Package: mcp (via r-universe)

October 26, 2026

Title Regression with Multiple Change Points

Version 0.4.0

Date 2026-10-25

URL https://lindeloev.github.io/mcp/

BugReports https://github.com/lindeloev/mcp/issues

Description Flexible and informed regression with Multiple Change Points. 'mcp' can infer change points in means, variances, autocorrelation structure, and any combination of these, as well as the parameters of the segments in between. All parameters are estimated with uncertainty and posterior predictive intervals are supported - also near the change points. 'mcp' supports hypothesis testing via Savage-Dickey density ratio, posterior contrasts, and PSIS-LOO model comparison. The 'mcp' framework is introduced in Lindeløv (2020) <doi:10.31219/osf.io/fzqxv>.

SystemRequirements JAGS (>= 4.0.0)

License GPL (>= 2)

Encoding UTF-8

Language en-US

LazyData true

Depends R (>= 4.1.0)

Config/testthat/parallel true

Config/testthat/edition 3

Imports future (>= 1.20.0), future.apply (>= 1.4), lifecycle (>= 1.0.0), rjags (>= 4.9), coda (>= 0.19.3), posterior (>= 1.0.0), loo (>= 2.1.0), bayesplot (>= 1.7.0), dplyr (>= 1.1.1), tidyr (>= 1.0.0), tibble (>= 2.1.3), stringr (>= 1.4.0), ggplot2 (>= 3.2.1), patchwork (>= 1.0.0), checkmate (>= 2.3.0), stats, rlang (>= 1.0.0), nlme

Suggests tidybayes (>= 3.0.0), rstantools (>= 2.0.0), hexbin, MASS, segmented, lme4, testthat (>= 3.1.0), vdiffr (>= 1.0.0), knitr, rmarkdown, withr

Enhances brms

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jonas Kristoffer Lindeløv [aut, cre] (ORCID:
<https://orcid.org/0000-0003-4565-0595>)

Maintainer Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Config/pak/sysreqs jags

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

Date/Publication 2026-10-26 15:20:11 UTC

RemoteUrl https://github.com/cran/mcp

RemoteRef HEAD

RemoteSha 61df46afa764c1d7294a8764a6e65ab949971e9e

Contents

mcp-package . . . 3
as_draws.mcpfit . . . 4
bernoulli . . . 6
demo_fit . . . 6
draws-index-mcp . . . 7
execute-mcp-model . . . 8
hypothesis . . . 14
interpolate_newdata . . . 16
is.mcpfit . . . 18
loo.mcpfit . . . 18
mcp . . . 20
mcp-formula . . . 25
mcp-priors . . . 28
mcp_columns . . . 30
mcp_example . . . 31
mcp_pars . . . 32
mcpfit-class . . . 33
model-accessors-mcpfit . . . 34
negbinomial . . . 35
plot.mcpfit . . . 35
plot_pars . . . 39
posterior-uncertainty-mcpfit . . . 41
posterior_epred.mcpfit . . . 42
pp_check.mcpfit . . . 45
print.mcplist . . . 48
print.mcptext . . . 48
prior_summary . . . 49
sd_to_prec . . . 50
summary.mcpfit . . . 51

Index **54**

`mcp-package` *mcp: Regression with Multiple Change Points*

Description

`mcp` is regression with multiple change points, inferring full Bayesian uncertainty for all parameters, including the change points themselves. At its simplest, it feels like `lm()` with one formula per segment. At its most flexible, it aims to be `brms` **for change points**:

- *GLMM and distributional regression:* Gaussian, binomial, Bernoulli, Poisson, and Negative Binomial families. Group-level (random) effects, AR/MA, and regression on distributional parameters (e.g., `sigma`, `shape`).
- *Express expert knowledge:* Specify the number of change points, the form of each segment, and restrict coefficients via priors. Compare hypotheses using `loo()` and `hypothesis()`.
- *Native feel:* `summary()`, `plot_pars()`, `pp_check()`, and `as_draws_df()` follow the conventions of `brms`, `rstanarm`, and `posterior`, adapted to change point regression. Inspect the sampled JAGS model in `fit$jags_code`.

You will find many aliases for change point regression in the literature, including piecewise linear, discontinuity, segmented, and broken stick regression.

See `mcp()` to fit models, mcp-formula for the model and formula syntax, and mcp-priors for priors.

Extended formulas and model features - and links to worked examples

- *Distributional regression:* Model residual variance and other distributional parameters across segments, e.g., `~ sigma(1 + x)` or `~ shape(1)`. Read more.
- *Time-series residuals:* Model serial dependence using `ar(p)` and `ma(q)` terms with a generalized link-scale recurrence. Read more.
- *Group-level effects:* Add hierarchical (random) intercepts, slopes, and change points, e.g., `~ 1 + (1|id)` or `~ cp((1|id)) + 0 + x`. Read more.
- *Families:* Binomial, Bernoulli, Poisson, and negative-binomial responses with several link functions. Read more.
- *Model comparison and hypothesis testing:* Compare models using PSIS-LOO (`loo()`) or WAIC (`waic()`), and test hypotheses (`hypothesis()`) using Savage-Dickey density ratios for point-null tests or posterior-to-prior odds for directional tests. Read more.
- *Fits, predictions, and forecasts:* `fitted()`, `predict()`, and `residuals()`, including forecasts beyond the data. Read more.
- *Missing responses:* Missing values in the response are imputed jointly with the parameters. Read more.

Options

- `mcp_quiet`: Logical. The default of `mcp(..., quiet)`. `options(mcp_quiet = TRUE)` suppresses routine JAGS output and mcp sampling-status messages for the session.
- `mcp_reparam`: Logical. `TRUE` (default) lets `mcp()` sample some default priors on scales that mix faster. This changes only `fit$jags_code` and sampling efficiency, not the model, the priors, or the posterior. `options(mcp_reparam = FALSE)` writes the plain model, which is mostly useful for seeing how `mcp` works. Ignored when `jags_code` is supplied. Read more in From formulas to sampler code.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

References

- Lindeløv, J. K. (2020). mcp: An R Package for Regression With Multiple Change Points. *OSF Preprints*. doi:10.31219/osf.io/fzqxv Introduces the `mcp` package, formula syntax, default priors, and workflow for regression with multiple change points across generalized linear and time-series models. Newer-than-2020 versions of the paper may be available at that link. Please cite the newest version.
- Carlin, B. P., Gelfand, A. E., & Smith, A. F. (1992). Hierarchical Bayesian Analysis of Changepoint Problems. *Applied Statistics*, 41(2), 389–405. doi:10.2307/2347570 Introduced Gibbs sampling for hierarchical Bayesian change-point models, the computational approach that BUGS and JAGS build on.

See Also

`mcp`

`as_draws.mcpfit` *Extract MCMC Draws from* `mcpfit` *Objects*

Description

Extract posterior or prior draws using **posterior**, **coda**, or the optional **tidybayes** package's S3 generics. Base R's `as.matrix()`, `as.array()`, and `as.data.frame()` return plain draws, like in **rstanarm** and **brms**, so **bayesplot** functions work directly on the fit.

Usage

```
## S3 method for class 'mcpfit'
as_draws(x, prior = FALSE, ...)

as_draws(x, ...)

as_draws_df(x, ...)
```

```
as_draws_array(x, ...)

as_draws_matrix(x, ...)

as_draws_rvars(x, ...)

## S3 method for class 'mcpfit'
as.matrix(x, prior = FALSE, ...)

## S3 method for class 'mcpfit'
as.array(x, prior = FALSE, ...)

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

Arguments

x	An mcpfit object.
prior	Logical. Extract prior draws (TRUE) instead of posterior draws (FALSE)? Errors if the requested draws are unavailable.
...	Passed to **posterior** or **tidybayes** format conversion functions.
row.names, optional	Unused. For compatibility with as.data.frame().

Value

A **posterior** draws object or a **coda** mcmc.list object. as.matrix() returns a draws-by-parameter matrix, as.array() an iteration-by-chain-by-parameter array, and as.data.frame() a draws-by-parameter data frame. The data frame has no chain information, so use as_draws_df() for chain-aware diagnostics such as posterior::rhat().

Examples

```
# Default posterior draws, with one row per iteration and chain
draws = as_draws(demo_fit)  # Return a posterior::draws object
head(as_draws_df(demo_fit))  # Convert draws to a data frame

# Other posterior formats are useful in different downstream packages
as_draws_matrix(demo_fit)[1:3, 1:3]  # Matrix of draws by parameter
as_draws_array(demo_fit)[1:2, , 1:2]  # Iteration-by-chain-by-parameter array
as_draws_rvars(demo_fit)[c("cp_1", "cp_2")]  # Random-variable representation

# Base R formats, e.g., for bayesplot
as.matrix(demo_fit)[1:3, 1:3]  # Plain matrix of draws by parameter
bayesplot::mcmc_trace(demo_fit, pars = c("cp_1", "cp_2"))

# mcp also supports the coda and tidybayes conventions
head(coda::as.mcmc(demo_fit)[[1]])  # First chain as a coda mcmc object
if (requireNamespace("tidybayes", quietly = TRUE))
```

```
head(tidybayes::tidy_draws(demo_fit))
```

`bernoulli` *Bernoulli Family for mcp*

Description

Binary responses (0 or 1). See Binomial and Bernoulli for worked examples.

Usage

```
bernoulli(link = "logit")
```

Arguments

`link`	Link function.

Examples

```
# Fit a binary-response model with a probit link
data = data.frame(time = 1:6, y = c(0, 0, 0, 1, 1, 1))
fit = mcp(list(y ~ 1), data, family = bernoulli(link = "probit"), par_x = "time", sample = FALSE)
mcp_pars(fit)$parameter  # Show the parameters of the fitted Bernoulli model
```

`demo_fit` *Pre-fitted example mcp model*

Description

`demo_fit` is an `mcpfit` object generated from the `mcp_example("demo")` model and data. It contains 1000 posterior and 1000 prior draws (two chains of 500 each).

Usage

```
demo_fit
```

Format

An `mcpfit` object.

Details

It is used to quickly demonstrate `mcp` functionality in various documentation.

See Also

`mcp_example` for the model and data used to create it, and `mcpfit-class` for the structure and methods of fitted models.

draws-index-mcp *Index* mcpfit *objects*

Description

Index variables, iterations, chains, and draws.

Usage

```
## S3 method for class 'mcpfit'
variables(x, ...)

## S3 method for class 'mcpfit'
niterations(x, ...)

## S3 method for class 'mcpfit'
nchains(x, ...)

## S3 method for class 'mcpfit'
ndraws(x, ...)

variables(x, ...)

ndraws(x)

nchains(x)

niterations(x)
```

Arguments

x	An mcpfit object or a posterior draws object.
...	Must be empty. Reserved for future use.

Details

These methods require posterior draws. To index prior draws, use e.g. ndraws(as_draws(fit, prior = TRUE)).

Value

variables() returns the names of the draws' variables, e.g., "cp_1" and "cp_1_id[John]". The other methods return an integer count of iterations, chains, or draws.

Functions

- `variables(mcpfit)`: Variable names of an `mcpfit` object, as in `as_draws(fit)`.
- `niterations(mcpfit)`: Number of iterations per chain of an `mcpfit` object.
- `nchains(mcpfit)`: Number of chains of an `mcpfit` object.

Examples

```
variables(demo_fit)
niterations(demo_fit)
nchains(as_draws(demo_fit, prior = TRUE))
```

---

execute-mcp-model *Fitted and predicted values of* mcp *models fits*

Description

Evaluate the model on data, either summarised (per data-row) or per draw. You can use draws from the prior (`prior = TRUE`), select a distributional parameter with `dpar`, and choose the response or linear-predictor scale with `scale` where applicable. See Fits and predictions for worked examples, including forecasts with future change points.

Usage

```
## S3 method for class 'mcpfit'
predict(
  object,
  newdata = NULL,
  summary = TRUE,
  width = 0.95,
  rate = FALSE,
  prior = FALSE,
  group_effects = TRUE,
  arma = TRUE,
  ndraws = NULL,
  draws_format = "tidy",
  nsamples = lifecycle::deprecated(),
  samples_format = lifecycle::deprecated(),
  varying = lifecycle::deprecated(),
  probs = lifecycle::deprecated(),
  ...
)

## S3 method for class 'mcpfit'
fitted(
  object,
  newdata = NULL,
```

```
  summary = TRUE,
  width = 0.95,
  rate = FALSE,
  prior = FALSE,
  dpar = "epred",
  group_effects = TRUE,
  arma = TRUE,
  ndraws = NULL,
  draws_format = "tidy",
  scale = "response",
  nsamples = lifecycle::deprecated(),
  samples_format = lifecycle::deprecated(),
  varying = lifecycle::deprecated(),
  probs = lifecycle::deprecated(),
  ...
)

log_lik(object, ...)

