_grid</code>	Numeric vector of sensitivity slopes. Positive values make larger unobserved values more likely to be missing.
<code>link</code>	Selection link, "logit" (the default) or "probit".
<code>N, m, seed, ...</code>	Passed to <code>gmm_impute()</code> ; a single seed makes the whole sweep reproducible and keeps the curve smooth across the grid.
<code>max_iter</code>	Maximum EM iterations per fit, passed to <code>gmm_impute()</code> . Default 500L; fits near missing at random can need more than the <code>gmm_impute()</code> default of 100.

Details

The mixture is refitted at each slope. Under this selection model the observed data carry information about the slope, but only through the assumed shape of the outcome distribution; a different shape can fit the observed data equally well at a different slope. The `loglik` column shows how strongly the data favour one slope over another under the mixture. Treat a large difference as a consequence of the model's shape, not as evidence about the missingness mechanism itself, and report the whole curve.

Value

A data frame with one row per grid value: `beta`, `estimate`, `std.error`, `conf.low`, `conf.high`, `fmi`, `loglik` (the observed-data log-likelihood of the selection model at the fitted mixture and intercept) and `converged` (whether the point fit converged). A warning names any slope whose fit did not converge.

See Also

[gmm_impute\(\)](#), [mnar\(\)](#).

Other imputation: [as_mids\(\)](#), [gmm_complete\(\)](#), [gmm_imputation\(\)](#), [gmm_impute\(\)](#), [mechanism](#), [proxy_fmi\(\)](#), [proxy_pool\(\)](#)

Examples

```
set.seed(1)
x1 <- rnorm(200)
y <- x1 + rnorm(200)
y[runif(200) < plogis(-0.4 + 0.8 * y)] <- NA      # MNAR on y
dat <- data.frame(x1 = x1, y = y)
proxy_mnar_sensitivity(dat, "y", beta_grid = c(0, 0.5, 1), m = 5L, seed = 1L)
```

proxy_overlap

Per-unit overlap / positivity diagnostic

Description

Flags units whose (treatment, covariate) configuration is poorly covered by the fitted joint – the proxy’s mass coverage is the positivity diagnostic. For each treatment arm the squared Mahalanobis distance to the nearest regime centre is converted to an upper-tail chi-square coverage probability; the reported coverage is the minimum across arms, since the treatment effect needs both arms supported. Units below `floor` are flagged and excluded from [proxy_policy_value\(\)](#) by default.

Usage

```
proxy_overlap(model, newdata, t1 = NULL, t0 = NULL, floor = 0.01)
```

Arguments

<code>model</code>	An uplift_model .
<code>newdata</code>	A data frame carrying the covariate columns.
<code>t1, t0</code>	The treated and control treatment values. Default the treatment levels observed at fit time (<code>model@treatment_levels</code>); a value matching neither observed level aborts.
<code>floor</code>	Coverage probability below which a unit is flagged. Default 0.01.

Value

A `data.table::data.table` with columns `id`, `coverage`, `overlap_flag`.

See Also

Other decision: `fit_uplift()`, `gmm_cf_mean()`, `gmm_cf_tail_prob()`, `gmm_cf_variance()`, `gmm_counterfactual()`, `gmm_intervene()`, `proxy_cate()`, `proxy_confounding_gap()`, `proxy_decide()`, `proxy_identification_report()`, `proxy_policy_value()`, `proxy_predict()`, `proxy_regime_segments()`, `proxy_retrospective_uplift()`, `proxy_uplift()`, `uplift_identification()`, `uplift_model()`

Examples

```
set.seed(1)
dat <- data.frame(y = stats::rnorm(200), t = stats::rbinom(200, 1L, 0.5),
                  x = stats::rnorm(200))
m <- fit_uplift(dat, "y", "t", "x", N = 1L, regime = "moment")
proxy_overlap(m, newdata = data.frame(x = c(0, 8)))
```

proxy_policy_value	<i>Off-line value of a targeting policy</i>
--------------------	---

Description

Estimates the expected value of deploying a per-unit targeting policy, $V(d) = E_X[\text{value} \cdot E[Y \mid \text{do}(T = d(X)), X] - \text{cost} \cdot d(X)]$, from the fitted model alone – no live A/B test. Units that fail the overlap diagnostic are excluded by default and their count is reported, never silently dropped.

