# Package 'CausalMixGPD'

April 21, 2026

**Type** Package

**Title** Bayesian Nonparametric Conditional Density Modeling in Causal Inference and Clustering with a Heavy-Tail Extension

**Version** 0.7.0

**Maintainer** Arnab Aich <aaich@fsu.edu>

**Description** The presence of a heavy tail is a feature of many scenarios when risk management involves extremely rare events. While parametric distributions may give adequate representation of the mode of data, they are likely to misrepresent heavy tails, and completely nonparametric approaches lack a rigorous mechanism for tail extrapolation; see Pickands (1975) <doi:10.1214/aos/1176343003>. The package 'CausalMixGPD' implements the semiparametric framework of Aich and Bhattacharya (2026) <doi:10.5281/zenodo.19620523> for Bayesian analysis of heavy-tailed outcomes by combining Dirichlet process mixture models for the body of the distribution with optional generalized Pareto tails. The method allows for unconditional and covariate-modulated mixtures, implements MCMC estimation using 'nimble', and extends to mixtures of different arms' outcomes with application to causal inference in the Rubin (1974) <doi:10.1037/h0037350> framework. Posterior summaries include density functions, quantiles, expected values, survival functions, and causal effects, with an emphasis on tail quantiles and functional measures sensitive to the tail.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 4.0.0), nimble

**Imports** stats, utils, ggplot2

**Suggests** testthat (>= 3.0.0), knitr, rmarkdown, here, cli, coda, ggmcmc, future, future.apply, crayon, DT, kableExtra, plotly, MatchIt, codetools

**VignetteBuilder** knitr

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**URL** https://arnabaich96.github.io/CausalMixGPD/pkgdown/

**BugReports** https://arnabaich96.github.io/CausalMixGPD/

**NeedsCompilation** no

**Author** Arnab Aich [aut, cre] (ORCID: <https://orcid.org/0009-0005-7801-6701>)

**Repository** CRAN

**Date/Publication** 2026-04-21 19:12:28 UTC

# Contents

amoroso *Amoroso distribution*

## Description

Scalar Amoroso utilities used as flexible positive-support base kernels and mixture components. The package parameterization uses `loc`, `scale`, `shape1`, and `shape2`, allowing the family to represent a broad range of skewed bulk shapes.

## Usage

```
dAmoroso(x, loc, scale, shape1, shape2, log = 0)

pAmoroso(q, loc, scale, shape1, shape2, lower.tail = 1, log.p = 0)

qAmoroso(p, loc, scale, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

rAmoroso(n, loc, scale, shape1, shape2)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `loc` | Numeric scalar location parameter. |
| `scale` | Numeric scalar scale parameter. |
| `shape1` | Numeric scalar first shape parameter. |
| `shape2` | Numeric scalar second shape parameter. |
| `log` | Logical; if `TRUE`, return the log-density (integer flag `0/1` in NIMBLE). |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq q)$. |
| `log.p` | Logical; if `TRUE`, probabilities are returned on the log scale. |
| `p` | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| `n` | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports `n = 1`. |

## Details

Writing `loc = a`, `scale = theta`, `shape1 = alpha`, and `shape2 = beta`, the transformed quantity $((X - a)/\theta)^\beta$ follows a gamma law with shape $\alpha$.

These uppercase NIMBLE-compatible functions are scalar (`x/q` and `n = 1`). For vectorized R usage, use `base_lowercase()`.

The Amoroso family used in the package is defined by the density

$$f(x) = \left|\frac{\beta}{\theta}\right| \frac{z^{\alpha\beta-1}\exp(-z^\beta)}{\Gamma(\alpha)}, \qquad z = \frac{x-a}{\theta},$$

on the side of the location parameter determined by the sign of $\theta$. Equivalently, $Z = ((X - a)/\theta)^{\beta}$ follows a Gamma distribution with shape $\alpha$ and unit scale. That representation explains why the quantile function is computed from a gamma quantile and then mapped back through the inverse transformation.

The mean exists whenever $\alpha + 1/\beta$ lies in the domain of the gamma function used by the moment formula. In the package this family serves as a flexible positive-support bulk kernel capable of reproducing gamma-like, Weibull-like, and other skewed shapes with a single parameterization.

## Value

Density/CDF/RNG functions return numeric scalars. The quantile function returns a numeric scalar or vector matching the length of `p`.

## Functions

- `dAmoroso()`: Density Function of Amoroso Distribution
- `pAmoroso()`: Distribution Function of Amoroso Distribution
- `qAmoroso()`: Quantile Function of Amoroso Distribution
- `rAmoroso()`: Sample generating Function of Amoroso Distribution

## See Also

`amoroso_mix()`, `amoroso_gpd()`, `base_lowercase()`, `kernel_support_table()`.

Other base bulk distributions: `InvGauss`, `cauchy`

## Examples

```
loc <- 0
scale <- 1.5
shape1 <- 2
shape2 <- 1.2

dAmoroso(1.0, loc, scale, shape1, shape2, log = 0)
pAmoroso(1.0, loc, scale, shape1, shape2, lower.tail = 1, log.p = 0)
qAmoroso(0.50, loc, scale, shape1, shape2)
qAmoroso(0.95, loc, scale, shape1, shape2)
replicate(10, rAmoroso(1, loc, scale, shape1, shape2))
```

---

# amoroso_gpd *Amoroso with a GPD tail*

---

## Description

Spliced family obtained by attaching a generalized Pareto tail above `threshold` to a single Amoroso bulk.

## Usage

```
dAmorosoGpd(
  x,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  log = 0
)

pAmorosoGpd(
  q,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rAmorosoGpd(n, loc, scale, shape1, shape2, threshold, tail_scale, tail_shape)

qAmorosoGpd(
  p,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| loc | Numeric scalar location parameter of the Amoroso bulk. |
| scale | Numeric scalar scale parameter of the Amoroso bulk. |

| | |
|---|---|
| shape1 | Numeric scalar first Amoroso shape parameter. |
| shape2 | Numeric scalar second Amoroso shape parameter. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Logical; if TRUE, return the log-density (integer flag 0/1 in NIMBLE). |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric tolerance for numerical inversion in qAmorosoGpd. |
| maxiter | Maximum iterations for numerical inversion in qAmorosoGpd. |

## Details

This is the single-component Amoroso splice. The Amoroso law controls the distribution below the threshold and the GPD controls exceedances above it, scaled so that the resulting CDF is continuous at the threshold. The ordinary mean of the spliced law exists only when the tail satisfies $\xi < 1$; otherwise users should rely on restricted means or quantile summaries.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qAmorosoGpd() returns a numeric vector with the same length as p.

## Functions

- dAmorosoGpd(): Density Function of Amoroso Distribution with GPD Tail
- pAmorosoGpd(): Cumulative Distribution Function of Amoroso Distribution with GPD Tail
- rAmorosoGpd(): Random Generation for Amoroso Distribution with GPD Tail
- qAmorosoGpd(): Quantile Function of Amoroso Distribution with GPD Tail

## See Also

amoroso_mix(), amoroso_mixgpd(), gpd(), amoroso_lowercase().

Other amoroso kernel families: amoroso_mix, amoroso_mixgpd

## Examples

```
loc <- 0
scale <- 1.5
shape1 <- 2
shape2 <- 1.2
threshold <- 3
tail_scale <- 1.0
tail_shape <- 0.2

dAmorosoGpd(4.0, loc, scale, shape1, shape2,
           threshold, tail_scale, tail_shape, log = 0)
pAmorosoGpd(4.0, loc, scale, shape1, shape2,
           threshold, tail_scale, tail_shape, lower.tail = 1, log.p = 0)
qAmorosoGpd(0.50, loc, scale, shape1, shape2,
           threshold, tail_scale, tail_shape)
qAmorosoGpd(0.95, loc, scale, shape1, shape2,
           threshold, tail_scale, tail_shape)
replicate(10, rAmorosoGpd(1, loc, scale, shape1, shape2,
                          threshold, tail_scale, tail_shape))
```

---

# amoroso_lowercase *Lowercase vectorized Amoroso distribution functions*

---

## Description

Vectorized R wrappers for the scalar Amoroso-kernel topics in this file.

## Usage

```
damorosomix(x, w, loc, scale, shape1, shape2, log = FALSE)

pamorosomix(q, w, loc, scale, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

qamorosomix(
  p,
  w,
  loc,
  scale,
  shape1,
  shape2,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

ramorosomix(n, w, loc, scale, shape1, shape2)
```

```
damorosomixgpd(
  x,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  log = FALSE
)

pamorosomixgpd(
  q,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qamorosomixgpd(
  p,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

ramorosomixgpd(
  n,
  w,
  loc,
  scale,
```

```
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape
)

damorosogpd(
  x,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  log = FALSE
)

pamorosogpd(
  q,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qamorosogpd(
  p,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

ramorosogpd(n, loc, scale, shape1, shape2, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector of quantiles. |
| `w` | Numeric vector of mixture weights. |
| `loc`, `scale`, `shape1`, `shape2` | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |
| `log` | Logical; if `TRUE`, return log-density. |
| `q` | Numeric vector of quantiles. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq x)$. |
| `log.p` | Logical; if `TRUE`, probabilities are on log scale. |
| `p` | Numeric vector of probabilities. |
| `tol`, `maxiter` | Tolerance and max iterations for numerical inversion. |
| `n` | Integer number of observations to generate. |
| `threshold`, `tail_scale`, `tail_shape` | GPD tail parameters (scalars). |

## Details

These are vectorized wrappers around the scalar Amoroso routines used internally by the package. They preserve the same location-scale-shape parameterization and the same piecewise splice logic for GPD tails. Quantile wrappers therefore continue to rely on the scalar numerical inversion or scalar GPD inverse exactly as documented for the uppercase functions.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `damorosomix()`: Amoroso mixture density (vectorized)
- `pamorosomix()`: Amoroso mixture distribution function (vectorized)
- `qamorosomix()`: Amoroso mixture quantile function (vectorized)
- `ramorosomix()`: Amoroso mixture random generation (vectorized)
- `damorosomixgpd()`: Amoroso mixture + GPD density (vectorized)
- `pamorosomixgpd()`: Amoroso mixture + GPD distribution function (vectorized)
- `qamorosomixgpd()`: Amoroso mixture + GPD quantile function (vectorized)
- `ramorosomixgpd()`: Amoroso mixture + GPD random generation (vectorized)
- `damorosogpd()`: Amoroso + GPD density (vectorized)
- `pamorosogpd()`: Amoroso + GPD distribution function (vectorized)
- `qamorosogpd()`: Amoroso + GPD quantile function (vectorized)
- `ramorosogpd()`: Amoroso + GPD random generation (vectorized)

## See Also

amoroso_mix(), amoroso_mixgpd(), amoroso_gpd(), bundle(), get_kernel_registry().

Other vectorized kernel helpers: base_lowercase, cauchy_mix_lowercase, gamma_lowercase, invgauss_lowercase, laplace_lowercase, lognormal_lowercase, normal_lowercase

## Examples

```
w <- c(0.6, 0.3, 0.1)
locs <- c(0.5, 0.5, 0.5)
scls <- c(1, 1.3, 1.6)
s1 <- c(2.5, 3, 4)
s2 <- c(1.2, 1.2, 1.2)

# Amoroso mixture
damorosomix(c(1, 2, 3), w = w, loc = locs, scale = scls, shape1 = s1, shape2 = s2)
ramorosomix(5, w = w, loc = locs, scale = scls, shape1 = s1, shape2 = s2)
```

---

# amoroso_mix *Amoroso mixture distribution*

---

## Description

Finite mixture of Amoroso components for flexible positive-support bulk modeling.

## Usage

```
dAmorosoMix(x, w, loc, scale, shape1, shape2, log = 0)

pAmorosoMix(q, w, loc, scale, shape1, shape2, lower.tail = 1, log.p = 0)

rAmorosoMix(n, w, loc, scale, shape1, shape2)

qAmorosoMix(
  p,
  w,
  loc,
  scale,
  shape1,
  shape2,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions treat the weights as non-negative and normalize them internally when needed. |
| loc | Numeric vector of length $K$ giving component locations. |
| scale | Numeric vector of length $K$ giving component scales. Negative values flip support for the corresponding component. |
| shape1 | Numeric vector of length $K$ giving the first Amoroso shape parameter for each component. |
| shape2 | Numeric vector of length $K$ giving the second Amoroso shape parameter for each component. |
| log | Logical; if TRUE, return the log-density (integer flag 0/1 in NIMBLE). |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot in quantile inversion. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k f_{Amoroso}(x \mid a_k, \theta_k, \alpha_k, \beta_k),$$

with normalized weights $\tilde{w}_k$. These scalar functions are NIMBLE-compatible; for vectorized R usage, use amoroso_lowercase().

The Amoroso family is especially useful for positive-support data because it can reproduce a wide range of skewed and heavy-right-tail shapes while remaining analytically tractable through its gamma transformation. The mixture CDF is

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k F_{Amoroso}(x \mid a_k, \theta_k, \alpha_k, \beta_k),$$

and random generation proceeds by selecting a component and sampling from that component.

Closed-form mixture quantiles are not available, so qAmorosoMix() inverts the mixture CDF numerically. The analytical mixture mean is the weighted average of the component means, $a_k + \theta_k \Gamma(\alpha_k + 1/\beta_k)/\Gamma(\alpha_k)$, whenever those component moments exist.

## Value

Mixture density/CDF/RNG functions return numeric scalars. `qAmorosoMix()` returns a numeric vector with the same length as `p`.

## Functions

- `dAmorosoMix()`: Density Function of Amoroso Mixture Distribution
- `pAmorosoMix()`: Cumulative Distribution Function of Amoroso Mixture Distribution
- `rAmorosoMix()`: Random Generation for Amoroso Mixture Distribution
- `qAmorosoMix()`: Quantile Function of Amoroso Mixture Distribution

## See Also

`amoroso_mixgpd()`, `amoroso_gpd()`, `amoroso_lowercase()`, `build_nimble_bundle()`, `kernel_support_table()`.

Other amoroso kernel families: `amoroso_gpd`, `amoroso_mixgpd`

## Examples

```
w <- c(0.60, 0.25, 0.15)
loc <- c(0, 1, 2)
scale <- c(1.0, 1.2, 1.6)
shape1 <- c(2, 4, 6)
shape2 <- c(1.0, 1.2, 1.5)

dAmorosoMix(2.0, w, loc, scale, shape1, shape2, log = 0)
pAmorosoMix(2.0, w, loc, scale, shape1, shape2, lower.tail = 1, log.p = 0)
qAmorosoMix(0.50, w, loc, scale, shape1, shape2)
qAmorosoMix(0.95, w, loc, scale, shape1, shape2)
replicate(10, rAmorosoMix(1, w, loc, scale, shape1, shape2))
```

---

# amoroso_mixgpd *Amoroso mixture with a GPD tail*

---

## Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to an Amoroso mixture bulk. Let $F_{mix}$ denote the Amoroso mixture CDF. The spliced CDF is $F(x) = F_{mix}(x)$ for $x < threshold$ and $F(x) = F_{mix}(threshold) + \{1 - F_{mix}(threshold)\}\, G(x)$ for $x \geq threshold$, where $G$ is the GPD CDF for exceedances above `threshold`.

## Usage

```
dAmorosoMixGpd(
  x,
  w,
  loc,
  scale,
```

```
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  log = 0
)

pAmorosoMixGpd(
  q,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rAmorosoMixGpd(
  n,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape
)

qAmorosoMixGpd(
  p,
  w,
  loc,
  scale,
  shape1,
  shape2,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
```

)

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions treat the weights as non-negative and normalize them internally when needed. |
| loc | Numeric vector of length $K$ giving component locations. |
| scale | Numeric vector of length $K$ giving component scales. |
| shape1 | Numeric vector of length $K$ giving the first Amoroso shape parameter for each component. |
| shape2 | Numeric vector of length $K$ giving the second Amoroso shape parameter for each component. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Logical; if TRUE, return the log-density (integer flag 0/1 in NIMBLE). |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot in quantile inversion. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The density, CDF, and RNG are implemented as nimbleFunctions for use in NIMBLE models. The quantile function is an R function that uses numerical inversion in the bulk region and the closed-form GPD quantile in the tail region.

The Amoroso mixture describes the bulk up to the threshold and the generalized Pareto describes exceedances above it. If $F_{mix}(u) = p_u$, then the splice uses

$$f(x) = \begin{cases} f_{mix}(x), & x < u, \\ (1 - p_u) g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

Bulk quantiles are computed numerically from the Amoroso mixture CDF and tail quantiles are obtained from the GPD inverse after rescaling the tail probability.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qAmorosoMixGpd() returns a numeric vector with the same length as p.

## Functions

- dAmorosoMixGpd(): Density Function of Amoroso Mixture Distribution with GPD Tail
- pAmorosoMixGpd(): Cumulative Distribution Function of Amoroso Mixture Distribution with GPD Tail
- rAmorosoMixGpd(): Random Generation for Amoroso Mixture Distribution with GPD Tail
- qAmorosoMixGpd(): Quantile Function of Amoroso Mixture Distribution with GPD Tail

## See Also

amoroso_mix(), amoroso_gpd(), gpd(), amoroso_lowercase(), dpmgpd().

Other amoroso kernel families: amoroso_gpd, amoroso_mix

## Examples

```
w <- c(0.60, 0.25, 0.15)
loc <- c(0, 1, 2)
scale <- c(1.0, 1.2, 1.6)
shape1 <- c(2, 4, 6)
shape2 <- c(1.0, 1.2, 1.5)
threshold <- 3
tail_scale <- 1.0
tail_shape <- 0.2

dAmorosoMixGpd(4.0, w, loc, scale, shape1, shape2,
               threshold, tail_scale, tail_shape, log = 0)
pAmorosoMixGpd(4.0, w, loc, scale, shape1, shape2,
               threshold, tail_scale, tail_shape, lower.tail = 1, log.p = 0)
qAmorosoMixGpd(0.50, w, loc, scale, shape1, shape2,
               threshold, tail_scale, tail_shape)
qAmorosoMixGpd(0.95, w, loc, scale, shape1, shape2,
               threshold, tail_scale, tail_shape)
replicate(10, rAmorosoMixGpd(1, w, loc, scale, shape1, shape2,
                             threshold, tail_scale, tail_shape))
```

---

# ate *Average treatment effects, marginal over the empirical covariate distribution*

---

## Description

ate() computes the posterior predictive average treatment effect.

## Usage

```
ate(
  fit,
  newdata = NULL,
  y = NULL,
  type = c("mean", "rmean"),
  cutoff = NULL,
  interval = "credible",
  level = 0.95,
  nsim_mean = 200L,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| fit | A "causalmixgpd_causal_fit" object from run_mcmc_causal(). |
| newdata | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| y | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| type | Character; type of mean treatment effect:<br>• "mean" (default): ordinary mean ATE<br>• "rmean": restricted-mean ATE (requires cutoff) |
| cutoff | Finite numeric cutoff for restricted mean; required for type = "rmean", ignored otherwise. |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| level | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| nsim_mean | Number of posterior predictive draws used by simulation-based mean targets. Ignored for analytical ordinary means. |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |

## Details

The default mean-scale estimand is

$$\mathrm{ATE} = E\{Y(1)\} - E\{Y(0)\},$$

where the expectation is taken with respect to the empirical training covariate distribution for conditional models.

When type = "rmean", the function instead computes a restricted-mean ATE using $E\{\min(Y(a), c)\}$ for each arm. For outcome kernels with a finite analytical mean, the ordinary mean path is analytical within each posterior draw; rmean remains simulation-based.

For unconditional causal models (X = NULL), the computation reduces to a direct contrast of the unconditional treated and control predictive laws.

## Value

An object of class "causalmixgpd_ate" containing the marginal ATE summary, optional intervals, and the arm-specific predictive objects used in the aggregation. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

att, cate, qte, ate_rmean, predict.causalmixgpd_causal_fit.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
ate(fit, interval = "credible", level = 0.90, nsim_mean = 100)
```

---

# ate_rmean *Restricted-mean ATE helper*

---

## Description

ate_rmean() is a convenience wrapper for restricted-mean treatment effects when the ordinary mean is unstable or undefined.

## Usage

```
ate_rmean(
  fit,
  newdata = NULL,
  cutoff,
  interval = "credible",
  level = 0.95,
  nsim_mean = 200L,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| fit | A "causalmixgpd_causal_fit" object from run_mcmc_causal(). |
| newdata | Optional data.frame or matrix of covariates for prediction. If NULL, uses the training covariates stored in fit. |

| | |
|---|---|
| cutoff | Finite numeric cutoff for the restricted mean. |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| level | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| nsim_mean | Number of posterior predictive draws used by simulation-based mean targets. Ignored for analytical ordinary means. |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |

## Details

The restricted-mean estimand replaces $Y(a)$ by $\min\{Y(a), c\}$, so the contrast remains finite even when the fitted GPD tail implies $\xi \geq 1$.

## Value

A "causalmixgpd_ate" object computed via ate for unconditional fits or cate for conditional fits. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

ate, cate, predict.mixgpd_fit.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          GPD = TRUE, components = 3,
                          mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb)
ate_rm <- ate_rmean(fit, cutoff = 10, interval = "credible")
```

---

# att *Average treatment effects standardized to treated covariates*

---

## Description

att() computes the average treatment effect on the treated.

## Usage

```
att(
  fit,
  newdata = NULL,
  y = NULL,
  type = c("mean", "rmean"),
  cutoff = NULL,
  interval = "credible",
  level = 0.95,
  nsim_mean = 200L,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| fit | A "causalmixgpd_causal_fit" object from run_mcmc_causal(). |
| newdata | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| y | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| type | Character; type of mean treatment effect:<br>• "mean" (default): ordinary mean ATE<br>• "rmean": restricted-mean ATE (requires cutoff) |
| cutoff | Finite numeric cutoff for restricted mean; required for type = "rmean", ignored otherwise. |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| level | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| nsim_mean | Number of posterior predictive draws used by simulation-based mean targets. Ignored for analytical ordinary means. |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |

## Details

The estimand is

$$\mathrm{ATT} = E\{Y(1) - Y(0) \mid A = 1\},$$

approximated by marginalizing over the empirical covariate distribution of treated units.

## Value

An object of class "causalmixgpd_ate" containing the ATT summary, optional intervals, and the arm-specific predictive objects used in the aggregation. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

ate, qtt, cate.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
att(fit, interval = "credible", nsim_mean = 100)
```

---

# base_lowercase *Lowercase vectorized distribution functions (base kernels)*

---

## Description

Vectorized R wrappers around the scalar base-kernel functions defined in this file. These helpers are intended for interactive R use, examples, testing, and checking numerical behavior outside compiled NIMBLE code.

## Usage

```
dgpd(x, threshold, scale, shape, log = FALSE)

pgpd(q, threshold, scale, shape, lower.tail = TRUE, log.p = FALSE)

qgpd(p, threshold, scale, shape, lower.tail = TRUE, log.p = FALSE)

rgpd(n, threshold, scale, shape)

dinvgauss(x, mean, shape, log = FALSE)

pinvgauss(q, mean, shape, lower.tail = TRUE, log.p = FALSE)

qinvgauss(
  p,
  mean,
  shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
```

```
)

rinvgauss(n, mean, shape)

damoroso(x, loc, scale, shape1, shape2, log = FALSE)

pamoroso(q, loc, scale, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

qamoroso(p, loc, scale, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

ramoroso(n, loc, scale, shape1, shape2)

dcauchy_vec(x, location, scale, log = FALSE)

pcauchy_vec(q, location, scale, lower.tail = TRUE, log.p = FALSE)

qcauchy_vec(p, location, scale, lower.tail = TRUE, log.p = FALSE)

rcauchy_vec(n, location, scale)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles. |
| threshold, scale, shape, mean, loc, shape1, shape2, location | Distribution parameters (scalars). |
| log | Logical; if TRUE, return log-density. |
| q | Numeric vector of quantiles. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq x)$. |
| log.p | Logical; if TRUE, probabilities are on log scale. |
| p | Numeric vector of probabilities. |
| n | Integer number of observations to generate. |
| tol, maxiter | Tolerance and max iterations for numerical inversion. |

## Details

The wrappers preserve the same parameterizations as the uppercase scalar functions, but accept vector inputs for x, q, or p and allow n > 1 for random generation.

Each lowercase helper is a vectorized R wrapper around the corresponding uppercase scalar routine documented in this file. The wrapper keeps the same parameterization and applies the scalar kernel repeatedly over the supplied evaluation points or simulation index. These helpers are therefore appropriate for interactive analysis, testing, and examples, whereas the uppercase functions are the building blocks used inside NIMBLE model code.

The wrappers do not change the underlying theory. For example, `qgpd()` still uses the closed-form GPD inverse, `qinvgauss()` still performs numerical inversion of the inverse Gaussian CDF, and `qamoroso()` still maps a gamma quantile through the Amoroso transformation. Random-generation wrappers call the corresponding scalar RNG repeatedly when n > 1.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `dgpd()`: GPD density (vectorized)
- `pgpd()`: GPD distribution function (vectorized)
- `qgpd()`: GPD quantile function (vectorized)
- `rgpd()`: GPD random generation (vectorized)
- `dinvgauss()`: Inverse Gaussian density (vectorized)
- `pinvgauss()`: Inverse Gaussian distribution function (vectorized)
- `qinvgauss()`: Inverse Gaussian quantile function (vectorized)
- `rinvgauss()`: Inverse Gaussian random generation (vectorized)
- `damoroso()`: Amoroso density (vectorized)
- `pamoroso()`: Amoroso distribution function (vectorized)
- `qamoroso()`: Amoroso quantile function (vectorized)
- `ramoroso()`: Amoroso random generation (vectorized)
- `dcauchy_vec()`: Cauchy density (vectorized)
- `pcauchy_vec()`: Cauchy distribution function (vectorized)
- `qcauchy_vec()`: Cauchy quantile function (vectorized)
- `rcauchy_vec()`: Cauchy random generation (vectorized)

## See Also

`gpd()`, `InvGauss()`, `amoroso()`, `cauchy()`, `build_nimble_bundle()`, `kernel_support_table()`.

Other vectorized kernel helpers: `amoroso_lowercase`, `cauchy_mix_lowercase`, `gamma_lowercase`, `invgauss_lowercase`, `laplace_lowercase`, `lognormal_lowercase`, `normal_lowercase`

## Examples

```
# GPD
dgpd(c(1.5, 2.0, 2.5), threshold = 1, scale = 0.8, shape = 0.2)
pgpd(c(1.5, 2.0), threshold = 1, scale = 0.8, shape = 0.2)
qgpd(c(0.5, 0.9), threshold = 1, scale = 0.8, shape = 0.2)
rgpd(5, threshold = 1, scale = 0.8, shape = 0.2)

# Inverse Gaussian
dinvgauss(c(1, 2, 3), mean = 2, shape = 5)
rinvgauss(5, mean = 2, shape = 5)

# Amoroso
damoroso(c(1, 2), loc = 0, scale = 1.5, shape1 = 2, shape2 = 1.2)
ramoroso(5, loc = 0, scale = 1.5, shape1 = 2, shape2 = 1.2)

# Cauchy
```

```
dcauchy_vec(c(-1, 0, 1), location = 0, scale = 1)
rcauchy_vec(5, location = 0, scale = 1)
```

---

`build_causal_bundle` *Build a causal bundle (design + two outcome arms)*

---

## Description

`build_causal_bundle()` is the detailed constructor behind `bundle` for causal analyses. It prepares:

- a propensity score (PS) design block for $A \mid X$,
- a control-arm outcome bundle for $Y(0)$,
- a treated-arm outcome bundle for $Y(1)$.

## Usage