## S3 method for class 'mcpfit'
log_lik(
  object,
  newdata = NULL,
  summary = FALSE,
  width = 0.95,
  prior = FALSE,
  group_effects = TRUE,
  arma = TRUE,
  ndraws = NULL,
  draws_format = "matrix",
  nsamples = lifecycle::deprecated(),
  samples_format = lifecycle::deprecated(),
  varying = lifecycle::deprecated(),
  probs = lifecycle::deprecated(),
  ...
)

## S3 method for class 'mcpfit'
residuals(
  object,
  newdata = NULL,
  summary = TRUE,
  width = 0.95,
  prior = FALSE,
  group_effects = TRUE,
  arma = TRUE,
  ndraws = NULL,
  draws_format = "tidy",
```

```
  nsamples = lifecycle::deprecated(),
  varying = lifecycle::deprecated(),
  probs = lifecycle::deprecated(),
  ...
)
```

Arguments

object	An mcpfit object.
newdata	A tibble or a data.frame containing predictors in the model. • If NULL (default), the original data is used. • Grouping factors must only contain levels from the fitted data. Use group_effects = FALSE to predict without group-level effects, e.g., for new groups. • For models with ar() or ma(): fitted(), residuals(), log_lik(), and predict() condition on the response history by default, so newdata must include the response. For fitted(), predict(), and residuals(), missing response histories are supported only in the original fitted data, using retained posterior imputations as histories. Predictions are fresh response draws, including at missing rows. With arma = "replicate", predict() and posterior_predict() generate fresh response series recursively, so their newdata need only contain predictors and required response auxiliaries. log_lik() is unavailable when a missing response enters a later observed history. • For models with y | weights(): Require the weights column except for fitted() and predict().
summary	Logical. Summarise the draws for each row in newdata (TRUE) or return the draws (FALSE)?
width	Width of the central interval (strictly between 0 and 1), returned in the .lower and .upper columns. FALSE omits the interval. Only in effect when summary == TRUE. To get custom quantiles, summarise the draws from summary = FALSE, e.g., using dplyr::reframe() (see examples).
rate	Logical scalar. For binomial models, return counts (rate = FALSE, the default for fitted() and predict()) or the observed or expected success proportion (rate = TRUE). Predictions and count-scale fitted values require a trials column in newdata. Distributional parameters such as dpar = "mu" evaluate the parameter itself (e.g., success probability) and are unaffected by rate. Other families do not support rate = TRUE. For rates per unit of exposure in models with offset(log(exposure)), set exposure = 1 in newdata.
prior	Logical. Evaluate prior draws (TRUE) instead of posterior draws (FALSE, default). The selected draws must be available; prior-only fits require prior = TRUE.
group_effects	Group-level effects to include. One of: • TRUE All group-level deviations. • FALSE or NA No group-level deviations, i.e., population-level predictions. Like re_formula = NA in {brms}. • "cp" or "predictor": All group-level deviations belonging to that part of the model.

- Character vector of group-level parameter names, e.g., `c("cp_1_id", "x_2_id")`. **[Experimental]** `*` is a wildcard, e.g., `"state*_1_id"` selects all levels of a factor and `"*_id"` selects all effects by `id`. See names using `ranef(fit)`. Can be combined, e.g., `c("cp", "sigma_*")`.

`arma` How to include AR and MA effects.

- `TRUE`: Condition on the observed response history, i.e., one-step-ahead evaluation of the GARMA residual recurrence. Requires the response variable in `newdata`.
- `FALSE`: Disregard AR and MA effects. For `family = gaussian()`, `predict()` uses only `sigma` for residuals.
- `"replicate"`: Generate fresh response series recursively instead of conditioning on the observed responses. Only for predictions: `predict()`, `posterior_predict()`, and `pp_check()` (where it is the default). Applies equally to prior and posterior draws.

For posterior evaluation of the original data, retained JAGS imputations supply missing GARMA histories. In models with group-level effects, this currently requires all such effects (`group_effects = TRUE`).

`ndraws` Integer or `NULL`. Number of posterior draws to return/summarise. `NULL` means "all". More draws trade speed for accuracy.

`draws_format` One of "tidy" or "matrix". Controls the output format when `summary == FALSE`.

`nsamples` Deprecated. Use `ndraws` instead.

`samples_format` Deprecated. Use `draws_format` instead.

`varying` Deprecated. Use `group_effects` instead.

`probs` Deprecated. Use `width` instead.

`...` Must be empty. Reserved for future use.

`dpar` What distributional parameter to evaluate. E.g.,

- `"epred"` (default): Expected response from the full model (or `NULL` for compatibility with brms etc.).
- `"mu"`: The conditional mean (or success probability per trial for binomial/bernoulli models), on the link or response scale.
- `"sigma"`: The standard deviation of the residuals. With `ar()` or `ma()`, it is the SD of the innovations, i.e., the part of the residuals not explained by the AR/MA terms.
- `"ar1"`, `"ar2"`, `"ma1"`, `"ma2"`, etc. depending on which AR or MA coefficient you want to evaluate.

`scale` One of

- `"response"`: return on the response scale, i.e., after applying the inverse link function.
- `"linear"`: return on the linear-predictor (link) scale, where the linear trends are modeled.

Details

`fitted()` and `posterior_epred()` evaluate the same expected responses; `predict()` and `posterior_predict()` evaluate the same response distributions. The `posterior_*()` methods return draws-by-observation matrices, while `fitted()` and `predict()` summarise by default and also offer tidy draws. For binomial models, the default response scale is counts. Use `rate = TRUE` for proportions or `fitted(..., dpar = "mu")` for success probabilities. During migration from v0.3.4, an omitted `rate` warns once per session per function when counts differ from proportions. Explicit `rate` settings do not warn.

`residuals(fit)` is equivalent to `fit$data[[mcp_columns(fit)$response]] - fitted(fit, ...)` (or `newdata[[mcp_columns(fit)$response]] - fitted(fit, ...)`), but with fixed arguments for `fitted`: `rate = FALSE, dpar = 'epred'`.

`log_lik()` defaults to an unsummarised draws-by-observation matrix, as used by `loo` and other posterior workflows. Non-default `group_effects` and `arma` settings evaluate conditional or counterfactual log-likelihoods (e.g., without group-level effects or serial correlation); they cannot be used in `loo()` or `waic()` because estimating information criteria for reduced models requires refitting.

Missing responses in the original data remain missing in the response column. `fitted()` returns their expected responses, while `predict()` generates fresh posterior predictive response draws (including at missing rows). In GARMA models, retained JAGS imputations supply the history used for later fitted and predicted rows. Read more.

Value

- If `summary = TRUE`: A tibble with the columns of `newdata` (or `fit$data` if `newdata` is `NULL`), followed by summaries for each row:
  - `.mean`: The mean of the draws.
  - `.sd`: The SD of the draws. With posterior draws (the default), this is the posterior predictive SD for `predict()` and the posterior SD of the evaluated quantity otherwise.
  - `.lower` and `.upper`: The lower and upper ends of the central interval given in `width`.

  For `predict()` with built-in families, these are the exact moments and quantiles of the posterior predictive distribution given the evaluated draws (the mixture over draws), so they do not vary with the seed. Two exceptions: `ndraws` selects draws at random, and `arma = "replicate"` conditions on randomly generated response histories, i.e., a Monte Carlo approximation.

  With `prior = TRUE`, these are the analogous prior summaries. mcp's own columns are dot-prefixed to avoid collisions with data columns.
- If `summary = FALSE` and `draws_format = "tidy"`: A `tidybayes` `tibble` with all the posterior draws (`Nd`) evaluated at each row in `newdata` (`Nn`), i.e., with `Nd x Nn` rows, ordered by `.row` and then `.draw`. If there are group-level effects, the returned data is expanded with the relevant levels for each row.

  The return columns are:
  - Predictors from `newdata`, plus its response column when supplied.
  - Draw descriptors: ".chain", ".iteration", ".draw" (see the `posterior` and `tidybayes` packages), and `.row`, the row number in the evaluated `newdata`.
  - Draw values: one column for each parameter in the model.
  - The estimate (matching tidybayes conventions): `.epred` for expected responses, `.linpred` with `scale = "linear"`, the `.dpar` for other dpars (e.g., `.mu` or `.sigma`), `.prediction`, `.residual`, or `.loglik`.

- If `summary = FALSE` and `draws_format = "matrix"`: An `N_draws` X `nrows(newdata)` matrix with fitted/predicted values (depending on the function). This format is used by `brms` and it's useful as `yrep` in `bayesplot::ppc_*` functions.

Functions

- `predict(mcpfit)`: Predictive Distribution
- `fitted(mcpfit)`: Expected response
- `log_lik(mcpfit)`: Pointwise log-likelihood
- `residuals(mcpfit)`: Residual distribution

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

See Also

`fitted.mcpfit` `predict.mcpfit` `residuals.mcpfit` `log_lik.mcpfit`

Examples

```
head(fitted(demo_fit))  # Expected response for each row of demo_fit$data
head(residuals(demo_fit))  # Residuals for each row of demo_fit$data
log_lik(demo_fit)[1:3, 1:3]  # Log-likelihood at each demo_fit$data

# All of the above take a range of arguments. E.g.,:

head(predict(demo_fit))  # Pointwise posterior predictive
head(predict(demo_fit, width = 0.8))  # 80% posterior predictive interval
head(predict(demo_fit, prior = TRUE))  # Prior predictive
head(fitted(demo_fit, summary = FALSE))  # Draws. Useful for plotting distributions.
head(fitted(demo_fit, dpar = "sigma"))  # Another model parameter

# Custom quantiles from the draws, here for fitted().
# Note that for predict(), this adds more Monte Carlo error than `width`.
fitted(demo_fit, summary = FALSE) |>
  dplyr::group_by(.row, time) |>
  dplyr::reframe(quantile = c(0.1, 0.5, 0.9), .epred = quantile(.epred, quantile)) |>
  head()

# Evaluate at novel data
novel_data = data.frame(time = c(-5, 20, 300))  # Only predictors are needed
head(predict(demo_fit, newdata = novel_data))

# Work with missing responses
missing_fit = mcp_example("missing", plot = FALSE)
fitted(missing_fit) |> dplyr::filter(is.na(y)) |> head()  # Expected responses for missing y
fitted(missing_fit, summary = FALSE) |> dplyr::filter(is.na(y)) |> head()  # Same, but draws
predict(missing_fit) |> dplyr::filter(is.na(y)) |> head()  # Posterior predictive for missing y
```

`hypothesis` *Test Hypotheses Concerning Individual Parameters*

Description

Returns posterior probabilities and Bayes factors for flexible hypotheses involving model parameters. The documentation for the argument `hypotheses` below shows examples of how to specify hypotheses, and read worked examples on the mcp website. For directional hypotheses, `hypothesis` executes the hypothesis string in a data-frame environment and summarises the proportion of posterior and prior draws where the expression evaluates to TRUE. The Bayes factor is the posterior odds divided by the prior odds. For equality hypotheses, a Savage-Dickey ratio is computed. Both kinds of Bayes factor require prior draws, so remember `mcp(..., sample = "both")`. This function is heavily inspired by the `hypothesis` function from the `brms` package. When `prior = TRUE`, the summary is based on prior draws and `BF` is `NA`.

Usage

```
hypothesis(x, ...)