Usage

```
proxy_policy_value(
  model,
  newdata,
  policy,
  value,
  cost = 0,
  t1 = NULL,
  t0 = NULL,
  exclude_low_overlap = TRUE
)
```

Arguments

<code>model</code>	An <code>uplift_model</code> .
<code>newdata</code>	A data frame carrying the covariate columns – the population the policy would be deployed on.

policy	A per-unit action specification: a 0/1 vector of length <code>nrow(newdata)</code> , a function of the <code>proxy_cate()</code> table returning actions, or one of the strings "all", "none", "optimal".
value	Numeric scalar – the value of one unit of outcome.
cost	Numeric scalar – the cost of treating one unit. Default 0.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (<code>model@treatment_levels</code>); a value matching neither observed level aborts.
exclude_low_overlap	Logical – drop overlap-flagged units from the average (and report the count). Default TRUE.

Value

A one-row `data.table::data.table` with columns `policy_value`, `n_used`, `n_excluded`, `n_treated`.

See Also

`proxy_decide()`, `proxy_overlap()`

Other decision: `fit_uplift()`, `gmm_cf_mean()`, `gmm_cf_tail_prob()`, `gmm_cf_variance()`, `gmm_counterfactual()`, `gmm_intervene()`, `proxy_cate()`, `proxy_confounding_gap()`, `proxy_decide()`, `proxy_identification_report()`, `proxy_overlap()`, `proxy_predict()`, `proxy_regime_segments()`, `proxy_retrospective_uplift()`, `proxy_uplift()`, `uplift_identification()`, `uplift_model()`

Examples

```
set.seed(1)
n <- 600L
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.5)
y <- 1 + (0.4 + x) * t + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 80L, seed = 1L)
nd <- data.frame(x = stats::rnorm(200))
proxy_policy_value(m, nd, policy = "optimal", value = 1, cost = 0.3)
```

proxy_pool

Pool a column mean across imputations

Description

Pools the mean of one column over the `m` completed datasets in a `gmm_imputation`, returning the estimate with a standard error, degrees of freedom, confidence interval, and fraction of missing information. The default method = "analytic" computes the between-imputation variance in closed form – the exact $m \rightarrow \infty$ limit, with no Monte-Carlo noise – from the mixture conditional, and splits the total variance into complete-data, imputation, and parameter parts. method = "rubin" instead applies Rubin's rules to the drawn completions (useful as a check).

Usage

```
proxy_pool(object, column, method = c("analytic", "rubin"))
```

Arguments

object	A gmm_imputation .
column	Name of a single numeric column whose mean is pooled.
method	"analytic" (the default) for the closed-form pooling, or "rubin" for Rubin's rules over the drawn completions.

Details

For a regression or any other model estimand, do not pool here: convert the imputations to a mice object with [as_mids\(\)](#) and pool with [mice::pool\(\)](#), which is the established workflow and reports the same diagnostics.

Value

A one-row data frame: term, estimate, std.error, statistic, df, conf.low, conf.high, fmi.

See Also

[proxy_fmi\(\)](#), [as_mids\(\)](#) to pool models with [mice::pool\(\)](#).

Other imputation: [as_mids\(\)](#), [gmm_complete\(\)](#), [gmm_imputation\(\)](#), [gmm_impute\(\)](#), [mechanism](#), [proxy_fmi\(\)](#), [proxy_mnar_sensitivity\(\)](#)

Examples

```
set.seed(1)
x1 <- rnorm(150); x2 <- x1 + rnorm(150)
x2[runif(150) < 0.3] <- NA
imp <- gmm_impute(cbind(x1, x2), N = 1L, m = 10L, seed = 1L)
proxy_pool(imp, "x2") # analytic column mean
```

proxy_predict	<i>Predicted outcome under a treatment (the seeing rung)</i>
---------------	--

Description

Per-unit predicted outcome $E[Y \mid do(T = t), X = x]$ – the first rung of the ladder, risk / response scoring. Under "ignorability" this is the component-gated conditional mean; under "latent_confounder" it is the regime-gated interventional mean. For a binary outcome with scale = "response" the prediction is the discretised predictive probability $P(Y > \text{threshold})$.

Usage

```
proxy_predict(
  model,
  newdata,
  t,
  scale = c("link", "response"),
  threshold = 0.5
)
```

Arguments

model	An uplift_model .
newdata	A data frame carrying the covariate columns.
t	The treatment value to predict the outcome under.
scale	One of "link" (default) or "response".
threshold	Decision threshold for the binary discretised predictive. Default 0.5.

Value

A [data.table::data.table](#) with columns id and prediction.