```
build_causal_bundle(
  y,
  X,
  A,
  backend = c("sb", "crp", "spliced"),
  kernel,
  GPD = FALSE,
  components = 10L,
  param_specs = NULL,
  mcmc_outcome = list(niter = 2000, nburnin = 500, thin = 1, nchains = 1, seed = 1),
  mcmc_ps = list(niter = 1000, nburnin = 250, thin = 1, nchains = 1, seed = 1),
  epsilon = 0.025,
  alpha_random = TRUE,
  ps_prior = list(mean = 0, sd = 2),
  include_intercept = TRUE,
  PS = "logit",
  ps_scale = c("logit", "prob"),
  ps_summary = c("mean", "median"),
  ps_clamp = 1e-06,
  monitor = c("core", "full"),
  monitor_latent = FALSE,
  monitor_v = FALSE
)
```

## Arguments

| | |
|---|---|
| `y` | Numeric outcome vector. |
| `X` | Design matrix or data.frame of covariates (N x P). |
| `A` | Binary treatment indicator (length N, values 0/1). |

backend — Character; the Dirichlet process representation for outcome models:

- "sb": stick-breaking truncation
- "crp": Chinese Restaurant Process
- "spliced": CRP with GPD tail splicing

If length 2, the first entry is used for treated (A=1) and the second for control (A=0).

kernel — Character kernel name for outcome models (must exist in get_kernel_registry()). If length 2:

- first entry: used for treated (A=1)
- second entry: used for control (A=0)

GPD — Logical; include GPD tail for outcomes if TRUE. If length 2:

- first entry: used for treated (A=1)
- second entry: used for control (A=0)

components — Integer >= 2; truncation parameter for outcome mixtures. If length 2:

- first entry: used for treated (A=1)
- second entry: used for control (A=0)

param_specs — Outcome parameter overrides (same structure as build_nimble_bundle()):

- a single list: used for both arms
- a list with con and trt entries: arm-specific overrides

mcmc_outcome — MCMC settings list for the outcome bundles.

mcmc_ps — MCMC settings list for the PS model.

epsilon — Numeric in [0,1) used by outcome bundles for posterior truncation summaries. If length 2:

- first entry: used for treated (A=1)
- second entry: used for control (A=0)

alpha_random — Logical; whether the outcome-model DP concentration parameter $\kappa$ is stochastic.

ps_prior — Normal prior for PS coefficients. List with mean and sd.

include_intercept — Logical; if TRUE, an intercept column is prepended to X in the PS model.

PS — Character or logical; controls propensity score estimation:

- "logit" (default): Logistic regression PS model
- "probit": Probit regression PS model
- "naive": Gaussian naive Bayes PS model
- FALSE: No PS estimation; outcome models use only X

The PS model choice is stored in bundle metadata for downstream use in prediction and summaries.

ps_scale — Scale used when augmenting outcomes with PS:

- "logit": augment on the logit (log-odds) scale
- "prob": augment on the probability scale

| | |
|---|---|
| `ps_summary` | Posterior summary for PS:<br>• `"mean"`: posterior mean of propensity scores<br>• `"median"`: posterior median of propensity scores |
| `ps_clamp` | Numeric epsilon for clamping PS values to $(\epsilon, 1-\epsilon)$. |
| `monitor` | Character monitor profile:<br>• `"core"` (default): monitors only the essential model parameters<br>• `"full"`: monitors all model nodes |
| `monitor_latent` | Logical; whether to monitor latent cluster labels (`z`) in outcome arms. |
| `monitor_v` | Logical; whether to monitor stick-breaking `v` terms for SB outcomes. |

## Details

The outcome bundles reuse the one-arm DPM plus optional GPD machinery. The PS block provides a shared adjustment object used by `run_mcmc_causal` and `predict.causalmixgpd_causal_fit`.

The causal bundle encodes the two arm-specific predictive laws $F_0(y \mid x)$ and $F_1(y \mid x)$. Downstream causal estimands are functionals of these two distributions:

$$\mathrm{ATE} = E\{Y(1)\} - E\{Y(0)\}, \qquad \mathrm{QTE}(\tau) = Q_1(\tau) - Q_0(\tau).$$

When `PS` is enabled, the package estimates a propensity score model $e(x) = \Pr(A = 1 \mid X = x)$ and uses a posterior summary of that score as an augmented covariate in the arm-specific outcome models. This mirrors the workflow described in the manuscript vignette.

## Value

A list of class `"causalmixgpd_causal_bundle"` containing the design bundle, two outcome bundles, training data, arm indices, and metadata required for posterior prediction and causal effect summaries.

## See Also

`bundle`, `run_mcmc_causal`, `predict.causalmixgpd_causal_fit`, `ate`, `qte`, `cate`, `cqte`.

## Examples

```
set.seed(1)
N <- 25
X <- cbind(x1 = rnorm(N), x2 = runif(N))
A <- rbinom(N, 1, plogis(0.3 + 0.5 * X[, 1]))
y <- rexp(N) + 0.1

cb <- build_causal_bundle(
  y = y,
  X = X,
  A = A,
  backend = "sb",
  kernel = "gamma",
  GPD = TRUE,
```

```
  components = 3,
  PS = "probit"
)
```

---

`build_code_from_spec` *Build NIMBLE model code from a compiled model spec*

---

## Description

Dispatches to the backend-specific code generators:

- `build_code_sb_from_spec()` for stick-breaking (`"sb"`)
- `build_code_crp_from_spec()` for CRP and spliced (`"crp"`, `"spliced"`)

## Usage

```
build_code_from_spec(spec)
```

## Arguments

| | |
|---|---|
| `spec` | A compiled model specification produced by `compile_model_spec()`. |

## Details

The model size is controlled by `spec$meta$components` only.

## Value

A `nimbleCode` object.

---

`build_constants_from_spec`
*Build constants list from a compiled model spec*

---

## Description

Produces a named list of constants to pass into `nimbleModel`. Constants include core sizes (`N`, `P`, `components`) and hyperparameters for priors implied by `spec$plan`.

## Usage

```
build_constants_from_spec(spec)
```

## Arguments

| | |
|---|---|
| `spec` | A compiled model specification produced by `compile_model_spec()`. |

## Details

This function is pre-run only; it does not compile or execute NIMBLE.

## Value

Named list of constants.

---

`build_dimensions_from_spec` *Build dimension declarations from a compiled model spec*

---

## Description

Returns a named list of array dimensions used by downstream builders (inits/monitors/code generation). Dimensions are derived solely from `spec$meta` and `spec$plan`. This function does not inspect data.

## Usage

```
build_dimensions_from_spec(spec)
```

## Arguments

| | |
|---|---|
| `spec` | A compiled model specification produced by `compile_model_spec()`. |

## Details

The model size is controlled by a single parameter: `components`. For SB this is the truncation level of the stick-breaking mixture. For CRP this is the maximum number of clusters represented in the finite model.

## Value

Named list of dimensions (integer vectors). Scalars are omitted.

`build_inits_from_spec` *Build initial values from a compiled model spec*

## Description

Produces a list of initial values suitable for passing to `nimbleModel`. The initial values are derived from `spec$plan` and are intended to be stable and support-respecting (e.g., positive parameters start positive).

## Usage

```
build_inits_from_spec(spec, seed = NULL, y = NULL, X = NULL)
```

## Arguments

| | |
|---|---|
| `spec` | A compiled model specification produced by `compile_model_spec()`. |
| `seed` | Optional seed (single integer or vector). If provided, the first element is used. |
| `y` | Optional numeric vector of observed outcomes used for heuristic initializations. |
| `X` | Optional numeric matrix of covariates used for link-mode parameter initializations. |

## Details

Notes:

- Uses only `components` as the model size parameter.
- SB: initializes stick breaks `v`; weights `w` are deterministic.
- CRP: initializes memberships `z` in `1:components`.
- Link-mode parameters initialize regression coefficients `beta_<param>` with shape `components x P`.
- Default GPD threshold under X is stochastic lognormal: initializes `threshold[1:N]` and scalar `sdlog_u` (non-link thresholds are scalar).

## Value

Named list of initial values.

build_monitors_from_spec *Build default monitors from a compiled model spec*

## Description

Returns the character vector of node names to monitor in MCMC. This is a pre-run builder used by `build_nimble_bundle()`.

## Usage

```
build_monitors_from_spec(spec, monitor_v = FALSE, monitor_latent = FALSE)
```

## Arguments

| | |
|---|---|
| `spec` | A compiled model specification produced by `compile_model_spec()`. |
| `monitor_v` | Logical; for SB, whether to also monitor stick breaks `v`. |
| `monitor_latent` | Logical; whether to monitor latent cluster labels `z`. |

## Details

Monitoring follows these rules:

- Always monitor concentration `kappa` (whether fixed or stochastic).
- SB: monitor `w[1:components]` and optionally `v[1:(components-1)]`.
- CRP: monitor `z[1:N]`.
- Bulk parameters:
  - dist/fixed: monitor `<param>[1:components]`
  - link: monitor `beta_<param>[1:components, 1:P]`
- GPD (if enabled):
  - threshold: monitor scalar `threshold` when not link-mode; `threshold[1:N]` for link-mode
  - if threshold is link-mode: monitor `beta_threshold[1:P]`
  - if threshold uses LN link-dist default: monitor `sdlog_u`
  - tail_scale: if link-mode, monitor `beta_tail_scale[1:P]`
  - tail_shape: monitor scalar `tail_shape` (fixed or dist)

## Value

Character vector of node names to monitor.

`build_nimble_bundle` *Build the explicit one-arm NIMBLE bundle*

## Description

`build_nimble_bundle()` is the detailed constructor behind `bundle` for one-arm models. It compiles the modeling plan into a self-contained object holding code-generation inputs, initialization rules, monitor policy, and stored MCMC defaults.

## Usage

```
build_nimble_bundle(
  y,
  X = NULL,
  ps = NULL,
  backend = c("sb", "crp", "spliced"),
  kernel,
  GPD = FALSE,
  components = 10L,
  param_specs = NULL,
  mcmc = list(niter = 2000, nburnin = 500, thin = 1, nchains = 1, seed = 1),
  epsilon = 0.025,
  alpha_random = TRUE,
  monitor = c("core", "full"),
  monitor_latent = FALSE,
  monitor_v = FALSE
)
```

## Arguments

| | |
|---|---|
| `y` | Numeric outcome vector. |
| `X` | Optional design matrix/data.frame (N x p) for conditional variants. |
| `ps` | Optional numeric vector (length N) of propensity scores. When provided, augments the design matrix for PS-adjusted outcome modeling. |
| `backend` | Character; the Dirichlet process representation:<br>• `"sb"`: stick-breaking truncation<br>• `"crp"`: Chinese Restaurant Process |
| `kernel` | Character kernel name (must exist in `get_kernel_registry()`). |
| `GPD` | Logical; whether a GPD tail is requested. |
| `components` | Integer >= 2. Single user-facing truncation parameter:<br>• SB: number of mixture components used in stick-breaking truncation<br>• CRP: maximum number of clusters represented in the finite NIMBLE model |
| `param_specs` | Optional list with entries `bulk` and `tail` to override defaults. |

| | |
|---|---|
| mcmc | Named list of MCMC settings (niter, nburnin, thin, nchains, seed). Stored in bundle. |
| epsilon | Numeric in [0,1). For downstream summaries/plots/prediction we keep the smaller k defined by either (i) cumulative mass >= 1 - epsilon or (ii) per-component weights >= epsilon, then renormalize. |
| alpha_random | Logical; whether the DP concentration parameter $\kappa$ is stochastic. |
| monitor | Character monitor profile:<br>• "core" (default): monitors only the essential model parameters<br>• "full": monitors all model nodes |
| monitor_latent | Logical; if TRUE, include latent cluster labels (z) in monitors. |
| monitor_v | Logical; if TRUE and backend is SB, include stick breaks (v) in monitors. |

## Details

The returned bundle encodes a finite approximation to a Dirichlet process mixture using either a stick-breaking ("sb") or Chinese restaurant process / spliced ("crp" / "spliced") representation.

For the bulk-only model, the target likelihood is the DPM predictive law

$$f(y \mid x) = \sum_{k=1}^{K} w_k(x) f_k(y \mid x, \theta_k).$$

When GPD = TRUE, the bundle augments the bulk model with a threshold $u(x)$ and generalized Pareto tail above that threshold, producing the spliced predictive distribution described in the manuscript vignette.

This function intentionally stops before model compilation and sampling. Use run_mcmc_bundle_manual or mcmc to execute the stored model definition.

The object contains:

- compiled model spec
- nimbleCode model code
- constants, data, explicit dimensions
- initialization function inits (stored as a function)
- monitor specification
- MCMC settings list (stored but not used for code generation)

## Value

A named list of class "causalmixgpd_bundle". Its primary components are spec, code, constants, dimensions, data, inits, monitors, and stored mcmc settings.

## See Also

bundle, run_mcmc_bundle_manual, predict.mixgpd_fit, kernel_support_table, get_kernel_registry.

## Examples

```
y <- abs(rnorm(25)) + 0.1
bundle <- build_nimble_bundle(
  y = y,
  backend = "sb",
  kernel = "normal",
  GPD = FALSE,
  components = 3,
  mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
)
bundle
```

---

# bundle *Build the workflow bundle used by the package fitters*

---

## Description

`bundle()` is the main workflow constructor. It converts raw inputs, a formula/data pair, or an already prepared bundle into the canonical object consumed by `mcmc`, `dpmix`, `dpmgpd`, `dpmix.causal`, and `dpmgpd.causal`.

## Usage

```
bundle(
  y = NULL,
  X = NULL,
  treat = NULL,
  data = NULL,
  formula = NULL,
  GPD = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `y` | Either a response vector or an existing bundle. |
| `X` | Optional design matrix/data.frame. |
| `treat` | Optional binary treatment indicator. |
| `data` | Optional data.frame used with `formula`. |
| `formula` | Optional formula. |
| `GPD` | Logical; include GPD tail in build mode. |
| `...` | Additional arguments passed to `build_nimble_bundle()` or `build_causal_bundle()`. |

## Details

For one-arm models the returned object represents a bulk Dirichlet process mixture, optionally augmented with a spliced generalized Pareto tail. For causal models the returned object contains two arm-specific outcome bundles plus an optional propensity score block.

The workflow is:

1. prepare a bundle with `bundle()`,
2. run posterior sampling with `mcmc` or one of the `dpmix*`/`dpmgpd*` wrappers,
3. inspect the fitted object with `summary.mixgpd_fit`, `params`, `predict.mixgpd_fit`, or the causal estimand helpers.

Setting `GPD = TRUE` requests the spliced bulk-tail model with conditional distribution

$$F(y \mid x) = F_{\text{bulk}}(y \mid x)\mathbf{1}\{y \le u(x)\} + [p_u(x) + \{1 - p_u(x)\}F_{\text{GPD}}(y \mid x)]\,\mathbf{1}\{y > u(x)\},$$

where $p_u(x)$ is the bulk probability below the threshold $u(x)$.

See the manuscript vignette for the DPM hierarchy, SB/CRP representations, and the spliced bulk-tail construction used throughout the package.

## Value

A `"causalmixgpd_bundle"` for one-arm models or a `"causalmixgpd_causal_bundle"` for causal models. The bundle stores code-generation inputs, monitor policy, and default MCMC settings, but it does not run MCMC.

## See Also

`build_nimble_bundle`, `build_causal_bundle`, `mcmc`, `dpmix`, `dpmgpd`.

---

`cate` *Conditional average treatment effects*

---

## Description

`cate()` evaluates treated-minus-control predictive means, or restricted means, at user-supplied covariate rows.

## Usage

```
cate(
  fit,
  newdata = NULL,
  type = c("mean", "rmean"),
  cutoff = NULL,
  interval = "credible",
  level = 0.95,
  nsim_mean = 200L,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| fit | A "causalmixgpd_causal_fit" object from run_mcmc_causal(). |
| newdata | Optional data.frame or matrix of covariates for prediction. If NULL, uses the training covariates stored in fit. |
| type | Character; type of mean treatment effect:<br>• "mean" (default): ordinary mean CATE<br>• "rmean": restricted-mean CATE (requires cutoff) |
| cutoff | Finite numeric cutoff for restricted mean; required for type = "rmean", ignored otherwise. |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| level | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| nsim_mean | Number of posterior predictive draws used by simulation-based mean targets. Ignored for analytical ordinary means. |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |

## Details

For each prediction row $x$, the conditional average treatment effect is

$$\mathrm{CATE}(x) = E\{Y(1) \mid x\} - E\{Y(0) \mid x\}.$$

With type = "rmean", the estimand becomes the conditional restricted mean contrast

$$E\{\min(Y(1), c) \mid x\} - E\{\min(Y(0), c) \mid x\},$$

which remains finite even when the ordinary mean is unstable under a heavy GPD tail. For outcome kernels with a finite analytical mean, the ordinary mean path is analytical within each posterior draw; rmean remains a separate simulation-based estimand.

This estimand is available only for conditional causal models with covariates. For marginal mean contrasts, use ate or att.

## Value

An object of class "causalmixgpd_ate" containing the CATE summary, optional intervals, and the treated/control prediction objects used to construct the effect. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

ate, att, cqte, ate_rmean, predict.causalmixgpd_causal_fit.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
cate(fit, newdata = X[1:5, , drop = FALSE])
cate(fit, interval = "credible", level = 0.90)  # 90% CI
cate(fit, interval = "hpd")  # HPD intervals
cate(fit, interval = NULL)   # No intervals
```

---

# cauchy *Cauchy distribution*

---

## Description

Scalar Cauchy utilities implemented for NIMBLE compatibility. These functions support symmetric heavy-tailed kernels on the real line and feed directly into the finite Cauchy mixture family.

## Usage

```
dCauchy(x, location, scale, log = 0)

pCauchy(q, location, scale, lower.tail = 1, log.p = 0)

rCauchy(n, location, scale)

qCauchy(p, location, scale, lower.tail = TRUE, log.p = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `location` | Numeric scalar location parameter. |
| `scale` | Numeric scalar scale parameter; must be positive. |
| `log` | Logical; if `TRUE`, return the log-density (integer flag `0/1` in NIMBLE). |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq q)$. |
| `log.p` | Logical; if `TRUE`, probabilities are returned on the log scale. |
| `n` | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports `n = 1`. |
| `p` | Numeric scalar probability in $(0, 1)$ for the quantile function. |

## Details

The density is

$$f(x) = \frac{1}{\pi s\{1 + ((x - \ell)/s)^2\}},$$

where `location` = `ell` and `scale` = `s` > `0`.

These uppercase NIMBLE-compatible functions are scalar (`x`/`q` and `n` = `1`). For vectorized R usage, use `base_lowercase()`.

The Cauchy law is a stable heavy-tailed distribution with undefined mean and variance. That is why the package allows the Cauchy kernel only as a bulk distribution and deliberately does not pair it with GPD tails in the kernel registry. For predictive summaries, ordinary means are not available under Cauchy kernels; medians, quantiles, survival curves, and restricted means remain well defined.

The distribution function is

$$F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x - \ell}{s}\right),$$

and the quantile is the corresponding inverse

$$Q(p) = \ell + s \tan\{\pi(p - 1/2)\}.$$

## Value

`dCauchy()` returns a numeric scalar density, `pCauchy()` returns a numeric scalar CDF, `rCauchy()` returns one random draw, and `qCauchy()` returns a numeric quantile.

## Functions

- `dCauchy()`: Cauchy density function
- `pCauchy()`: Cauchy distribution function
- `rCauchy()`: Cauchy random generation
- `qCauchy()`: Cauchy quantile function

## See Also

`cauchy_mix()`, `base_lowercase()`, `kernel_support_table()`.

Other base bulk distributions: `InvGauss`, `amoroso`

## Examples

```
location <- 0
scale <- 1.5

dCauchy(0.5, location, scale, log = 0)
pCauchy(0.5, location, scale, lower.tail = 1, log.p = 0)
qCauchy(0.50, location, scale)
qCauchy(0.95, location, scale)
replicate(10, rCauchy(1, location, scale))
```

cauchy_mix *Cauchy mixture distribution*

## Description

Finite mixture of Cauchy components for symmetric heavy-tailed bulk modeling on the real line.

## Usage

```
dCauchyMix(x, w, location, scale, log = 0)

pCauchyMix(q, w, location, scale, lower.tail = 1, log.p = 0)

rCauchyMix(n, w, location, scale)

qCauchyMix(
  p,
  w,
  location,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions normalize w internally when needed. |
| location, scale | Numeric vectors of length $K$ giving component locations and scales. |
| log | Logical; if TRUE, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k f_C(x \mid \ell_k, s_k),$$

with normalized weights $\tilde{w}_k$. These scalar functions are NIMBLE-compatible; for vectorized R usage, use `cauchy_mix_lowercase()`.

The mixture CDF is the weighted average of component CDFs,

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k \left\{ \frac{1}{2} + \frac{1}{\pi} \arctan\left( \frac{x - \ell_k}{s_k} \right) \right\}.$$

Random generation first selects a component according to the normalized weights and then draws from the chosen Cauchy law by inverse-CDF sampling.

Because each Cauchy component has undefined mean and variance, the mixture also lacks an ordinary mean in general. That is why the package exposes Cauchy kernels for densities, CDFs, quantiles, medians, survival functions, and restricted means, but not for ordinary predictive means.

## Value

Density/CDF/RNG functions return numeric scalars. `qCauchyMix()` returns a numeric vector with the same length as `p`.

## Functions

- `dCauchyMix()`: Cauchy mixture density
- `pCauchyMix()`: Cauchy mixture distribution function
- `rCauchyMix()`: Cauchy mixture random generation
- `qCauchyMix()`: Cauchy mixture quantile function

## See Also

`cauchy()`, `cauchy_mix_lowercase()`, `build_nimble_bundle()`, `kernel_support_table()`.

## Examples

```
w <- c(0.50, 0.30, 0.20)
location <- c(-2, 0, 3)
scale <- c(1.0, 0.7, 1.5)

dCauchyMix(0.5, w = w, location = location, scale = scale, log = FALSE)
pCauchyMix(0.5, w = w, location = location, scale = scale,
           lower.tail = TRUE, log.p = FALSE)
qCauchyMix(0.50, w = w, location = location, scale = scale)
qCauchyMix(0.95, w = w, location = location, scale = scale)
replicate(10, rCauchyMix(1, w = w, location = location, scale = scale))
```

`cauchy_mix_lowercase` *Lowercase vectorized Cauchy mixture distribution functions*

## Description

Vectorized R wrappers for the scalar Cauchy mixture functions in this file.

## Usage

```
dcauchymix(x, w, location, scale, log = FALSE)

pcauchymix(q, w, location, scale, lower.tail = TRUE, log.p = FALSE)

qcauchymix(
  p,
  w,
  location,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rcauchymix(n, w, location, scale)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector of quantiles. |
| `w` | Numeric vector of mixture weights. |
| `location, scale` | Numeric vectors of component parameters. |
| `log` | Logical; if `TRUE`, return log-density. |
| `q` | Numeric vector of quantiles. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq x)$. |
| `log.p` | Logical; if `TRUE`, probabilities are on log scale. |
| `p` | Numeric vector of probabilities. |
| `tol, maxiter` | Tolerance and max iterations for numerical inversion. |
| `n` | Integer number of observations to generate. |

## Details

These are vectorized R wrappers around the scalar Cauchy-mixture routines. They retain the same location-scale parameterization and the same inverse-CDF logic for simulation and quantiles. The lowercase functions do not alter the heavy-tail theory of the underlying Cauchy components; they apply the scalar routines elementwise to vector inputs in R.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `dcauchymix()`: Cauchy mixture density (vectorized)
- `pcauchymix()`: Cauchy mixture distribution function (vectorized)
- `qcauchymix()`: Cauchy mixture quantile function (vectorized)
- `rcauchymix()`: Cauchy mixture random generation (vectorized)

## See Also

`cauchy_mix()`, `cauchy()`, `bundle()`, `get_kernel_registry()`.

Other vectorized kernel helpers: `amoroso_lowercase`, `base_lowercase`, `gamma_lowercase`, `invgauss_lowercase`, `laplace_lowercase`, `lognormal_lowercase`, `normal_lowercase`

## Examples

```
w <- c(0.6, 0.3, 0.1)
loc <- c(-1, 0, 1)
scl <- c(1, 1.2, 2)

dcauchymix(c(-2, 0, 2), w = w, location = loc, scale = scl)
rcauchymix(5, w = w, location = loc, scale = scl)
```

---

`causal_alt_pos500_p3_k3`

*causal_alt_pos500_p3_k3 dataset*

---

## Description

Causal dataset (N=500, p=3) with different positive-support kernels by arm. Intended for alternating-kernel causal vignettes (GPD=FALSE).

## Usage

```
causal_alt_pos500_p3_k3
```

## Format

A list with:

**y** Numeric outcome vector.

**A** Binary treatment indicator (0/1).

**X** data.frame with x1-x3.

**meta** List with N, support, p, K0, K1, tail, exceed_frac.

**truth** List with kernel0, kernel1, params0, params1, tail_params.

## Examples

```
head(causal_alt_pos500_p3_k3$X)
```

---

`causal_alt_pos500_p5_k4_tail`
*causal_alt_pos500_p5_k4_tail dataset*

---

## Description

Causal dataset (N=500, p=5) with different positive-support kernels by arm and tail-designed exceedances (GPD=TRUE).

## Usage

```
causal_alt_pos500_p5_k4_tail
```

## Format

A list with:

**y** Numeric outcome vector.

**A** Binary treatment indicator (0/1).

**X** data.frame with x1-x5.

**meta** List with N, support, p, K0, K1, tail, exceed_frac.

**truth** List with kernel0, kernel1, params0, params1, tail_params.

## Examples

```
head(causal_alt_pos500_p5_k4_tail$X)
```

---

`causal_alt_real500_p4_k2`
*causal_alt_real500_p4_k2 dataset*

---

## Description

Causal dataset (N=500, p=4) with different real-line kernels by arm. Intended for alternating-kernel causal vignettes (GPD=FALSE).

## Usage

```
causal_alt_real500_p4_k2
```

## Format

A list with:

**y** Numeric outcome vector.

**A** Binary treatment indicator (0/1).

**X** data.frame with x1-x4.

**meta** List with N, support, p, K0, K1, tail, exceed_frac.

**truth** List with kernel0, kernel1, params0, params1, tail_params.

## Examples

```
head(causal_alt_real500_p4_k2$X)
```

---

# causal_pos500_p3_k2 *causal_pos500_p3_k2 dataset*

---

## Description

Causal dataset (N=500, p=3) with the same positive-support kernel for both arms. Intended for same-kernel causal baselines (GPD=FALSE).

## Usage

```
causal_pos500_p3_k2
```

## Format

A list with:

**y** Numeric outcome vector.

**A** Binary treatment indicator (0/1).

**X** data.frame with x1-x3.

**meta** List with N, support, p, K0, K1, tail, exceed_frac.

**truth** List with kernel0, kernel1, params0, params1, tail_params.

## Examples

```
head(causal_pos500_p3_k2$X)
```

`check_glue_validity` *Validate bulk+tail glue for MixGPD predictive distribution*

## Description

This diagnostic checks whether the implied predictive distribution behaves like a valid distribution (monotone CDF in $[0, 1]$, nonnegative density, and sensible behavior around the threshold when a GPD tail is enabled).

## Usage

```
check_glue_validity(
  fit,
  x = NULL,
  grid = NULL,
  n_draws = 50L,
  tol = 1e-08,
  check_continuity = TRUE,
  eps = 1e-06
)
```

## Arguments

| | |
|---|---|
| `fit` | A `mixgpd_fit` object. |
| `x` | Optional design matrix for conditional models. If `NULL`, uses training `X`. |
| `grid` | Numeric evaluation grid. If `NULL`, defaults to a grid based on training `y`. |
| `n_draws` | Number of posterior draws to check (sampled without replacement when possible). |
| `tol` | Numerical tolerance for monotonicity/range checks. |
| `check_continuity` | Logical; if `TRUE` and GPD is enabled, checks continuity at the threshold. |
| `eps` | Small offset used for threshold continuity check. |

## Details

The check is performed draw-by-draw on a user-specified grid. It is intended for development, debugging, and CI (not for routine large-scale use).

## Value

A list with per-check pass/fail flags and summaries of violations.

cqte *Conditional quantile treatment effects*

## Description

`cqte()` evaluates treated-minus-control predictive quantiles at user-supplied covariate rows.

## Usage

```
cqte(
  fit,
  probs = c(0.1, 0.5, 0.9),
  newdata = NULL,
  interval = "credible",
  level = 0.95,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| `fit` | A `"causalmixgpd_causal_fit"` object from `run_mcmc_causal()`. |
| `probs` | Numeric vector of probabilities in (0, 1) specifying the quantile levels of the outcome distribution to estimate treatment effects at. |
| `newdata` | Optional data.frame or matrix of covariates for prediction. If `NULL`, uses the training covariates stored in `fit`. |
| `interval` | Character or NULL; type of credible interval:<br>• `NULL`: no interval<br>• `"credible"` (default): equal-tailed quantile intervals<br>• `"hpd"`: highest posterior density intervals |
| `level` | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| `show_progress` | Logical; if TRUE, print step messages and render progress where supported. |

## Details

For each prediction row $x$, the conditional quantile treatment effect is

$$\mathrm{CQTE}(\tau, x) = Q_1(\tau \mid x) - Q_0(\tau \mid x).$$

This estimand is available only for conditional causal models with covariates. For marginal quantile contrasts over the empirical covariate distribution, use `qte` or `qtt`.