## S3 method for class 'mcpfit'
hypothesis(x, hypotheses, width = 0.95, prior = FALSE, ...)
```

Arguments

`x`	An `mcpfit` object.
`...`	Must be empty. Reserved for future use.
`hypotheses`	String representation of a logical test involving model parameters. Takes R code that evaluates to TRUE or FALSE in a vectorized way.

Directional hypotheses are specified using <, >, <=, or >=. `hypothesis` returns the posterior probability $P(H \mid \text{data})$ and the Bayes factor in favor of the stated hypothesis H:

$$\text{BF}_{10} = \frac{P(H \mid \text{data})/(1 - P(H \mid \text{data}))}{P(H)/(1 - P(H))}$$

where $P(H)$ is the prior probability of H. The Bayes factor requires both prior and posterior draws from `mcp(sample = "both")`. For example:

- `"cp_1 > 30"`: the first change point is above 30.
- `"Intercept_1 > Intercept_2"`: the intercept is greater in segment 1 than 2.
- `"x_2 - x_1 <= 3"`: slope 2 minus slope 1 is at most 3.
- `"Intercept_1 > -2 & Intercept_1 < 2"`: Intercept_1 is between -2 and 2 (an interval hypothesis). This can be useful as a Region Of Practical Equivalence test (ROPE).
- `"cp_1^2 < 30 | (log(x_1) + log(x_2)) > 5"`: be creative.

- "`cp_1_id[John]` > `cp_1_id[Mary]`": John's change point is later than Mary's, as estimated through the group-level change-point deviation for `id` at change point 1 (starting segment 2). Note that `` ` `` are required around names with `[level]`.

Equality hypotheses use the equal sign (=) and a Savage-Dickey density ratio: posterior density divided by prior density at the tested point equality $\theta = \theta_0$:

$$\mathrm{BF}_{01} = \frac{p(\theta = \theta_0 \mid \text{data})}{p(\theta = \theta_0)}$$

where θ is the evaluated parameter (or affine contrast), θ_0 is the hypothesized null value, $p(\theta = \theta_0 \mid \text{data})$ is the posterior density, and $p(\theta = \theta_0)$ is the prior density. This is a Bayes factor for a nested point-null model against the fitted continuous model ($\mathrm{BF}_{10} = 1/\mathrm{BF}_{01}$). Prior and posterior draws are required, using `mcp(sample = "both")`.

The point-null model's nuisance prior is the fitted model's conditional prior at the equality. Equality tests are limited to named scalar parameters and affine contrasts.

Examples:

- `"cp_1 = 30"`: compare the point-null model `cp_1 = 30` with the continuous alternative.
- `"Intercept_1 - Intercept_2 = 0"`: compare equal segment intercepts with the continuous alternative.

`width` Float. The width of the central posterior interval (between 0 and 1).

`prior` Logical. Summarise prior draws (`TRUE`) instead of posterior draws (`FALSE`, default)?

Value

A tibble with a row per hypothesis and the following columns:

- `hypothesis` is the hypothesis; often re-arranged to test against zero.
- `mean` is the posterior mean of the left-hand side of the hypothesis, or the prior mean when `prior = TRUE`.
- `sd` is the posterior standard deviation of ditto, or the prior SD when `prior = TRUE`.
- `lower` is the lower end of the central posterior interval of width `width`, or the corresponding prior interval when `prior = TRUE`.
- `upper` is the upper end of ditto.
- `prob` is the posterior probability of a directional hypothesis, or the prior probability when `prior = TRUE`. It is `NA` for equality hypotheses, which compare models rather than an event within the fitted model.
- `BF` is the Bayes factor in favor of the hypothesis. For "=" it is the Savage-Dickey density ratio. It is `0` when all posterior (but not prior) draws lie on one side of the tested value; increase `iter` for a finite estimate. For directional hypotheses, it is the posterior odds divided by the prior odds. It is `Inf` (or `0`) when all (or no) posterior draws satisfy the hypothesis, i.e., the evidence exceeds what the number of draws can resolve; increase `iter` for a finite estimate. It is `NA` when `prior = TRUE`, when prior draws are not available (e.g., `sample = "post"`), or when all or no prior draws satisfy the hypothesis.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
# demo_fit contains both posterior and prior draws
# A directional hypothesis returns its posterior probability and Bayes factor
hypothesis(demo_fit, "cp_1 > 30")

# Combine directional statements for an interval (a ROPE-style hypothesis)
hypothesis(demo_fit, "cp_1 > 20 & cp_1 < 30")

# Evaluate several directional hypotheses at once
hypothesis(demo_fit, c("cp_1 > 30", "cp_2 > 70"))

# Inspect the corresponding prior probability without a Bayes factor
hypothesis(demo_fit, "cp_1 > 30", prior = TRUE)
```

---

interpolate_newdata *Returns a tibble with all combos of predictors*

Description

[Experimental]

Usage

```
interpolate_newdata(
  fit,
  by = NULL,
  x_values = NULL,
  at = NULL,
  arma = NULL,
  prior = FALSE
)
```

Arguments

fit	An mcpfit object.
by	Character vector of categorical or group-level columns to evaluate separately. Categorical model predictors are always evaluated separately.
x_values	Numeric vector of x-values to evaluate at.
at	Named list setting additional continuous predictors to fixed values. They default to their observed means. Family response auxiliaries can also be supplied as explicit scalar design values; e.g., at = list(N = 20).
arma	How AR/MA terms will be evaluated (see predict.mcpfit()). Defaults to TRUE when the fit includes ar() or ma() terms.

- TRUE: Preserve the observed rows and response history for conditional AR/MA evaluation.
- FALSE: Interpolate unconditional trends.
- "replicate": Preserve the observed rows, because AR/MA terms depend on the previous rows.

prior — Logical. Increase the resolution around the change points of the prior draws (TRUE) instead of the posterior draws (FALSE, default).

Details

This function synthesizes predictors for all combinations of predictor values. It is used internally in plot.mcpfit() and may be useful if you want to build your own custom plot.

The par_x variable will be interpolated with higher resolution around the change points where the values can change abruptly, but lower resolution in between to speed up the computation.

Categorical variables and requested grouping factors are combined factorially (all level combinations). Additional continuous predictors are held at their observed means, or at values supplied through at. Family-specific response auxiliaries are not interpolated. Supply binomial trial counts as a scalar in at or use newdata for a varying design.

Likelihood weights are needed only when evaluating log_lik() and therefore need not be supplied here.

Value

A tibble with

- A column for the change-point variable (par_x)
- All combinations of categorical predictor levels
- Fixed values for additional continuous predictors

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
# Get predictors for a fit
newdata = interpolate_newdata(demo_fit)

# Fit summary
head(fitted(demo_fit, newdata))

# Predictions for each draw
prediction = predict(demo_fit, newdata, summary = FALSE)
head(prediction)

# Custom plot
library(ggplot2)
plotdata = fitted(demo_fit, newdata)  # `width = .95` by default
ggplot(plotdata, aes(x = time, y = .mean)) +
```

```
  geom_ribbon(aes(ymin = .lower, ymax = .upper), alpha = 0.3) +
  geom_line(lwd = 2) +
  geom_point(aes(y = response), data = demo_fit$data)
```

`is.mcpfit` *Checks if the Argument is an* `mcpfit` *Object*

Description

Checks if the Argument is an `mcpfit` Object

Usage

```
is.mcpfit(x)

is_mcpfit(x)
```

Arguments

`x`	An R object.

Value

Logical scalar (`TRUE` if `x` is an `mcpfit` object, `FALSE` otherwise).

`loo.mcpfit` *Information Criteria for Model Comparison*

Description

Compare models using `loo_compare` (re-exported by mcp) and `loo_model_weights`. Read more in `loo` and see worked examples on the mcp website.

Usage

```
## S3 method for class 'mcpfit'
loo(x, ..., ndraws = NULL, nsamples = lifecycle::deprecated())

## S3 method for class 'mcpfit'
waic(x, ..., ndraws = NULL, nsamples = lifecycle::deprecated())
```

Arguments

x	An mcpfit object.
...	For loo(): further arguments passed to loo, e.g., cores or save_psis. For waic(): must be empty.
ndraws	Integer or NULL. Target number of posterior draws used for the log-likelihood or information criterion. Draws are balanced across chains, so the actual number may be rounded. NULL uses all draws.
nsamples	Deprecated. Use ndraws instead.

Details

Observationwise PSIS-LOO and WAIC are problematic for AR/MA models because both treat individual conditional likelihood terms as validation units. In PSIS-LOO, a held-out response also remains in the conditioning history of later terms. Prefer leave-future-out or blocked cross-validation, which are not currently implemented in mcp. When a missing response enters a later observed GARMA history, loo(), waic(), and log_lik() with arma = TRUE are unavailable: the observed-data likelihood requires integrating over that missing history, which mcp does not currently implement.

loo() and waic() evaluate the likelihood of the fitted model, including all group-level effects and AR/MA terms. Evaluating an information criterion with fitted components dropped post-hoc violates the PSIS identity because draws come from the full model's posterior; comparing a reduced model requires refitting it. log_lik(fit, group_effects = FALSE) and log_lik(fit, arma = FALSE) remain available as conditional or counterfactual diagnostics.

When ndraws is supplied, draws are balanced across chains and thinned at evenly spaced midpoint iterations. This preserves MCMC chain identities and chronological order, allowing relative_eff() to be computed directly.

Value

a loo or psis_loo object.

Functions

- waic(mcpfit): Computes WAIC on mcpfit objects

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
# Define two models and sample them
# future::plan(future::multisession, workers = 3)  # Uncomment for parallel sampling
set.seed(42)
data = data.frame(x = seq(-1, 1, length.out = 100))
data$y = 1 + 2 * data$x + rnorm(100, sd = 0.3)
model1 = list(y ~ 1 + x)
model2 = list(y ~ 1)
```

```
fit1 = mcp(model1, data, seed = 42)
fit2 = mcp(model2, data, par_x = "x", seed = 42)

# Compute LOO for each and compare (works for waic(fit) too)
loo1 = loo(fit1)
loo2 = loo(fit2)
loo_compare(loo1, loo2)
```

---

mcp *Fit Multiple Linear Segments And Their Change Points*

Description

Given a model (a list of segment formulas), mcp infers the posterior distributions of the parameters of each segment as well as the change points between segments. See details or the mcp website.

Usage

```
mcp(
  model,
  data,
  prior = list(),
  family = gaussian(),
  par_x = NULL,
  sample = "post",
  chains = 3,
  iter = 3000,
  warmup = 1000,
  inits = NULL,
  jags_code = NULL,
  seed = NULL,
  diagnostics = list(),
  quiet = getOption("mcp_quiet", FALSE),
  cores = lifecycle::deprecated(),
  adapt = lifecycle::deprecated()
)
```

Arguments

model	A list of formulas, one for each segment, in the format `model = list(response ~ predictors, ~ cp(...) + predictors)`, where `cp()` is optional. See mcp-formula for how segments connect and the full formula syntax. Read more.
data	Table-like data in long format (data.frame, tibble, data.table, etc.) with syntactic column names. The response variable can contain missing values, which will be imputed jointly with the parameters. Read more.

prior — Named list, e.g., `prior = list(cp_1 = "dunif(0, 100)", Intercept_1 = "dnorm(0, 1)")`. List names are parameter names (`cp_1`, `Intercept_1`, `x_2`, `sigma_1`, etc.) and values are a distribution string (e.g., `"dnorm(0, 1) T(0, )"`), a constant (e.g., `-2.1`), or the name of another parameter to share its value. *OBS: Distributions use JAGS names but SD/scale instead of JAGS precision, so* `dnorm(0, 0.001)` *is a very narrow prior.* Weakly informative defaults are used where priors are not specified; see them with `prior_summary()`. See mcp-priors for the full syntax. Read more.

family — A supported family and link function:

- `gaussian(link = "identity")` or `gaussian(link = "log")`
- `binomial()` or `bernoulli()` with `link = "logit"`, `"probit"`, or `"identity"`
- `poisson(link = "log")` or `poisson(link = "identity")`
- `negbinomial(link = "log", link_shape = "log")`

Read more.

par_x — String. Name of the change-point variable, i.e., the data column on which change points are located. `NULL` (default) auto-detects it as the only continuous variable in the formulas.

sample — One of

- `"post"`: Sample the posterior.
- `"prior"`: Sample only the prior. Use `prior = TRUE` explicitly in plots, summaries, and other methods to select prior draws.
- `"both"`: Sample both prior and posterior. Plots, summaries, etc. will default to using the posterior. Use `prior = TRUE` to select prior draws.
- `"none"` or `FALSE`: Do not sample. Returns an mcpfit object without draws (an "empty fit"). This is useful if you only want to check prior strings (`fit$prior`), the JAGS model (`fit$jags_code`), etc.

chains — Positive integer. Number of chains to run.

iter — Positive integer. Number of post-warmup draws from each chain. The total number of draws is `iter * chains`.

warmup — Positive integer. Number of initial iterations per chain which are discarded before sampling. Set higher if needed for sampler adaptation; use diagnostics to assess convergence.

inits — A named list of initial values for the parameters, e.g., `list(Intercept_1 = 10)`, or a list of such lists with one per chain. This can be useful if a model fails to converge (read more). Names must be parameters in `mcp_pars()`. Parameters with constant priors and parameters that are reparameterized for faster sampling cannot be initialized (see `mcp_reparam` in mcp-package). Initial values for change points must be ordered. Defaults to `NULL` where all chains start at equally spaced change points and the prior medians. User-provided `inits` take precedence.

jags_code — String. JAGS code to sample instead of the code that `mcp` writes. This is an expert option for edits to `fit$jags_code` that keep the parameters in `mcp_pars()`, the data, and the likelihood. Only the parameters in `mcp_pars()` are monitored. R-side simulation and prediction methods continue to use the `mcp` formulas (with

warning), so `fitted()`, `predict()`, `plot()`, `loo()`, etc. are wrong if the custom code changes the formula or likelihood. The written JAGS code is not a stable interface and may change between `mcp` versions. `jags_code` is planned for deprecation once a future mcp release brings custom families and tooling to run `fit$jags_code` directly in JAGS. Read more about how `mcp` writes JAGS code.