See Also

[proxy_cate\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
dat <- data.frame(y = stats::rnorm(200), t = stats::rbinom(200, 1L, 0.5),
                  x = stats::rnorm(200))
m <- fit_uplift(dat, "y", "t", "x", N = 1L, regime = "moment")
proxy_predict(m, data.frame(x = c(-1, 0, 1)), t = 1)
```

proxy_regime_segments *The fitted regimes as an interpretable segment table*

Description

Exposes the K mixture components as decision segments: each regime's prevalence (weight), its within-segment treatment effect (the within-class treatment slope), its residual standard deviation, and its covariate centre. This is the interpretable by-product the closed-form reading gives for free.

Usage

```
proxy_regime_segments(model, t1 = NULL, t0 = NULL)
```

Arguments

model	An uplift_model .
t1, t0	The treated and control treatment values used to scale the within-segment effect. Default the treatment levels observed at fit time (model@treatment_levels).

Details

When each regime holds a single treatment arm, as often happens with a binary treatment, every within-segment effect is zero and a warning of class `proxymix_constant_treatment` is raised.

Value

A `data.table::data.table` with columns `regime`, `weight`, `effect`, `sigma`, and one column per covariate centre.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 600L
u <- stats::rbinom(n, 1L, 0.5) # unobserved group
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.3 + 0.4 * u)
y <- 1 + 5 * u + 0.5 * t + x + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 80L, seed = 1L)
proxy_regime_segments(m)
```

proxy_retrospective_uplift

Retrospective (counterfactual-mean) uplift for observed units

Description

For each observed unit (y, t, x) , the counterfactual-mean uplift of moving from t_0 to t_1 , $E[Y_{t_1} | y, t, x] - E[Y_{t_0} | y, t, x]$, computed by [gmm_counterfactual\(\)](#). Unlike [proxy_cate\(\)](#), the abduction gate uses the observed outcome y as well, sharpening the per-unit estimate. Only the counterfactual *mean* is identified; the spread is not (see [gmm_cf_variance\(\)](#)).

Usage

```
proxy_retrospective_uplift(model, observed, t1 = NULL, t0 = NULL)
```

Arguments

model	An uplift_model .
observed	A data frame carrying the outcome, treatment, and covariate columns of the observed units.
t1, t0	The treated and control treatment values. Default the treatment levels observed at fit time (model@treatment_levels); a value matching neither observed level aborts.

Value

A [data.table::data.table](#) with columns id, y_obs, t_obs, cf_mean_t1, retro_uplift.

See Also

[gmm_counterfactual\(\)](#), [proxy_cate\(\)](#)

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
n <- 400L
x <- stats::rnorm(n)
t <- stats::rbinom(n, 1L, 0.5)
y <- 1 + (0.5 + x) * t + stats::rnorm(n, sd = 0.5)
dat <- data.frame(y = y, t = t, x = x)
m <- fit_uplift(dat, "y", "t", "x", N = 2L, regime = "sample",
               max_iter = 80L, seed = 1L)
proxy_retrospective_uplift(m, observed = dat[1:5, ])
```

proxy_uplift

Uplift (alias of [proxy_cate\(\)](#) for a binary treatment)

Description

For a binary treatment, the uplift is exactly the conditional average treatment effect. This is a thin alias of [proxy_cate\(\)](#) kept for the next-best-action vocabulary.

Usage

```
proxy_uplift(model, newdata, ...)
```

Arguments

`model` An [uplift_model](#).
`newdata` A data frame carrying the covariate columns.
`...` Forwarded to [fit_proxymix\(\)](#) inside the "mc" refits.

Value

A `data.table::data.table` – see [proxy_cate\(\)](#).

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift_identification\(\)](#), [uplift_model\(\)](#)

Examples

```
set.seed(1)
dat <- data.frame(y = stats::rnorm(200), t = stats::rbinom(200, 1L, 0.5),
                  x = stats::rnorm(200))
m <- fit_uplift(dat, "y", "t", "x", N = 1L, regime = "moment")
proxy_uplift(m, newdata = data.frame(x = 0))
```

rgmm

Sample from a Gaussian mixture

Description

Draws `n` independent samples from a Gaussian mixture.

Usage

```
rgmm(n, g)
```

Arguments

`n` Number of samples (positive integer scalar).
`g` A [gmm](#) (or [gmm_fit](#)) object.

Value

A numeric matrix of dimension `n` by `p`.

See Also

Other ops: [dgmm\(\)](#), [gmm_canonicalise\(\)](#), [gmm_conditionalise\(\)](#), [gmm_divergence\(\)](#), [gmm_kld\(\)](#), [gmm_marginalise\(\)](#), [gmm_mean\(\)](#), [gmm_modes\(\)](#), [pgmm\(\)](#)

Examples