If the fit includes a PS block, the same PS adjustment is applied to both arm predictions before differencing.

## Value

An object of class `"causalmixgpd_qte"` containing the CQTE summary, the probability grid, and the treated/control prediction objects used to construct the effect. The returned object includes a top-level `$fit_df` data frame for direct extraction.

## See Also

`qte`, `qtt`, `cate`, `predict.causalmixgpd_causal_fit`.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
cqte(fit, probs = c(0.5, 0.9), newdata = X[1:5, , drop = FALSE])
cqte(fit, probs = c(0.5, 0.9), interval = "credible", level = 0.90)  # 90% CI
cqte(fit, probs = c(0.5, 0.9), interval = "hpd")  # HPD intervals
cqte(fit, probs = c(0.5, 0.9), interval = NULL)   # No intervals
```

---

dpmgpd *Fit a one-arm Dirichlet process mixture with a spliced GPD tail*

---

## Description

`dpmgpd()` is the one-step convenience wrapper for the spliced bulk-tail model. It combines `bundle` and `mcmc` for one-arm data.

## Usage

```
dpmgpd(
  y = NULL,
  X = NULL,
  treat = NULL,
  data = NULL,
  mcmc = list(),
  formula = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| y | Either a response vector or a bundle object. |
| X | Optional design matrix/data.frame. |
| treat | Optional binary treatment indicator. If supplied, this wrapper errors; use dpmgpd.causal() for causal models. |
| data | Optional data.frame used with formula. |
| mcmc | Named list of run arguments passed to mcmc() (including optional performance controls such as parallel_chains, workers, timing, and z_update_every). |
| formula | Optional formula. |
| ... | Additional build arguments passed to build_nimble_bundle. |

### Details

This wrapper targets the posterior predictive distribution obtained by combining a flexible bulk DPM with a generalized Pareto exceedance model above the threshold $u(x)$. In the tail region the predictive density is proportional to

$$\{1 - p_u(x)\} f_{\mathrm{GPD}}(y \mid x), \qquad y > u(x),$$

where $p_u(x)$ is the posterior bulk mass below the threshold.

Use this wrapper when upper-tail behavior matters for inference, prediction, or extrapolation of extreme quantiles and survival probabilities.

### Value

A fitted object of class "mixgpd_fit".

### See Also

build_nimble_bundle, bundle, dpmix, predict.mixgpd_fit, summary.mixgpd_fit.

---

## dpmgpd.causal *Fit a causal two-arm Dirichlet process mixture with a spliced GPD tail*

---

### Description

dpmgpd.causal() is the highest-level causal fitting wrapper. It builds or accepts a causal bundle, runs posterior sampling for the treated and control arms, and returns a single causal fit ready for prediction and effect estimation.

## Usage

```
dpmgpd.causal(
  y = NULL,
  X = NULL,
  treat = NULL,
  data = NULL,
  mcmc = list(),
  formula = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| y | Either a response vector or a causal bundle object. |
| X | Optional design matrix/data.frame. |
| treat | Binary treatment indicator. |
| data | Optional data.frame used with formula. |
| mcmc | Named list of run arguments passed to mcmc() (including optional performance controls such as parallel_arms, workers, timing, and z_update_every). |
| formula | Optional formula. |
| ... | Additional build arguments passed to build_causal_bundle. |

## Details

The arm-specific predictive distributions $F_1(y \mid x)$ and $F_0(y \mid x)$ inherit the spliced bulk-tail structure. Downstream causal estimands are computed as functionals of these two predictive laws, for example

$$\mathrm{QTE}(\tau) = Q_1(\tau) - Q_0(\tau), \qquad \mathrm{ATE} = E(Y_1) - E(Y_0).$$

## Value

A fitted object of class "causalmixgpd_causal_fit".

## See Also

build_causal_bundle, bundle, dpmix.causal, predict.causalmixgpd_causal_fit, ate, qte, cate, cqte.

`dpmgpd.cluster` *Fit a clustering-only bulk-tail model*

## Description

Variant of `dpmix.cluster()` that augments the cluster kernel with a generalized Pareto tail. This is the clustering analogue of the spliced bulk-tail workflow used by `dpmgpd()`.

## Usage

```
dpmgpd.cluster(
  formula,
  data,
  type = c("weights", "param", "both"),
  default = "weights",
  mcmc = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| `formula` | Model formula. The response must be present in `data`. |
| `data` | Data frame containing the response and optional predictors. |
| `type` | Clustering mode:<br>• `"weights"`: links mixture weights to predictors<br>• `"param"`: links kernel parameters to predictors<br>• `"both"`: links both weights and kernel parameters to predictors |
| `default` | Default mode used when `type` is omitted. |
| `mcmc` | MCMC control list passed into the cluster bundle. |
| `...` | Additional arguments passed to `build_cluster_bundle()`, including kernel settings, prior overrides, component counts, and monitoring controls. |

## Details

For observations above a component-specific threshold, the component density is spliced as

$$f(y) = (1 - F_{bulk}(u))g_{GPD}(y \mid u, \sigma_u, \xi_u), \qquad y \geq u,$$

so cluster assignment can be informed by both central behavior and tail behavior.

This interface is preferable when cluster separation is driven by upper-tail differences rather than bulk-only shape or location differences.

## Value

Object of class `dpmixgpd_cluster_fit`.

## See Also

dpmix.cluster(), predict.dpmixgpd_cluster_fit(), dpmgpd(), sim_bulk_tail().

Other cluster workflow: dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

# dpmix *Fit a one-arm Dirichlet process mixture without a GPD tail*

---

## Description

dpmix() is the one-step convenience wrapper for the bulk-only model. It combines bundle and mcmc for one-arm data.

## Usage

```
dpmix(
  y = NULL,
  X = NULL,
  treat = NULL,
  data = NULL,
  mcmc = list(),
  formula = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| y | Either a response vector or a bundle object. |
| X | Optional design matrix/data.frame. |
| treat | Optional binary treatment indicator. If supplied, this wrapper errors; use dpmix.causal() for causal models. |
| data | Optional data.frame used with formula. |
| mcmc | Named list of run arguments passed to mcmc() (including optional performance controls such as parallel_chains, workers, timing, and z_update_every). |
| formula | Optional formula. |
| ... | Additional build arguments passed to build_nimble_bundle. |

### Details

The fitted model targets the posterior predictive bulk distribution

$$f(y \mid x) = \int f(y \mid x, \theta)\, d\Pi(\theta),$$

without the spliced tail augmentation used by dpmgpd.

Use this wrapper when the outcome support is adequately modeled by the bulk kernel alone. If you need threshold exceedance modeling or extreme-quantile extrapolation, use dpmgpd instead.

### Value

A fitted object of class "mixgpd_fit".

### See Also

build_nimble_bundle, bundle, dpmgpd, predict.mixgpd_fit, summary.mixgpd_fit.

---

## dpmix.causal *Fit a causal two-arm Dirichlet process mixture without a GPD tail*

---

### Description

dpmix.causal() fits a causal model with separate treated and control outcome mixtures and, when requested, a propensity score block. It is the bulk-only companion to dpmgpd.causal.

### Usage

```
dpmix.causal(
  y = NULL,
  X = NULL,
  treat = NULL,
  data = NULL,
  mcmc = list(),
  formula = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| y | Either a response vector or a causal bundle object. |
| X | Optional design matrix/data.frame. |
| treat | Binary treatment indicator. |
| data | Optional data.frame used with formula. |
| mcmc | Named list of run arguments passed to mcmc() (including optional performance controls such as parallel_arms, workers, timing, and z_update_every). |
| formula | Optional formula. |
| ... | Additional build arguments passed to build_causal_bundle. |

## Details

The resulting fit supports conditional outcome prediction $F_a(y \mid x)$ for $a \in \{0, 1\}$, followed by causal functionals such as `ate`, `qte`, `cate`, and `cqte`.

## Value

A fitted object of class `"causalmixgpd_causal_fit"`.

## See Also

`build_causal_bundle`, `bundle`, `dpmgpd.causal`, `predict.causalmixgpd_causal_fit`, `ate`, `qte`.

---

`dpmix.cluster` *Fit a clustering-only bulk model*

---

## Description

Build and fit a Dirichlet-process mixture for clustering without causal estimands or posterior prediction for a response surface. This interface focuses on latent partition recovery from a formula specification and returns a cluster-fit object that can be summarized, plotted, or converted into labels and posterior similarity matrices with `predict.dpmixgpd_cluster_fit()`.

## Usage

```
dpmix.cluster(
  formula,
  data,
  type = c("weights", "param", "both"),
  default = "weights",
  mcmc = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| `formula` | Model formula. The response must be present in `data`. |
| `data` | Data frame containing the response and optional predictors. |
| `type` | Clustering mode:<br>• `"weights"`: links mixture weights to predictors<br>• `"param"`: links kernel parameters to predictors<br>• `"both"`: links both weights and kernel parameters to predictors |
| `default` | Default mode used when `type` is omitted. |
| `mcmc` | MCMC control list passed into the cluster bundle. |
| `...` | Additional arguments passed to `build_cluster_bundle()`, including kernel settings, prior overrides, component counts, and monitoring controls. |

## Details

The fitted model targets a latent partition $z_1, \ldots, z_n$ with component-specific kernel parameters. Depending on `type`, predictors can enter through the gating probabilities

$$\Pr(z_i = k \mid x_i) = \pi_k(x_i)$$

or through linked kernel parameters for each component. The returned fit stores posterior draws of the latent cluster labels and associated parameters; the representative clustering is extracted later by `predict.dpmixgpd_cluster_fit()` using Dahl's least-squares rule.

Use `type = "weights"` or `type = "both"` only when the formula includes predictors and when an explicit number of `components` is supplied. Otherwise the builder stops before fitting.

## Value

Object of class `dpmixgpd_cluster_fit`.

## See Also

`dpmgpd.cluster()`, `predict.dpmixgpd_cluster_fit()`, `summary.dpmixgpd_cluster_fit()`, `plot.dpmixgpd_cluster_fit()`, `build_nimble_bundle()`, `dpmix()`.

Other cluster workflow: `dpmgpd.cluster()`, `plot.dpmixgpd_cluster_bundle()`, `plot.dpmixgpd_cluster_fit()`, `plot.dpmixgpd_cluster_labels()`, `plot.dpmixgpd_cluster_psm()`, `predict.dpmixgpd_cluster_fit()`, `print.dpmixgpd_cluster_bundle()`, `print.dpmixgpd_cluster_fit()`, `print.dpmixgpd_cluster_labels()`, `print.dpmixgpd_cluster_psm()`, `summary.dpmixgpd_cluster_bundle()`, `summary.dpmixgpd_cluster_fit()`, `summary.dpmixgpd_cluster_labels()`, `summary.dpmixgpd_cluster_psm()`

---

# ess_summary *Effective sample size summaries for fitted models*

---

## Description

`ess_summary()` reports effective sample size diagnostics for posterior draws, optionally scaled by wall-clock time.

## Usage

```
ess_summary(
  fit,
  params = NULL,
  per_chain = TRUE,
  wall_time = NULL,
  robust = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| fit | A "mixgpd_fit" or "causalmixgpd_causal_fit" object. |
| params | Optional character vector of parameter names/patterns. If NULL, a fixed canonical set is auto-resolved. |
| per_chain | Logical; if TRUE, include per-chain ESS rows. |
| wall_time | Optional numeric total MCMC time in seconds. If NULL, uses fit$timing$mcmc when available. |
| robust | Logical; if TRUE, ignore missing parameters. |
| ... | Unused. |

### Details

This is a convergence and efficiency diagnostic, not a model summary. For causal fits the function evaluates each outcome arm separately and tags the rows accordingly.

### Value

Object of class "mixgpd_ess_summary" with elements table, overall, and meta.

### See Also

summary.mixgpd_fit, plot.mixgpd_fit, params.

---

## fitted.mixgpd_fit *Fitted values on the training design*

---

### Description

fitted.mixgpd_fit() is a thin training-data wrapper around predict.mixgpd_fit for conditional models.

### Usage

```
## S3 method for class 'mixgpd_fit'
fitted(
  object,
  type = c("mean", "median", "quantile"),
  p = 0.5,
  level = 0.95,
  interval = "credible",
  seed = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A fitted object of class "mixgpd_fit" (must have covariates). |
| type | Which fitted functional to return:<br>• "mean": posterior predictive mean<br>• "median": posterior predictive median<br>• "quantile": posterior predictive quantile at level p |
| p | Quantile level used when type = "quantile". |
| level | Credible level for confidence intervals (default 0.95 for 95 percent credible intervals). |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| seed | Random seed used for deterministic fitted values. |
| ... | Unused. |

## Details

The method returns posterior predictive fitted values on the observed design matrix. It is available only when the fitted model stored covariates.

## Value

A data frame with columns for fitted values, optional intervals, and residuals computed on the training sample.

## See Also

predict.mixgpd_fit, residuals.mixgpd_fit, plot.mixgpd_fitted.

## Examples

```
# Conditional model (with covariates X)
y <- abs(stats::rnorm(25)) + 0.1
X <- data.frame(x1 = stats::rnorm(25), x2 = stats::runif(25))
bundle <- build_nimble_bundle(y = y, X = X, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                             mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
fitted(fit)
fitted(fit, level = 0.90)
fitted(fit, interval = "hpd")  # HPD intervals
fitted(fit, interval = NULL)   # No intervals
```

gamma_gpd *Gamma with a GPD tail*

## Description

Spliced family obtained by attaching a generalized Pareto tail above `threshold` to a single gamma bulk distribution.

## Usage

```
dGammaGpd(x, shape, scale, threshold, tail_scale, tail_shape, log = 0)

pGammaGpd(
  q,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rGammaGpd(n, shape, scale, threshold, tail_scale, tail_shape)

qGammaGpd(
  p,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `shape` | Numeric scalar Gamma shape parameter. |
| `scale` | Numeric scalar scale parameter for the Gamma bulk. |
| `threshold` | Numeric scalar threshold at which the GPD tail is attached. |
| `tail_scale` | Numeric scalar GPD scale parameter; must be positive. |

| | |
|---|---|
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric tolerance for numerical inversion in qGammaGpd. |
| maxiter | Maximum iterations for numerical inversion in qGammaGpd. |

## Details

This topic combines a single gamma bulk with a generalized Pareto exceedance model. If $F_\Gamma(u)$ is the bulk probability below the threshold, then the splice replaces the upper tail by $\{1-F_\Gamma(u)\}g_{GPD}(x)$ while leaving the lower region unchanged. The resulting distribution is continuous at the threshold and preserves the gamma body exactly below $u$.

The ordinary mean is finite only when the GPD shape satisfies $\xi < 1$. For heavier tails, predictive mean summaries should be replaced by restricted means or quantile summaries.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qGammaGpd() returns a numeric vector with the same length as p.

## Functions

- dGammaGpd(): Gamma + GPD tail density
- pGammaGpd(): Gamma + GPD tail distribution function
- rGammaGpd(): Gamma + GPD tail random generation
- qGammaGpd(): Gamma + GPD tail quantile function

## See Also

gamma_mix(), gamma_mixgpd(), gpd(), gamma_lowercase().

Other gamma kernel families: gamma_mix, gamma_mixgpd

## Examples

```
scale <- 2.5
shape <- 4
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dGammaGpd(4.0, scale = scale, shape = shape,
         threshold = threshold, tail_scale = tail_scale,
```

```
         tail_shape = tail_shape, log = 0)
pGammaGpd(4.0, scale = scale, shape = shape,
         threshold = threshold, tail_scale = tail_scale,
         tail_shape = tail_shape, lower.tail = 1, log.p = 0)
qGammaGpd(0.50, scale = scale, shape = shape,
         threshold = threshold, tail_scale = tail_scale,
         tail_shape = tail_shape)
qGammaGpd(0.95, scale = scale, shape = shape,
         threshold = threshold, tail_scale = tail_scale,
         tail_shape = tail_shape)
replicate(10, rGammaGpd(1, scale = scale, shape = shape,
                        threshold = threshold,
                        tail_scale = tail_scale,
                        tail_shape = tail_shape))
```

---

gamma_lowercase *Lowercase vectorized gamma distribution functions*

---

## Description

Vectorized R wrappers for the scalar gamma-kernel topics in this file.

## Usage

```
dgammamix(x, w, shape, scale, log = FALSE)

pgammamix(q, w, shape, scale, lower.tail = TRUE, log.p = FALSE)

qgammamix(
  p,
  w,
  shape,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rgammamix(n, w, shape, scale)

dgammamixgpd(
  x,
  w,
  shape,
  scale,
  threshold,
```

```
  tail_scale,
  tail_shape,
  log = FALSE
)

pgammamixgpd(
  q,
  w,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qgammamixgpd(
  p,
  w,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rgammamixgpd(n, w, shape, scale, threshold, tail_scale, tail_shape)

dgammagpd(x, shape, scale, threshold, tail_scale, tail_shape, log = FALSE)

pgammagpd(
  q,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qgammagpd(
  p,
```

```
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

rgammagpd(n, shape, scale, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles. |
| w | Numeric vector of mixture weights. |
| shape, scale | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |
| log | Logical; if TRUE, return log-density. |
| q | Numeric vector of quantiles. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq x)$. |
| log.p | Logical; if TRUE, probabilities are on log scale. |
| p | Numeric vector of probabilities. |
| tol, maxiter | Tolerance and max iterations for numerical inversion. |
| n | Integer number of observations to generate. |
| threshold, tail_scale, tail_shape | GPD tail parameters (scalars). |

## Details

These wrappers are vectorized interfaces to the scalar gamma and gamma-plus-GPD routines. They preserve the package's shape-scale parameterization and the same splice definition used in the fitted-model prediction code. Quantile wrappers delegate to the scalar inversion code rather than implementing separate approximations.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- dgammamix(): Gamma mixture density (vectorized)
- pgammamix(): Gamma mixture distribution function (vectorized)
- qgammamix(): Gamma mixture quantile function (vectorized)
- rgammamix(): Gamma mixture random generation (vectorized)
- dgammamixgpd(): Gamma mixture + GPD density (vectorized)
- pgammamixgpd(): Gamma mixture + GPD distribution function (vectorized)

- qgammamixgpd(): Gamma mixture + GPD quantile function (vectorized)
- rgammamixgpd(): Gamma mixture + GPD random generation (vectorized)
- dgammagpd(): Gamma + GPD density (vectorized)
- pgammagpd(): Gamma + GPD distribution function (vectorized)
- qgammagpd(): Gamma + GPD quantile function (vectorized)
- rgammagpd(): Gamma + GPD random generation (vectorized)

## See Also

gamma_mix(), gamma_mixgpd(), gamma_gpd(), bundle(), get_kernel_registry().

Other vectorized kernel helpers: amoroso_lowercase, base_lowercase, cauchy_mix_lowercase, invgauss_lowercase, laplace_lowercase, lognormal_lowercase, normal_lowercase

## Examples

```
w <- c(0.55, 0.3, 0.15)
shp <- c(2, 4, 6)
scl <- c(1, 2.5, 5)

# Gamma mixture
dgammamix(c(1, 2, 3), w = w, shape = shp, scale = scl)
rgammamix(5, w = w, shape = shp, scale = scl)

# Gamma mixture + GPD
dgammamixgpd(c(2, 3, 4), w = w, shape = shp, scale = scl,
             threshold = 3, tail_scale = 0.9, tail_shape = 0.2)

# Gamma + GPD (single component)
dgammagpd(c(2, 3, 4), shape = 4, scale = 2.5, threshold = 3,
          tail_scale = 0.9, tail_shape = 0.2)
```

---

# gamma_mix *Gamma mixture distribution*

---

## Description

Finite mixture of gamma components for positive-support bulk modeling. The scalar functions in this topic are the compiled building blocks behind the gamma bulk kernel family.

## Usage

```
dGammaMix(x, w, shape, scale, log = 0)

pGammaMix(q, w, shape, scale, lower.tail = 1, log.p = 0)

rGammaMix(n, w, shape, scale)
```

```
qGammaMix(
  p,
  w,
  shape,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions normalize w internally when needed. |
| shape, scale | Numeric vectors of length $K$ giving Gamma shape and scale parameters. |
| log | Logical; if TRUE, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k f_\Gamma(x \mid \alpha_k, \theta_k), \qquad x > 0,$$

with normalized weights $\tilde{w}_k$. For vectorized R usage, use gamma_lowercase().

Under the package parameterization, each component has density $f_\Gamma(x \mid \alpha, \theta) = x^{\alpha-1} \exp(-x/\theta)/\{\Gamma(\alpha)\theta^\alpha\}$ on $x > 0$. The mixture CDF is therefore

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k F_\Gamma(x \mid \alpha_k, \theta_k).$$

Random generation first selects a component according to the normalized mixture weights and then draws from the corresponding gamma distribution. Since finite gamma mixtures do not have closed form quantiles, qGammaMix() obtains them numerically by inverting the mixture CDF.

The analytical mean is

$$E(X) = \sum_{k=1}^{K} \tilde{w}_k \alpha_k \theta_k.$$

This expression is reused in posterior predictive mean calculations for gamma-based fits.

## Value

Density/CDF/RNG functions return numeric scalars. `qGammaMix()` returns a numeric vector with the same length as `p`.

## Functions

- `dGammaMix()`: Gamma mixture density
- `pGammaMix()`: Gamma mixture distribution function
- `rGammaMix()`: Gamma mixture random generation
- `qGammaMix()`: Gamma mixture quantile function

## See Also

`gamma_mixgpd()`, `gamma_gpd()`, `gamma_lowercase()`, `build_nimble_bundle()`, `kernel_support_table()`.

Other gamma kernel families: `gamma_gpd`, `gamma_mixgpd`

## Examples

```
w <- c(0.55, 0.30, 0.15)
scale <- c(1.0, 2.5, 5.0)
shape <- c(2, 4, 6)

dGammaMix(2.0, w = w, scale = scale, shape = shape, log = 0)
pGammaMix(2.0, w = w, scale = scale, shape = shape, lower.tail = 1, log.p = 0)
qGammaMix(0.50, w = w, scale = scale, shape = shape)
qGammaMix(0.95, w = w, scale = scale, shape = shape)
replicate(10, rGammaMix(1, w = w, scale = scale, shape = shape))
```

---

# gamma_mixgpd *Gamma mixture with a GPD tail*

---

## Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to a gamma mixture bulk.

## Usage

```
dGammaMixGpd(x, w, shape, scale, threshold, tail_scale, tail_shape, log = 0)

pGammaMixGpd(
  q,
  w,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
```

```
  lower.tail = 1,
  log.p = 0
)

rGammaMixGpd(n, w, shape, scale, threshold, tail_scale, tail_shape)

qGammaMixGpd(
  p,
  w,
  shape,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. |
| shape, scale | Numeric vectors of length $K$ giving Gamma shape and scale parameters. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Logical; if TRUE, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The gamma mixture governs the body of the distribution up to the threshold $u$. Beyond $u$, only the remaining survival mass is modeled by the GPD, giving

$$f(x) = \begin{cases} f_{mix}(x), & x < u, \\ \{1 - F_{mix}(u)\} g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

This is the positive-support analogue of the normal and lognormal splice families. Bulk quantiles are still found by numerical inversion, while tail quantiles use the explicit GPD inverse.

## Value

Spliced density/CDF/RNG functions return numeric scalars. `qGammaMixGpd()` returns a numeric vector with the same length as `p`.

## Functions

- `dGammaMixGpd()`: Gamma mixture + GPD tail density
- `pGammaMixGpd()`: Gamma mixture + GPD tail distribution function
- `rGammaMixGpd()`: Gamma mixture + GPD tail random generation
- `qGammaMixGpd()`: Gamma mixture + GPD tail quantile function

## See Also

`gamma_mix()`, `gamma_gpd()`, `gpd()`, `gamma_lowercase()`, `dpmgpd()`.

Other gamma kernel families: `gamma_gpd`, `gamma_mix`

## Examples

```
w <- c(0.55, 0.30, 0.15)
scale <- c(1.0, 2.5, 5.0)
shape <- c(2, 4, 6)
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dGammaMixGpd(4.0, w = w, scale = scale, shape = shape,
             threshold = threshold, tail_scale = tail_scale,
             tail_shape = tail_shape, log = 0)
pGammaMixGpd(4.0, w = w, scale = scale, shape = shape,
             threshold = threshold, tail_scale = tail_scale,
             tail_shape = tail_shape, lower.tail = 1, log.p = 0)
qGammaMixGpd(0.50, w = w, scale = scale, shape = shape,
             threshold = threshold, tail_scale = tail_scale,
             tail_shape = tail_shape)
qGammaMixGpd(0.95, w = w, scale = scale, shape = shape,
             threshold = threshold, tail_scale = tail_scale,
             tail_shape = tail_shape)
replicate(10, rGammaMixGpd(1, w = w, scale = scale, shape = shape,
                           threshold = threshold,
                           tail_scale = tail_scale,
                           tail_shape = tail_shape))
```

## get_kernel_registry *Get kernel registry*

### Description

Get kernel registry

### Usage

```
get_kernel_registry()
```

### Details

This accessor returns the registry created by `init_kernel_registry()`. The returned object is a named list keyed by kernel name. Each kernel definition describes which bulk parameters are present, how those parameters may depend on covariates, whether a GPD tail is allowed, and which density or mean functions should be dispatched for the supported backends.

Downstream builders use this registry as the package-level reference for kernel-specific implementation metadata. Reading it is appropriate when you need to inspect what the package believes a kernel can do before constructing or debugging a model specification.

### Value

A list of kernel metadata.

### Examples

```
init_kernel_registry()
reg <- get_kernel_registry()
reg$normal$bulk_params
```

## get_tail_registry *Get tail registry*

### Description

Get tail registry

### Usage

```
get_tail_registry()
```

## Details

The tail registry records the generalized Pareto splice used by bulk-tail models. It stores the tail parameter names `threshold`, `tail_scale`, and `tail_shape`, together with the support each parameter must satisfy and the modeling modes the builders may assign to them.

In mathematical terms, for a threshold $u$ the upper tail is represented with a generalized Pareto law for excesses above $u$. Accessing this registry is useful when inspecting how the package encodes those tail parameters before model compilation.

## Value

A list of tail metadata.

## Examples

```
init_kernel_registry()
tail <- get_tail_registry()
tail$params
```

---

# gpd *Generalized Pareto distribution*

---

## Description

Scalar generalized Pareto distribution (GPD) utilities for threshold exceedances above `threshold`. These NIMBLE-compatible functions provide the tail component used by the spliced bulk-tail families elsewhere in the package.

## Usage

```
dGpd(x, threshold, scale, shape, log = 0)

pGpd(q, threshold, scale, shape, lower.tail = 1, log.p = 0)

rGpd(n, threshold, scale, shape)

qGpd(p, threshold, scale, shape, lower.tail = TRUE, log.p = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `threshold` | Numeric scalar threshold at which the GPD is attached. |
| `scale` | Numeric scalar GPD scale parameter; must be positive. |
| `shape` | Numeric scalar GPD shape parameter. |
| `log` | Logical; if `TRUE`, return the log-density (integer flag `0/1` in NIMBLE). |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |

| | |
|---|---|
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |

## Details

The parameterization is

$$G(x) = 1 - \left(1 + \xi \frac{x-u}{\sigma_u}\right)^{-1/\xi}, \qquad x \geq u,$$

where threshold = u, scale = sigma_u > 0, and shape = xi. When shape approaches zero, the distribution reduces to the exponential tail limit.