`seed` — `NULL` or a positive integer. Seed for the JAGS random-number generators. If `NULL` (default), the seed is drawn from R's random-number generator, so `set.seed()` before `mcp()` also makes sampling reproducible.

`diagnostics` — Named list of diagnostic warning thresholds. Available elements are `rhat = 1.01`, `ess_bulk = 400`, `ess_tail = 400`, `ar = 0.10`, and `ma = 0.10`. An empty list uses these defaults; a partial list overrides only the supplied values. Set an element to `NULL` to disable that diagnostic, or use `FALSE` to disable all configurable diagnostic warnings. In `summary.mcpfit()`, `NULL` inherits the settings used to fit the model, while a list or `FALSE` overrides the diagnostic footer.

`quiet` — Logical. Suppress routine JAGS output and mcp sampling-status messages? Defaults to `FALSE`. Use `options(mcp_quiet = TRUE)` to change the default for the session.

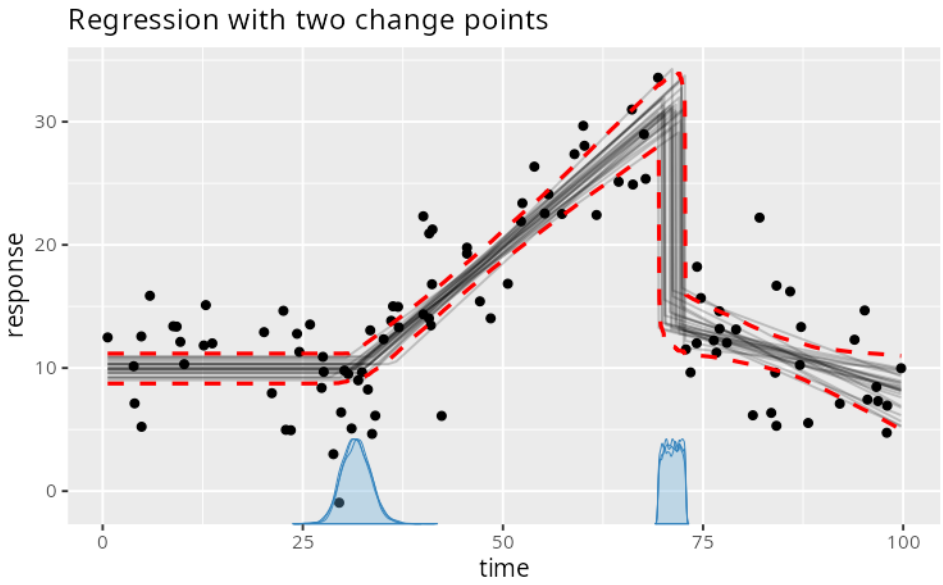
`cores` — Deprecated and ignored. Configure parallel processing with a future plan instead, for example `future::plan(future::multisession, workers = 3)`. With the default future plan, chains are sampled sequentially. The argument remains available for backwards compatibility.

`adapt` — Deprecated; use `warmup` instead.

Details

Here is the demo model, which ships already fitted as `demo_fit` (see also `mcp_example("demo")`):

```
model = list(
  response ~ 1,  # Plateau in the first segment (Intercept_1)
  ~ 0 + time,    # Joined slope (time_2) in segment 2 which starts at cp_1
  ~ 1 + time     # Disjoined slope (Intercept_3, time_3) at cp_2
)
```



Segment 2 continues from where the plateau left off, while segment 3 starts afresh with a new intercept. See mcp-formula for the rules of how segments connect, the full formula syntax, and the underlying model. See mcp-priors for how to specify priors. See the mcp website for worked examples of all model features.

Value

An `mcpfit` object.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

References

- Lindeløv, J. K. (2020). mcp: An R Package for Regression With Multiple Change Points. *OSF Preprints*. doi:10.31219/osf.io/fzqxv Introduces the `mcp` package, formula syntax, default priors, and workflow for regression with multiple change points across generalized linear and time-series models. Newer-than-2020 versions of the paper may be available at that link. Please cite the newest version.
- Carlin, B. P., Gelfand, A. E., & Smith, A. F. (1992). Hierarchical Bayesian Analysis of Changepoint Problems. *Applied Statistics*, 41(2), 389–405. doi:10.2307/2347570 Introduced Gibbs sampling for hierarchical Bayesian change-point models, the computational approach that BUGS and JAGS build on.

Examples

```
# Define the segments using formulas. A change point is estimated between each formula.
model = list(
  response ~ 1,  # Plateau in the first segment (Intercept_1)
```

```
  ~ 0 + time,    # Joined slope (time_2) in segment 2 which starts at cp_1
  ~ 1 + time     # Disjoined slope (Intercept_3, time_3) at cp_2
)

# Fit it and sample the prior too.
# future::plan(future::multisession, workers = 3)  # Uncomment for parallel sampling
data = mcp_example_data("demo")  # Simulated data example
demo_fit = mcp(model, data = data, sample = "both", seed = 42)

# See parameter estimates
summary(demo_fit)

# Visual inspection of the results
plot(demo_fit)  # Visualization of model fit/predictions
plot_pars(demo_fit)  # Parameter distributions
pp_check(demo_fit)  # Prior/Posterior predictive checks

# Test a hypothesis
hypothesis(demo_fit, "cp_1 > 30")

# Make predictions
head(fitted(demo_fit))
head(predict(demo_fit))
head(predict(demo_fit, newdata = data.frame(time = c(55.545, 80, 132))))

# Compare to a one-intercept-only model (no change points) with default prior
model_null = list(response ~ 1)
fit_null = mcp(model_null, data = data, par_x = "time")  # fit another model here
demo_loo = loo(demo_fit)
null_loo = loo(fit_null)
loo_compare(demo_loo, null_loo)

# Inspect the prior. Useful for prior predictive checks.
summary(demo_fit, prior = TRUE)
plot(demo_fit, prior = TRUE)

# Show all priors. Default priors are added where you don't provide any
prior_summary(demo_fit)

# Set priors and re-run
prior = list(
  Intercept_1 = 15,
  time_2 = "dt(0, 2, 1) T(0, )",  # t-dist slope. Truncated to positive.
  cp_2 = "dunif(min(time), 80)"   # change point to segment 3 before 80 (and after cp_1).
)

fit3 = mcp(model, data = data, prior = prior, seed = 42)

# Reuse coefficients in later segments using same() (e.g., Intercept_1 in segment 3)
model_same = list(
  response ~ 1,
  ~ 0 + time,
  ~ same(1, as = 1) + time  # Reuse Intercept_1 instead of estimating Intercept_3
```

```
)
fit_same = mcp(model_same, data = data, sample = FALSE)

# Show the JAGS model
demo_fit$jags_code
```

`mcp-formula` *Model formulas in mcp*

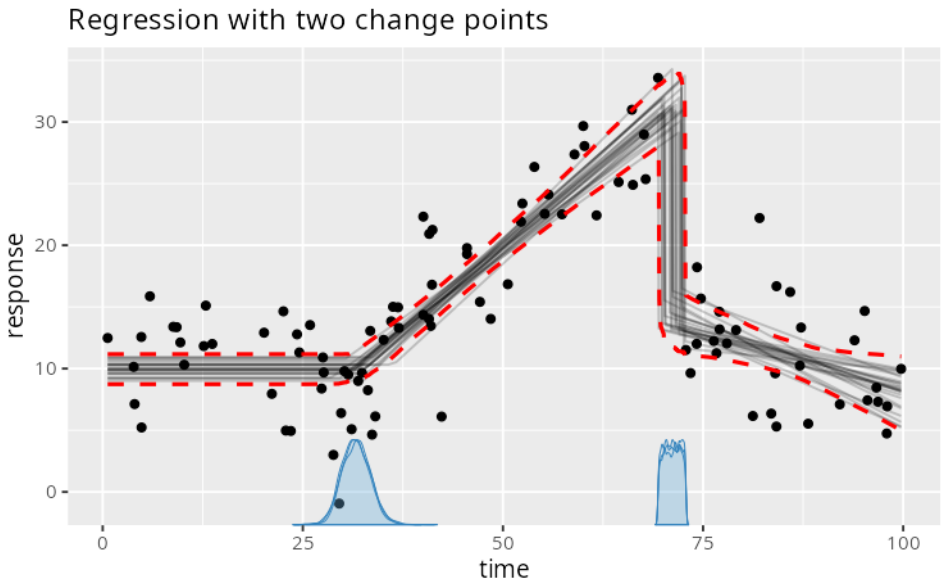
Description

The `model` argument of `mcp()` is a list of formulas, one for each segment. A change point is estimated between each pair of consecutive segments. This page describes how segments connect, the formula syntax, and the underlying model. See mcp-priors for priors on the resulting parameters.

Details

Here is the demo model (see `mcp_example("demo")`):

```
model = list(
  response ~ 1,  # Plateau in the first segment (Intercept_1)
  ~ 0 + time,    # Joined slope (time_2) in segment 2 which starts at cp_1
  ~ 1 + time     # Disjoined slope (Intercept_3, time_3) at cp_2
)
```



Segment 2 continues from where the plateau left off, while segment 3 starts afresh with a new intercept.

The modeling options and behavior are described below. They are described in a more applied and comprehensive way on the mcp website.

Three rules govern how segments connect (with `x` as the change-point variable, i.e., `time` above):

1. **Only included terms get coefficients.** A segment includes the terms in its formula, plus an intercept unless removed with `0 +`. Each included term gets a new coefficient (`x_2`, `stateB_2`) or reuses an existing one if wrapped in `same()`.
2. **x-terms are measured from the change point.** Only `x`, `x:z`, `state:x`, and `I(x^2)` start at zero at the segment's change point and stop growing at the next change point.
3. **Joined or disjoined.** Without an intercept (`0 + ...`), a segment is *joined*: it continues from where the earlier intercept and x-terms left off. With an intercept, it is *disjoined* and starts afresh.

`sigma` and `shape` are required by the likelihood, so they include an intercept in segment 1. They are joined in later segments that do not include them. `ar()` and `ma()` are not required, so a segment that does not include them has no AR/MA component. Each segment after the first starts at a change point, so it always includes a change-point intercept (`cp_1`, `cp_2`, ...). Use `cp()` to add terms to it, e.g., `cp((1 | id))`.

Formula syntax. The first segment has the format `response ~ predictors`. Later segments use the same response, so they are one-sided, `~ predictors`, and each starts at a change point that can be modeled using `cp()`, e.g., `~ cp((1 | id)) + 0 + x`. `cp()` is the change-point part of the formula and the rest is the predictor part.

1. Response (segment 1 only):

- `y ~ ...`: Standard continuous, binary, or count response (Gaussian, Bernoulli, Poisson, negative binomial).
- `successes | trials(total) ~ ...`: Binomial response (`family = binomial()`). Read more.
- `y | weights(w) ~ ...`: Observation log-likelihood weights (multiplies each observation's log-likelihood contribution by `w > 0`; affects posterior inference and `log_lik()`, but not predictions).
- `y | trials(total) + weights(w) ~ ...`: Combine response auxiliaries using `+`.

2. Change-point part (`cp()`, segments 2+):

- `~ x` (i.e., `cp()` not included): Population-level change point (default). Same as `~ cp(1) + x`.
- `~ cp((1 | id)) + x`: Group-level change-point deviations around the population-level change point. Read more.

3. Predictor part (all segments): Read more

- `~ 1 + x`: Disjoined slope with a new segment intercept.
- `~ 0 + x`: Joined slope (no new intercept).
- `~ 1`: Plateau (intercept only, no slope).
- `~ x:state + I(x^2) + exp(z)`: Extended terms, interactions, and R-side bases (`scale()`, `poly()`, `splines::ns()`). Bases are evaluated before sampling and reused for `newdata`. Except for `I(x^k)`, transformations of `x` (e.g., `I(x / 2)`) use the original values of `x`, as in `lm()`, and do not join.