```
g <- gmm(weights = c(0.5, 0.5),
          means = list(c(-1, 0), c(1, 0)),
          covariances = list(diag(2), diag(2)))
x <- rgmm(50L, g)
dim(x)
```

select_N

*Select the number of mixture components***Description**

Fits every candidate component count and chooses one by a regime-appropriate criterion. With samples (regime ii) the choice is the smallest BIC. With an evaluable-only target (regime iii) each candidate is scored by its held-out validation KLD – an independent importance draw the fit never trained on – and the choice follows the one-standard-error rule: the smallest N whose validation score is within one Monte Carlo standard error of the best score. The scored table is returned alongside the choice, and callers who prefer to choose by eye can ignore the recommendation.

Usage

```
select_N(
  target,
  candidates = 1:6,
  regime = c("auto", "sample", "kld"),
  seed = NULL,
  ...
)
```

Arguments

target	A gmm_target .
candidates	Integer vector of component counts to try.
regime	"auto" (default: "sample" when the target carries samples, "kld" otherwise), "sample", or "kld".
seed	Optional integer seed shared across candidates (paired fits).
...	Forwarded to the regime's fitter (fit_em_samples() or fit_kld_em()).

Value

A list of class proxymix_selection: best_n (the chosen count), best_fit (its [gmm_fit](#)), and table (one row per candidate with the criterion values).

See Also

Other fitting: [fit_em_samples\(\)](#), [fit_kld_em\(\)](#), [fit_moment_match\(\)](#), [from_kde\(\)](#), [from_objective\(\)](#)

Examples

```
sel <- select_N(banana_target(), candidates = 1:3,
               is_size = 1500L, max_iter = 20L, seed = 1L)
sel$table
sel$best_n
```

tidy.gmm

*Tidy a Gaussian mixture into a component table***Description**

A broom-style `tidy()` method: one row per component, with the weight, the mean coordinates (`mean_1, ...`), and the marginal variances (`var_1, ...`). Available as `generics::tidy(g)` (or `broom::tidy(g)`) when the `generics` package is installed.

Value

A data frame with K rows.

Arguments

Called as `generics::tidy(x, ...)`, where `x` is a [gmm](#) (or [gmm_fit](#)) and `...` is ignored.

See Also

Other classes: [autoplot.gmm_fit](#), [glance.gmm_fit](#), [gmm\(\)](#), [gmm_counterfactual_law\(\)](#), [gmm_dim\(\)](#), [gmm_fit\(\)](#), [gmm_n_components\(\)](#), [gmm_target\(\)](#), [gmm_weights\(\)](#), [is_proposal\(\)](#)

Examples

```
g <- gmm(weights = c(0.3, 0.7), means = list(c(-1, 0), c(2, 1)),
         covariances = list(diag(2), 0.5 * diag(2)))
generics::tidy(g)
```

uplift_identification *Identification-report object***Description**

The structured return type of [proxy_identification_report\(\)](#). Print it for the executive one-pager.

Usage

```
uplift_identification(  
  estimand = character(0),  
  assume = character(0),  
  n_units = integer(0),  
  overlap_pct = numeric(0),  
  confounding_gap_mean = numeric(0),  
  confounding_gap_max = numeric(0),  
  K = integer(0),  
  outcome_type = "continuous"  
)
```

Arguments

- estimand Character – the target estimand.
- assume Character – the identification regime.
- n_units Integer – units the report covers.
- overlap_pct Numeric – percentage of units with adequate overlap.
- confounding_gap_mean, confounding_gap_max
 Numeric – mean and maximum absolute confounding gap over the population.
- K Integer – the number of fitted regimes.
- outcome_type Character – the outcome scale.

Value

An S7 object of class `uplift_identification`.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_model\(\)](#)

<code>uplift_model</code>	<i>A fitted uplift / next-best-action model</i>
---------------------------	---

Description

The object returned by `fit_uplift()`: a joint `gmm_fit` over (outcome, treatment, covariates) together with the column roles, the identification assumption it will be read under, the outcome type, and the training sample (retained so that resampling standard errors and overlap diagnostics are available). The decision verbs dispatch on this class.

Usage