These uppercase NIMBLE-compatible functions are scalar (x/q and n = 1). For vectorized R usage, use base_lowercase().

The associated density is

$$g(x) = \frac{1}{\sigma_u}\left(1 + \xi \frac{x-u}{\sigma_u}\right)^{-1/\xi - 1},$$

on the support where $1+\xi(x-u)/\sigma_u > 0$. When $\xi = 0$, this reduces to the exponential density with mean excess $\sigma_u$. The GPD has finite mean only when $\xi < 1$, and finite variance only when $\xi < 1/2$. Those existence conditions matter for downstream predictive means in the package: spliced models with $\xi \geq 1$ require restricted means rather than ordinary means.

If a bulk distribution has CDF $F_{bulk}$, the package's spliced families use the tail construction

$$F(x) = \begin{cases} F_{bulk}(x), & x < u, \\ F_{bulk}(u) + \{1 - F_{bulk}(u)\}G(x), & x \geq u. \end{cases}$$

## Value

dGpd() returns a numeric scalar density, pGpd() returns a numeric scalar CDF, rGpd() returns one random draw, and qGpd() returns a numeric quantile.

## Functions

- dGpd(): Generalized Pareto density function
- pGpd(): Generalized Pareto distribution function
- rGpd(): Generalized Pareto random generation
- qGpd(): Generalized Pareto quantile function

## See Also

base_lowercase(), normal_gpd(), lognormal_gpd(), gamma_gpd(), InvGauss_gpd(), laplace_gpd(), amoroso_gpd().

## Examples

```
threshold <- 1
tail_scale <- 0.8
tail_shape <- 0.2

dGpd(1.5, threshold, tail_scale, tail_shape, log = 0)
pGpd(1.5, threshold, tail_scale, tail_shape, lower.tail = 1, log.p = 0)
qGpd(0.50, threshold, tail_scale, tail_shape)
qGpd(0.95, threshold, tail_scale, tail_shape)
replicate(10, rGpd(1, threshold, tail_scale, tail_shape))
```

---

# init_kernel_registry *Initialize kernel registries*

---

## Description

Creates/refreshes registries used by the model specification compiler and code generators. Each kernel entry stores bulk parameters, supports, default regression/link behavior, and distribution signatures for SB/CRP backends.

## Usage

```
init_kernel_registry()
```

## Details

The kernel registry is the package-level contract that keeps model building, prediction, and documentation aligned. Each entry records the natural bulk parameters for one kernel, the support constraints they must satisfy, the default covariate-link strategy, and the backend-specific distribution names used when generating NIMBLE code.

The companion tail registry records the generalized Pareto tail parameters $u$ (threshold), $\sigma_u$ (tail scale), and $\xi_u$ (tail shape) together with their support and allowed modeling modes. Calling `init_kernel_registry()` makes those contracts available in the package namespace so later builders can validate requests without duplicating lookup logic.

## Value

Invisibly returns TRUE.

## Examples

```
init_kernel_registry()
reg <- get_kernel_registry()
names(reg)
tail <- get_tail_registry()
tail$params
```

InvGauss *Inverse Gaussian (Wald) distribution*

## Description

Scalar inverse Gaussian utilities under the $(\mu, \lambda)$ parameterization, where `mean = mu > 0` and `shape = lambda > 0`. These functions are used directly and as building blocks for inverse-Gaussian mixtures and spliced inverse-Gaussian-plus-GPD families.

## Usage

```
dInvGauss(x, mean, shape, log = 0)

pInvGauss(q, mean, shape, lower.tail = 1, log.p = 0)

rInvGauss(n, mean, shape)

qInvGauss(
  p,
  mean,
  shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `mean` | Numeric scalar mean parameter $\mu > 0$. |
| `shape` | Numeric scalar shape parameter $\lambda > 0$. |
| `log` | Logical; if `TRUE`, return the log-density (integer flag `0/1` in NIMBLE). |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq q)$. |
| `log.p` | Logical; if `TRUE`, probabilities are returned on the log scale. |
| `n` | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports `n = 1`. |
| `p` | Numeric scalar giving the probability for the quantile. |
| `tol` | Numeric scalar tolerance passed to `stats::uniroot`. |
| `maxiter` | Integer maximum number of iterations for `stats::uniroot`. |

## Details

The density is

$$f(x) = \left(\frac{\lambda}{2\pi x^3}\right)^{1/2} \exp\left\{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right\}, \qquad x > 0.$$

These uppercase NIMBLE-compatible functions are scalar (x/q and n = 1). For vectorized R usage, use base_lowercase().

The inverse Gaussian is the first-passage-time distribution of a Brownian motion with positive drift. Under the $(\mu, \lambda)$ parameterization used here, the mean is $E(X) = \mu$ and the variance is $\mathrm{Var}(X) = \mu^3/\lambda$. The implementation follows that parameterization throughout the package, so inverse-Gaussian mixture and splice families inherit the same interpretation.

The distribution function is evaluated through the standard normal representation

$$F(x) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right) + \exp\left(\frac{2\lambda}{\mu}\right)\Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right),$$

and the quantile is obtained numerically because no simple closed form is available.

## Value

dInvGauss() returns a numeric scalar density, pInvGauss() returns a numeric scalar CDF, rInvGauss() returns one random draw, and qInvGauss() returns a numeric quantile.

## Functions

- dInvGauss(): Inverse Gaussian density function
- pInvGauss(): Inverse Gaussian distribution function
- rInvGauss(): Inverse Gaussian random generation
- qInvGauss(): Inverse Gaussian quantile function

## See Also

InvGauss_mix(), InvGauss_gpd(), base_lowercase(), build_nimble_bundle().

Other base bulk distributions: amoroso, cauchy

## Examples

```
mean <- 2
shape <- 5

dInvGauss(2.0, mean, shape, log = 0)
pInvGauss(2.0, mean, shape, lower.tail = 1, log.p = 0)
qInvGauss(0.50, mean, shape)
qInvGauss(0.95, mean, shape)
replicate(10, rInvGauss(1, mean, shape))
```

`InvGauss_gpd` *Inverse Gaussian with a GPD tail*

## Description

Spliced family obtained by attaching a generalized Pareto tail above `threshold` to a single inverse Gaussian bulk.

## Usage

```
dInvGaussGpd(x, mean, shape, threshold, tail_scale, tail_shape, log = 0)

pInvGaussGpd(
  q,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rInvGaussGpd(n, mean, shape, threshold, tail_scale, tail_shape)

qInvGaussGpd(
  p,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `mean` | Numeric scalar mean parameter $\mu > 0$. |
| `shape` | Numeric scalar shape parameter $\lambda > 0$. |
| `threshold` | Numeric scalar threshold at which the GPD tail is attached. |
| `tail_scale` | Numeric scalar GPD scale parameter; must be positive. |

| | |
|---|---|
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric tolerance for numerical inversion in qInvGaussGpd. |
| maxiter | Maximum iterations for numerical inversion in qInvGaussGpd. |

## Details

This is the one-component version of InvGauss_mixgpd(). The inverse Gaussian governs the bulk region and the generalized Pareto governs exceedances over the threshold. The splice is continuous at $u$ because the GPD is scaled by the inverse-Gaussian survival probability at the threshold.

The ordinary mean of the spliced law exists only when the GPD tail has $\xi < 1$. When that condition fails, the package uses restricted means or quantile-based summaries instead of an ordinary mean.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qInvGaussGpd() returns a numeric vector with the same length as p.

## Functions

- dInvGaussGpd(): Inverse Gaussian + GPD tail density
- pInvGaussGpd(): Inverse Gaussian + GPD tail distribution function
- rInvGaussGpd(): Inverse Gaussian + GPD tail random generation
- qInvGaussGpd(): Inverse Gaussian + GPD tail quantile function

## See Also

InvGauss_mix(), InvGauss_mixgpd(), gpd(), invgauss_lowercase().

Other inverse-gaussian kernel families: InvGauss_mix, InvGauss_mixgpd

## Examples

```
mean <- 2.5
shape <- 6
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dInvGaussGpd(4.0, mean = mean, shape = shape,
             threshold = threshold, tail_scale = tail_scale,
```

```
            tail_shape = tail_shape, log = 0)
pInvGaussGpd(4.0, mean = mean, shape = shape,
            threshold = threshold, tail_scale = tail_scale,
            tail_shape = tail_shape, lower.tail = 1, log.p = 0)
qInvGaussGpd(0.50, mean = mean, shape = shape,
            threshold = threshold, tail_scale = tail_scale,
            tail_shape = tail_shape)
qInvGaussGpd(0.95, mean = mean, shape = shape,
            threshold = threshold, tail_scale = tail_scale,
            tail_shape = tail_shape)
replicate(10, rInvGaussGpd(1, mean = mean, shape = shape,
                           threshold = threshold,
                           tail_scale = tail_scale,
                           tail_shape = tail_shape))
```

---

`invgauss_lowercase` *Lowercase vectorized inverse Gaussian distribution functions*

---

## Description

Vectorized R wrappers for the scalar inverse-Gaussian-kernel topics in this file.

## Usage

```
dinvgaussmix(x, w, mean, shape, log = FALSE)

pinvgaussmix(q, w, mean, shape, lower.tail = TRUE, log.p = FALSE)

qinvgaussmix(
  p,
  w,
  mean,
  shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rinvgaussmix(n, w, mean, shape)

dinvgaussmixgpd(
  x,
  w,
  mean,
  shape,
  threshold,
  tail_scale,
```

```
  tail_shape,
  log = FALSE
)

pinvgaussmixgpd(
  q,
  w,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qinvgaussmixgpd(
  p,
  w,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rinvgaussmixgpd(n, w, mean, shape, threshold, tail_scale, tail_shape)

dinvgaussgpd(x, mean, shape, threshold, tail_scale, tail_shape, log = FALSE)

pinvgaussgpd(
  q,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qinvgaussgpd(
  p,
  mean,
```

```
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rinvgaussgpd(n, mean, shape, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector of quantiles. |
| `w` | Numeric vector of mixture weights. |
| `mean, shape` | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |
| `log` | Logical; if `TRUE`, return log-density. |
| `q` | Numeric vector of quantiles. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq x)$. |
| `log.p` | Logical; if `TRUE`, probabilities are on log scale. |
| `p` | Numeric vector of probabilities. |
| `tol, maxiter` | Tolerance and max iterations for numerical inversion. |
| `n` | Integer number of observations to generate. |
| `threshold, tail_scale, tail_shape` | GPD tail parameters (scalars). |

## Details

These functions are vectorized R front ends to the scalar inverse-Gaussian and splice routines. They retain the $(\mu, \lambda)$ parameterization used everywhere else in the package and apply the scalar evaluator repeatedly over the supplied input vector or draw index.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `dinvgaussmix()`: Inverse Gaussian mixture density (vectorized)
- `pinvgaussmix()`: Inverse Gaussian mixture distribution function (vectorized)
- `qinvgaussmix()`: Inverse Gaussian mixture quantile function (vectorized)
- `rinvgaussmix()`: Inverse Gaussian mixture random generation (vectorized)
- `dinvgaussmixgpd()`: Inverse Gaussian mixture + GPD density (vectorized)
- `pinvgaussmixgpd()`: Inverse Gaussian mixture + GPD distribution function (vectorized)

- `qinvgaussmixgpd()`: Inverse Gaussian mixture + GPD quantile function (vectorized)
- `rinvgaussmixgpd()`: Inverse Gaussian mixture + GPD random generation (vectorized)
- `dinvgaussgpd()`: Inverse Gaussian + GPD density (vectorized)
- `pinvgaussgpd()`: Inverse Gaussian + GPD distribution function (vectorized)
- `qinvgaussgpd()`: Inverse Gaussian + GPD quantile function (vectorized)
- `rinvgaussgpd()`: Inverse Gaussian + GPD random generation (vectorized)

## See Also

`InvGauss_mix()`, `InvGauss_mixgpd()`, `InvGauss_gpd()`, `bundle()`, `get_kernel_registry()`.

Other vectorized kernel helpers: `amoroso_lowercase`, `base_lowercase`, `cauchy_mix_lowercase`, `gamma_lowercase`, `laplace_lowercase`, `lognormal_lowercase`, `normal_lowercase`

## Examples

```
w <- c(0.6, 0.3, 0.1)
mu <- c(1, 1.5, 2)
lam <- c(2, 3, 4)

# Inverse Gaussian mixture
dinvgaussmix(c(1, 2, 3), w = w, mean = mu, shape = lam)
rinvgaussmix(5, w = w, mean = mu, shape = lam)
```

---

`InvGauss_mix` *Inverse Gaussian mixture distribution*

---

## Description

Finite mixture of inverse Gaussian components for positive-support bulk modeling. Each component is parameterized by `mean[j]` and `shape[j]`.

## Usage

```
dInvGaussMix(x, w, mean, shape, log = 0)

pInvGaussMix(q, w, mean, shape, lower.tail = 1, log.p = 0)

rInvGaussMix(n, w, mean, shape)

qInvGaussMix(
  p,
  w,
  mean,
  shape,
  lower.tail = TRUE,
```

```
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions normalize w internally when needed. |
| mean, shape | Numeric vectors of length $K$ giving component means and shapes. |
| log | Logical; if TRUE, return the log-density (integer flag 0/1 in NIMBLE). |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot in quantile inversion. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The scalar functions in this topic are the compiled building blocks for inverse-Gaussian bulk kernels. For vectorized R usage, use invgauss_lowercase().

The mixture distribution is

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k F_{IG}(x \mid \mu_k, \lambda_k),$$

where each inverse Gaussian component has mean $\mu_k$ and variance $\mu_k^3/\lambda_k$. Random generation selects a component using the normalized weights and then generates from the corresponding inverse Gaussian law. Quantiles are computed numerically because the finite-mixture inverse CDF is not available in closed form.

The analytical mixture mean is

$$E(X) = \sum_{k=1}^{K} \tilde{w}_k \mu_k.$$

That expression is used by the package whenever inverse-Gaussian mixtures contribute to posterior predictive means.

## Value

Density/CDF/RNG functions return numeric scalars. qInvGaussMix() returns a numeric vector with the same length as p.

### Functions

- dInvGaussMix(): Inverse Gaussian mixture density
- pInvGaussMix(): Inverse Gaussian mixture distribution function
- rInvGaussMix(): Inverse Gaussian mixture random generation
- qInvGaussMix(): Inverse Gaussian mixture quantile function

### See Also

InvGauss_mixgpd(), InvGauss_gpd(), invgauss_lowercase(), build_nimble_bundle(), kernel_support_table().

Other inverse-gaussian kernel families: InvGauss_gpd, InvGauss_mixgpd

### Examples

```
w <- c(0.55, 0.30, 0.15)
mean <- c(1.0, 2.5, 5.0)
shape <- c(2, 4, 8)

dInvGaussMix(2.0, w = w, mean = mean, shape = shape, log = 0)
pInvGaussMix(2.0, w = w, mean = mean, shape = shape,
             lower.tail = 1, log.p = 0)
qInvGaussMix(0.50, w = w, mean = mean, shape = shape)
qInvGaussMix(0.95, w = w, mean = mean, shape = shape)
replicate(10, rInvGaussMix(1, w = w, mean = mean, shape = shape))
```

---

## InvGauss_mixgpd *Inverse Gaussian mixture with a GPD tail*

---

### Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to an inverse Gaussian mixture bulk.

### Usage

```
dInvGaussMixGpd(x, w, mean, shape, threshold, tail_scale, tail_shape, log = 0)

pInvGaussMixGpd(
  q,
  w,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
```

```
)

rInvGaussMixGpd(n, w, mean, shape, threshold, tail_scale, tail_shape)

qInvGaussMixGpd(
  p,
  w,
  mean,
  shape,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. |
| mean, shape | Numeric vectors of length $K$ giving component means and shapes. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. For portability inside NIMBLE, the RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot in quantile inversion. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

This family keeps the inverse-Gaussian mixture body below the threshold $u$ and attaches a generalized Pareto exceedance law to the residual survival probability above $u$. If $F_{mix}(u) = p_u$, then the tail density is $(1 - p_u)g_{GPD}(x \mid u, \sigma_u, \xi)$.

Quantile evaluation is piecewise. For probabilities at or below $p_u$, the function solves the mixture inverse numerically; above $p_u$, it rescales the upper-tail probability and applies the GPD inverse directly.

## Value

Spliced density/CDF/RNG functions return numeric scalars. `qInvGaussMixGpd()` returns a numeric vector with the same length as `p`.

## Functions

- `dInvGaussMixGpd()`: Inverse Gaussian mixture + GPD tail density
- `pInvGaussMixGpd()`: Inverse Gaussian mixture + GPD tail distribution function
- `rInvGaussMixGpd()`: Inverse Gaussian mixture + GPD tail random generation
- `qInvGaussMixGpd()`: Inverse Gaussian mixture + GPD tail quantile function

## See Also

`InvGauss_mix()`, `InvGauss_gpd()`, `gpd()`, `invgauss_lowercase()`, `dpmgpd()`.

Other inverse-gaussian kernel families: `InvGauss_gpd`, `InvGauss_mix`

## Examples

```
w <- c(0.55, 0.30, 0.15)
mean <- c(1.0, 2.5, 5.0)
shape <- c(2, 4, 8)
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dInvGaussMixGpd(4.0, w = w, mean = mean, shape = shape,
                threshold = threshold, tail_scale = tail_scale,
                tail_shape = tail_shape, log = 0)
pInvGaussMixGpd(4.0, w = w, mean = mean, shape = shape,
                threshold = threshold, tail_scale = tail_scale,
                tail_shape = tail_shape, lower.tail = 1, log.p = 0)
qInvGaussMixGpd(0.50, w = w, mean = mean, shape = shape,
                threshold = threshold, tail_scale = tail_scale,
                tail_shape = tail_shape)
qInvGaussMixGpd(0.95, w = w, mean = mean, shape = shape,
                threshold = threshold, tail_scale = tail_scale,
                tail_shape = tail_shape)
replicate(10, rInvGaussMixGpd(1, w = w, mean = mean, shape = shape,
                              threshold = threshold,
                              tail_scale = tail_scale,
                              tail_shape = tail_shape))
```

## kernel_support_table *Kernel support matrix*

### Description

Returns a data frame summarizing each kernel's supported features.

### Usage

```
kernel_support_table(round = TRUE)
```

### Arguments

| | |
|---|---|
| round | Logical; TRUE to replace logical values with symbols. |

### Details

The returned table is a compact view of the registry contracts. Each row corresponds to one kernel, while the logical columns indicate whether that kernel can be paired with a GPD tail, whether covariate-linked parameter models are defined, and whether the stick-breaking (sb) and Chinese-restaurant (crp) backends are implemented.

This helper is intended for inspection and reporting rather than model fitting. It is a quick way to verify that a requested combination of kernel, tail, and backend is supported before calling higher-level workflow constructors.

### Value

data.frame with columns kernel, gpd, covariates, sb, crp.

## laplace_gpd *Laplace with a GPD tail*

### Description

Spliced family obtained by attaching a generalized Pareto tail above threshold to a single Laplace bulk.

### Usage

```
dLaplaceGpd(x, location, scale, threshold, tail_scale, tail_shape, log = 0)

pLaplaceGpd(
  q,
  location,
  scale,
```

```
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rLaplaceGpd(n, location, scale, threshold, tail_scale, tail_shape)

qLaplaceGpd(
  p,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| location | Numeric scalar location parameter for the Laplace bulk. |
| scale | Numeric scalar scale parameter for the Laplace bulk. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Logical; if TRUE, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |

## Details

This topic pairs a single Laplace bulk distribution with a generalized Pareto exceedance tail. The splice is continuous at the threshold because the tail density is multiplied by the Laplace survival probability at that threshold.

The ordinary mean exists only when the GPD tail has $\xi < 1$. If the fitted tail is too heavy for an ordinary mean to exist, the package directs users to restricted means or quantiles rather than returning an unstable mean summary.

## Value

Spliced density/CDF/RNG functions return numeric scalars. `qLaplaceGpd()` returns a numeric vector with the same length as `p`.

## Functions

- `dLaplaceGpd()`: Laplace + GPD tail density
- `pLaplaceGpd()`: Laplace + GPD tail distribution function
- `rLaplaceGpd()`: Laplace + GPD tail random generation
- `qLaplaceGpd()`: Laplace + GPD tail quantile function

## See Also

`laplace_mix()`, `laplace_MixGpd()`, `gpd()`, `laplace_lowercase()`.

Other laplace kernel families: `laplace_MixGpd`, `laplace_mix`

## Examples

```
location <- 0.5
scale <- 1.0
threshold <- 1
tail_scale <- 1.0
tail_shape <- 0.2

dLaplaceGpd(2.0, location, scale, threshold, tail_scale, tail_shape, log = FALSE)
pLaplaceGpd(2.0, location, scale, threshold, tail_scale, tail_shape,
           lower.tail = TRUE, log.p = FALSE)
qLaplaceGpd(0.50, location, scale, threshold, tail_scale, tail_shape)
qLaplaceGpd(0.95, location, scale, threshold, tail_scale, tail_shape)
replicate(10, rLaplaceGpd(1, location, scale, threshold, tail_scale, tail_shape))
```

---

# laplace_lowercase *Lowercase vectorized Laplace distribution functions*

---

## Description

Vectorized R wrappers for the scalar Laplace-kernel topics in this file.

## Usage

```
dlaplacemix(x, w, location, scale, log = FALSE)

plaplacemix(q, w, location, scale, lower.tail = TRUE, log.p = FALSE)

qlaplacemix(
  p,
  w,
```

```
  location,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rlaplacemix(n, w, location, scale)

dlaplacemixgpd(
  x,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  log = FALSE
)

plaplacemixgpd(
  q,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qlaplacemixgpd(
  p,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rlaplacemixgpd(n, w, location, scale, threshold, tail_scale, tail_shape)
```

```
dlaplacegpd(x, location, scale, threshold, tail_scale, tail_shape, log = FALSE)

plaplacegpd(
  q,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qlaplacegpd(
  p,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

rlaplacegpd(n, location, scale, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector of quantiles. |
| `w` | Numeric vector of mixture weights. |
| `location`, `scale` | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |
| `log` | Logical; if `TRUE`, return log-density. |
| `q` | Numeric vector of quantiles. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq x)$. |
| `log.p` | Logical; if `TRUE`, probabilities are on log scale. |
| `p` | Numeric vector of probabilities. |
| `tol`, `maxiter` | Tolerance and max iterations for numerical inversion. |
| `n` | Integer number of observations to generate. |
| `threshold`, `tail_scale`, `tail_shape` | GPD tail parameters (scalars). |

## Details

These helpers vectorize the scalar Laplace and Laplace-plus-GPD routines for interactive R use. They retain the same location-scale parameterization and the same splice definition as the uppercase

functions. Quantiles continue to use the scalar root-finding or piecewise logic rather than a separate approximation.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `dlaplacemix()`: Laplace mixture density (vectorized)
- `plaplacemix()`: Laplace mixture distribution function (vectorized)
- `qlaplacemix()`: Laplace mixture quantile function (vectorized)
- `rlaplacemix()`: Laplace mixture random generation (vectorized)
- `dlaplacemixgpd()`: Laplace mixture + GPD density (vectorized)
- `plaplacemixgpd()`: Laplace mixture + GPD distribution function (vectorized)
- `qlaplacemixgpd()`: Laplace mixture + GPD quantile function (vectorized)
- `rlaplacemixgpd()`: Laplace mixture + GPD random generation (vectorized)
- `dlaplacegpd()`: Laplace + GPD density (vectorized)
- `plaplacegpd()`: Laplace + GPD distribution function (vectorized)
- `qlaplacegpd()`: Laplace + GPD quantile function (vectorized)
- `rlaplacegpd()`: Laplace + GPD random generation (vectorized)

## See Also

`laplace_mix()`, `laplace_MixGpd()`, `laplace_gpd()`, `bundle()`, `get_kernel_registry()`.

Other vectorized kernel helpers: `amoroso_lowercase`, `base_lowercase`, `cauchy_mix_lowercase`, `gamma_lowercase`, `invgauss_lowercase`, `lognormal_lowercase`, `normal_lowercase`

## Examples

```
w <- c(0.6, 0.3, 0.1)
loc <- c(0, 1, -2)
scl <- c(1, 0.9, 1.1)

# Laplace mixture
dlaplacemix(c(-1, 0, 1), w = w, location = loc, scale = scl)
rlaplacemix(5, w = w, location = loc, scale = scl)
```

`laplace_mix` *Laplace (double exponential) mixture distribution*

## Description

Finite mixture of Laplace components for real-valued bulk modeling. The scalar functions in this topic are the NIMBLE-compatible building blocks for Laplace-based kernels.

## Usage

```
dLaplaceMix(x, w, location, scale, log = 0)

pLaplaceMix(q, w, location, scale, lower.tail = 1, log.p = 0)

rLaplaceMix(n, w, location, scale)

qLaplaceMix(
  p,
  w,
  location,
  scale,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `w` | Numeric vector of mixture weights of length $K$. The functions normalize `w` internally when needed. |
| `location` | Numeric vector of length $K$ giving component locations. |
| `scale` | Numeric vector of length $K$ giving component scales. |
| `log` | Logical; if `TRUE`, return the log-density. |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |
| `lower.tail` | Logical; if `TRUE` (default), probabilities are $P(X \leq q)$. |
| `log.p` | Logical; if `TRUE`, probabilities are returned on the log scale. |
| `n` | Integer giving the number of draws. The RNG implementation supports `n = 1`. |
| `p` | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| `tol` | Numeric scalar tolerance passed to `stats::uniroot`. |
| `maxiter` | Integer maximum iterations for `stats::uniroot`. |

## Details

The mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k f_{Lap}(x \mid \mu_k, b_k),$$

with normalized weights $\tilde{w}_k$. For vectorized R usage, use `laplace_lowercase()`.

Each component is a Laplace law with density $f(x \mid \mu, b) = (2b)^{-1} \exp\{-|x-\mu|/b\}$. The mixture CDF is the corresponding weighted average of component CDFs, and random generation selects a component first and then samples from that component. As with the other finite mixtures in the package, the quantile has no closed form and is therefore obtained numerically.

The analytical mean of the mixture is simply

$$E(X) = \sum_{k=1}^{K} \tilde{w}_k \mu_k.$$

That is the formula used in downstream predictive mean calculations for Laplace-based fits.

## Value

Density/CDF/RNG functions return numeric scalars. `qLaplaceMix()` returns a numeric vector with the same length as `p`.

## Functions

- `dLaplaceMix()`: Laplace mixture density
- `pLaplaceMix()`: Laplace mixture distribution function
- `rLaplaceMix()`: Laplace mixture random generation
- `qLaplaceMix()`: Laplace mixture quantile function

## See Also

`laplace_MixGpd()`, `laplace_gpd()`, `laplace_lowercase()`, `build_nimble_bundle()`, `kernel_support_table()`.