- `~ 1 + offset(log(exposure))`: Offset with a fixed coefficient of 1. If one segment includes an offset, all segments must include one. The same applies within `sigma()`, etc., and within `ar()`/`ma()` in the segments that include them. Read more.
- `~ 1 + (1 | id)`: Group-level intercepts (or `(1 + x || id)` for independent slopes and intercepts). Read more.
- `~ sigma(1 + x)`: Distributional parameters on the link scale (e.g., log residual SD), with coefficients `sigma_Intercept_2` and `sigma_x_2`. Without `sigma()` anywhere in the model, `sigma_1` is the residual SD itself. Read more.
- `~ ar(1) + ma(1)`: Autoregressive and moving-average time-series residuals on the link scale (accepts regression formulas; include them in every segment where they apply). Set `series = id` and `threshold` once in any term; they apply to all segments. Read more.
- `~ 1 + same(x)` or `~ 0 + same(z, as = 1)`: Reuse a coefficient from the preceding segment (default) or from segment `as`. Reused coefficients are estimated jointly from all segments where they are included, not fitted in one segment and copied to another. Read more.

The mcp model

An `mcp` model divides the change-point variable x into K segments separated by ordered change points $\tau_1 < \cdots < \tau_{K-1}$. In each segment $k \in \{1, \ldots, K\}$, the linear predictor η_i measures x from the change point τ_{k-1}:

$$\eta_i = \alpha_k + \beta_{k,1}(x_i - \tau_{k-1}) \quad (\text{with } \tau_0 = 0)$$

where α_k is the value at the start of segment k:

$$\alpha_k = \begin{cases} \beta_{k,0}, & \text{disjoined } (\sim \texttt{1 + x}, \text{ or } k = 1) \\ \alpha_{k-1} + \beta_{k-1,1}(\tau_{k-1} - \tau_{k-2}), & \text{joined } (\sim \texttt{0 + x}) \end{cases}$$

That is, a disjoined segment starts at its own intercept $\beta_{k,0}$, while a joined segment continues from where segment $k - 1$ left off.

Here, $\beta_{k,1}$ is the slope on x in segment k. Intercepts and slopes are absolute values (not changes relative to preceding segments). Other variables (e.g., `+ z + state`) add $\sum_j \gamma_{k,j} z_{j,i}$ in the segments where they are included. They use their original values and are not part of α_k, which is why they can make a joined segment start with a jump.

The inverse-linked parameter is $\mu_i = g^{-1}(\eta_i)$ via link function $g(\mu_i) = \eta_i$, representing the expected response for most families (or the success probability for binomial models, where the expected count is $n_i \mu_i$). Distributional parameters (`sigma()`, `shape()`, etc.) and autoregressive terms (`ar()`, `ma()`) follow this exact same segmented structure on their respective link scales. Since each of them has its own segmented linear predictor, `mcp` calls them all distributional parameters (`dpar`): `mu`, `sigma`, `shape`, and each AR/MA lag (`ar1`, `ar2`, `ma1`, ...), e.g., `fitted(fit, dpar = "ar1")` or `fixef(fit, dpar = "sigma")`. See more details on the mcp website.

Time-series residuals (link-scale observation-driven GARMA)

Autoregressive (`ar(p)`) and moving-average (`ma(q)`) terms define a finite conditional recurrence on the link scale (generalized autoregressive moving-average, GARMA). They support Gaussian

(`identity`), binomial (`logit`), Bernoulli (`logit`), Poisson (`log`), and negative-binomial (`log`) families. If η_t^{reg} is the ordinary linear predictor from the segment formulas and η_t is the linear predictor including serial dependence, the recurrence decomposes into components:

$$\text{AR}_t = \sum_{j=1}^{p} \phi_{j,t} \left[g(y^*_{t-j}) - \eta^{\text{reg}}_{t-j} \right]$$
$$\text{MA}_t = \sum_{k=1}^{q} \theta_{k,t} \left[g(y^*_{t-k}) - \eta_{t-k} \right]$$
$$\eta_t = \eta_t^{\text{reg}} + \text{AR}_t + \text{MA}_t$$

where $\phi_{j,t}$ is the lag-j autoregressive (AR) coefficient at time t, $\theta_{k,t}$ is the lag-k moving-average (MA) coefficient at time t, $g(\cdot)$ is the link function, and y_t^* is the observation, constrained by a threshold constant c (argument `threshold = 0.1` in `ar()` / `ma()`) to keep residuals finite on the link scale, e.g., $y_t^* = \max(y_t, c)$ for zero counts in Poisson models. Read more.

Implications:

- For an N-order component, the last N values *before* the segment's change point are input to the first η_t in the segment.
- AR and MA components apply only in segments where they are included.
- AR coefficients are not jointly constrained to stationarity; nor MA coefficients to invertibility.
- See the arma article for more details.

See Also

`mcp()`, mcp-priors, `mcp_pars()` for the parameter names of a fitted model.

`mcp-priors` *Priors in mcp*

Description

The `prior` argument of `mcp()` is a named list. Names are parameter names (`cp_1`, `Intercept_1`, `x_2`, `sigma_1`, etc.; see mcp-formula for how they arise) and the values specify their priors. Use `prior_summary()` to see the resolved priors of a fitted model, including defaults. See Working with priors for worked examples and prior predictive checks.

Details

Each value in the list is either

- A distribution string (e.g., `Intercept_1 = "dnorm(0, 1) T(0, )"`). Distributions use JAGS names and argument order, including `T()` truncation, but with conventional scales instead of JAGS precision: SD for `dnorm()`, SD on the log scale for `dlnorm()` (`sdlog` in `stats::dlnorm()`), and scale for `dt()`, `ddexp()`, and `dlogis()`. Change points also accept `dirichlet(alpha)`, and all priors accept data-dependent terms such as `sd(y)` (see below).

- A numerical value (e.g., `Intercept_1 = -2.1`) fixing the parameter to that constant.
- A model parameter name (e.g., `Intercept_2 = "Intercept_1"`), sharing that parameter's value. Note that reusing coefficients via `same()` in the segment formulas (e.g., `~ same(1)`) is generally preferred. If two group-level deviations are shared via the prior, they will need to have the same grouping variable.

Regression priors

Weakly informative defaults are used where priors are not specified. These are the conventional defaults of `rstanarm` for intercepts and coefficients and of `brms` for scale parameters, with a few documented deviations. They are designed for stable estimation and prediction, but should be justified before hypothesis testing. Use `prior_summary(fit, verbose = TRUE)` to see the default priors of a model and the rules behind them. For more details, read the article on mcp default priors for all families, links, and distributional parameters, and for how the defaults deviate from `rstanarm` and `brms`.

Change point priors

Change points are always ordered: each change point keeps its specified prior, restricted to ordered change points. Use `plot_pars(fit, prior = TRUE)` to see the resulting marginal priors. The default is a flat Dirichlet prior on the segment lengths, written as `dunif(min(x), max(x))` on each change point (read more). Set a minimum segment length using the previous change point in the lower bound, e.g., `cp_2 = "dunif(cp_1 + 20, max(x))"`. Read more about setting priors on change points.

Notes on priors

- *Change point priors* without `dunif()` or `T()` are truncated to the observed range of x, e.g., `dnorm(30, 10)` becomes `dnorm(30, 10) T(min(x), max(x))`.
- *Data-dependent terms:* If `mcp` encounters a data-dependent term like `min(time)`, `max(time)`, `mean(response)`, `median(response)`, `sd(response)`, or `mad(response)` in the prior string, they are resolved from the model data so a numerical value is passed to JAGS. The following terms are also allowed: `n_segments()` and `n_cp()`. The older constants `MINX`, `MAXX`, `MEANX`, `SDX`, `MINY`, `MAXY`, `MEANY`, `SDY`, and `N_CP` remain accepted with a deprecation warning.
- *Group-level change points:* Group-specific change points follow a hierarchical normal distribution around their population-level change point, truncated so that they remain in the observed range and ordered. Read more.
- *Parameterization:* Prior strings use conventional scale parameterizations. `mcp` converts these to the parameterization required by JAGS when generating code: inverse variance for `dnorm()`, `dt()`, and `dlnorm()`, and inverse scale for `ddexp()` and `dlogis()`. Use `prior_summary(fit)` for resolved priors and `prior_summary(fit, verbose = TRUE)` for their rules and descriptions.

See Also

`mcp()`, `prior_summary()`, mcp-formula

Examples

```
model = list(
  response ~ 1,
  ~ 0 + time,
  ~ 1 + time
)
prior = list(
  cp_1 = "dunif(20, 50)",         # Distribution
  time_2 = "dnorm(0, 2) T(0, )",  # Truncated distribution
  Intercept_3 = 15,               # Constant
  time_3 = "time_2"               # Shared with another parameter
)
fit = mcp(model, data = mcp_example_data("demo"), prior = prior, sample = "none")
prior_summary(fit)
```

`mcp_columns` *Model data columns*

Description

[Experimental]

Usage

```
mcp_columns(fit)
```

Arguments

`fit`	An `mcpfit` object.

Details

Return the resolved data-column roles for an `mcpfit` object. In particular, `par_x` is the change-point variable chosen automatically or supplied to `mcp()`. Family-defined response auxiliaries, such as `trials` and `weights`, are listed when relevant.

Value

A named list with `par_x`, `response`, and `series`, plus any family-defined response-auxiliary column roles.

Examples

```
# Show the predictor, response, and auxiliary-column roles
mcp_columns(demo_fit)
```

`mcp_example` *Run example models*

Description

Run example models

Usage

```
mcp_example(name, sample = "post", plot = TRUE)

mcp_example_data(name)
```

Arguments

`name`	Name of the example. One of: • `"ar"`: One change point in autoregressive residuals (the `ar1` dpar) • `"binomial"`: Binomial with two change points. Much like `"demo"` on a logit scale. • `"demo"`: Two change points between intercepts and joined/disjoined slopes. • `"group_mu"`: Group-level deviations in the predictor part (random intercepts/slopes) across a change point. • `"group_cp"`: Group-level change-point deviations (random effects). • `"intercepts"`: An intercept-only change point. • `"missing"`: Missing data imputation (NAs in response variable `y`). • `"multiple"`: Multiple regression with categorical predictors and interactions. • `"sigma"`: A change in "sigma" dpar, including a slope on sigma. • `"void"`: Missing data around change points.
`sample`	One of • `"post"`: Sample the posterior. • `"prior"`: Sample only the prior. Use `prior = TRUE` explicitly in plots, summaries, and other methods to select prior draws. • `"both"`: Sample both prior and posterior. Plots, summaries, etc. will default to using the posterior. Use `prior = TRUE` to select prior draws. • `"none"` or `FALSE`: Do not sample. Returns an mcpfit object without draws (an "empty fit"). This is useful if you only want to check prior strings (`fit$prior`), the JAGS model (`fit$jags_code`), etc.
`plot`	Logical. Plot the fitted example? Requires sample != "none".

Value

An `mcpfit`, enriched with an `$example_code` field. It contains the code to reproduce the data and the fit.

Functions

- `mcp_example_data()`: Conveniently get simulated data only.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
fit = mcp_example("multiple")
print(fit$example_code) # See how the data was simulated

# Without sampling
empty = mcp_example("binomial", sample = "none", plot = FALSE)
print(empty)
print(empty$example_code)

# Now sample this model
fit2 = mcp(empty$model, empty$data, family = empty$family, seed = 42)
plot(fit2)
```

---

mcp_pars *Model parameters*

Description

[Experimental]

Usage

```
mcp_pars(fit, scope = NULL, role = NULL)
```

Arguments

`fit`	An `mcpfit` object.
`scope`	Optional parameter scope(s): `"population"` or `"group"`.
`role`	Optional parameter role(s): `"change_point"`, `"coefficient"` (regression coefficients of all distributional parameters, e.g., `mu`, `sigma`, and `ar1`), or `"group_sd"`. Combine with `scope` to select population-level parameters or their group-level deviations, e.g., `scope = "group", role = "change_point"` for group-level change points.

Details

Return the canonical parameter definitions for an `mcpfit` object. This works before sampling, so you can discover the model parameters before setting priors or inits.

Value

A tibble with one row per parameter definition. `parameter` is the model parameter name; `dpar` is its distributional parameter (e.g., `"mu"`, `"sigma"`, or `"ar1"`; see mcp-formula), as accepted by the dpar argument of `fitted.mcpfit()`, `fixef.mcpfit()`, etc., and `NA` for change points; `group_col` and `population_parameter` describe group-level effects.

Examples

```
# Show every parameter in the model
print(mcp_pars(demo_fit), width = Inf)

# Select population-level coefficients
mcp_pars(demo_fit, scope = "population", role = "coefficient")$parameter
```

`mcpfit-class` *Class* `mcpfit` *of Models Fitted with the* **mcp** *Package*

Description

Models fitted with the `mcp` function are represented as an `mcpfit` object which contains the user input (model, data, family), derived model characteristics (prior, parameter names, and jags code), and the fit (prior and/or posterior MCMC draws).

Details

See `methods(class = "mcpfit")` for an overview of available methods.

Components:

- `call`: The matched call to `mcp()`.
- `model`: A list of user-provided formulas.
- `data`: The user-provided data frame reduced to model-used columns.
- `family`: An `mcpfamily` object.
- `prior`: A named list of priors.
- `mcmc_post` and `mcmc_prior`: `mcmc.list` objects with posterior and prior draws, respectively. Do not access these directly; use as_draws(fit) or similar.
- `jags_code`: A string with JAGS code; use `cat(fit$jags_code)` to show it. Read more.
- `simulate`: A function to simulate data from supplied parameter values.
- `version`: A named list of package_versions for `mcp`, `R`, `rjags`, and `jags`, e.g., `fit$version$mcp >= "0.4.0"`. `rjags` and `jags` are `NULL` for fits with `sample = "none"`. `NULL` for fits made with mcp < 0.4.0.
- `.internal`: Information used internally by mcp.

`model-accessors-mcpfit`

Extract Model Information from an `mcpfit`

Description

Standard R accessors for the model formulas, family, fitting data, and number of observations stored in an `mcpfit`.

Usage

```
## S3 method for class 'mcpfit'
family(object, ...)