```
uplift_model(
  fit = NULL,
  roles = list(),
  assume = "ignorability",
  outcome_type = "continuous",
  data = NULL,
  n_train = integer(0),
  treatment_levels = c(0, 1),
  name = "uplift_model",
  metadata = list()
)
```

Arguments

<code>fit</code>	The joint gmm_fit over the stacked (outcome, treatment, covariates) coordinates, in that column order.
<code>roles</code>	A list with integer indices outcome, treatment, covariate and the matching outcome_name, treatment_name, covariate_names.
<code>assume</code>	One of "ignorability" (the default uplift assumption) or "latent_confounder" (the do-operator reading; always flagged).
<code>outcome_type</code>	One of "continuous", "binary", "count".
<code>data</code>	The n by p numeric training matrix cbind(outcome, treatment, covariates).
<code>n_train</code>	Integer scalar – the training sample size.
<code>treatment_levels</code>	Numeric length-2 vector c(t0, t1) – the control and treated values used at fit time (the observed treatment arms).
<code>name</code>	Human-readable name.
<code>metadata</code>	Optional list of descriptors (e.g. the K-selection trace).

Value

An S7 object of class `uplift_model`.

See Also

Other decision: [fit_uplift\(\)](#), [gmm_cf_mean\(\)](#), [gmm_cf_tail_prob\(\)](#), [gmm_cf_variance\(\)](#), [gmm_counterfactual\(\)](#), [gmm_intervene\(\)](#), [proxy_cate\(\)](#), [proxy_confounding_gap\(\)](#), [proxy_decide\(\)](#), [proxy_identification_report\(\)](#), [proxy_overlap\(\)](#), [proxy_policy_value\(\)](#), [proxy_predict\(\)](#), [proxy_regime_segments\(\)](#), [proxy_retrospective_uplift\(\)](#), [proxy_uplift\(\)](#), [uplift_identification\(\)](#)

Index

* classes

- `autoplot.gmm_fit`, 5
- `glance.gmm_fit`, 26
- `gmm`, 27
- `gmm_counterfactual_law`, 40
- `gmm_dim`, 41
- `gmm_fit`, 49
- `gmm_n_components`, 64
- `gmm_target`, 69
- `gmm_weights`, 73
- `is_proposal`, 79
- `tidy.gmm`, 105

* decision

- `fit_uplift`, 20
- `gmm_cf_mean`, 33
- `gmm_cf_tail_prob`, 33
- `gmm_cf_variance`, 34
- `gmm_counterfactual`, 39
- `gmm_intervene`, 57
- `proxy_cate`, 88
- `proxy_confounding_gap`, 89
- `proxy_decide`, 90
- `proxy_identification_report`, 94
- `proxy_overlap`, 96
- `proxy_policy_value`, 97
- `proxy_predict`, 99
- `proxy_regime_segments`, 100
- `proxy_retrospective_uplift`, 101
- `proxy_uplift`, 102
- `uplift_identification`, 105
- `uplift_model`, 106

* diagnostics

- `bic_aic`, 7
- `ess_summary`, 11
- `ess_trace`, 12
- `gmm_anneal_path`, 30
- `gmm_conditional_entropy`, 37
- `gmm_entropy`, 43
- `gmm_evidence`, 45

- `gmm_fit_ensemble`, 50
- `gmm_fit_quality`, 51
- `gmm_independence_graph`, 55
- `gmm_mutual_information`, 63
- `hellinger_mc`, 74
- `kld_trace`, 81
- `proxy_functional_ci`, 92

* fitting

- `fit_em_samples`, 12
- `fit_kld_em`, 14
- `fit_moment_match`, 18
- `from_kde`, 22
- `from_objective`, 24
- `select_N`, 104

* imputation

- `as_mids`, 4
- `gmm_complete`, 35
- `gmm_imputation`, 52
- `gmm_impute`, 54
- `mechanism`, 83
- `proxy_fmi`, 92
- `proxy_mnar_sensitivity`, 95
- `proxy_pool`, 98

* init

- `init_kmeans`, 75
- `init_moment_seed`, 75