Other laplace kernel families: `laplace_MixGpd`, `laplace_gpd`

## Examples

```
w <- c(0.50, 0.30, 0.20)
location <- c(-1, 0.5, 2.0)
scale <- c(1.0, 0.7, 1.4)

dLaplaceMix(0.8, w = w, location = location, scale = scale, log = FALSE)
pLaplaceMix(0.8, w = w, location = location, scale = scale,
            lower.tail = TRUE, log.p = FALSE)
qLaplaceMix(0.50, w = w, location = location, scale = scale)
qLaplaceMix(0.95, w = w, location = location, scale = scale)
replicate(10, rLaplaceMix(1, w = w, location = location, scale = scale))
```

laplace_MixGpd *Laplace mixture with a GPD tail*

## Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to a Laplace mixture bulk.

## Usage

```
dLaplaceMixGpd(
  x,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  log = 0
)

pLaplaceMixGpd(
  q,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rLaplaceMixGpd(n, w, location, scale, threshold, tail_scale, tail_shape)

qLaplaceMixGpd(
  p,
  w,
  location,
  scale,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. |
| location | Numeric vector of length $K$ giving component locations. |
| scale | Numeric vector of length $K$ giving component scales. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Logical; if TRUE, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq q)$. |
| log.p | Logical; if TRUE, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum iterations for stats::uniroot. |

## Details

This family keeps the Laplace mixture body below the threshold and replaces the upper tail with a generalized Pareto exceedance model scaled by the residual survival mass at the threshold. The tail density is therefore

$$f(x) = \{1 - F_{mix}(u)\} g_{GPD}(x \mid u, \sigma_u, \xi), \qquad x \geq u.$$

Bulk quantiles are found numerically from the mixture CDF, and tail quantiles are computed by rescaling the upper-tail probability and applying the GPD inverse.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qLaplaceMixGpd() returns a numeric vector with the same length as p.

## Functions

- dLaplaceMixGpd(): Laplace mixture + GPD tail density
- pLaplaceMixGpd(): Laplace mixture + GPD tail distribution function
- rLaplaceMixGpd(): Laplace mixture + GPD tail random generation
- qLaplaceMixGpd(): Laplace mixture + GPD tail quantile function

## See Also

laplace_mix(), laplace_gpd(), gpd(), laplace_lowercase(), dpmgpd().

Other laplace kernel families: laplace_gpd, laplace_mix

## Examples

```
w <- c(0.50, 0.30, 0.20)
location <- c(-1, 0.5, 2.0)
scale <- c(1.0, 0.7, 1.4)
threshold <- 1
tail_scale <- 1.0
tail_shape <- 0.2

dLaplaceMixGpd(2.0, w = w, location = location, scale = scale,
               threshold = threshold, tail_scale = tail_scale,
               tail_shape = tail_shape, log = FALSE)
pLaplaceMixGpd(2.0, w = w, location = location, scale = scale,
               threshold = threshold, tail_scale = tail_scale,
               tail_shape = tail_shape, lower.tail = TRUE, log.p = FALSE)
qLaplaceMixGpd(0.50, w = w, location = location, scale = scale,
               threshold = threshold, tail_scale = tail_scale,
               tail_shape = tail_shape)
qLaplaceMixGpd(0.95, w = w, location = location, scale = scale,
               threshold = threshold, tail_scale = tail_scale,
               tail_shape = tail_shape)
replicate(10, rLaplaceMixGpd(1, w = w, location = location, scale = scale,
                             threshold = threshold,
                             tail_scale = tail_scale,
                             tail_shape = tail_shape))
```

---

# lognormal_gpd *Lognormal with a GPD tail*

---

## Description

Spliced family obtained by attaching a generalized Pareto tail above `threshold` to a single lognormal bulk.

## Usage

```
dLognormalGpd(x, meanlog, sdlog, threshold, tail_scale, tail_shape, log = 0)

pLognormalGpd(
  q,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)
```

```
rLognormalGpd(n, meanlog, sdlog, threshold, tail_scale, tail_shape)

qLognormalGpd(
  p,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| meanlog | Numeric scalar log-mean parameter for the Lognormal bulk. |
| sdlog | Numeric scalar log-standard deviation for the Lognormal bulk. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |

## Details

This is the single-lognormal counterpart of lognormal_mixgpd(). If $F_{LN}(u)$ denotes the bulk probability below the threshold, then the spliced density is

$$f(x) = \begin{cases} f_{LN}(x \mid \mu, \sigma), & x < u, \\ \{1 - F_{LN}(u)\} g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

The ordinary mean is finite only when the GPD tail has $\xi < 1$; otherwise the package requires restricted means or quantiles.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qLognormalGpd() returns a numeric vector with the same length as p.

## Functions

- dLognormalGpd(): Lognormal + GPD tail density
- pLognormalGpd(): Lognormal + GPD tail distribution function
- rLognormalGpd(): Lognormal + GPD tail random generation
- qLognormalGpd(): Lognormal + GPD tail quantile function

## See Also

lognormal_mix(), lognormal_mixgpd(), gpd(), lognormal_lowercase().

Other lognormal kernel families: lognormal_mix, lognormal_mixgpd

## Examples

```
meanlog <- 0.4
sdlog <- 0.35
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dLognormalGpd(4.0, meanlog, sdlog, threshold, tail_scale, tail_shape, log = FALSE)
pLognormalGpd(4.0, meanlog, sdlog, threshold, tail_scale, tail_shape,
              lower.tail = TRUE, log.p = FALSE)
qLognormalGpd(0.50, meanlog, sdlog, threshold, tail_scale, tail_shape)
qLognormalGpd(0.95, meanlog, sdlog, threshold, tail_scale, tail_shape)
replicate(10, rLognormalGpd(1, meanlog, sdlog, threshold, tail_scale, tail_shape))
```

---

# lognormal_lowercase *Lowercase vectorized lognormal distribution functions*

---

## Description

Vectorized R wrappers for the scalar lognormal-kernel topics in this file.

## Usage

```
dlognormalmix(x, w, meanlog, sdlog, log = FALSE)

plognormalmix(q, w, meanlog, sdlog, lower.tail = TRUE, log.p = FALSE)

qlognormalmix(
  p,
  w,
  meanlog,
  sdlog,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
```

```
  maxiter = 200
)

rlognormalmix(n, w, meanlog, sdlog)

dlognormalmixgpd(
  x,
  w,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  log = FALSE
)

plognormalmixgpd(
  q,
  w,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qlognormalmixgpd(
  p,
  w,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rlognormalmixgpd(n, w, meanlog, sdlog, threshold, tail_scale, tail_shape)

dlognormalgpd(
  x,
  meanlog,
  sdlog,
```

```
  threshold,
  tail_scale,
  tail_shape,
  log = FALSE
)

plognormalgpd(
  q,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qlognormalgpd(
  p,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

rlognormalgpd(n, meanlog, sdlog, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles. |
| w | Numeric vector of mixture weights. |
| meanlog, sdlog | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |
| log | Logical; if TRUE, return log-density. |
| q | Numeric vector of quantiles. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq x)$. |
| log.p | Logical; if TRUE, probabilities are on log scale. |
| p | Numeric vector of probabilities. |
| tol, maxiter | Tolerance and max iterations for numerical inversion. |
| n | Integer number of observations to generate. |
| threshold, tail_scale, tail_shape | GPD tail parameters (scalars). |

## Details

These are direct vectorized wrappers around the scalar lognormal routines. They keep the same parameterization, support restrictions, and bulk-tail splice, while allowing ordinary vector inputs in R. Quantile wrappers continue to use the scalar inversion logic, so there is no separate approximation layer in the lowercase API.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- `dlognormalmix()`: Lognormal mixture density (vectorized)
- `plognormalmix()`: Lognormal mixture distribution function (vectorized)
- `qlognormalmix()`: Lognormal mixture quantile function (vectorized)
- `rlognormalmix()`: Lognormal mixture random generation (vectorized)
- `dlognormalmixgpd()`: Lognormal mixture + GPD density (vectorized)
- `plognormalmixgpd()`: Lognormal mixture + GPD distribution function (vectorized)
- `qlognormalmixgpd()`: Lognormal mixture + GPD quantile function (vectorized)
- `rlognormalmixgpd()`: Lognormal mixture + GPD random generation (vectorized)
- `dlognormalgpd()`: Lognormal + GPD density (vectorized)
- `plognormalgpd()`: Lognormal + GPD distribution function (vectorized)
- `qlognormalgpd()`: Lognormal + GPD quantile function (vectorized)
- `rlognormalgpd()`: Lognormal + GPD random generation (vectorized)

## See Also

`lognormal_mix()`, `lognormal_mixgpd()`, `lognormal_gpd()`, `bundle()`, `get_kernel_registry()`.

Other vectorized kernel helpers: `amoroso_lowercase`, `base_lowercase`, `cauchy_mix_lowercase`, `gamma_lowercase`, `invgauss_lowercase`, `laplace_lowercase`, `normal_lowercase`

## Examples

```
w <- c(0.6, 0.3, 0.1)
ml <- c(0, 0.3, 0.6)
sl <- c(0.4, 0.5, 0.6)

# Lognormal mixture
dlognormalmix(c(1, 2, 3), w = w, meanlog = ml, sdlog = sl)
rlognormalmix(5, w = w, meanlog = ml, sdlog = sl)

# Lognormal mixture + GPD
dlognormalmixgpd(c(2, 3, 4), w = w, meanlog = ml, sdlog = sl,
                 threshold = 2.5, tail_scale = 0.5, tail_shape = 0.2)
```

`lognormal_mix` *Lognormal mixture distribution*

## Description

Finite mixture of lognormal components for positive-support bulk modeling. The scalar functions in this topic are the NIMBLE-compatible building blocks for the lognormal bulk kernel family.

## Usage

```
dLognormalMix(x, w, meanlog, sdlog, log = 0)

pLognormalMix(q, w, meanlog, sdlog, lower.tail = 1, log.p = 0)

rLognormalMix(n, w, meanlog, sdlog)

qLognormalMix(
  p,
  w,
  meanlog,
  sdlog,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `w` | Numeric vector of mixture weights of length $K$. The functions normalize `w` internally when needed. |
| `meanlog, sdlog` | Numeric vectors of length $K$ giving component log-means and log-standard deviations. |
| `log` | Integer flag `0/1`; if `1`, return the log-density. |
| `q` | Numeric scalar giving the point at which the distribution function is evaluated. |
| `lower.tail` | Integer flag `0/1`; if `1` (default), probabilities are $P(X \leq q)$. |
| `log.p` | Integer flag `0/1`; if `1`, probabilities are returned on the log scale. |
| `n` | Integer giving the number of draws. The RNG implementation supports `n = 1`. |
| `p` | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| `tol` | Numeric scalar tolerance passed to `stats::uniroot`. |
| `maxiter` | Integer maximum number of iterations for `stats::uniroot`. |

## Details

The mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k f_{LN}(x \mid \mu_k, \sigma_k), \qquad x > 0,$$

with normalized weights $\tilde{w}_k$. For vectorized R usage, use `lognormal_lowercase()`.

Each component satisfies $\log X_k \sim N(\mu_k, \sigma_k^2)$, so the mixture CDF is

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k \Phi\left(\frac{\log x - \mu_k}{\sigma_k}\right), \qquad x > 0.$$

Random generation proceeds by drawing a component index with probability $\tilde{w}_k$ and then sampling from the corresponding lognormal law. Because a finite mixture of lognormals does not admit a closed-form inverse CDF, `qLognormalMix()` computes quantiles by numerical inversion.

The analytical mixture mean is

$$E(X) = \sum_{k=1}^{K} \tilde{w}_k \exp(\mu_k + \sigma_k^2/2),$$

which is the expression used by the package whenever an ordinary predictive mean exists.

## Value

Density/CDF/RNG functions return numeric scalars. `qLognormalMix()` returns a numeric vector with the same length as `p`.

## Functions

- `dLognormalMix()`: Lognormal mixture density
- `pLognormalMix()`: Lognormal mixture distribution function
- `rLognormalMix()`: Lognormal mixture random generation
- `qLognormalMix()`: Lognormal mixture quantile function

## See Also

`lognormal_mixgpd()`, `lognormal_gpd()`, `lognormal_lowercase()`, `build_nimble_bundle()`, `kernel_support_table()`.

Other lognormal kernel families: `lognormal_gpd`, `lognormal_mixgpd`

## Examples

```
w <- c(0.60, 0.25, 0.15)
meanlog <- c(-0.2, 0.6, 1.2)
sdlog <- c(0.4, 0.3, 0.5)

dLognormalMix(2.0, w = w, meanlog = meanlog, sdlog = sdlog, log = FALSE)
```

```
pLognormalMix(2.0, w = w, meanlog = meanlog, sdlog = sdlog,
              lower.tail = TRUE, log.p = FALSE)
qLognormalMix(0.50, w = w, meanlog = meanlog, sdlog = sdlog)
qLognormalMix(0.95, w = w, meanlog = meanlog, sdlog = sdlog)
replicate(10, rLognormalMix(1, w = w, meanlog = meanlog, sdlog = sdlog))
```

---

`lognormal_mixgpd` *Lognormal mixture with a GPD tail*

---

## Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to a lognormal mixture bulk. Let $F_{mix}$ be the Lognormal mixture CDF. The spliced CDF is $F(x) = F_{mix}(x)$ for $x < threshold$ and $F(x) = F_{mix}(threshold) + \{1 - F_{mix}(threshold)\}G(x)$ for $x \geq threshold$, where $G$ is the GPD CDF for exceedances above `threshold`.

## Usage

```
dLognormalMixGpd(
  x,
  w,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  log = 0
)

pLognormalMixGpd(
  q,
  w,
  meanlog,
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rLognormalMixGpd(n, w, meanlog, sdlog, threshold, tail_scale, tail_shape)

qLognormalMixGpd(
  p,
  w,
  meanlog,
```

```
  sdlog,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. |
| meanlog, sdlog | Numeric vectors of length $K$ giving component log-means and log-standard deviations. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot in quantile inversion. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

The density, CDF, and RNG are implemented as nimbleFunctions. The quantile is an R function: it uses numerical inversion in the bulk region and the closed-form GPD quantile in the tail.

Let $F_{mix}$ be the lognormal-mixture CDF and let $u$ denote the threshold. The splice uses the bulk law below $u$ and attaches a GPD to the residual survival mass above $u$. The density therefore becomes

$$f(x) = \begin{cases} f_{mix}(x), & x < u, \\ \{1 - F_{mix}(u)\} g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

The quantile is computed piecewise: bulk quantiles are obtained numerically from the mixture CDF, whereas tail quantiles use the closed-form GPD inverse after rescaling the upper-tail probability.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qLognormalMixGpd() returns a numeric vector with the same length as p.

### Functions

- dLognormalMixGpd(): Lognormal mixture + GPD tail density
- pLognormalMixGpd(): Lognormal mixture + GPD tail distribution function
- rLognormalMixGpd(): Lognormal mixture + GPD tail random generation
- qLognormalMixGpd(): Lognormal mixture + GPD tail quantile function

### See Also

lognormal_mix(), lognormal_gpd(), gpd(), lognormal_lowercase(), dpmgpd().

Other lognormal kernel families: lognormal_gpd, lognormal_mix

### Examples

```
w <- c(0.60, 0.25, 0.15)
meanlog <- c(-0.2, 0.6, 1.2)
sdlog <- c(0.4, 0.3, 0.5)
threshold <- 3
tail_scale <- 0.9
tail_shape <- 0.2

dLognormalMixGpd(4.0, w = w, meanlog = meanlog, sdlog = sdlog,
                 threshold = threshold, tail_scale = tail_scale,
                 tail_shape = tail_shape, log = FALSE)
pLognormalMixGpd(4.0, w = w, meanlog = meanlog, sdlog = sdlog,
                 threshold = threshold, tail_scale = tail_scale,
                 tail_shape = tail_shape, lower.tail = TRUE, log.p = FALSE)
qLognormalMixGpd(0.50, w = w, meanlog = meanlog, sdlog = sdlog,
                 threshold = threshold, tail_scale = tail_scale,
                 tail_shape = tail_shape)
qLognormalMixGpd(0.95, w = w, meanlog = meanlog, sdlog = sdlog,
                 threshold = threshold, tail_scale = tail_scale,
                 tail_shape = tail_shape)
replicate(10, rLognormalMixGpd(1, w = w, meanlog = meanlog, sdlog = sdlog,
                               threshold = threshold,
                               tail_scale = tail_scale,
                               tail_shape = tail_shape))
```

---

## mcmc *Run posterior sampling from a prepared bundle*

---

### Description

mcmc() is the generic workflow runner. It dispatches to run_mcmc_bundle_manual for one-arm bundles and to run_mcmc_causal for causal bundles.

### Usage

```
mcmc(b, ...)
```

### Arguments

| | |
|---|---|
| b | A non-causal or causal bundle. |
| ... | Optional MCMC overrides (niter, nburnin, thin, nchains, seed, waic) and runner controls (show_progress, quiet). |

### Details

This wrapper is useful when you want a two-stage workflow: build first, inspect or modify the bundle, then sample. Named MCMC arguments supplied through ... override the settings stored in the bundle before execution.

The returned fit represents posterior draws from the finite SB/CRP approximation encoded in the bundle. Downstream summaries therefore target posterior predictive quantities such as $f(y \mid x)$, $F(y \mid x)$, and derived treatment-effect functionals.

### Value

A fitted object of class "mixgpd_fit" or "causalmixgpd_causal_fit".

### See Also

bundle, run_mcmc_bundle_manual, run_mcmc_causal, predict.mixgpd_fit.

---

## nc_pos200_k3 *nc_pos200_k3 dataset*

---

### Description

Positive-support, bulk-only mixture dataset with K=3 components and no covariates. Intended for non-causal bulk-only positive-kernel vignettes (gamma/lognormal/invgauss/amoroso).

### Usage

```
nc_pos200_k3
```

### Format

A list with:

**y** Numeric outcome vector.

**X** NULL.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

### Examples

```
head(nc_pos200_k3$y)
```

## nc_posX100_p3_k2 *nc_posX100_p3_k2 dataset*

### Description

Positive-support dataset with covariates (p=3) and K=2 mixture components. Intended for covariate and prediction vignettes (GPD=FALSE).

### Usage

```
nc_posX100_p3_k2
```

### Format

A list with:

**y** Numeric outcome vector.

**X** data.frame with x1-x3.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

### Examples

```
head(nc_posX100_p3_k2$X)
```

## nc_posX100_p4_k3 *nc_posX100_p4_k3 dataset*

### Description

Positive-support dataset with covariates (p=4) and K=3 mixture components. Intended for covariate and prediction vignettes (GPD=FALSE).

### Usage

```
nc_posX100_p4_k3
```

### Format

A list with:

**y** Numeric outcome vector.

**X** data.frame with x1-x4.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

## Examples

```
head(nc_posX100_p4_k3$X)
```

---

`nc_posX100_p5_k4` *nc_posX100_p5_k4 dataset*

---

## Description

Positive-support dataset with covariates (p=5) and K=4 mixture components. Intended for covariate and prediction vignettes (GPD=FALSE).

## Usage

```
nc_posX100_p5_k4
```

## Format

A list with:

**y** Numeric outcome vector.

**X** data.frame with x1-x5.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

## Examples

```
head(nc_posX100_p5_k4$X)
```

---

`nc_pos_tail200_k4` *nc_pos_tail200_k4 dataset*

---

## Description

Positive-support, tail-designed mixture dataset with K=4 components and no covariates. Intended for GPD vignettes (gamma/lognormal/invgauss/amoroso with GPD=TRUE).

## Usage

```
nc_pos_tail200_k4
```

## Format

A list with:

**y** Numeric outcome vector.

**X** NULL.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

## Examples

```
head(nc_pos_tail200_k4$y)
```

---

`nc_real200_k2` *nc_real200_k2 dataset*

---

## Description

Real-line, bulk-only mixture dataset with K=2 components and no covariates. Intended for non-causal bulk-only vignettes (normal/laplace/cauchy, GPD=FALSE).

## Usage

```
nc_real200_k2
```

## Format

A list with:

**y** Numeric outcome vector.

**X** NULL.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

## Examples

```
head(nc_real200_k2$y)
```

## nc_realX100_p3_k2 *nc_realX100_p3_k2 dataset*

### Description

Real-line dataset with covariates (p=3) and K=2 mixture components. Intended for covariate and prediction vignettes (GPD=FALSE).

### Usage

```
nc_realX100_p3_k2
```

### Format

A list with:

**y** Numeric outcome vector.

**X** data.frame with x1-x3.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

### Examples

```
head(nc_realX100_p3_k2$X)
```

## nc_realX100_p5_k3 *nc_realX100_p5_k3 dataset*

### Description

Real-line dataset with covariates (p=5) and K=3 mixture components. Intended for covariate and prediction vignettes (GPD=FALSE).

### Usage

```
nc_realX100_p5_k3
```

### Format

A list with:

**y** Numeric outcome vector.

**X** data.frame with x1-x5.

**meta** List with n, support, p, K_true, tail, exceed_frac, seed.

**truth** List with kernel, weights, params, threshold, tail_params.

## Examples

```
head(nc_realX100_p5_k3$X)
```

---

`normal_gpd` *Normal with a GPD tail*

---

## Description

Spliced family obtained by attaching a generalized Pareto tail above `threshold` to a single normal bulk component.

## Usage

```
dNormGpd(x, mean, sd, threshold, tail_scale, tail_shape, log = 0)

pNormGpd(
  q,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
)

rNormGpd(n, mean, sd, threshold, tail_scale, tail_shape)

qNormGpd(
  p,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | Numeric scalar giving the point at which the density is evaluated. |
| `mean` | Numeric scalar mean parameter for the Normal bulk. |
| `sd` | Numeric scalar standard deviation for the Normal bulk. |
| `threshold` | Numeric scalar threshold at which the GPD tail is attached. |
| `tail_scale` | Numeric scalar GPD scale parameter; must be positive. |

| | |
|---|---|
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |

## Details

This is the single-component version of normal_mixgpd(). If $\Phi_u = \Phi((u - \mu)/\sigma)$ denotes the normal bulk probability below the threshold, then the density is

$$f(x) = \begin{cases} \phi(x \mid \mu, \sigma^2), & x < u, \\ (1 - \Phi_u) g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

The distribution is continuous at $u$ by construction, although the derivative generally changes there because the tail is modeled by a different family.

The ordinary mean exists only when the GPD tail satisfies $\xi < 1$. When that condition fails, downstream mean prediction is intentionally blocked and the package directs the user to restricted means or quantile-based summaries instead.

## Value

Spliced density/CDF/RNG functions return numeric scalars. qNormGpd() returns a numeric vector with the same length as p.

## Functions

- dNormGpd(): Normal + GPD tail density
- pNormGpd(): Normal + GPD tail distribution function
- rNormGpd(): Normal + GPD tail random generation
- qNormGpd(): Normal + GPD tail quantile function

## See Also

normal_mix(), normal_mixgpd(), gpd(), normal_lowercase().

Other normal kernel families: normal_mix, normal_mixgpd

## Examples

```
mean <- 0.5
sd <- 1.0
threshold <- 2
tail_scale <- 1.0
tail_shape <- 0.2
```

```
dNormGpd(3.0, mean, sd, threshold, tail_scale, tail_shape, log = FALSE)
pNormGpd(3.0, mean, sd, threshold, tail_scale, tail_shape,
        lower.tail = TRUE, log.p = FALSE)
qNormGpd(0.50, mean, sd, threshold, tail_scale, tail_shape)
qNormGpd(0.95, mean, sd, threshold, tail_scale, tail_shape)
replicate(10, rNormGpd(1, mean, sd, threshold, tail_scale, tail_shape))
```

---

`normal_lowercase` *Lowercase vectorized normal distribution functions*

---

## Description

Vectorized R wrappers for the scalar normal-kernel topics in this file. These helpers are meant for interactive use and examples rather than direct use inside NIMBLE code.

## Usage

```
dnormmix(x, w, mean, sd, log = FALSE)

pnormmix(q, w, mean, sd, lower.tail = TRUE, log.p = FALSE)

qnormmix(
  p,
  w,
  mean,
  sd,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rnormmix(n, w, mean, sd)

dnormmixgpd(x, w, mean, sd, threshold, tail_scale, tail_shape, log = FALSE)

pnormmixgpd(
  q,
  w,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)
```

```
qnormmixgpd(
  p,
  w,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)

rnormmixgpd(n, w, mean, sd, threshold, tail_scale, tail_shape)

dnormgpd(x, mean, sd, threshold, tail_scale, tail_shape, log = FALSE)

pnormgpd(
  q,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

qnormgpd(
  p,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE
)

rnormgpd(n, mean, sd, threshold, tail_scale, tail_shape)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles. |
| w | Numeric vector of mixture weights. |
| mean, sd | Numeric vectors (mix) or scalars (base+gpd) of component parameters. |

| | |
|---|---|
| log | Logical; if TRUE, return log-density. |
| q | Numeric vector of quantiles. |
| lower.tail | Logical; if TRUE (default), probabilities are $P(X \leq x)$. |
| log.p | Logical; if TRUE, probabilities are on log scale. |
| p | Numeric vector of probabilities. |
| tol, maxiter | Tolerance and max iterations for numerical inversion. |
| n | Integer number of observations to generate. |
| threshold, tail_scale, tail_shape | GPD tail parameters (scalars). |

## Details

These wrappers vectorize the scalar normal-kernel routines for ordinary R use. They preserve the same formulas, parameter meanings, and tail construction as the uppercase functions; the only change is that x, q, p, and n may now be length greater than one.

For the mixture quantile and splice quantile functions, the numerical and piecewise logic is delegated directly to the corresponding scalar routine. As a result, the lowercase helpers are faithful front ends rather than separate implementations.

## Value

Numeric vector of densities, probabilities, quantiles, or random variates.

## Functions

- dnormmix(): Normal mixture density (vectorized)
- pnormmix(): Normal mixture distribution function (vectorized)
- qnormmix(): Normal mixture quantile function (vectorized)
- rnormmix(): Normal mixture random generation (vectorized)
- dnormmixgpd(): Normal mixture + GPD density (vectorized)
- pnormmixgpd(): Normal mixture + GPD distribution function (vectorized)
- qnormmixgpd(): Normal mixture + GPD quantile function (vectorized)
- rnormmixgpd(): Normal mixture + GPD random generation (vectorized)
- dnormgpd(): Normal + GPD density (vectorized)
- pnormgpd(): Normal + GPD distribution function (vectorized)
- qnormgpd(): Normal + GPD quantile function (vectorized)
- rnormgpd(): Normal + GPD random generation (vectorized)

## See Also

normal_mix(), normal_mixgpd(), normal_gpd(), bundle(), get_kernel_registry().

Other vectorized kernel helpers: amoroso_lowercase, base_lowercase, cauchy_mix_lowercase, gamma_lowercase, invgauss_lowercase, laplace_lowercase, lognormal_lowercase

## Examples

```
w <- c(0.6, 0.25, 0.15)
mu <- c(-1, 0.5, 2)
sig <- c(1, 0.7, 1.3)

# Normal mixture
dnormmix(c(0, 1, 2), w = w, mean = mu, sd = sig)
rnormmix(5, w = w, mean = mu, sd = sig)

# Normal mixture + GPD
dnormmixgpd(c(1, 2, 3), w = w, mean = mu, sd = sig,
            threshold = 2, tail_scale = 1, tail_shape = 0.2)

# Normal + GPD (single component)
dnormgpd(c(1, 2, 3), mean = 0.5, sd = 1, threshold = 2,
         tail_scale = 1, tail_shape = 0.2)
```

---

# normal_mix *Normal mixture distribution*

---

## Description

Finite mixture of normal components for real-valued bulk modeling. This topic provides the scalar mixture density, CDF, RNG, and quantile functions used by the bulk-only normal kernel in the package registry.

## Usage

```
dNormMix(x, w, mean, sd, log = 0)

pNormMix(q, w, mean, sd, lower.tail = 1, log.p = 0)

rNormMix(n, w, mean, sd)

qNormMix(
  p,
  w,
  mean,
  sd,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. The functions normalize w internally when needed. |
| mean, sd | Numeric vectors of length $K$ giving component means and standard deviations. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

With weights $w_k$, means $\mu_k$, and standard deviations $\sigma_k$, the mixture density is

$$f(x) = \sum_{k=1}^{K} \tilde{w}_k \phi(x \mid \mu_k, \sigma_k^2),$$

where $\tilde{w}_k = w_k / \sum_j w_j$. These uppercase functions are scalar NIMBLE-compatible building blocks. For vectorized R usage, use normal_lowercase().