## S3 method for class 'mcpfit'
nobs(object, ...)

## S3 method for class 'mcpfit'
model.frame(formula, ...)

## S3 method for class 'mcpfit'
formula(x, segment = NULL, ...)
```

Arguments

`object`, `x`, `formula`	An `mcpfit` object.
`...`	Must be empty. Reserved for future use.
`segment`	`NULL` to return all segment formulas, or a positive integer selecting one segment.

Value

`formula()` returns the complete list of segment formulas, or one formula when `segment` is supplied. `family()` returns an `mcpfamily`. `model.frame()` returns the data retained in the fit. `nobs()` returns the number of observed response values (excluding missing responses).

Examples

```
formula(demo_fit)  # Show all segment formulas
formula(demo_fit, segment = 2)  # Show the formula for segment 2
family(demo_fit)  # Show the response family and link
head(model.frame(demo_fit))  # Show the top rows of fitting data
nobs(demo_fit)  # Count observed response rows
```

negbinomial *Negative Binomial for mcp*

Description

Parameterized as mu (the conditional mean) and shape (the same quantity as size in rnbinom()). Thus Var(y) = mu + mu^2 / shape, which approaches the Poisson variance as shape approaches infinity. Read more.

Usage

```
negbinomial(link = "log", link_shape = "log")
```

Arguments

link	Link function for mu.
link_shape	Link function for shape when it is modeled using shape().

Details

shape is required by the likelihood, so it includes an intercept in segment 1. Later segments that do not include shape() are joined. For example, y ~ 1 + x + shape(1 + x) models both the mean and shape. Regression coefficients for both dpars are on their link scales, e.g., shape_Intercept_1 and shape_x_1.

Without shape(), shape_1 is the shape itself with a dinvgamma(0.4, 0.3) prior, as in brms, much like the implicit sigma_1 in gaussian().

Examples

```
# Fit an overdispersed count model with the default log links
data = data.frame(time = 1:6, count = c(1, 2, 8, 3, 12, 5))
fit = mcp(list(count ~ 1), data, family = negbinomial(), par_x = "time", sample = FALSE)
mcp_pars(fit)$parameter  # Show the mean and shape parameters
```

plot.mcpfit *Plot full fits*

Description

Plot prior or posterior model draws on top of data.

Usage

```
## S3 method for class 'mcpfit'
plot(
  x,
  interval = "none",
  width = 0.95,
  facet_by = NULL,
  color_by = "auto",
  lines = 25,
  geom_data = "point",
  cp_dens = TRUE,
  rate = TRUE,
  prior = FALSE,
  arma = TRUE,
  ndraws = 500,
  scale = "response",
  at = NULL,
  q_fit = lifecycle::deprecated(),
  q_predict = lifecycle::deprecated(),
  nsamples = lifecycle::deprecated(),
  ...
)

plot_dpar(
  x,
  dpar = "epred",
  interval = "none",
  width = 0.95,
  facet_by = NULL,
  color_by = "auto",
  lines = 25,
  cp_dens = TRUE,
  prior = FALSE,
  arma = TRUE,
  ndraws = 500,
  scale = "response",
  at = NULL,
  ...
)
```

Arguments

x	An mcpfit object
interval	Which central intervals to plot as dashed lines: • "none" (default): No intervals. • "fitted": Interval of the fitted values (expected response), as in fitted(). Red lines.

- "predict": Posterior predictive interval, as in predict(). Green lines.
- c("fitted", "predict"): Both. plot_dpar() supports only "none" and "fitted".

For other quantiles, see the plotting article for computing them from fitted(..., summary = FALSE) or predict(..., summary = FALSE) and adding them as a ggplot2 layer.

width — Width of the central interval(s) given in interval (strictly between 0 and 1). For example, width = 0.8 plots the 10% and 90% quantiles.

facet_by — Character vector. Names of categorical data columns to split to facets. Can be grouping-factor or categorical predictor columns.

color_by — A character vector naming categorical predictor or grouping columns to color by. Multiple columns are combined as an interaction. Curves and quantiles remain separate for grouping columns not mapped to color. "auto" colors by the categorical predictor if the model has exactly one and it is not in facet_by. NULL disables color.

lines — Positive integer or FALSE. The number of fitted lines (draws). It is the number of joint posterior draws shown for every curve. FALSE or lines = 0 plots no lines. Note that lines always plot fitted values - not predicted. For posterior predictive intervals, use interval = "predict".

geom_data — String. One of "point", "line" (good for time-series), or FALSE (do not plot).

cp_dens — TRUE/FALSE. Plot posterior densities of the change point(s)?

rate — Logical scalar. For binomial models, plot the observed or expected success proportion (rate = TRUE, the default here) or counts (rate = FALSE). This default is the opposite of fitted() and predict(), which return counts by default; plotting proportions keeps curves comparable across varying trial counts. Predictions and count-scale fitted values require a trials column in newdata/at. Other families do not support rate = TRUE, so the default only applies to binomial and Bernoulli models.

prior — Logical. Evaluate prior draws (TRUE) instead of posterior draws (FALSE, default). The selected draws must be available; prior-only fits require prior = TRUE.

arma — How to include AR and MA effects.

- TRUE: Condition on the observed response history, i.e., one-step-ahead evaluation of the GARMA residual recurrence. Requires the response variable in newdata.
- FALSE: Disregard AR and MA effects. For family = gaussian(), predict() uses only sigma for residuals.
- "replicate": Generate fresh response series recursively instead of conditioning on the observed responses. Only for predictions: predict(), posterior_predict(), and pp_check() (where it is the default). Applies equally to prior and posterior draws.

For posterior evaluation of the original data, retained JAGS imputations supply missing GARMA histories. In models with group-level effects, this currently requires all such effects (group_effects = TRUE).

ndraws — Integer or NULL. Number of posterior draws to return/summarise. NULL means "all". More draws trade speed for accuracy.

`scale` One of

- `"response"`: return on the response scale, i.e., after applying the inverse link function.
- `"linear"`: return on the linear-predictor (link) scale, where the linear trends are modeled.

`at` Named list setting additional continuous predictors to fixed values. They default to their observed means. Family-specific response auxiliaries can be supplied as explicit scalar design values. Passed to `interpolate_newdata()`.

`q_fit`, `q_predict`
Deprecated. Use `interval` and `width` instead.

`nsamples` Deprecated. Use `ndraws` instead.

`...` Must be empty. Reserved for future use.

`dpar` What distributional parameter to evaluate. E.g.,

- `"epred"` (default): Expected response from the full model (or `NULL` for compatibility with brms etc.).
- `"mu"`: The conditional mean (or success probability per trial for binomial/bernoulli models), on the link or response scale.
- `"sigma"`: The standard deviation of the residuals. With `ar()` or `ma()`, it is the SD of the innovations, i.e., the part of the residuals not explained by the AR/MA terms.
- `"ar1"`, `"ar2"`, `"ma1"`, `"ma2"`, etc. depending on which AR or MA coefficient you want to evaluate.

Details

`plot()` evaluates the model on posterior draws like `fitted()` and `predict()`. These represent the (joint) posterior distribution. Interval summaries are based on `ndraws` draws, and change-point densities are simple densities on all available draws.

Value

A **ggplot2** object.

Functions

- `plot_dpar()`: Plot distributional parameters

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

See Also

`plot_pars()`, `pp_check()`

Examples

```
# Typical usage. demo_fit is an mcpfit object.
plot(demo_fit)

plot(demo_fit, prior = TRUE)  # The prior

plot(demo_fit, lines = 0, interval = "fitted")  # 95% central interval without lines
plot(demo_fit, interval = "predict", width = 0.8)  # 80% posterior predictive interval
plot_dpar(demo_fit, dpar = "sigma", lines = 100)  # Residual standard deviation on y

# Show a panel for each group-level effect
# plot(fit, facet_by = "my_column")

# Customize plots using regular ggplot2
library(ggplot2)
plot(demo_fit) + theme_bw(15) + ggtitle("Great plot!")
```

---

plot_pars *Plot individual parameters*

Description

Plot many types of plots of parameter estimates. See examples for typical use cases.

Usage

```
plot_pars(
  fit,
  pars = "population",
  regex_pars = character(0),
  type = "combo",
  ncol = 1,
  prior = FALSE,
  npars = 5,
  ask = TRUE
)
```

Arguments

fit	An mcpfit object.
pars	Character vector. One of: • Vector of parameter names. • "population" plots all population-level parameters. • "group" plots all group-level deviations (random effects). To plot a particular group-level effect, use regex_pars = "^name".

regex_pars	Vector of regular expressions. This will typically just be the beginning of the parameter name(s), i.e., "^cp_" plots all change points, "^my_group_effect" plots all levels of a particular group-level effect, and "^cp_|^my_group_effect" plots both.
type	String or vector of strings. Calls `bayesplot::mcmc_>>type<<()`. Common calls are "combo", "trace", and "dens_overlay". Current options are 'acf', 'acf_bar', 'areas', 'areas_ridges', 'combo', 'dens', 'dens_chains', 'dens_overlay', 'hex', 'hist', 'intervals', 'pairs', 'rank_hist', 'rank_overlay', 'trace', 'trace_highlight', and 'violin'.
ncol	Number of columns in plot. This is useful when you have many parameters and only one plot type.
prior	Logical. Plot prior draws (TRUE) instead of posterior draws (FALSE, default)? Useful for `mcp(..., sample = "both")`. Prior draws show the marginal priors, e.g., of ordered change points, rather than the specified priors in `prior_summary()`.
npars	Positive integer or NULL / Inf. Maximum number of parameters plotted per page. Set to NULL or Inf to plot all parameters on a single page. The default of 5 follows `brms::plot.brmsfit()`.
ask	Logical. In an interactive session, prompt before printing each page after the first. Only used when there are multiple pages.

Details

For other type, it calls `bayesplot::mcmc_type()`. Use these directly on `fit` if you want finer control of plotting, e.g., `bayesplot::mcmc_dens(fit)`. There are also a number of useful plots in the **coda** package, e.g., `coda::gelman.plot(coda::as.mcmc(fit))` and `coda::crosscorr.plot(coda::as.mcmc(fit))`

In any case, if you see a few erratic lines or parameter estimates, this is a sign that you may want to increase argument 'warmup' and 'iter' in `mcp`. Read more.

Up to npars parameters are shown on each page. Multi-page plots are printed in sequence; in interactive use, ask = TRUE pauses between pages.

Value

A **ggplot2** object when all selected parameters fit on one page. For multiple pages, an invisible list of **ggplot2** objects.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

See Also

`plot.mcpfit()`, `plot_dpar()`, `pp_check()`

Examples

```
# Typical usage. demo_fit is an mcpfit object.
plot_pars(demo_fit)
```

```
# More options
plot_pars(demo_fit, regex_pars = "^cp_")  # Plot only change points
plot_pars(demo_fit, pars = c("Intercept_3", "time_3"))  # Plot these parameters
plot_pars(demo_fit, type = c("trace", "violin"), regex_pars = "^cp_")  # Combine plots

# Some plots only take pairs. hex is good to assess identifiability
plot_pars(demo_fit, type = "hex", pars = c("cp_1", "time_2"))

# Visualize the priors:
plot_pars(demo_fit, prior = TRUE, regex_pars = "^cp_")

# Useful for group-level effects:
# plot_pars(my_fit, pars = "group", ncol = 3)  # plot all group-level deviations
# plot_pars(my_fit, regex_pars = "my_group_effect", ncol = 3)  # one group-level effect
# pages = plot_pars(my_fit, pars = "group", ask = FALSE)
# pages[[1]]  # Inspect or customize one page

# Customize multi-column ggplots using "&" instead of "+" (patchwork)
library(ggplot2)
library(patchwork)
plot_pars(demo_fit, regex_pars = "cp") & theme_gray(15)
```

`posterior-uncertainty-mcpfit` *Prior and Posterior Covariance and Central Intervals for* `mcpfit` *Objects*

Description

Summarise the joint and marginal uncertainty of population-level model parameters using posterior or prior draws.

Usage

```
## S3 method for class 'mcpfit'
vcov(object, correlation = FALSE, pars = NULL, dpar = "mu", prior = FALSE, ...)