- `init_random`, 76
- `init_warm_start`, 77
- `multi_start_best_of`, 85

* interface

- `fit_proxymix`, 19

* interop

- `gmm_target_from_posterior`, 71

* operators

- `gmm_affine`, 28
- `gmm_aggregate`, 29
- `gmm_convolve`, 38
- `gmm_counterfactual`, 39
- `gmm_filter`, 47

- gmm_intervene, 57
- gmm_missing, 61
- gmm_mix, 62
- gmm_observe, 65
- gmm_product, 66
- gmm_reduce, 67
- * **ops**
 - dgmm, 8
 - gmm_canonicalise, 32
 - gmm_conditionalise, 36
 - gmm_divergence, 42
 - gmm_kld, 58
 - gmm_marginalise, 59
 - gmm_mean, 60
 - gmm_modes, 62
 - pgmm, 86
 - rgmm, 103
- * **proposals**
 - is_mvn, 78
 - is_mvt, 78
 - is_uniform, 80
 - proposal_uniform, 87
- * **targets**
 - banana_target, 6
 - donut_target, 9
 - epanechnikov_target, 10
 - gmm_target_from_samples, 72
 - maxent_target, 81
 - mixture_target, 84
- as_mids, 4
- as_mids(), 35, 53, 55, 84, 92, 96, 99
- autoplot.gmm_fit, 5, 27, 41, 50, 65, 70, 73, 80, 105
- banana_target, 6
- banana_target(), 9, 10, 73, 82, 85
- bic_aic, 7
- bic_aic(), 11, 12, 31, 37, 43, 46, 51, 52, 56, 64, 74, 81, 93
- censored (mechanism), 83
- censored(), 54, 55
- data.table::data.table, 89–91, 97, 98, 100–103
- dgmm, 8
- dgmm(), 27, 32, 36, 42, 59, 60, 63, 86, 103
- donut_target, 9
- donut_target(), 6, 10, 73, 82, 85
- epanechnikov_target, 10
- epanechnikov_target(), 6, 9, 73, 82, 85
- ess_summary, 11
- ess_summary(), 7, 12, 23, 31, 37, 43, 46, 51, 52, 56, 64, 74, 81, 93
- ess_trace, 12
- ess_trace(), 7, 11, 31, 37, 43, 46, 50–52, 56, 64, 74, 81, 93
- fit_em_samples, 12
- fit_em_samples(), 17–19, 23, 26, 28, 31, 32, 68, 104
- fit_kld_em, 14
- fit_kld_em(), 10, 14, 18, 19, 22–26, 31, 32, 56, 70, 71, 79, 82, 104
- fit_moment_match, 18
- fit_moment_match(), 14, 17, 19, 23, 26, 32, 104
- fit_proxymix, 19
- fit_proxymix(), 5, 10, 21, 32, 49, 71, 79, 82, 88, 103
- fit_uplift, 20
- fit_uplift(), 33–35, 40, 58, 89–91, 94, 97, 98, 100–103, 106, 107
- from_kde, 22
- from_kde(), 14, 17, 18, 26, 104
- from_objective, 24
- from_objective(), 14, 17, 18, 23, 63, 104
- glance.gmm_fit, 6, 26, 27, 41, 50, 65, 70, 73, 80, 105
- gmm, 8, 13, 15, 27, 28, 29, 32, 36–39, 41–44, 47–49, 51, 53, 56–58, 60–69, 73, 75–77, 85, 86, 93, 103, 105
- gmm(), 6, 27, 41, 50, 52, 65, 70, 73, 80, 105
- gmm_affine, 28
- gmm_affine(), 29, 38, 40, 47, 48, 58, 61, 62, 66–69, 86
- gmm_aggregate, 29
- gmm_aggregate(), 29, 38, 40, 48, 58, 61, 62, 66, 67, 69
- gmm_anneal_path, 30
- gmm_anneal_path(), 7, 11–13, 16, 37, 43, 46, 51, 52, 56, 64, 74, 81, 93
- gmm_canonicalise, 32
- gmm_canonicalise(), 8, 13, 16, 18, 19, 23, 36, 42, 59, 60, 63, 86, 103

- `gmm_cf_mean`, 33
- `gmm_cf_mean()`, 21, 34, 35, 40, 58, 89–91, 94, 97, 98, 100–103, 106, 107
- `gmm_cf_tail_prob`, 33
- `gmm_cf_tail_prob()`, 21, 33, 35, 40, 58, 89–91, 94, 97, 98, 100–103, 106, 107
- `gmm_cf_variance`, 34
- `gmm_cf_variance()`, 21, 33, 34, 39, 40, 58, 89–91, 94, 97, 98, 100–103, 106, 107
- `gmm_complete`, 35
- `gmm_complete()`, 4, 52, 53, 55, 84, 92, 96, 99
- `gmm_conditional_entropy`, 37
- `gmm_conditional_entropy()`, 7, 11, 12, 31, 43, 46, 51, 52, 56, 64, 74, 81, 93
- `gmm_conditionalise`, 36