If $F_k$ denotes the $k$-th component CDF, then the mixture distribution function is

$$F(x) = \sum_{k=1}^{K} \tilde{w}_k F_k(x) = \sum_{k=1}^{K} \tilde{w}_k \Phi\left(\frac{x - \mu_k}{\sigma_k}\right).$$

Random generation first draws a component index with probability $\tilde{w}_k$ and then generates from the corresponding normal law. The quantile function has no closed form for a general finite mixture, so qNormMix() solves $F(x) = p$ numerically by bracketing the root and applying stats::uniroot().

The mixture mean is

$$E(X) = \sum_{k=1}^{K} \tilde{w}_k \mu_k,$$

which is the analytical mean used by the package when a normal-mixture draw contributes to a posterior predictive mean calculation.

## Value

Density/CDF/RNG functions return numeric scalars. qNormMix() returns a numeric vector with the same length as p.

### Functions

- dNormMix(): Normal mixture density
- pNormMix(): Normal mixture distribution function
- rNormMix(): Normal mixture random generation
- qNormMix(): Normal mixture quantile function

### See Also

normal_mixgpd(), normal_gpd(), normal_lowercase(), build_nimble_bundle(), kernel_support_table().

Other normal kernel families: normal_gpd, normal_mixgpd

### Examples

```
w <- c(0.60, 0.25, 0.15)
mean <- c(-1, 0.5, 2.0)
sd <- c(1.0, 0.7, 1.3)

dNormMix(0.5, w = w, mean = mean, sd = sd, log = FALSE)
pNormMix(0.5, w = w, mean = mean, sd = sd,
        lower.tail = TRUE, log.p = FALSE)
qNormMix(0.50, w = w, mean = mean, sd = sd)
qNormMix(0.95, w = w, mean = mean, sd = sd)
replicate(10, rNormMix(1, w = w, mean = mean, sd = sd))
```

---

## normal_mixgpd *Normal mixture with a GPD tail*

---

### Description

Spliced bulk-tail family formed by attaching a generalized Pareto tail to a normal mixture bulk. This matches the structure used by the package's normal mixgpd kernels.

### Usage

```
dNormMixGpd(x, w, mean, sd, threshold, tail_scale, tail_shape, log = 0)

pNormMixGpd(
  q,
  w,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = 1,
  log.p = 0
```

```
)

rNormMixGpd(n, w, mean, sd, threshold, tail_scale, tail_shape)

qNormMixGpd(
  p,
  w,
  mean,
  sd,
  threshold,
  tail_scale,
  tail_shape,
  lower.tail = TRUE,
  log.p = FALSE,
  tol = 1e-10,
  maxiter = 200
)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar giving the point at which the density is evaluated. |
| w | Numeric vector of mixture weights of length $K$. |
| mean, sd | Numeric vectors of length $K$ giving component means and standard deviations. |
| threshold | Numeric scalar threshold at which the GPD tail is attached. |
| tail_scale | Numeric scalar GPD scale parameter; must be positive. |
| tail_shape | Numeric scalar GPD shape parameter. |
| log | Integer flag 0/1; if 1, return the log-density. |
| q | Numeric scalar giving the point at which the distribution function is evaluated. |
| lower.tail | Integer flag 0/1; if 1 (default), probabilities are $P(X \leq q)$. |
| log.p | Integer flag 0/1; if 1, probabilities are returned on the log scale. |
| n | Integer giving the number of draws. The RNG implementation supports n = 1. |
| p | Numeric scalar probability in $(0, 1)$ for the quantile function. |
| tol | Numeric scalar tolerance passed to stats::uniroot. |
| maxiter | Integer maximum number of iterations for stats::uniroot. |

## Details

If $F_{mix}$ denotes the normal-mixture CDF, the spliced CDF is

$$F(x) = \begin{cases} F_{mix}(x), & x < u, \\ F_{mix}(u) + \{1 - F_{mix}(u)\}G(x), & x \geq u, \end{cases}$$

where threshold = u and $G$ is the GPD CDF.

The construction keeps the normal mixture unchanged below the threshold $u$ and replaces the upper tail by a generalized Pareto exceedance model. Writing $F_{mix}(u) = p_u$, the spliced density is

$$f(x) = \begin{cases} f_{mix}(x), & x < u, \\ \{1 - p_u\} g_{GPD}(x \mid u, \sigma_u, \xi), & x \geq u. \end{cases}$$

This formulation preserves total probability because the GPD is attached only to the residual survival mass above the bulk threshold.

The quantile is piecewise. If $p \leq p_u$, `qNormMixGpd()` inverts the bulk mixture CDF; otherwise it rescales the tail probability to $(p - p_u)/(1 - p_u)$ and applies the closed-form GPD quantile. That same piecewise logic is what the fitted-model prediction code uses draw by draw.

## Value

Spliced density/CDF/RNG functions return numeric scalars. `qNormMixGpd()` returns a numeric vector with the same length as `p`.

## Functions

- `dNormMixGpd()`: Normal mixture + GPD tail density
- `pNormMixGpd()`: Normal mixture + GPD tail distribution function
- `rNormMixGpd()`: Normal mixture + GPD tail random generation
- `qNormMixGpd()`: Normal mixture + GPD tail quantile function

## See Also

`normal_mix()`, `normal_gpd()`, `gpd()`, `normal_lowercase()`, `dpmgpd()`.

Other normal kernel families: `normal_gpd`, `normal_mix`

## Examples

```
w <- c(0.60, 0.25, 0.15)
mean <- c(-1, 0.5, 2.0)
sd <- c(1.0, 0.7, 1.3)
threshold <- 2
tail_scale <- 1.0
tail_shape <- 0.2

dNormMixGpd(3.0, w, mean, sd, threshold, tail_scale, tail_shape, log = FALSE)
pNormMixGpd(3.0, w, mean, sd, threshold, tail_scale, tail_shape,
           lower.tail = TRUE, log.p = FALSE)
qNormMixGpd(0.50, w, mean, sd, threshold, tail_scale, tail_shape)
qNormMixGpd(0.95, w, mean, sd, threshold, tail_scale, tail_shape)
replicate(10, rNormMixGpd(1, w, mean, sd, threshold, tail_scale, tail_shape))
```

params *Extract posterior mean parameters in natural shape*

## Description

`params()` reshapes posterior mean summaries back into the parameter layout implied by the fitted model specification.

## Usage

```
params(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"mixgpd_fit"`. |
| `...` | Unused. |

## Details

This extractor is intended for structural inspection of the fitted model. Scalar quantities remain scalar, component-specific parameters are returned as vectors, and linked regression blocks are returned as matrices with covariate names as columns when available. If propensity-score adjustment is active for a linked bulk parameter, its coefficient is folded into the returned beta matrix as a leading `"PropScore"` column.

For a spliced model, the extractor returns posterior means of the bulk mixture parameters together with component-level threshold, tail-scale, and tail-shape terms. When tail terms are link-mode, the corresponding component-by-covariate beta blocks are returned.

## Value

An object of class `"mixgpd_params"` (a named list). For causal fits, `params()` returns a treated/control pair and includes a `ps` block when a propensity-score model was fitted.

## See Also

`summary.mixgpd_fit`, `predict.mixgpd_fit`, `ess_summary`.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                            mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
params(fit)
p <- params(fit)
```

plot.causalmixgpd_ate *Plot ATE-style effect summaries*

## Description

plot.causalmixgpd_ate() visualizes objects returned by ate, att, cate, and ate_rmean. The type parameter controls the plot style. When type is omitted, cate() objects default to "effect":

- "both" (default): Returns a list with both trt_control (treated vs control means) and treatment_effect (ATE curve) plots
- "effect": ATE curve/points vs index/PS with pointwise CI error bars
- "arms": Treated mean vs control mean, with pointwise CI error bars

## Usage

```
## S3 method for class 'causalmixgpd_ate'
plot(
  x,
  y = NULL,
  type = c("both", "effect", "arms"),
  plotly = getOption("CausalMixGPD.plotly", FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| x | Object of class causalmixgpd_ate. |
| y | Ignored. |
| type | Character; plot type:<br>• "both" (default): returns a list with both arm means and treatment-effect plots<br>• "effect": ATE curve/points with pointwise CI error bars<br>• "arms": treated vs control mean with pointwise CI error bars |
| plotly | Logical; if TRUE, convert the ggplot2 output to a plotly / htmlwidget representation via .wrap_plotly(). Defaults to getOption("CausalMixGPD.plotly", FALSE). |
| ... | Additional arguments passed to ggplot2 functions. |

## Details

The effect panel visualizes the posterior summary of the treatment contrast on the mean scale, namely $E(Y^1) - E(Y^0)$ or its conditional or treated-standardized analogue. The arms panel instead shows the treated and control mean predictions whose difference defines that contrast.

For cate() objects, the x-axis follows the prediction profiles; otherwise it uses the estimated propensity score when available or a simple index order. This keeps the comparison aligned with how the effect object was standardized.

## Value

A list of ggplot objects with elements `trt_control` and `treatment_effect` (if `type="both"`), or a single ggplot object (if `type` is `"effect"` or `"arms"`).

## See Also

`ate`, `cate`, `summary.causalmixgpd_ate`.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
X_new <- X[1:5, , drop = FALSE]
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
ate_result <- cate(fit, newdata = X_new, interval = "credible")
plot(ate_result)  # CATE default: effect plot
plot(ate_result, type = "effect")  # single ATE plot
plot(ate_result, type = "arms")    # single arms plot
```

---

# plot.causalmixgpd_causal_fit

*Plot the treated and control outcome fits from a causal model*

---

## Description

`plot.causalmixgpd_causal_fit()` is a convenience router to the underlying one-arm diagnostic plots for the treated and control fits.

## Usage

```
## S3 method for class 'causalmixgpd_causal_fit'
plot(x, arm = "both", ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"causalmixgpd_causal_fit"` object. |
| `arm` | Integer or character; `1` or `"treated"` for treatment, `0` or `"control"` for control. |
| `...` | Additional arguments forwarded to the underlying outcome plot method. |

## Details

Each arm-specific outcome model is itself a `mixgpd_fit`, so this method delegates to `plot.mixgpd_fit()` for the selected arm. With `arm = "both"`, it returns a named list of treated and control diagnostics so the two fitted outcome models can be assessed side by side.

These are MCMC diagnostics for the nuisance outcome models, not plots of causal estimands. Use `plot()` on objects from `predict.causalmixgpd_causal_fit()`, `qte()`, or `ate()` when the goal is to visualize treatment effects rather than chain behavior.

## Value

The result of the underlying plot call (invisibly).

## See Also

`plot.mixgpd_fit`, `predict.causalmixgpd_causal_fit`, `ate`, `qte`.

---

# plot.causalmixgpd_causal_predict

*Plot causal prediction outputs*

---

## Description

S3 method for visualizing causal predictions from `predict.causalmixgpd_causal_fit()`. For mean/quantile, plots treated/control and treatment effect versus PS (or index). For `type = "sample"`, plots arm-level posterior predictive samples alongside treatment-effect samples. For density/prob, plots treated/control values versus y.

## Usage

```
## S3 method for class 'causalmixgpd_causal_predict'
plot(x, y = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `causalmixgpd_causal_predict`. |
| `y` | Ignored. |
| `...` | Additional arguments passed to ggplot2 functions. |

## Details

The causal prediction object carries arm-specific predictions together with the implied contrast. For mean predictions, the contrast is $m_1(x) - m_0(x)$. For quantile predictions, the contrast is $Q_{Y^1}(\tau \mid x) - Q_{Y^0}(\tau \mid x)$. The plotting method keeps those arm and contrast views synchronized.

Unlike `plot.causalmixgpd_causal_fit()`, which diagnoses MCMC behavior inside the outcome models, this method visualizes predictive quantities after posterior integration. It is therefore the natural plotting method once the user has already accepted the fitted-model diagnostics.

## Value

A ggplot object or a list of ggplot objects.

---

`plot.causalmixgpd_qte` *Plot QTE-style effect summaries*

---

## Description

`plot.causalmixgpd_qte()` visualizes objects returned by `qte`, `qtt`, and `cqte`. The `type` parameter controls the plot style. When `type` is omitted, `cqte()` objects default to `"effect"` and, when multiple quantile levels are present, `facet_by = "id"`. Whenever quantile index appears on the x-axis, it is shown as an ordered categorical axis with equidistant spacing:

- `"both"` (default): Returns a list with both `trt_control` (treated vs control quantile curves) and `treatment_effect` (QTE curve) plots
- `"effect"`: QTE curve vs quantile levels (`probs`) with pointwise CI error bars
- `"arms"`: Treated and control quantile curves vs `probs`, with pointwise CI error bars

## Usage

```
## S3 method for class 'causalmixgpd_qte'
plot(
  x,
  y = NULL,
  type = c("both", "effect", "arms"),
  facet_by = c("tau", "id"),
  plotly = getOption("CausalMixGPD.plotly", FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `causalmixgpd_qte`. |
| `y` | Ignored. |
| `type` | Character; plot type:<br>• `"both"` (default): returns a list with both arm curves and treatment-effect plots<br>• `"effect"`: QTE curve with pointwise CI error bars<br>• `"arms"`: treated and control quantile curves with pointwise CI error bars |
| `facet_by` | Character; faceting strategy when multiple prediction points exist:<br>• `"tau"` (default): facets by quantile level<br>• `"id"`: facets by prediction point |
| `plotly` | Logical; if `TRUE`, convert the `ggplot2` output to a `plotly` / `htmlwidget` representation via `.wrap_plotly()`. Defaults to `getOption("CausalMixGPD.plotly", FALSE)`. |
| `...` | Additional arguments passed to ggplot2 functions. |

## Details

The effect view emphasizes the quantile contrast $\tau \mapsto Q_{Y^1}(\tau) - Q_{Y^0}(\tau)$, while the arms view shows the treated and control quantile functions that generate that contrast. For conditional CQTE objects, faceting can separate covariate profiles so the same quantile contrast is compared across prediction settings.

These graphics visualize posterior summaries of the effect object itself. They are therefore downstream of model fitting and downstream of the causal prediction step.

## Value

A list of ggplot objects with elements `trt_control` and `treatment_effect` (if `type="both"`), or a single ggplot object (if `type` is `"effect"` or `"arms"`).

## See Also

`qte`, `cqte`, `summary.causalmixgpd_qte`.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
X_new <- X[1:5, , drop = FALSE]
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
qte_result <- cqte(fit, probs = c(0.1, 0.5, 0.9), newdata = X_new)
plot(qte_result)  # CQTE default: effect plot (faceted by id when needed)
plot(qte_result, type = "effect")  # single QTE plot
plot(qte_result, type = "arms")    # single arms plot
```

---

# plot.dpmixgpd_cluster_bundle

*Plot a cluster bundle*

---

## Description

Produce a compact graphical summary of the cluster bundle metadata.

## Usage

```
## S3 method for class 'dpmixgpd_cluster_bundle'
plot(x, plotly = getOption("CausalMixGPD.plotly", FALSE), ...)
```

### Arguments

| | |
|---|---|
| x | A cluster bundle. |
| plotly | Logical; if TRUE, convert the ggplot2 output to a plotly / htmlwidget representation via .wrap_plotly(). Defaults to getOption("CausalMixGPD.plotly", FALSE). |
| ... | Unused. |

### Details

The bundle plot is a metadata display rather than an inferential graphic. It mirrors the structural fields reported by print() and summary() in a single panel so the pre-MCMC clustering specification can be reviewed in a figure-oriented workflow or notebook.

Because the object has not been sampled yet, no representative partition or posterior uncertainty is shown here. Use plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), or plot.dpmixgpd_cluster_psm() after fitting when you need substantive clustering output.

### Value

A ggplot2 object or a plotly/htmlwidget object when plotly = TRUE.

### See Also

summary.dpmixgpd_cluster_bundle(), dpmix.cluster(), dpmgpd.cluster().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

## plot.dpmixgpd_cluster_fit *Plot a cluster fit*

---

### Description

Visualize either the posterior similarity matrix, the posterior number of occupied clusters, the size distribution of the representative clusters, or cluster-specific response summaries.

### Usage

```
## S3 method for class 'dpmixgpd_cluster_fit'
plot(
  x,
  which = c("psm", "k", "sizes", "summary"),
  burnin = NULL,
  thin = NULL,
```

```
  psm_max_n = 2000L,
  top_n = 5L,
  order_by = c("size", "label"),
  plotly = getOption("CausalMixGPD.plotly", FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| x | A cluster fit. |
| which | Plot type:<br>• "psm": posterior similarity matrix heatmap<br>• "k": posterior number of occupied clusters<br>• "sizes": bar chart of representative cluster sizes<br>• "summary": cluster-specific response summaries |
| burnin | Number of initial posterior draws to discard. |
| thin | Keep every thin-th posterior draw. |
| psm_max_n | Maximum training sample size allowed for PSM plotting. |
| top_n | Number of populated representative clusters to display for which = "sizes" or which = "summary". Use NULL to display all populated clusters. |
| order_by | Ordering rule for cluster displays:<br>• "size": decreasing cluster size<br>• "label": ascending cluster label |
| plotly | Logical; if TRUE, convert the ggplot2 output to a plotly / htmlwidget representation via .wrap_plotly(). Defaults to getOption("CausalMixGPD.plotly", FALSE). |
| ... | Unused. |

## Details

This plot method exposes the main posterior diagnostics for clustering. The which = "k" view tracks the number of occupied clusters across retained draws, which = "psm" visualizes pairwise co-clustering probabilities, which = "sizes" displays the size profile of the representative partition, and which = "summary" shows response summaries conditional on the selected representative labels.

The representative partition is obtained from predict.dpmixgpd_cluster_fit() using Dahl's least-squares rule. As a result, the sizes and summary views describe that representative clustering rather than the full posterior distribution over partitions.

## Value

A ggplot2 object or a plotly/htmlwidget object when plotly = TRUE.

## See Also

predict.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_psm(), plot.dpmixgpd_cluster_labels().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

# plot.dpmixgpd_cluster_labels

*Plot cluster labels*

---

## Description

Visualize representative cluster sizes, assignment certainty, or cluster-specific response summaries. For `type = "summary"`, the response view is shown as boxplots ordered by cluster size or label. When `x` comes from `predict(..., newdata = ...)`, only clusters represented in the new sample are displayed.

## Usage

```
## S3 method for class 'dpmixgpd_cluster_labels'
plot(
  x,
  type = c("sizes", "certainty", "summary"),
  top_n = 5L,
  order_by = c("size", "label"),
  plotly = getOption("CausalMixGPD.plotly", FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Cluster labels object. |
| `type` | Plot type:<br>• `"sizes"`: bar chart of representative cluster sizes<br>• `"certainty"`: assignment certainty distribution<br>• `"summary"`: cluster-specific response boxplots |
| `top_n` | Number of populated representative clusters to display for `type = "sizes"` or `type = "summary"`. Use `NULL` to display all populated clusters. |
| `order_by` | Ordering rule for cluster displays:<br>• `"size"`: decreasing cluster size<br>• `"label"`: ascending cluster label |

| | |
|---|---|
| plotly | Logical; if TRUE, convert the ggplot2 output to a plotly / htmlwidget representation via .wrap_plotly(). Defaults to getOption("CausalMixGPD.plotly", FALSE). |
| ... | Unused. |

## Details

This method visualizes the representative partition stored in a dpmixgpd_cluster_labels object. The sizes view emphasizes the empirical distribution of the selected clusters, the certainty view summarizes the assignment scores $\max_k p_{ik}$, and the summary view compares the attached response data across representative clusters.

For new-data prediction, the plots are always interpreted relative to the representative training clusters. That is why only clusters observed in the predicted sample are shown even though the training partition may contain additional occupied groups.

## Value

A ggplot2 object or a plotly/htmlwidget object when plotly = TRUE.

## See Also

summary.dpmixgpd_cluster_labels(), predict.dpmixgpd_cluster_fit().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

# plot.dpmixgpd_cluster_psm

*Plot a cluster posterior similarity matrix*

---

## Description

Heatmap of pairwise posterior co-clustering probabilities.

## Usage

```
## S3 method for class 'dpmixgpd_cluster_psm'
plot(
  x,
  psm_max_n = x$psm_max_n %||% 2000L,
  plotly = getOption("CausalMixGPD.plotly", FALSE),
  ...
)
```

### Arguments

| | |
|---|---|
| x | Cluster PSM object. |
| psm_max_n | Maximum allowed matrix size for plotting. |
| plotly | Logical; if TRUE, convert the ggplot2 output to a plotly / htmlwidget representation via .wrap_plotly(). Defaults to getOption("CausalMixGPD.plotly", FALSE). |
| ... | Unused. |

### Details

The heatmap visualizes the matrix

$$\mathrm{PSM}_{ij} \approx \frac{1}{S}\sum_{s=1}^{S} I(z_i^{(s)} = z_j^{(s)}),$$

so larger values indicate pairs of observations that are stably allocated to the same cluster over the retained posterior draws.

Because the PSM is an $n \times n$ object, plotting and even storing it becomes expensive for large n. The psm_max_n argument is therefore a deliberate guard against accidental quadratic memory use.

### Value

A ggplot2 object or a plotly/htmlwidget object when plotly = TRUE.

### See Also

predict.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_psm(), plot.dpmixgpd_cluster_fit().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

## plot.mixgpd_fit *Plot MCMC diagnostics for a MixGPD fit (ggmcmc backend)*

---

### Description

Uses ggmcmc to produce standard MCMC diagnostic plots. Works with 1+ chains.

### Usage

```
## S3 method for class 'mixgpd_fit'
plot(x, family = "auto", params = NULL, nLags = 50, ...)
```

## Arguments

| | |
|---|---|
| x | A fitted object of class "mixgpd_fit". |
| family | Character vector of plot names (ggmcmc plot types) or a single one. Use "auto" (or "all") to include all plots supported for the available number of chains/parameters. Supported types:<br>• "histogram": posterior histograms<br>• "density": posterior density curves<br>• "traceplot": MCMC trace plots<br>• "running": running mean plots<br>• "compare_partial": partial chain comparisons<br>• "autocorrelation": autocorrelation plots<br>• "crosscorrelation": cross-correlation matrix<br>• "Rhat": Gelman–Rubin R-hat (2+ chains)<br>• "grb": Gelman–Rubin–Brooks (2+ chains)<br>• "effective": effective sample size<br>• "geweke": Geweke diagnostic<br>• "caterpillar": caterpillar/forest plots<br>• "pairs": pairwise scatter plots (2+ params) |
| params | Optional parameter selector:<br>• character vector of parameter patterns (exact names or partial matches)<br>• a single regex string (e.g. "alpha\|threshold\|tail_")<br>• NULL (default): plots all monitored parameters |
| nLags | Number of lags for autocorrelation (ggmcmc). |
| ... | Passed through to the underlying ggmcmc plotting functions when applicable. |

## Details

The supported plots diagnose posterior simulation quality rather than data fit. Depending on the selected family, they show chain traces, marginal posterior densities, autocorrelation, cross-correlation, running means, or Gelman-style convergence summaries for the monitored parameters.

These graphics should be read before interpreting posterior summaries or treatment-effect results. Poor mixing or strong autocorrelation in the MCMC output can invalidate downstream summaries even when the fitted model itself is correctly specified.

## Value

Invisibly returns a named list of ggplot objects.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                            mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
```

```
plot(fit, family = c("traceplot", "density"))
```

---

`plot.mixgpd_fitted` *Plot fitted values diagnostics*

---

## Description

S3 method for visualizing fitted values from `fitted.mixgpd_fit()`. Produces a 2-panel figure: Q-Q plot and residuals vs fitted.

## Usage

```
## S3 method for class 'mixgpd_fitted'
plot(x, y = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `mixgpd_fitted` from `fitted.mixgpd_fit()`. |
| `y` | Ignored; included for S3 compatibility. |
| `...` | Additional arguments (ignored). |

## Details

These diagnostics compare the fitted values implied by the posterior summary on the training design against the observed responses. The first panel checks how closely fitted and observed values align, while the second panel looks for residual structure that would indicate lack of fit or remaining mean trends.

This method is distinct from posterior predictive simulation on new data. It is a training-sample diagnostic built from `fitted.mixgpd_fit()` and the corresponding residuals.

## Value

Invisibly returns a list with the two plots.

`plot.mixgpd_predict` *Plot prediction results*

## Description

Generates type-specific visualizations for prediction objects returned by `predict.mixgpd_fit()`. Each prediction type produces a tailored plot:

- `quantile`: Quantile indices vs estimates with credible intervals
- `sample`: Histogram of samples with density overlay
- `mean`: Histogram density with posterior mean vertical line and CI bounds
- `density`: Density values vs evaluation points
- `survival`: Survival function (decreasing y values)

## Usage

```
## S3 method for class 'mixgpd_predict'
plot(x, y = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | A prediction object returned by `predict.mixgpd_fit()`. |
| `y` | Ignored; included for S3 compatibility. |
| `...` | Additional arguments passed to ggplot2 functions. |

## Details

The plotting method is tied to the predictive functional stored in the input object. Quantile and mean outputs display posterior point summaries and intervals, density and survival outputs show evaluated functions on the supplied grid, and posterior samples are visualized as empirical predictive draws.

In every case the plot reflects the quantity requested from `predict.mixgpd_fit()` after integrating over the retained posterior draws. It is therefore distinct from parameter-level summaries and from chain diagnostics.

## Value

Invisibly returns the ggplot object.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                             mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
```

```
# Quantile prediction with plot
pred_q <- predict(fit, type = "quantile", index = c(0.25, 0.5, 0.75))
plot(pred_q)

# Sample prediction with plot
pred_s <- predict(fit, type = "sample", nsim = 500)
plot(pred_s)

# Mean prediction with plot
pred_m <- predict(fit, type = "mean", nsim_mean = 300)
plot(pred_m)
```

---

`predict.causalmixgpd_causal_fit` *Predict arm-specific and contrast-scale quantities from a causal fit*

---

## Description

`predict.causalmixgpd_causal_fit()` is the causal counterpart to `predict.mixgpd_fit`. It coordinates the treated and control arm predictions so that both sides use the same covariate rows and the same PS adjustment.

## Usage

```
## S3 method for class 'causalmixgpd_causal_fit'
predict(
  object,
  newdata = NULL,
  y = NULL,
  ps = NULL,
  id = NULL,
  type = c("mean", "quantile", "density", "survival", "prob", "sample"),
  p = NULL,
  index = NULL,
  nsim = NULL,
  interval = "credible",
  probs = c(0.025, 0.5, 0.975),
  store_draws = TRUE,
  nsim_mean = 200L,
  ncores = 1L,
  show_progress = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_causal_fit" object returned by run_mcmc_causal(). |
| newdata | Optional new data. If NULL, uses training design (if stored). |
| y | Numeric vector of evaluation points (required for type="density" or "survival"). |
| ps | Optional numeric vector of propensity scores aligned with newdata. When provided, the supplied scores are used instead of recomputing them from the stored PS model (needed only for custom inputs). |
| id | Optional identifier for prediction rows. Provide either a column name in newdata or a vector of length nrow(newdata). The id column is excluded from analysis. |
| type | Prediction type:<br>• "mean": posterior predictive mean treatment effect<br>• "quantile": posterior predictive quantile treatment effect<br>• "density": arm-specific posterior predictive densities<br>• "survival": arm-specific posterior predictive survival functions<br>• "prob": arm-specific posterior predictive probabilities<br>• "sample": paired posterior predictive samples |
| p | Numeric vector of probabilities for quantiles (required for type="quantile"). |
| index | Alias for p; numeric vector of quantile levels. |
| nsim | Number of posterior predictive samples when type = "sample". |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| probs | Quantiles for credible interval bands. |
| store_draws | Logical; whether to store treatment-effect sample draws in the returned object when type = "sample". |
| nsim_mean | Number of posterior predictive samples used by simulation-based mean targets. Ignored for analytical type = "mean"; still used for type = "rmean". |
| ncores | Number of CPU cores to use for parallel prediction (if supported). |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |
| ... | Additional arguments forwarded to per-arm predict.mixgpd_fit calls. |

## Details

For each prediction row $x$, the function evaluates arm-specific posterior predictive quantities based on $F_1(y \mid x)$ and $F_0(y \mid x)$. Mean and quantile outputs are returned on the treatment-effect scale, while density, survival, and probability outputs retain both arm-specific curves. For outcome kernels with a finite analytical mean, the mean path uses analytical per-draw means; restricted means remain simulation-based.