## S3 method for class 'mcpfit'
confint(object, parm, level = 0.95, prior = FALSE, ...)
```

Arguments

`object`	An `mcpfit` object.
`correlation`	Return the correlation matrix instead of the covariance matrix?
`pars`	Optional names of population-level parameters to extract, or `"population"` for all population-level parameters.

dpar	Distributional parameter(s) to select when pars = NULL, e.g., "mu", "sigma", or "ar1".
prior	Logical. Use prior draws instead of posterior draws?
...	Must be empty. Reserved for future use.
parm	Optional names or positions of population-level parameters to include in the intervals.
level	Width of the central interval.

Details

As in lme4, vcov() covers the coefficients of fixef() (by default those of mu), while confint() covers all population-level parameters, including change points.

Value

vcov() returns a covariance or correlation matrix. confint() returns a two-column matrix of central intervals.

Examples

```
# Posterior covariance of the primary-response coefficients, matching fixef().
vcov(demo_fit)

# Central posterior intervals for all population-level parameters, or a selection.
confint(demo_fit)
confint(demo_fit, parm = "cp_1")
confint(demo_fit, parm = c("Intercept_1", "time_2"), level = 0.8)
confint(demo_fit, prior = TRUE)

# Include change points, residual SDs, group-level SDs, and AR/MA parameters.
vcov(demo_fit, pars = "population")

# Inspect posterior parameter correlations across the full population model.
# Useful to quickly check identifiability (high correlation). Inspecting
# `bayesplot::mcmc_pairs(as_draws(demo_fit))` is better, though.
vcov(demo_fit, pars = "population", correlation = TRUE)
```

---

posterior_epred.mcpfit

Posterior prediction draws for mcpfit *objects*

Description

Methods for the {rstantools} posterior-prediction generics. They return a draws-by-observation matrix and enable {tidybayes} workflows such as add_epred_draws(), add_predicted_draws(), and add_linpred_draws(). mcp exports these generics, so they work without {rstantools}. If {rstantools} or {brms} masks them, their generics dispatch to the same methods.

Usage

```
## S3 method for class 'mcpfit'
posterior_epred(
  object,
  newdata = NULL,
  draws = NULL,
  ndraws = NULL,
  re.form = NULL,
  re_formula = NULL,
  dpar = NULL,
  seed = NULL,
  rate = FALSE,
  ...
)

## S3 method for class 'mcpfit'
posterior_predict(
  object,
  newdata = NULL,
  draws = NULL,
  ndraws = NULL,
  re.form = NULL,
  re_formula = NULL,
  seed = NULL,
  rate = FALSE,
  arma = TRUE,
  ...
)

## S3 method for class 'mcpfit'
posterior_linpred(
  object,
  transform = FALSE,
  newdata = NULL,
  draws = NULL,
  ndraws = NULL,
  re.form = NULL,
  re_formula = NULL,
  dpar = NULL,
  seed = NULL,
  ...
)

posterior_epred(object, ...)

posterior_predict(object, ...)

posterior_linpred(object, ...)
```

Arguments

object	An mcpfit object.
newdata	A tibble or a data.frame containing predictors in the model. • If NULL (default), the original data is used. • Grouping factors must only contain levels from the fitted data. Use group_effects = FALSE to predict without group-level effects, e.g., for new groups. • For models with ar() or ma(): fitted(), residuals(), log_lik(), and predict() condition on the response history by default, so newdata must include the response. For fitted(), predict(), and residuals(), missing response histories are supported only in the original fitted data, using retained posterior imputations as histories. Predictions are fresh response draws, including at missing rows. With arma = "replicate", predict() and posterior_predict() generate fresh response series recursively, so their newdata need only contain predictors and required response auxiliaries. log_lik() is unavailable when a missing response enters a later observed history. • For models with y | weights(): Require the weights column except for fitted() and predict().
draws, ndraws	Number of posterior draws to return. draws follows the {rstantools} convention; ndraws is the mcp spelling. Supply at most one.
re.form, re_formula	Group-level effects to include. NULL includes all effects and NA excludes them.
dpar	Distributional parameter for posterior_epred() and posterior_linpred(); NULL uses the expected response.
seed	Optional integer seed for draw selection and posterior prediction.
rate	Logical scalar. For binomial models, return counts (rate = FALSE, the default for fitted() and predict()) or the observed or expected success proportion (rate = TRUE). Predictions and count-scale fitted values require a trials column in newdata. Distributional parameters such as dpar = "mu" evaluate the parameter itself (e.g., success probability) and are unaffected by rate. Other families do not support rate = TRUE. For rates per unit of exposure in models with offset(log(exposure)), set exposure = 1 in newdata.
...	Must be empty. Reserved for future use.
arma	How to include AR and MA effects. • TRUE: Condition on the observed response history, i.e., one-step-ahead evaluation of the GARMA residual recurrence. Requires the response variable in newdata. • FALSE: Disregard AR and MA effects. For family = gaussian(), predict() uses only sigma for residuals. • "replicate": Generate fresh response series recursively instead of conditioning on the observed responses. Only for predictions: predict(), posterior_predict(), and pp_check() (where it is the default). Applies equally to prior and posterior draws.

For posterior evaluation of the original data, retained JAGS imputations supply missing GARMA histories. In models with group-level effects, this currently requires all such effects (`group_effects = TRUE`).

`transform` For `posterior_linpred()`, return the inverse-linked parameter on the response scale instead of the linear predictor. For binomial models, this returns the success probability rather than expected counts.

Details

For GARMA models, `posterior_predict()` conditions on the observed response history, just like `predict()`. Use `arma = "replicate"` in either method to generate fresh response histories recursively. Missing responses in the original data are filled with retained imputations only to supply histories. With `arma = TRUE`, `posterior_predict()` draws from the response distributions whose means `posterior_epred()` returns, including at missing rows. These methods require posterior draws. For prior prediction, use `predict()` with `prior = TRUE`; `arma` selects the same behavior.

For binomial models, `posterior_epred()` and `posterior_predict()` (and corresponding `{tidybayes}` workflows such as `add_epred_draws()`) follow `{brms}` and `{rstantools}` conventions by returning values on the outcome count scale (`rate = FALSE`), i.e., expected counts $E[Y] = n\mu$ and simulated counts in $\{0, \ldots, n\}$, matching `fitted()` and `predict()`. Use `rate = TRUE` for proportions. To obtain the success probability parameter μ on the $[0, 1]$ scale regardless of trial counts, pass `dpar = "mu"`. In contrast, `posterior_linpred(transform = TRUE)` returns the inverse-linked parameter directly, evaluating success probabilities μ in $[0, 1]$ rather than counts.

Value

A numeric `N_draws` by `nrow(newdata)` matrix.

See Also

`fitted.mcpfit()`, `predict.mcpfit()`

---

pp_check.mcpfit *Posterior Predictive Checks For Mcpfit Objects*

Description

Plot posterior (default) or prior (`prior = TRUE`) predictive checks. This is a convenience wrapper around the `bayesplot::ppc_*()` methods. See prior predictive checks for an example.

Usage

```
## S3 method for class 'mcpfit'
pp_check(
  object,
  type = "dens_overlay",
  facet_by = NULL,
  newdata = NULL,
```

```
  prior = FALSE,
  group_effects = TRUE,
  arma = "replicate",
  ndraws = 100,
  nsamples = lifecycle::deprecated(),
  varying = lifecycle::deprecated(),
  ...
)
```

Arguments

`object`	An `mcpfit` object.
`type`	String specifying the type of plot. See `bayesplot::available_ppc()` for available plot types, omitting the `"ppc_"` prefix (e.g., `"dens_overlay"`, `"ribbon"`, `"intervals"`).
`facet_by`	Name of a grouping column used by group-level effects.
`newdata`	A `tibble` or a `data.frame` containing predictors in the model. • If `NULL` (default), the original data is used. • Grouping factors must only contain levels from the fitted data. Use `group_effects = FALSE` to predict without group-level effects, e.g., for new groups. • For models with `ar()` or `ma()`: `fitted()`, `residuals()`, `log_lik()`, and `predict()` condition on the response history by default, so `newdata` must include the response. For `fitted()`, `predict()`, and `residuals()`, missing response histories are supported only in the original fitted data, using retained posterior imputations as histories. Predictions are fresh response draws, including at missing rows. With `arma = "replicate"`, `predict()` and `posterior_predict()` generate fresh response series recursively, so their `newdata` need only contain predictors and required response auxiliaries. `log_lik()` is unavailable when a missing response enters a later observed history. • For models with `y | weights()`: Require the weights column except for `fitted()` and `predict()`.
`prior`	Logical. Evaluate prior draws (`TRUE`) instead of posterior draws (`FALSE`, default). The selected draws must be available; prior-only fits require `prior = TRUE`.
`group_effects`	Group-level effects to include. One of: • `TRUE` All group-level deviations. • `FALSE` or `NA` No group-level deviations, i.e., population-level predictions. Like `re_formula = NA` in {brms}. • `"cp"` or `"predictor"`: All group-level deviations belonging to that part of the model. • Character vector of group-level parameter names, e.g., `c("cp_1_id", "x_2_id")`. **[Experimental]** `*` is a wildcard, e.g., `"state*_1_id"` selects all levels of a factor and `"*_id"` selects all effects by `id`. See names using `ranef(fit)`. Can be combined, e.g., `c("cp", "sigma_*")`.
`arma`	How to include AR and MA effects.

- TRUE: Condition on the observed response history, i.e., one-step-ahead evaluation of the GARMA residual recurrence. Requires the response variable in newdata.
- FALSE: Disregard AR and MA effects. For family = gaussian(), predict() uses only sigma for residuals.
- "replicate": Generate fresh response series recursively instead of conditioning on the observed responses. Only for predictions: predict(), posterior_predict(), and pp_check() (where it is the default). Applies equally to prior and posterior draws.

For posterior evaluation of the original data, retained JAGS imputations supply missing GARMA histories. In models with group-level effects, this currently requires all such effects (group_effects = TRUE).

ndraws	Integer or NULL. Number of posterior draws to return/summarise. NULL means "all". More draws trade speed for accuracy.
nsamples	Deprecated. Use ndraws instead.
varying	Deprecated. Use group_effects instead.
...	Further arguments passed to bayesplot::ppc_type(y, yrep, ...)

Details

Missing responses are omitted from the observed-data check. LOO predictive checks use posterior draws and the original fitted data, so they do not support prior = TRUE or newdata. For GARMA models, they default to arma = TRUE to match the one-step-ahead log-likelihood used for the LOO weights.

Value

A ggplot2 object for single plots. Enriched by patchwork for faceted plots.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

See Also

plot.mcpfit(), plot_pars(), predict.mcpfit()

Examples

```
pp_check(demo_fit)
pp_check(demo_fit, type = "ecdf_overlay")
#pp_check(some_group_fit, type = "loo_intervals", facet_by = "id")
```

print.mcplist *Print mcplist*

Description

Shows a list in a more condensed format using `str(list)`.

Usage

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

Arguments

`x`	An `mcpfit` object, or an `mcpsummary` returned by `summary()`.
`...`	Must be empty. Reserved for future use.

Value

`x`, invisibly.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
print(demo_fit$model)
print(demo_fit$prior)
```

print.mcptext *Nice Printing of Multiline Texts*

Description

Useful for `print(fit$jags_code)`, `print(mcp_demo$example_code)`, etc.

Usage

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

Arguments

`x`	Character, often with newlines.
`...`	Must be empty. Reserved for future use.

Value

x, invisibly.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
mytext = "line1 = 2\n line2 = 'horse'"
class(mytext) = "mcptext"
print(mytext)
```

---

prior_summary *Summarise priors used by an mcp model*

Description

[Experimental]

Usage

```
prior_summary(object, ...)