- `gmm_conditionalise()`, 8, 27, 32, 37, 42, 54–56, 59–61, 63, 86, 103
- `gmm_convolve`, 38
- `gmm_convolve()`, 29, 40, 48, 58, 61, 62, 66, 67, 69
- `gmm_counterfactual`, 39
- `gmm_counterfactual()`, 21, 29, 33–35, 38, 40, 48, 58, 61, 62, 66, 67, 69, 89–91, 94, 97, 98, 100–103, 106, 107
- `gmm_counterfactual_law`, 33, 34, 40
- `gmm_counterfactual_law()`, 6, 27, 41, 50, 65, 70, 73, 80, 105
- `gmm_cov (gmm_mean)`, 60
- `gmm_cov()`, 73, 92
- `gmm_covariances (gmm_weights)`, 73
- `gmm_dim`, 41
- `gmm_dim()`, 6, 27, 41, 50, 65, 70, 73, 80, 105
- `gmm_divergence`, 42
- `gmm_divergence()`, 8, 32, 36, 43, 59, 60, 63, 64, 68, 86, 103
- `gmm_entropy`, 43
- `gmm_entropy()`, 7, 11, 12, 31, 37, 42, 46, 51, 52, 56, 64, 74, 81, 93
- `gmm_eos_test`, 44
- `gmm_evidence`, 45
- `gmm_evidence()`, 7, 11, 12, 31, 37, 43, 51, 52, 56, 64, 74, 81, 93
- `gmm_filter`, 47
- `gmm_filter()`, 29, 38, 40, 44, 45, 58, 61, 62, 66, 67, 69
- `gmm_fit`, 5, 7, 8, 11–13, 17–19, 23, 26, 28, 29, 32, 36–39, 41–43, 46, 47, 49, 50, 51, 56–58, 60–68, 73, 74, 77, 81, 85, 86, 103–107
- `gmm_fit()`, 6, 27, 41, 65, 70, 73, 80, 105
- `gmm_fit_ensemble`, 50
- `gmm_fit_ensemble()`, 7, 11, 12, 31, 37, 43, 46, 52, 56, 64, 74, 81, 93
- `gmm_fit_quality`, 51
- `gmm_fit_quality()`, 7, 11, 12, 17, 31, 37, 43, 46, 51, 56, 64, 73, 74, 81, 93
- `gmm_imputation`, 4, 35, 52, 55, 92, 98, 99
- `gmm_imputation()`, 4, 35, 55, 84, 92, 96, 99
- `gmm_impute`, 54
- `gmm_impute()`, 4, 35, 52, 53, 83, 84, 92, 95, 96, 99
- `gmm_independence_graph`, 55
- `gmm_independence_graph()`, 7, 11, 12, 31, 37, 43, 46, 51, 52, 64, 74, 81, 93
- `gmm_intervene`, 57
- `gmm_intervene()`, 21, 29, 33–35, 38, 40, 48, 61, 62, 66, 67, 69, 89–91, 94, 97, 98, 100–103, 106, 107
- `gmm_kld`, 58
- `gmm_kld()`, 8, 32, 36, 42, 43, 60, 63, 86, 103
- `gmm_marginalise`, 59
- `gmm_marginalise()`, 5, 8, 27, 32, 36, 42, 59, 60, 63, 86, 103
- `gmm_mean`, 60
- `gmm_mean()`, 8, 32, 36, 42, 59, 60, 63, 73, 86, 92, 103
- `gmm_means (gmm_weights)`, 73
- `gmm_missing`, 61
- `gmm_missing()`, 29, 38, 40, 48, 58, 62, 66, 67, 69
- `gmm_mix`, 62
- `gmm_mix()`, 29, 38, 40, 48, 58, 61, 66, 67, 69
- `gmm_modes`, 62
- `gmm_modes()`, 8, 24, 26, 32, 36, 42, 59, 60, 86, 103
- `gmm_mutual_information`, 63
- `gmm_mutual_information()`, 7, 11, 12, 31, 37, 43, 46, 51, 52, 56, 74, 81, 93
- `gmm_n_components`, 64
- `gmm_n_components()`, 6, 27, 41, 50, 70, 73, 80, 105
- `gmm_observe`, 65
- `gmm_observe()`, 29, 38, 40, 47, 48, 58, 61, 62, 67–69
- `gmm_product`, 66
- `gmm_product()`, 29, 38, 40, 48, 58, 61, 62, 66,

- 69
- gmm_reduce, 67
- gmm_reduce(), 29, 38, 40, 47, 48, 58, 61, 62, 66, 67
- gmm_target, 6, 9, 10, 13, 15, 18, 19, 30, 50, 69, 72, 73, 81, 82, 84, 104
- gmm_target(), 6, 27, 41, 50, 65, 70, 73, 80, 105
- gmm_target_from_posterior, 71
- gmm_target_from_samples, 72
- gmm_target_from_samples(), 6, 9, 10, 70, 82, 85
- gmm_weights, 73
- gmm_weights(), 6, 27, 41, 50, 65, 70, 80, 105
- hellinger_mc, 74
- hellinger_mc(), 7, 11, 12, 31, 37, 43, 46, 51, 52, 56, 64, 81, 93
- init_kmeans, 75
- init_kmeans(), 13, 76, 77, 85
- init_moment_seed, 75
- init_moment_seed(), 18, 75, 77, 85