If a PS model is stored in the fit, the same estimated score is supplied to both arms unless the user overrides it with ps. This is the main prediction entry point used internally by ate, qte, cate, and cqte.

## Value

For "mean" and "quantile", a causal prediction object whose $fit component reports treated-minus-control posterior summaries. For "density", "survival", and "prob", the $fit component contains side-by-side treated and control summaries evaluated on the supplied y grid. For "sample", the returned object contains paired treated, control, and treatment-effect posterior predictive samples. Sample outputs also include long-form data frames $fit_df, $trt_fit_df, and $con_fit_df for direct extraction.

## See Also

predict.mixgpd_fit, ate, qte, cate, cqte.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb)
predict(fit, newdata = X[1:10, , drop = FALSE], type = "quantile", index = c(0.25, 0.5, 0.75))
predict(fit, newdata = X[1:10, ], type = "mean", interval = "hpd")  # HPD intervals
predict(fit, newdata = X[1:10, ], type = "mean", interval = NULL)   # No intervals
```

---

# predict.dpmixgpd_cluster_fit

*Predict labels or similarity matrices from a cluster fit*

---

## Description

Convert posterior draws from a dpmixgpd_cluster_fit object into either a representative clustering or a posterior similarity matrix (PSM). This is the main post-processing step for the cluster workflow after dpmix.cluster() or dpmgpd.cluster().

## Usage

```
## S3 method for class 'dpmixgpd_cluster_fit'
predict(
  object,
  newdata = NULL,
  type = c("label", "psm"),
  burnin = NULL,
  thin = NULL,
  return_scores = FALSE,
  psm_max_n = 2000L,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A fitted cluster object. |
| `newdata` | Optional new data containing the response and predictors required by the original formula. New-data prediction is available only for `type = "label"`. |
| `type` | Prediction target:<br>• `"label"`: representative partition via Dahl's least-squares rule<br>• `"psm"`: posterior similarity matrix on the training sample |
| `burnin` | Number of initial posterior draws to discard. |
| `thin` | Keep every `thin`-th posterior draw. |
| `return_scores` | Logical; if `TRUE` and `type = "label"`, include the matrix of Dahl-cluster assignment scores. |
| `psm_max_n` | Maximum training sample size allowed for `type = "psm"`. |
| `...` | Unused. |

## Details

Let $z_i^{(s)}$ denote the latent cluster label for observation $i$ at posterior draw $s$. The posterior similarity matrix is

$$\mathrm{PSM}_{ij} = \Pr(z_i = z_j \mid y) \approx \frac{1}{S}\sum_{s=1}^{S} I(z_i^{(s)} = z_j^{(s)}).$$

The returned label solution is the Dahl representative partition, obtained by choosing the draw whose adjacency matrix is closest to the PSM in squared error.

For `newdata`, the function combines draw-specific component weights and component densities to produce posterior assignment scores relative to the representative training clusters. Returned `newdata` label objects also carry the training labels and response data needed for comparative `plot(..., type = "summary")` displays. A PSM is not defined for `newdata`, so `type = "psm"` is restricted to the training sample.

Computing the PSM is $O(n^2)$ in the training sample size, so `psm_max_n` guards against accidental large matrix allocations.

## Value

A `dpmixgpd_cluster_labels` object when `type = "label"` or a `dpmixgpd_cluster_psm` object when `type = "psm"`.

## See Also

`dpmix.cluster()`, `dpmgpd.cluster()`, `summary.dpmixgpd_cluster_fit()`, `plot.dpmixgpd_cluster_fit()`, `summary.dpmixgpd_cluster_labels()`, `summary.dpmixgpd_cluster_psm()`.

Other cluster workflow: `dpmgpd.cluster()`, `dpmix.cluster()`, `plot.dpmixgpd_cluster_bundle()`, `plot.dpmixgpd_cluster_fit()`, `plot.dpmixgpd_cluster_labels()`, `plot.dpmixgpd_cluster_psm()`,

print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

`predict.mixgpd_fit` *Posterior predictive summaries from a fitted one-arm model*

---

## Description

`predict.mixgpd_fit()` is the central distributional prediction method for fitted one-arm models.

## Usage

```
## S3 method for class 'mixgpd_fit'
predict(
  object,
  newdata = NULL,
  y = NULL,
  ps = NULL,
  id = NULL,
 type = c("density", "survival", "quantile", "sample", "mean", "rmean", "median", "fit"),
  p = NULL,
  index = NULL,
  nsim = NULL,
  level = 0.95,
  interval = "credible",
  probs = c(0.025, 0.5, 0.975),
  store_draws = TRUE,
  nsim_mean = 200L,
  cutoff = NULL,
  ncores = 1L,
  show_progress = TRUE,
  ndraws_pred = NULL,
  chunk_size = NULL,
  parallel = FALSE,
  workers = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"mixgpd_fit"`. |
| `newdata` | Optional new data. If `NULL`, uses training design (if stored). |
| `y` | Numeric vector of evaluation points (required for `type="density"` or `"survival"`). |
| `ps` | Optional numeric vector of propensity scores for conditional prediction. Used when the model was fit with propensity score augmentation. |

`id` Optional identifier for prediction rows. Provide either a column name in `newdata` or a vector of length `nrow(newdata)`. The id column is excluded from analysis.

`type` Prediction type:

- `"density"`: Posterior predictive density f(y | x, data)
- `"survival"`: Posterior predictive survival S(y | x, data) = 1 - F(y | x, data)
- `"quantile"`: Posterior predictive quantiles Q(p | x, data)
- `"sample"`: Posterior predictive samples Y^rep ~ f(y | x, data)
- `"mean"`: Posterior predictive mean E(Y | x, data) (averaged over posterior parameter uncertainty)
- `"rmean"`: Posterior predictive restricted mean $E[\min(Y, cutoff) \mid x, data]$
- `"median"`: Posterior predictive median (quantile at p=0.5)
- `"fit"`: Per-observation posterior predictive draws

Note: `type="mean"` returns the posterior predictive mean, which integrates over parameter uncertainty. This differs from the mean of a single model distribution.

`p` Numeric vector of probabilities for quantiles (required for `type="quantile"`).

`index` Alias for `p`; numeric vector of quantile levels.

`nsim` Number of posterior predictive samples (for `type="sample"`).

`level` Credible level for credible intervals (default 0.95 for 95 percent intervals).

`interval` Character or NULL; type of credible interval:

- `NULL`: no interval
- `"credible"` (default): equal-tailed quantile intervals
- `"hpd"`: highest posterior density intervals

`probs` Quantiles for credible interval bands.

`store_draws` Logical; whether to store all posterior draws (for `type="sample"`).

`nsim_mean` Number of posterior predictive samples used by simulation-based mean targets. Ignored for analytical `type = "mean"`; still used for `type = "rmean"`.

`cutoff` Finite numeric cutoff for `type="rmean"` (restricted mean).

`ncores` Number of CPU cores to use for parallel prediction (if supported).

`show_progress` Logical; if TRUE, print step messages and render progress where supported.

`ndraws_pred` Optional integer subsample of posterior draws for prediction speed. If NULL and `nrow(newdata) > 20000`, defaults to 200.

`chunk_size` Optional row chunk size for large `newdata` prediction. If NULL and `nrow(newdata) > 20000`, defaults to 10000.

`parallel` Logical; if TRUE, enable parallel prediction (alias for setting `ncores > 1`).

`workers` Optional integer worker count (alias for `ncores`).

`...` Unused.

## Details

The method works with posterior predictive functionals rather than raw model parameters. Supported output types include:

- `"density"` for $f(y \mid x)$,
- `"survival"` for $S(y \mid x) = 1 - F(y \mid x)$,
- `"quantile"` for $Q(\tau \mid x)$,
- `"mean"` for $E(Y \mid x)$,
- `"rmean"` for $E\{\min(Y, c) \mid x\}$,
- `"sample"` and `"fit"` for draw-level predictive output.

For spliced models these predictions integrate over both the DPM bulk and the GPD tail using component-specific tail parameters, including link-mode tail coefficients when present. For kernels with a finite analytical mean, `type = "mean"` computes the posterior-draw mean analytically and then summarizes those draw-level means across the posterior. The `type = "rmean"` path remains a separate posterior predictive simulation pipeline.

For kernels with an analytical mean, `type = "mean"` is computed analytically within each posterior draw and then summarized over draws. For GPD-tail fits this analytical path is used when the tail shape parameter satisfies $\xi < 1$. If the mean does not exist analytically for the chosen kernel or if any required GPD tail has $\xi \geq 1$, the ordinary mean is undefined and the function errors with a message directing you to `type = "rmean"` or other summaries that remain well defined.

## Value

A list with elements:

- `fit`: numeric vector/matrix for `type = "sample"`, otherwise a data frame with `estimate`/`lower`/`upper` columns (posterior means over draws) plus any index columns (e.g. `id`, `y`, `index`).
- `fit_df`: a machine-readable data frame view of the prediction output. For non-sample types this aliases `fit`; for `type = "sample"` it is a long-form data frame with draw indices and sampled values.
- `lower`, `upper`: reserved for backward compatibility (typically `NULL`).
- `type`, `grid`: metadata.

## See Also

`summary.mixgpd_fit`, `fitted.mixgpd_fit`, `residuals.mixgpd_fit`, `predict.causalmixgpd_causal_fit`.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                            mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
pr <- predict(fit, type = "quantile", p = c(0.5, 0.9))
pr_surv <- predict(fit, y = sort(y), type = "survival")
```

```
pr_cdf <- list(fit = 1 - pr_surv$fit)
# HPD intervals
pr_hpd <- predict(fit, type = "quantile", p = c(0.5, 0.9), interval = "hpd")
# No intervals
pr_none <- predict(fit, type = "quantile", p = c(0.5, 0.9), interval = NULL)
# Restricted mean (finite under heavy tails)
pr_rmean <- predict(fit, type = "rmean", cutoff = 10, interval = "credible")
```

---

`print.causalmixgpd_ate` *Print an ATE-style effect object*

---

## Description

`print.causalmixgpd_ate()` prints a compact summary for objects produced by `ate`, `att`, `cate`, or `ate_rmean`.

## Usage

```
## S3 method for class 'causalmixgpd_ate'
print(x, digits = 3, max_rows = 6, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"causalmixgpd_ate"` object from `ate()`. |
| `digits` | Number of digits to display. |
| `max_rows` | Maximum number of estimate rows to display. |
| `...` | Unused. |

## Details

These objects summarize posterior treatment contrasts on the mean scale. For the marginal average treatment effect,

$$\Delta = E(Y^1) - E(Y^0).$$

`att()` changes the standardization target to the treated population, `cate()` conditions on supplied covariate profiles, and `ate_rmean()` replaces the ordinary mean by a restricted mean $\int_0^c S_a(t)\,dt$ up to the chosen truncation point.

The print method shows the main effect table and setup metadata, but it is not a full diagnostic report. Use `summary()` for tabular summaries and `plot()` for graphical inspection of the same treatment-effect object.

## Value

The object `x`, invisibly.

## See Also

summary.causalmixgpd_ate, plot.causalmixgpd_ate, ate, cate.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
a <- ate(fit, interval = "credible")
print(a)
```

---

# print.causalmixgpd_bundle

*Print a one-arm workflow bundle*

---

## Description

print.causalmixgpd_bundle() gives a compact structural summary of the pre-run bundle created by build_nimble_bundle.

## Usage

```
## S3 method for class 'causalmixgpd_bundle'
print(x, code = FALSE, max_code_lines = 200L, ...)
```

## Arguments

| | |
|---|---|
| x | A "causalmixgpd_bundle" object. |
| code | Logical; if TRUE, print the generated NIMBLE model code. |
| max_code_lines | Integer; maximum number of code lines to print when code=TRUE. |
| ... | Unused. |

## Details

The bundle is the compiled representation of the predictive model before MCMC. For a bulk-only fit, the underlying target law is

$$f(y \mid x) = \sum_{k=1}^{K} w_k(x) f_k(y \mid x, \theta_k).$$

When a GPD tail is enabled, the same bulk mixture is spliced to a generalized Pareto tail above the threshold recorded in the bundle specification.

`print()` is intentionally brief. It is meant to confirm that the stored backend, kernel, truncation size, covariate structure, and code-generation artifacts match the intended model before you compile and sample with `run_mcmc_bundle_manual`.

## Value

The object `x`, invisibly.

## See Also

`summary.causalmixgpd_bundle`, `mcmc`, `run_mcmc_bundle_manual`.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = FALSE, components = 3)
print(bundle)
print(bundle, code = TRUE, max_code_lines = 30)
```

---

# print.causalmixgpd_causal_bundle

*Print a causal workflow bundle*

---

## Description

`print.causalmixgpd_causal_bundle()` gives a compact structural summary of the pre-run causal bundle created by `build_causal_bundle`.

## Usage

```
## S3 method for class 'causalmixgpd_causal_bundle'
print(x, code = FALSE, max_code_lines = 200L, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"causalmixgpd_causal_bundle"` object. |
| `code` | Logical; if TRUE, print generated NIMBLE code for each block. |
| `max_code_lines` | Integer; maximum number of code lines to print when `code=TRUE`. |
| `...` | Unused. |

## Details

A causal bundle collects three pre-MCMC building blocks: the optional propensity-score model for $e(x) = \Pr(A = 1 \mid X = x)$, the control outcome model for $Y^0$, and the treated outcome model for $Y^1$. The printed output aligns those blocks side by side so the user can verify that the treated and control outcome specifications are coherent before sampling.

No causal estimand is computed at this stage. The bundle only records the structural assumptions that will later support estimands such as $E(Y^1 - Y^0 \mid X = x)$ or $Q_{Y^1}(\tau \mid X = x) - Q_{Y^0}(\tau \mid X = x)$.

## Value

The input object (invisibly).

## See Also

summary.causalmixgpd_causal_bundle, run_mcmc_causal, ate, qte.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal")
print(cb)
```

---

# print.causalmixgpd_causal_fit

*Print a fitted causal model*

---

## Description

print.causalmixgpd_causal_fit() provides a compact overview of the fitted treated/control outcome blocks and the PS component when present.

## Usage

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

## Arguments

| | |
|---|---|
| x | A "causalmixgpd_causal_fit" object. |
| ... | Unused. |

## Details

A fitted causal object combines posterior draws for the treated outcome model, the control outcome model, and optionally the propensity-score model. Those fitted blocks are the ingredients used later to evaluate causal estimands such as $\mu_1(x) - \mu_0(x)$ or $Q_{Y^1}(\tau \mid x) - Q_{Y^0}(\tau \mid x)$.

The print method is deliberately high level. It identifies which models were fitted and whether GPD tails are active, but it does not report posterior summaries or treatment-effect estimates. Use `summary()`, `predict()`, or the dedicated causal estimand helpers for inferential output.

## Value

The input object (invisibly).

## See Also

`summary.causalmixgpd_causal_fit`, `predict.causalmixgpd_causal_fit`, `ate`, `qte`.

---

# print.causalmixgpd_causal_fit_plots

*Print method for paired causal-fit diagnostic plots*

---

## Description

Print method for paired causal-fit diagnostic plots

## Usage

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

## Arguments

| | |
|---|---|
| `x` | Object of class `causalmixgpd_causal_fit_plots`. |
| `...` | Additional arguments passed to the stored plot-print methods. |

## Details

When `plot.causalmixgpd_causal_fit()` is called with `arm = "both"`, the result is a named pair of treated and control diagnostic-plot objects. This print method renders those two stored plot collections one after the other so arm-specific diagnostics remain clearly separated.

It is a formatting helper and does not recompute any diagnostics.

## Value

Invisibly returns the input object.

## print.causalmixgpd_causal_predict_plots

*Print method for causal prediction plots*

### Description

Print method for causal prediction plots

### Usage

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

### Arguments

| | |
|---|---|
| x | Object of class causalmixgpd_causal_predict_plots. |
| ... | Additional arguments (ignored). |

### Details

The causal prediction plotting methods can return either a single plot or a named list of plots. This print method renders those stored plot objects in sequence so both arm-level and contrast-level graphics appear in console or notebook workflows without manual extraction.

It is a display helper only and does not modify the underlying prediction summaries.

### Value

Invisibly returns the input object.

## print.causalmixgpd_ps_bundle

*Print a propensity score bundle*

### Description

Print a propensity score bundle

### Usage

```
## S3 method for class 'causalmixgpd_ps_bundle'
print(x, code = FALSE, max_code_lines = 200L, ...)
```

### Arguments

| | |
|---|---|
| x | A "causalmixgpd_ps_bundle" object. |
| code | Logical; if TRUE, print generated NIMBLE code for the PS model. |
| max_code_lines | Integer; maximum number of code lines to print when code=TRUE. |
| ... | Unused. |

### Details

A PS bundle is the pre-sampling representation of the treatment-assignment model $e(x) = \Pr(A = 1 \mid X = x)$. Depending on the stored model type, the latent linear predictor is later mapped to a probability through a logit link, a probit link, or a naive Bayes factorization.

The printed output is limited to the structural PS choices because posterior draws do not exist yet. Use this method as a quick check that the requested treatment model was encoded correctly before fitting the full causal bundle.

### Value

The input object (invisibly).

---

## print.causalmixgpd_ps_fit

*Print a propensity score fit*

---

### Description

Print a propensity score fit

### Usage

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

### Arguments

| | |
|---|---|
| x | A "causalmixgpd_ps_fit" object. |
| ... | Unused. |

### Details

A propensity-score fit models the treatment assignment probability $e(x) = \Pr(A = 1 \mid X = x)$. The printed header identifies which PS family was fitted, but it intentionally omits coefficient-level summaries.

Use summary() on the same object when you need posterior means, spread, and intervals for the monitored PS parameters. The compact print method is mainly an identity check inside larger causal workflows.

## Value

The input object (invisibly).

---

`print.causalmixgpd_qte`

*Print a QTE-style effect object*

---

## Description

`print.causalmixgpd_qte()` prints a compact summary for objects produced by `qte`, `qtt`, or `cqte`.

## Usage

```
## S3 method for class 'causalmixgpd_qte'
print(x, digits = 3, max_rows = 6, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"causalmixgpd_qte"` object from `qte()`. |
| `digits` | Number of digits to display. |
| `max_rows` | Maximum number of estimate rows to display. |
| `...` | Unused. |

## Details

These objects store posterior summaries of quantile treatment contrasts. In the marginal case,

$$\Delta(\tau) = Q_{Y^1}(\tau) - Q_{Y^0}(\tau).$$

For `qtt()`, the same contrast is standardized to the treated covariate distribution, and for `cqte()` it is evaluated conditionally at the supplied covariate profiles.

The print method is intentionally compact: it reports the prediction setup and the resulting effect table, but it does not attempt to reproduce all posterior draws. Use `summary()` or `plot()` on the same object for more structured reporting.

## Value

The object `x`, invisibly.

## See Also

`summary.causalmixgpd_qte`, `plot.causalmixgpd_qte`, `qte`, `cqte`.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
q <- qte(fit, probs = c(0.25, 0.5, 0.75), interval = "credible")
print(q)
```

---

# print.dpmixgpd_cluster_bundle

*Print a cluster bundle*

---

## Description

Compact display for a dpmixgpd_cluster_bundle before MCMC is run.

## Usage

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

## Arguments

| | |
|---|---|
| x | A cluster bundle. |
| ... | Unused. |

## Details

A cluster bundle is the pre-sampling representation of the latent partition model. It stores the formula-derived design, kernel choice, truncation level, and the rule by which predictors enter the clustering model, but it does not yet contain posterior draws of the latent labels $z_1, \ldots, z_n$.

print() is intentionally brief. It is meant to confirm that the bundle matches the requested clustering structure before you run MCMC with run_cluster_mcmc() or the higher-level wrappers dpmix.cluster() and dpmgpd.cluster().

## Value

x, invisibly.

## See Also

dpmix.cluster(), dpmgpd.cluster(), summary.dpmixgpd_cluster_bundle().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

# print.dpmixgpd_cluster_fit

*Print a cluster fit*

---

## Description

Compact display for a fitted clustering object.

## Usage

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

## Arguments

| | |
|---|---|
| x | A cluster fit. |
| ... | Unused. |

## Details

A fitted cluster object contains posterior draws for the latent labels and associated component parameters. The printed header identifies the model family that generated those draws, including whether the fit used a bulk-only kernel or a spliced bulk-tail specification.

The printed components value is the truncation size used by the sampler. It is not the same thing as the number of occupied clusters in the Dahl representative partition, which is computed later by predict.dpmixgpd_cluster_fit() and summarized by summary.dpmixgpd_cluster_fit().

## Value

x, invisibly.

## See Also

summary.dpmixgpd_cluster_fit(), predict.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_fit().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

print.dpmixgpd_cluster_labels *Print cluster labels*

## Description

Compact display for a representative clustering.

## Usage

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

## Arguments

| | |
|---|---|
| x | Cluster labels object. |
| ... | Unused. |

## Details

A dpmixgpd_cluster_labels object represents one partition, usually the Dahl representative partition for the training data or the induced allocation of newdata to those representative clusters. The printed output therefore describes the selected labels and their sizes, not the full posterior uncertainty over alternative partitions.

When the object comes from predict(..., return_scores = TRUE), richer assignment information is carried alongside the labels and can be inspected with summary.dpmixgpd_cluster_labels() or plot.dpmixgpd_cluster_labels().

## Value

x, invisibly.

## See Also

predict.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_labels().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

print.dpmixgpd_cluster_psm *Print a cluster posterior similarity matrix*

## Description

Compact display for a posterior similarity matrix.

## Usage

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

## Arguments

| | |
|---|---|
| x | Cluster PSM object. |
| ... | Unused. |

## Details

A posterior similarity matrix records pairwise co-clustering probabilities on the training sample. Its $(i, j)$ entry is the posterior probability that observations $i$ and $j$ share the same latent cluster across the retained MCMC draws.

The printed header reports only matrix size and bookkeeping information. Use summary.dpmixgpd_cluster_psm() for numerical summaries and plot.dpmixgpd_cluster_psm() for a visual heatmap.

## Value

x, invisibly.

## See Also

predict.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_psm(), plot.dpmixgpd_cluster_psm().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

print.mixgpd_fit *Print a one-arm fitted model*

## Description

print.mixgpd_fit() gives a compact header for a fitted one-arm model. It is meant as a quick identity check rather than a full posterior summary.

## Usage

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

## Arguments

| | |
|---|---|
| x | A fitted object of class "mixgpd_fit". |
| ... | Unused. |

## Details

The fitted object represents posterior draws from a bulk mixture model, or from its spliced bulk-tail extension when GPD = TRUE. For the bulk part, the predictive law has the mixture form

$$f(y \mid x) = \sum_{k=1}^{K} w_k(x) f_k(y \mid x, \theta_k).$$

When a GPD tail is active, exceedances above the threshold are instead routed through the generalized Pareto tail attached to the same bulk mixture.

The print method reports only the model identity and basic metadata. Use summary() for parameter-level posterior summaries, predict() for predictive functionals, and plot() for chain diagnostics.

## Value

x invisibly.

## See Also

summary.mixgpd_fit, params, predict.mixgpd_fit.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                            mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
print(fit)
```

## print.mixgpd_fitted_plots

*Print method for fitted value plots*

### Description

Print method for fitted value plots

### Usage

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

### Arguments

| | |
|---|---|
| x | Object of class mixgpd_fitted_plots. |
| ... | Additional arguments (ignored). |

### Details

The fitted-value diagnostic object stores two plot panels. This print method renders them in sequence so both the observed-versus-fitted comparison and the residual-versus-fitted comparison are shown together.

It is a display helper only and does not recompute fitted values or residuals.

### Value

Invisibly returns the input object.

## print.mixgpd_fit_plots

*Print method for mixgpd_fit diagnostic plots*

### Description

Print method for mixgpd_fit diagnostic plots

### Usage

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

### Arguments

| | |
|---|---|
| x | Object of class mixgpd_fit_plots. |
| ... | Additional arguments (ignored). |

## Details

Diagnostic plotting for `mixgpd_fit` can return a named collection of ggmcmc graphics. This print method iterates through that collection and prints each stored diagnostic plot with a section label so trace, density, and related views can be read in order.

The method performs no additional posterior computation.

## Value

Invisibly returns the input object.

---

`print.mixgpd_predict_plots`
*Print method for prediction plots*

---

## Description

Print method for prediction plots

## Usage

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

## Arguments

| | |
|---|---|
| `x` | Object of class `mixgpd_predict_plots`. |
| `...` | Additional arguments (ignored). |

## Details

Prediction plotting methods may return a single plot or a richer plot object with an additional wrapper class. This print method temporarily drops that wrapper class so the underlying graphics object uses its native print method.

The stored predictive summaries are not changed.

## Value

Invisibly returns the input object.

`print.mixgpd_summary` *Print a MixGPD summary object*

## Description

Print a MixGPD summary object

## Usage

```
## S3 method for class 'mixgpd_summary'
print(x, digits = 3, max_rows = 60, show_ess = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `"mixgpd_summary"` object. |
| `digits` | Number of digits to print. |
| `max_rows` | Maximum rows to print. |
| `show_ess` | Logical; if `TRUE`, include the `ess` column when present. |
| `...` | Unused. |

## Details

This method formats the output of `summary.mixgpd_fit()`. It prints the model metadata, any stored WAIC value, the effective truncation information induced by `epsilon`, and the parameter-level posterior summary table.

The printed rows correspond to monitored posterior parameters. They are not predictions of densities, quantiles, or means, which should instead be obtained from `predict.mixgpd_fit()`.

## Value

`x` invisibly.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                             mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
summary(fit)
```

## print.summary.causalmixgpd_ate
*Print an ATE summary*

### Description

Print an ATE summary

### Usage

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

### Arguments

| | |
|---|---|
| x | A "summary.causalmixgpd_ate" object. |
| digits | Number of digits to display. |
| ... | Unused. |

### Details

This method formats the object returned by summary.causalmixgpd_ate(). It prints the prediction design, interval settings, optional model metadata, and the resulting treatment-effect table on the mean or restricted-mean scale.

The method is purely a reporting layer. All posterior aggregation has already been completed by the corresponding summary constructor.

### Value

The object x, invisibly.

## print.summary.causalmixgpd_causal_fit
*Print a causal-model summary object*

### Description

Print a causal-model summary object

### Usage

```
## S3 method for class 'summary.causalmixgpd_causal_fit'
print(x, digits = 3, max_rows = 60, ...)
```

### Arguments

| | |
|---|---|
| x | A "summary.causalmixgpd_causal_fit" object. |
| digits | Number of digits to print in summary tables. |
| max_rows | Maximum rows to print from each summary table. |
| ... | Unused. |

### Details

This is a formatter for the object returned by summary.causalmixgpd_causal_fit(). It prints the propensity-score summary first when that block is present, followed by the control and treated outcome summaries on the same scale of posterior diagnostics.