## S3 method for class 'mcpfit'
prior_summary(object, verbose = FALSE, ...)
```

Arguments

object	An mcpfit object.
...	Must be empty. Reserved for future use.
verbose	Logical. Include rule, description, source, and kind.

Details

Shows the specified priors in the syntax of mcp(..., prior), resolved to numbers. As in prior, scales are SD/scale rather than JAGS precision (see mcp-priors). bounds are the lower and upper bounds of each prior, i.e., the arguments of dunif() or T(), or the support of the distribution. Change points are also "ordered": their specified priors are restricted to ordered change points. Use plot_pars(fit, prior = TRUE) to see the marginal priors after this restriction. verbose = TRUE also displays the underlying rule, description, source, and kind. Numbers are rounded to 5 significant digits for display. fit$prior holds the exact values used in the model. Parameter names within a prior describe its dependence on other parameters, e.g., the default prior on Intercept_1 given x_1.

prior_summary() shares its generic with **rstantools** (used by brms and rstanarm), so it works for all of these models regardless of package load order.

Value

A tibble with one row per model parameter, ordered and labeled the same way as `summary()`: change points first, then `mu`, then the other distributional parameters, then AR/MA lags - each with `segment` and `dpar` columns.

See Also

mcp-priors for how to specify priors, and Default priors for the rules behind them.

Examples

```
prior_summary(demo_fit)  # Show the resolved priors and their bounds
prior_summary(demo_fit, verbose = TRUE)  # Include their rules and sources
```

---

sd_to_prec *Transform an mcp prior to the parameterization used by JAGS*

Description

[Deprecated]

Usage

```
sd_to_prec(prior_str)
```

Arguments

`prior_str`	String. A JAGS prior. Can be truncated, e.g. `dt(3, 2, 1) T(my_var, )`.

Details

This function is deprecated. JAGS uses precision rather than SD for some distributions. For example, this function converts `dnorm(4.2, 1.3)` into `dnorm(4.2, 1/1.3^2)`. It allows users to specify priors using conventional scale parameters before they are translated to JAGS code. Users normally do not need to call this function.

Value

A string

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

summary.mcpfit *Summarise mcpfit objects*

Description

Summarise parameter estimates and model diagnostics.

Usage

```
## S3 method for class 'mcpfit'
summary(
  object,
  width = 0.95,
  digits = 2,
  prior = FALSE,
  verbose = FALSE,
  diagnostics = NULL,
  ...
)

## S3 method for class 'mcpsummary'
print(x, digits = attr(x, "mcpsummary")$digits, ...)

## S3 method for class 'mcpfit'
fixef(object, width = 0.95, prior = FALSE, verbose = FALSE, dpar = "mu", ...)

## S3 method for class 'mcpfit'
ranef(object, width = 0.95, prior = FALSE, verbose = FALSE, ...)

## S3 method for class 'mcpfit'
coef(object, width = 0.95, prior = FALSE, verbose = FALSE, ...)

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

Arguments

object	An mcpfit object.
width	Float. The width of the central posterior interval (between 0 and 1).
digits	Positive integer. Number of significant digits of sd when printing the summary. Defaults to 2. mean, lower, upper, and sim are rounded to the same decimal places in each row. The returned tibble retains the unrounded values.
prior	Logical. Summarise prior draws (TRUE) instead of posterior draws (FALSE, default)?
verbose	Logical. Include the segment and dpar columns. Defaults to FALSE for a compact summary.

diagnostics	Named list of diagnostic warning thresholds. Available elements are rhat = 1.01, ess_bulk = 400, ess_tail = 400, ar = 0.10, and ma = 0.10. An empty list uses these defaults; a partial list overrides only the supplied values. Set an element to NULL to disable that diagnostic, or use FALSE to disable all configurable diagnostic warnings. In summary.mcpfit(), NULL inherits the settings used to fit the model, while a list or FALSE overrides the diagnostic footer.
...	Must be empty. Reserved for future use.
x	An mcpfit object, or an mcpsummary returned by summary().
dpar	Distributional parameter(s) whose regression coefficients to return, e.g., "mu", "sigma", or "ar1" (see mcp-formula). For modeled distributional parameters such as sigma(), these coefficients are on the link scale.

Value

An mcpsummary tibble with parameter estimates and MCMC diagnostics. It prints with model information, separate tables for change points and population-level parameters, and convergence warnings. Rows are ordered by change point first, then mu, then the other distributional parameters, then AR/MA lags - each ascending by segment. Note: The change point distributions are often not unimodal and symmetric so the intervals can be deceiving. Plot them using plot_pars(fit).

Columns:

- segment (only with verbose = TRUE) is the segment the parameter belongs to. For change points, it is the segment that the change point starts, e.g., 2 for cp_1.
- dpar (only with verbose = TRUE) is the distributional parameter (e.g., "mu", "sigma", or "ar1"; see mcp-formula) as accepted by the dpar argument of fixef.mcpfit(), fitted.mcpfit(), etc. It is NA for change points. See also mcp_pars().
- mean is the posterior mean
- sd is the posterior standard deviation.
- lower and upper are the lower and upper ends of the central posterior interval given in width.
- rhat is the rank-normalized split-Rhat convergence diagnostic.
- ess_bulk and ess_tail are the bulk and tail effective sample sizes. Low effective sample sizes are also obvious as poor mixing in trace plots (see plot_pars(fit)). Read how to deal with such problems in Diagnosing and fixing problems.

Group-level change-point deviations (cp_i_id) follow a hierarchical normal distribution around the population-level change point before truncation. The group-specific change points are truncated to the observed range of x and ordered, so the population-level change point is generally close to their mean, but not exactly. Predictor group-level deviations (such as Intercept_1_id) use standard hierarchical zero-mean priors, without these bounds. Read more.

For simulated data, the summary contains two additional columns so that it is easy to inspect whether the model can recover the parameters. They are shown when the response was simulated using fit$simulate() from a model with the same parameters, family, and link. Run simulation and summary multiple times to get a sense of the robustness.

- sim is the value used to generate the data.

- `match` is `"OK"` if `sim` is contained in the central posterior interval (`lower` to `upper`). For change points starting a disjoined segment, `sim` only needs to be between the same two adjacent observed `x` as the interval, since the likelihood is identical anywhere between them. Group-level deviations in `ranef()` are relative to their population-level parameter, so they share its error; `coef()` checks the group-specific values directly.

Functions

- `print(mcpsummary)`: Print an `mcpsummary` returned by `summary()`: model information, change point and population-level parameters, and diagnostics.
- `fixef(mcpfit)`: Population-level coefficients (fixed effects) of `mcpfit`.
- `ranef(mcpfit)`: Group-level deviations (random effects) of `mcpfit`. Change-point deviations are relative to their population-level change point; `cp_i_<group>_sd` (e.g., `cp_1_id_sd`) is the SD of their normal distribution before truncation. Use `coef()` for group-specific values.
- `coef(mcpfit)`: Group-specific coefficients and change points of `mcpfit`: the population-level parameter plus each group-level deviation, as in `lme4` and `brms`. For example, `cp_1` for `group_col = "id"` and `level = "a"` summarises the draws of `cp_1 + cp_1_id[a]`, which is the change point of group `a`. Deviations without a population-level parameter, e.g., from `(0 + x || id)`, are added to zero. Coefficients of distributional parameters such as `sigma()` are on the link scale. For simulated data, `sim` is the sum of the simulated values.
- `print(mcpfit)`: Print the posterior summary of an `mcpfit` object, or the prior summary with `prior = TRUE`. Without the requested draws, only the model is shown.

Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

Examples

```
# Typical usage
summary(demo_fit)
summary(demo_fit, width = 0.8, digits = 4)  # Set interval width

# Get the results as a tibble
results = summary(demo_fit)
results$mean

# Group-level deviations (random effects)
# ranef(my_fit)

# Summarise prior
summary(demo_fit, prior = TRUE)
```

Index

∗ **datasets**
demo_fit, 6

as.array.mcpfit *(as_draws.mcpfit)*, 4
as.data.frame(), *5*
as.data.frame.mcpfit *(as_draws.mcpfit)*, 4
as.matrix.mcpfit *(as_draws.mcpfit)*, 4
as.mcmc.mcpfit *(as_draws.mcpfit)*, 4
as_draws *(as_draws.mcpfit)*, 4
as_draws.mcpfit, 4
as_draws_array *(as_draws.mcpfit)*, 4
as_draws_df *(as_draws.mcpfit)*, 4
as_draws_matrix *(as_draws.mcpfit)*, 4
as_draws_rvars *(as_draws.mcpfit)*, 4

bayesplot::available_ppc(), *46*
bernoulli, 6

coef.mcpfit *(summary.mcpfit)*, 51
confint.mcpfit *(posterior-uncertainty-mcpfit)*, 41

demo_fit, 6
draws-index-mcp, 7

execute-mcp-model, 8

family.mcpfit *(model-accessors-mcpfit)*, 34
fitted *(execute-mcp-model)*, 8
fitted.mcpfit, *13*
fitted.mcpfit(), *33*, *45*, *52*
fixef *(summary.mcpfit)*, 51
fixef(), *42*
fixef.mcpfit(), *33*, *52*
formula.mcpfit *(model-accessors-mcpfit)*, 34
future, *22*

hypothesis, 14

interpolate_newdata, 16
is.mcpfit, 18
is_mcpfit *(is.mcpfit)*, 18

log_lik *(execute-mcp-model)*, 8
log_lik.mcpfit, *13*
LOO *(loo.mcpfit)*, 18
loo, *18, 19*
loo *(loo.mcpfit)*, 18
loo.mcpfit, 18
loo_compare, *18*
loo_compare *(loo.mcpfit)*, 18
loo_model_weights, *18*

mcmc.list, *33*
mcp, *4*, 20, *33*, *40*
mcp(), *3, 4*, *25*, *28–30*
mcp-formula, *3*, *20*, *23*, 25, *28, 29*, *33*, *52*
mcp-package, 3, *21*
mcp-priors, *3*, *21*, *23*, *25*, 28, *28*, *49, 50*
mcp_columns, 30
mcp_example, *6*, 31
mcp_example_data *(mcp_example)*, 31
mcp_formula *(mcp-formula)*, 25
mcp_pars, 32
mcp_pars(), *21*, *28*, *52*
mcp_priors *(mcp-priors)*, 28
mcpfit, *5, 6*, *14*, *19*, *23*, *36*, *39*, *48*, *51–53*
mcpfit *(mcpfit-class)*, 33
mcpfit-class, 33
model-accessors-mcpfit, 34
model.frame.mcpfit *(model-accessors-mcpfit)*, 34

nchains *(draws-index-mcp)*, 7
ndraws *(draws-index-mcp)*, 7
negbinomial, 35
niterations *(draws-index-mcp)*, 7

nobs.mcpfit *(*model-accessors-mcpfit*)*, 34

package_version, *33*
plot *(*plot.mcpfit*)*, 35
plot.mcpfit, 35
plot.mcpfit(), *40*, *47*
plot_dpar *(*plot.mcpfit*)*, 35
plot_dpar(), *40*
plot_pars, 39
plot_pars(), *38*, *47*
posterior-uncertainty-mcpfit, 41
posterior_epred
        *(*posterior_epred.mcpfit*)*, 42
posterior_epred.mcpfit, 42
posterior_linpred
        *(*posterior_epred.mcpfit*)*, 42
posterior_predict
        *(*posterior_epred.mcpfit*)*, 42
posterior_predict(), *10*, *44*, *46*
pp_check *(*pp_check.mcpfit*)*, 45
pp_check(), *38*, *40*
pp_check.mcpfit, 45
predict *(*execute-mcp-model*)*, 8
predict.mcpfit, *13*
predict.mcpfit(), *16*, *45*, *47*
print *(*summary.mcpfit*)*, 51
print.mcplist, 48
print.mcptext, 48
prior_summary, 49
prior_summary(), *21*, *28*, *29*, *40*

ranef *(*summary.mcpfit*)*, 51
residuals *(*execute-mcp-model*)*, 8
residuals.mcpfit, *13*

sd_to_prec, 50
stats::dlnorm(), *28*
summary *(*summary.mcpfit*)*, 51
summary.mcpfit, 51
summary.mcpfit(), *22*, *52*

tidy_draws.mcpfit *(*as_draws.mcpfit*)*, 4

variables *(*draws-index-mcp*)*, 7
vcov.mcpfit
        *(*posterior-uncertainty-mcpfit*)*,
        41

WAIC *(*loo.mcpfit*)*, 18
waic *(*loo.mcpfit*)*, 18
waic.mcpfit *(*loo.mcpfit*)*, 18