- init_random, 76
- init_random(), 75–77, 85
- init_warm_start, 77
- init_warm_start(), 75–77, 85
- is_mvn, 78
- is_mvn(), 79, 80, 87
- is_mvt, 78
- is_mvt(), 78, 80, 87
- is_proposal, 15, 22, 23, 78, 79, 79, 80, 87
- is_proposal(), 6, 27, 41, 50, 65, 70, 73, 105
- is_uniform, 80
- is_uniform(), 15, 78, 79, 87
- kld_trace, 81
- kld_trace(), 7, 11, 12, 31, 37, 43, 46, 50–52, 56, 64, 74, 93
- mar (mechanism), 83
- mar(), 54, 55
- maxent_target, 81
- maxent_target(), 6, 9, 10, 73, 85
- mechanism, 4, 35, 53, 55, 83, 92, 96, 99
- mice::pool(), 4, 99
- mixture_target, 84
- mixture_target(), 6, 9, 10, 73, 82
- mnar (mechanism), 83
- mnar(), 54, 55, 96
- multi_start_best_of, 85
- multi_start_best_of(), 75–77
- pgmm, 86
- pgmm(), 8, 32, 36, 42, 59, 60, 63, 92, 103
- proposal_mvn (proposal_uniform), 87
- proposal_mvt (proposal_uniform), 87
- proposal_uniform, 87
- proposal_uniform(), 78–80
- proxy_cate, 57, 88
- proxy_cate(), 20, 21, 33–35, 40, 58, 90, 91, 94, 97, 98, 100–103, 106, 107
- proxy_confounding_gap, 89
- proxy_confounding_gap(), 21, 33–35, 40, 58, 88, 89, 91, 94, 97, 98, 100–103, 106, 107
- proxy_decide, 90
- proxy_decide(), 20, 21, 33–35, 40, 58, 89, 90, 94, 97, 98, 100–103, 106, 107
- proxy_fmi, 92
- proxy_fmi(), 4, 35, 52, 53, 55, 84, 96, 99
- proxy_functional_ci, 92
- proxy_functional_ci(), 7, 11, 12, 31, 37, 43, 46, 50–52, 56, 64, 74, 81
- proxy_identification_report, 94
- proxy_identification_report(), 20, 21, 33–35, 40, 58, 89–91, 97, 98, 100–103, 105–107
- proxy_mnar_sensitivity, 95
- proxy_mnar_sensitivity(), 4, 35, 53, 55, 84, 92, 99
- proxy_overlap, 96
- proxy_overlap(), 21, 33–35, 40, 58, 89–91, 94, 98, 100–103, 106, 107
- proxy_policy_value, 97
- proxy_policy_value(), 20, 21, 33–35, 40, 58, 89–91, 94, 96, 97, 100–103, 106, 107
- proxy_pool, 98
- proxy_pool(), 4, 35, 52, 53, 55, 84, 92, 96
- proxy_predict, 99
- proxy_predict(), 21, 33–35, 40, 58, 89–91, 94, 97, 98, 101–103, 106, 107
- proxy_regime_segments, 100
- proxy_regime_segments(), 21, 33–35, 40, 58, 89–91, 94, 97, 98, 100, 102, 103, 106, 107
- proxy_retrospective_uplift, 101

`proxy_retrospective_uplift()`, [21](#), [33–35](#),
[40](#), [58](#), [89–91](#), [94](#), [97](#), [98](#), [100](#), [101](#),
[103](#), [106](#), [107](#)
`proxy_uplift`, [102](#)
`proxy_uplift()`, [21](#), [33–35](#), [40](#), [58](#), [89–91](#),
[94](#), [97](#), [98](#), [100–102](#), [106](#), [107](#)

`qgmm` (`pgmm`), [86](#)

`rgmm`, [103](#)
`rgmm()`, [8](#), [27](#), [32](#), [36](#), [42](#), [59](#), [60](#), [63](#), [86](#)

`select_N`, [104](#)
`select_N()`, [14](#), [17](#), [18](#), [23](#), [26](#)

`tidy.gmm`, [6](#), [27](#), [41](#), [50](#), [65](#), [70](#), [73](#), [80](#), [105](#)

`uplift_identification`, [105](#)
`uplift_identification()`, [21](#), [33–35](#), [40](#),
[58](#), [89–91](#), [94](#), [97](#), [98](#), [100–103](#), [107](#)
`uplift_model`, [20](#), [21](#), [88](#), [90](#), [91](#), [94](#), [96](#), [97](#),
[100–103](#), [106](#)
`uplift_model()`, [21](#), [33–35](#), [40](#), [58](#), [89–91](#),
[94](#), [97](#), [98](#), [100–103](#), [106](#)