No new computation is performed here. The method arranges the stored summary tables so that the three fitted blocks can be inspected together.

### Value

x invisibly.

---

## print.summary.causalmixgpd_ps_fit

*Print a propensity-score summary object*

---

### Description

Print a propensity-score summary object

### Usage

```
## S3 method for class 'summary.causalmixgpd_ps_fit'
print(x, digits = 3, max_rows = 60, show_ess = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | A "summary.causalmixgpd_ps_fit" object. |
| digits | Number of digits to print in summary tables. |
| max_rows | Maximum rows to print from the summary table. |
| show_ess | Logical; if TRUE, include the ess column when present. |
| ... | Unused. |

### Details

This is a display method for the object returned by summary.causalmixgpd_ps_fit(). It prints the PS model identity, the effective data dimension used by that model, and the posterior summary table for the monitored parameters.

The method does not recompute propensity scores or refit the model. It is a formatting layer over already computed posterior summaries.

### Value

x invisibly.

---

`print.summary.causalmixgpd_qte` *Print a QTE summary*

---

### Description

Print a QTE summary

### Usage

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

### Arguments

| | |
|---|---|
| `x` | A "`summary.causalmixgpd_qte`" object. |
| `digits` | Number of digits to display. |
| `...` | Unused. |

### Details

This formatter displays the summary object returned by `summary.causalmixgpd_qte()`. It reports the quantile grid, interval configuration, model metadata when available, and the tabulated quantile effect summaries.

No additional causal computations are performed here. The method simply turns the stored summary tables into a readable report.

### Value

The object `x`, invisibly.

qte *Quantile treatment effects, marginal over the empirical covariate distribution*

## Description

`qte()` returns the marginal quantile treatment effect implied by the causal fit.

## Usage

```
qte(
  fit,
  probs = c(0.1, 0.5, 0.9),
  newdata = NULL,
  y = NULL,
  interval = "credible",
  level = 0.95,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| `fit` | A `"causalmixgpd_causal_fit"` object from `run_mcmc_causal()`. |
| `probs` | Numeric vector of probabilities in (0, 1) specifying the quantile levels of the outcome distribution to estimate treatment effects at. |
| `newdata` | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| `y` | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| `interval` | Character or NULL; type of credible interval:<br>• `NULL`: no interval<br>• `"credible"` (default): equal-tailed quantile intervals<br>• `"hpd"`: highest posterior density intervals |
| `level` | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| `show_progress` | Logical; if TRUE, print step messages and render progress where supported. |

## Details

The package computes

$$\mathrm{QTE}(\tau) = Q_1^m(\tau) - Q_0^m(\tau),$$

where $Q_a^m(\tau)$ is the arm-$a$ posterior predictive marginal quantile obtained by averaging over the empirical training covariate distribution.

For unconditional causal models (`X = NULL`), this reduces to a direct contrast of the arm-level unconditional predictive distributions.

## Value

An object of class "causalmixgpd_qte" containing the marginal QTE summary, the probability grid, and the arm-specific predictive objects used in the aggregation. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

qtt, cqte, ate, predict.causalmixgpd_causal_fit.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
qte(fit, probs = c(0.5, 0.9))
```

---

# qtt *Quantile treatment effects standardized to treated covariates*

---

## Description

qtt() computes the quantile treatment effect on the treated.

## Usage

```
qtt(
  fit,
  probs = c(0.1, 0.5, 0.9),
  newdata = NULL,
  y = NULL,
  interval = "credible",
  level = 0.95,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| fit | A "causalmixgpd_causal_fit" object from run_mcmc_causal(). |
| probs | Numeric vector of probabilities in (0, 1) specifying the quantile levels of the outcome distribution to estimate treatment effects at. |
| newdata | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |

| | |
|---|---|
| y | Ignored for marginal estimands. If supplied, a warning is issued and training data are used. |
| interval | Character or NULL; type of credible interval:<br>• NULL: no interval<br>• "credible" (default): equal-tailed quantile intervals<br>• "hpd": highest posterior density intervals |
| level | Numeric credible level for intervals (default 0.95 for 95 percent CI). |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |

## Details

The estimand is

$$\mathrm{QTT}(\tau) = Q_1^t(\tau) - Q_0^t(\tau),$$

where marginalization is over the empirical covariate distribution of the treated units only.

## Value

An object of class "causalmixgpd_qte" containing the QTT summary, the probability grid, and the arm-specific predictive objects used in the aggregation. The returned object includes a top-level $fit_df data frame for direct extraction.

## See Also

qte, cqte, att.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
qtt(fit, probs = c(0.5, 0.9))
```

---

# residuals.mixgpd_fit *Residual diagnostics on the training design*

---

## Description

residuals.mixgpd_fit() computes residual diagnostics for conditional fits on the original training data.

## Usage

```
## S3 method for class 'mixgpd_fit'
residuals(
  object,
  type = c("raw", "pit"),
  fitted_type = c("mean", "median"),
  pit = c("plugin", "bayes_mean", "bayes_draw"),
  pit_seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class `"mixgpd_fit"` (must have covariates). |
| `type` | Residual type:<br>• `"raw"`: observed minus fitted values<br>• `"pit"`: probability integral transform residuals (see `pit` argument) |
| `fitted_type` | For `type = "raw"`, use fitted means or medians. |
| `pit` | PIT mode for `type = "pit"`:<br>• `"plugin"`: plug-in PIT using the posterior mean CDF.<br>• `"bayes_mean"`: Bayesian PIT using draw-wise CDFs averaged over draws.<br>• `"bayes_draw"`: Bayesian PIT using a single draw-wise CDF per observation.<br>Bayesian PIT modes drop invalid posterior draws using the same validation rules as prediction and attach diagnostics via `attr(res, "pit_diagnostics")`. |
| `pit_seed` | Optional integer seed for reproducible `bayes_draw` sampling. |
| `...` | Unused. |

## Details

Raw residuals are based on posterior predictive fitted means or medians. PIT residuals instead assess calibration through the posterior predictive CDF. The plug-in PIT uses a posterior mean CDF, while the Bayesian PIT variants work draw by draw.

This method is not available for unconditional models because no training design matrix is stored for observation-specific fitted values.

## Value

Numeric vector of residuals with length equal to the training sample size. PIT variants attach diagnostic metadata as attributes.

## See Also

`fitted.mixgpd_fit`, `predict.mixgpd_fit`, `plot.mixgpd_fitted`.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
X <- data.frame(x1 = stats::rnorm(25), x2 = stats::runif(25))
bundle <- build_nimble_bundle(y = y, X = X, backend = "sb", kernel = "lognormal",
                              GPD = FALSE, components = 3,
                             mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
pit_plugin <- residuals(fit, type = "pit", pit = "plugin")
pit_bayes_mean <- residuals(fit, type = "pit", pit = "bayes_mean", pit_seed = 1L)
pit_bayes_draw <- residuals(fit, type = "pit", pit = "bayes_draw", pit_seed = 1L)
attr(pit_bayes_draw, "pit_diagnostics")
```

---

`run_mcmc_bundle_manual` *Run posterior sampling for a prepared one-arm bundle*

---

## Description

`run_mcmc_bundle_manual()` is the explicit runner for objects created by `build_nimble_bundle`. It compiles the stored NIMBLE code, executes MCMC, and returns a `"mixgpd_fit"` object.

## Usage

```
run_mcmc_bundle_manual(
  bundle,
  show_progress = TRUE,
  quiet = FALSE,
  parallel_chains = FALSE,
  workers = NULL,
  timing = FALSE,
  z_update_every = NULL
)
```

## Arguments

| | |
|---|---|
| `bundle` | A `causalmixgpd_bundle` from `build_nimble_bundle()`. |
| `show_progress` | Logical; if TRUE, print step messages and render progress where supported. |
| `quiet` | Logical; if TRUE, suppress console status messages. Set to FALSE to see progress messages during MCMC setup and execution. |
| `parallel_chains` | Logical; run chains concurrently when `nchains` > 1. |
| `workers` | Optional integer number of workers for parallel execution. |
| `timing` | Logical; if TRUE, include stage timings (`build`, `compile`, `mcmc`) in `fit$timing`. |
| `z_update_every` | Integer >= 1 controlling latent cluster-label update cadence. |

## Details

The resulting fit supports posterior summaries of the model parameters as well as posterior predictive functionals such as $f(y \mid x)$, $S(y \mid x)$, $Q(\tau \mid x)$, and restricted means.

If `parallel_chains = TRUE`, chains are run concurrently when the stored MCMC configuration uses more than one chain. If the bundle was built with latent cluster labels monitored, the `z_update_every` argument controls how frequently those latent indicators are refreshed during sampling.

## Value

A fitted object of class `"mixgpd_fit"` containing posterior draws, model metadata, and cached objects used by downstream S3 methods.

## See Also

`build_nimble_bundle`, `mcmc`, `summary.mixgpd_fit`, `predict.mixgpd_fit`.

## Examples

```
library(nimble)
y <- abs(rnorm(25)) + 0.1
bundle <- build_nimble_bundle(
  y = y,
  backend = "sb",
  kernel = "normal",
  GPD = FALSE,
  components = 3,
  mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
)
fit <- run_mcmc_bundle_manual(bundle, show_progress = FALSE)
fit
```

---

# run_mcmc_causal *Run posterior sampling for a causal bundle*

---

## Description

`run_mcmc_causal()` executes the PS block (when enabled) and the two arm-specific outcome models prepared by `build_causal_bundle`, then returns a single `"causalmixgpd_causal_fit"` object.

## Usage

```
run_mcmc_causal(
  bundle,
  show_progress = TRUE,
  quiet = FALSE,
  parallel_arms = FALSE,
```

```
  workers = NULL,
  timing = FALSE,
  z_update_every = NULL
)
```

## Arguments

| | |
|---|---|
| bundle | A "causalmixgpd_causal_bundle" from build_causal_bundle(). |
| show_progress | Logical; if TRUE, print step messages and render progress where supported. |
| quiet | Logical; if TRUE, suppress step messages and progress display. |
| parallel_arms | Logical; if TRUE, run control and treated outcome arms in parallel. |
| workers | Optional integer workers for parallel arm execution. |
| timing | Logical; if TRUE, return arm and total timings in $timing. |
| z_update_every | Integer >= 1 passed to arm-level outcome MCMC. |

## Details

The fitted object contains the posterior draws needed to evaluate arm-level predictive distributions $F_1(y \mid x)$ and $F_0(y \mid x)$, followed by marginal or conditional causal contrasts. When PS = FALSE in the bundle, the PS block is skipped and outcome prediction uses only the original covariates.

## Value

A list of class "causalmixgpd_causal_fit" containing the fitted treated/control outcome models, optional PS fit, the original bundle, and timing metadata when requested.

## See Also

build_causal_bundle, mcmc, predict.causalmixgpd_causal_fit, ate, qte.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
                          mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb)
```

`sim_bulk_tail` *Simulate positive bulk-tail data*

## Description

Generate synthetic outcomes with a light-to-moderate bulk and a heavier upper tail. The sample is assembled from a lognormal-gamma bulk and a shifted tail sample, then sorted. This generator is intended for examples, help pages, and workflow checks rather than as a formal generative model matching the full package hierarchy exactly.

## Usage

```
sim_bulk_tail(n = 200, tail_prob = 0.12, seed = NULL)
```

## Arguments

| | |
|---|---|
| `n` | Integer sample size. |
| `tail_prob` | Approximate tail probability $\Pr(X > u)$ used to split the sample into bulk and tail draws. |
| `seed` | Optional random seed for reproducibility. |

## Details

The generator approximates a spliced sample

$$X \sim (1 - \pi_u) F_{bulk} + \pi_u F_{tail},$$

where $\pi_u =$ `tail_prob`. The bulk component is itself a simple two-component mixture, while the tail component is a shifted positive distribution that produces larger values.

Use this helper when you need a fast toy sample for `bundle()`, `dpmix()`, or `dpmgpd()`. It should not be interpreted as posterior predictive simulation from a fitted object.

## Value

Numeric vector of length `n` containing positive outcomes sorted in ascending order.

## See Also

`sim_causal_qte()`, `sim_survival_tail()`, `bundle()`, `dpmgpd()`.

Other simulation helpers: `sim_causal_qte()`, `sim_survival_tail()`

`sim_causal_qte` *Simulate causal quantile-treatment-effect data*

## Description

Generate a treatment indicator, covariates, and a continuous outcome with both location and tail heterogeneity. The resulting structure is intended for examples involving `dpmix.causal()`, `dpmgpd.causal()`, `qte()`, and `cqte()`.

## Usage

```
sim_causal_qte(n = 300, seed = NULL)
```

## Arguments

| | |
|---|---|
| `n` | Integer sample size. |
| `seed` | Optional random seed. |

## Details

Treatment assignment is generated from a logistic propensity score

$$\Pr(T = 1 \mid X) = \text{logit}^{-1}(\eta(X)),$$

and the observed outcome combines baseline covariate effects, an average treatment shift, and a covariate-dependent tail amplification for treated units. This produces data where marginal and conditional quantile effects differ across the outcome distribution.

The returned list can be converted directly into the arguments expected by the causal fitting wrappers after minor formatting.

## Value

List with components `y`, `t`, and `X`; `A` is included as a backward-compatible alias for `t`.

## See Also

`sim_bulk_tail()`, `dpmgpd.causal()`, `qte()`, `cqte()`.

Other simulation helpers: `sim_bulk_tail()`, `sim_survival_tail()`

`sim_survival_tail` *Simulate censored survival-style tail data*

## Description

Generate event times, censoring times, an event indicator, and covariates for examples where right tail behavior and positive support matter.

## Usage

```
sim_survival_tail(n = 250, seed = NULL)
```

## Arguments

| | |
|---|---|
| `n` | Integer sample size. |
| `seed` | Optional random seed. |

## Details

Event times are sampled from an exponential model with covariate-dependent mean, then censored by an independent uniform censoring time. The observed time is

$$\tilde{T} = \min(T, C), \qquad \Delta = I(T \leq C).$$

This helper is mainly for experimentation and stress-testing positive-support kernels; it does not implement a dedicated survival model from the package API.

## Value

Data frame containing observed time `time`, event indicator `status`, and covariates.

## See Also

`sim_bulk_tail()`, `build_nimble_bundle()`, `dpmgpd()`.

Other simulation helpers: `sim_bulk_tail()`, `sim_causal_qte()`

summary.causalmixgpd_ate *Summarize an ATE-style effect object*

## Description

summary.causalmixgpd_ate() converts ATE, ATT, CATE, or restricted-mean output into a tabular summary suitable for reporting.

## Usage

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

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_ate" object from ate(). |
| ... | Unused. |

## Details

The summary reorganizes posterior treatment-effect output on the mean scale. Depending on the source object, the target estimand is a marginal ATE, treated-standardized ATT, conditional ATE, or a restricted-mean contrast based on

$$\int_0^c \{S_1(t) - S_0(t)\}\, dt.$$

The returned object stores overall metadata, effect tables, and interval summaries in a reporting-friendly format. It does not refit the model or recompute arm-specific predictions.

## Value

An object of class "summary.causalmixgpd_ate" with overall, ate_stats, effect_table, ci_summary, meta, and the original object.

## See Also

print.causalmixgpd_ate, plot.causalmixgpd_ate, ate, cate.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
```

```
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
a <- ate(fit, interval = "credible")
summary(a)
```

---

`summary.causalmixgpd_bundle` *Summarize a one-arm workflow bundle*

---

## Description

`summary.causalmixgpd_bundle()` prints the structural contents of a bundle before MCMC is run.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | A "causalmixgpd_bundle" object. |
| `...` | Unused. |

## Details

The summary is meant for workflow validation rather than inference. It shows:

- the model metadata (backend, kernel, components, covariates, GPD flag),
- the prior/parameter table derived from `spec$plan`,
- the nodes that will be monitored during MCMC.

This is the recommended checkpoint after `build_nimble_bundle` and before `run_mcmc_bundle_manual`.

## Value

An invisible list with elements `meta`, `priors`, and `monitors`.

## See Also

`build_nimble_bundle`, `print.causalmixgpd_bundle`, `run_mcmc_bundle_manual`.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = FALSE, components = 3)
summary(bundle)
```

summary.causalmixgpd_causal_bundle *Summarize a causal workflow bundle*

## Description

summary.causalmixgpd_causal_bundle() is the bundle-level validation checkpoint for the causal workflow.

## Usage

```
## S3 method for class 'causalmixgpd_causal_bundle'
summary(object, code = FALSE, max_code_lines = 200L, ...)
```

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_causal_bundle" object. |
| code | Logical; if TRUE, print generated NIMBLE code for each block. |
| max_code_lines | Integer; maximum number of code lines to print when code=TRUE. |
| ... | Unused. |

## Details

This summary is meant to be read before posterior sampling. It reports the stored propensity-score specification, the treated and control outcome model definitions, and the sample split across treatment arms. In other words, it verifies the model ingredients for the causal decomposition $(e(x), f_0(y \mid x), f_1(y \mid x))$.

Since no MCMC has been run yet, the summary contains only structural information. Posterior treatment-effect summaries become available after run_mcmc_causal through functions such as ate and qte.

## Value

The input object (invisibly).

## See Also

print.causalmixgpd_causal_bundle, run_mcmc_causal.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal")
summary(cb)
```

summary.causalmixgpd_causal_fit
*Summarize a fitted causal model*

## Description

summary.causalmixgpd_causal_fit() returns posterior summaries for the fitted PS block (when present) and both arm-specific outcome models.

## Usage

```
## S3 method for class 'causalmixgpd_causal_fit'
summary(object, pars = NULL, ps_pars = NULL, probs = c(0.025, 0.5, 0.975), ...)
```

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_causal_fit" object. |
| pars | Optional character vector of outcome-model parameters to summarize in both treatment arms. Passed to summary.mixgpd_fit. |
| ps_pars | Optional character vector of PS-model parameters to summarize. If NULL, all monitored PS parameters are summarized. |
| probs | Numeric vector of posterior quantiles to report. |
| ... | Unused. |

## Details

This summary stays at the model-parameter level. It aggregates posterior summaries for the nuisance model $e(x)$ and for the arm-specific outcome models $f_0(y \mid x)$ and $f_1(y \mid x)$, but it does not yet collapse those pieces into treatment-effect functionals.

That separation is intentional. Parameters and treatment effects answer different questions: summary.causalmixgpd_causal summarizes posterior draws of the fitted model, whereas ate(), att(), cate(), qte(), qtt(), and cqte() transform those draws into causal contrasts.

## Value

An object of class "summary.causalmixgpd_causal_fit" with elements ps, outcome, and probs.

## See Also

print.causalmixgpd_causal_fit, predict.causalmixgpd_causal_fit, ate, qte, cate, cqte.

summary.causalmixgpd_ps_fit *Summarize a propensity score fit*

## Description

summary.causalmixgpd_ps_fit() returns posterior summaries for the monitored PS-model parameters.

## Usage

```
## S3 method for class 'causalmixgpd_ps_fit'
summary(object, pars = NULL, probs = c(0.025, 0.5, 0.975), ...)
```

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_ps_fit" object. |
| pars | Optional character vector of PS parameters to summarize. If NULL, summarize all monitored parameters. |
| probs | Numeric vector of posterior quantiles to report. |
| ... | Unused. |

## Details

The summary is parameter based. For logit and probit models, it summarizes the posterior draws of the coefficients that determine the latent linear predictor, which is then mapped to $e(x)$ by the chosen link function. For the naive Bayes option, it summarizes the class-conditional parameters used to factorize the treatment-assignment model.

This function does not compute fitted propensity scores for specific covariate rows. It summarizes the posterior distribution of the PS model itself, which is the nuisance model later used by causal prediction and treatment-effect standardization.

## Value

An object of class "summary.causalmixgpd_ps_fit" with elements model and table.

summary.causalmixgpd_qte *Summarize a QTE-style effect object*

## Description

summary.causalmixgpd_qte() converts QTE, QTT, or CQTE output into a tabular summary suitable for reporting.

## Usage

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

## Arguments

| | |
|---|---|
| object | A "causalmixgpd_qte" object from qte(). |
| ... | Unused. |

## Details

The summary reorganizes the posterior effect object into reporting tables. The target estimand remains a quantile contrast,

$$\Delta(\tau) = Q_{Y^1}(\tau) - Q_{Y^0}(\tau),$$

with the appropriate marginal, treated-standardized, or conditional interpretation depending on whether the source object came from qte(), qtt(), or cqte().

Besides the effect table itself, the summary records the quantile grid, the interval settings, and per-quantile distributional summaries when posterior draws are available. This makes the object convenient for reporting and downstream printing without recomputing the estimand.

## Value

An object of class "summary.causalmixgpd_qte" with overall, quantile_summary, effect_table, ci_summary, meta, and the original object.

## See Also

print.causalmixgpd_qte, plot.causalmixgpd_qte, qte, cqte.

## Examples

```
N <- 25
X <- data.frame(x1 = stats::rnorm(N))
A <- stats::rbinom(N, 1, 0.5)
y <- abs(stats::rnorm(N)) + 0.1
mcmc_small <- list(niter = 100, nburnin = 50, thin = 1, nchains = 1, seed = 1)
cb <- build_causal_bundle(y = y, X = X, A = A, backend = "sb", kernel = "normal",
```

```
                          components = 3, mcmc_outcome = mcmc_small, mcmc_ps = mcmc_small)
fit <- run_mcmc_causal(cb, show_progress = FALSE)
q <- qte(fit, probs = c(0.25, 0.5, 0.75), interval = "credible")
summary(q)
```

---

`summary.dpmixgpd_cluster_bundle` *Summarize a cluster bundle*

---

## Description

Report the modeling choices encoded in a cluster bundle before fitting.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | A cluster bundle. |
| `...` | Unused. |

## Details

This summary is a pre-flight check for the clustering workflow. It reports the latent-partition design, the chosen kernel family, whether a GPD tail will be spliced above the threshold, the effective sample and predictor dimensions, and the monitor set that will be carried into MCMC.

Because no posterior simulation has occurred yet, the summary describes only the assumed model structure. Quantities such as representative labels, pairwise co-clustering probabilities, and cluster-specific summaries become available only after the fitted object has been created and post-processed.

## Value

Summary list containing kernel choice, GPD flag, dimensions, component count, and monitor configuration.

## See Also

`print.dpmixgpd_cluster_bundle()`, `plot.dpmixgpd_cluster_bundle()`, `predict.dpmixgpd_cluster_fit()`.

Other cluster workflow: `dpmgpd.cluster()`, `dpmix.cluster()`, `plot.dpmixgpd_cluster_bundle()`, `plot.dpmixgpd_cluster_fit()`, `plot.dpmixgpd_cluster_labels()`, `plot.dpmixgpd_cluster_psm()`, `predict.dpmixgpd_cluster_fit()`, `print.dpmixgpd_cluster_bundle()`, `print.dpmixgpd_cluster_fit()`, `print.dpmixgpd_cluster_labels()`, `print.dpmixgpd_cluster_psm()`, `summary.dpmixgpd_cluster_fit()`, `summary.dpmixgpd_cluster_labels()`, `summary.dpmixgpd_cluster_psm()`

summary.dpmixgpd_cluster_fit *Summarize a cluster fit*

## Description

Summarize the posterior clustering induced by the Dahl representative partition.

## Usage

```
## S3 method for class 'dpmixgpd_cluster_fit'
summary(
  object,
  burnin = NULL,
  thin = NULL,
  top_n = 5L,
  order_by = c("size", "label"),
  vars = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A cluster fit. |
| `burnin` | Number of initial posterior draws to discard. |
| `thin` | Keep every `thin`-th posterior draw. |
| `top_n` | Number of populated clusters to profile when descriptive summaries are available. |
| `order_by` | Ordering rule for descriptive cluster profiles:<br>• `"size"`: decreasing cluster size<br>• `"label"`: ascending cluster label |
| `vars` | Optional character vector of numeric columns to summarize within each cluster. |
| `...` | Unused. |

## Details

This summary is based on `predict.dpmixgpd_cluster_fit()` with `type = "label"`. The reported cluster count $K^*$ is the number of unique labels in the representative partition rather than the number of components available in the truncated sampler.

## Value

Summary list with the number of retained clusters, cluster sizes, optional cluster-level descriptive summaries, and the burn-in/thinning settings used to construct the summary.

## See Also

predict.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_psm()

---

# summary.dpmixgpd_cluster_labels

*Summarize cluster labels*

---

## Description

Summarize a representative clustering for training data or new observations.

## Usage

```
## S3 method for class 'dpmixgpd_cluster_labels'
summary(object, top_n = 5L, order_by = c("size", "label"), vars = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | Cluster labels object. |
| top_n | Number of populated clusters to profile when attached data are available. |
| order_by | Ordering rule for descriptive cluster profiles:<br>• "size": decreasing cluster size<br>• "label": ascending cluster label |
| vars | Optional character vector of numeric columns to summarize within each cluster. |
| ... | Unused. |

## Details

If score or probability matrices are attached, certainty is summarized by the rowwise maxima $\max_k p_{ik}$, which quantify how strongly each observation is assigned to its selected cluster. When the labels object also carries attached training or prediction data, the summary includes descriptive mean/sd profiles for the first populated clusters.

## Value

Summary list containing cluster sizes, optional cluster-level descriptive summaries, and, when available, assignment-certainty summaries.

## See Also

predict.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), summary.dpmixgpd_cluster_fit().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_psm()

---

# summary.dpmixgpd_cluster_psm

*Summarize a cluster posterior similarity matrix*

---

## Description

Summarize pairwise posterior co-clustering probabilities.

## Usage

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

## Arguments

| | |
|---|---|
| object | Cluster PSM object. |
| ... | Unused. |

## Details

The diagonal of a posterior similarity matrix is always close to one, while off-diagonal values near one indicate highly stable co-clustering across retained posterior draws.

## Value

Summary list with matrix size and basic summaries of the similarity entries.

## See Also

predict.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_fit().

Other cluster workflow: dpmgpd.cluster(), dpmix.cluster(), plot.dpmixgpd_cluster_bundle(), plot.dpmixgpd_cluster_fit(), plot.dpmixgpd_cluster_labels(), plot.dpmixgpd_cluster_psm(), predict.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_bundle(), print.dpmixgpd_cluster_fit(), print.dpmixgpd_cluster_labels(), print.dpmixgpd_cluster_psm(), summary.dpmixgpd_cluster_bundle(), summary.dpmixgpd_cluster_fit(), summary.dpmixgpd_cluster_labels()

summary.mixgpd_fit *Summarize posterior draws from a one-arm fitted model*

## Description

summary.mixgpd_fit() computes posterior summaries for monitored model parameters.

## Usage

```
## S3 method for class 'mixgpd_fit'
summary(object, pars = NULL, probs = c(0.025, 0.5, 0.975), ...)
```

## Arguments

| | |
|---|---|
| object | A fitted object of class "mixgpd_fit". |
| pars | Optional character vector of parameters to summarize. If NULL, summarize all (excluding v's). |
| probs | Numeric vector of quantiles to report. |
| ... | Unused. |

## Details

The returned table is a parameter-level summary of the posterior draws, not a predictive summary. Use predict.mixgpd_fit for posterior predictive quantities such as densities, survival probabilities, quantiles, and means.

The summary respects the stored truncation metadata and reports WAIC if it was requested during MCMC.

## Value

An object of class "mixgpd_summary".

## See Also

print.mixgpd_fit, params, predict.mixgpd_fit, ess_summary.

## Examples

```
y <- abs(stats::rnorm(25)) + 0.1
bundle <- build_nimble_bundle(y = y, backend = "sb", kernel = "normal",
                              GPD = TRUE, components = 3,
                            mcmc = list(niter = 100, nburnin = 50, thin = 1, nchains = 1))
fit <- run_mcmc_bundle_manual(bundle)
summary(fit, pars = c("alpha", "threshold"))
```

# Index