

# Package ‘mcp’

March 17, 2024

**Title** Regression with Multiple Change Points

**Version** 0.3.4

**Date** 2024-03-14

**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 prediction intervals are supported - also near the change points. 'mcp' supports hypothesis testing via Savage-Dickey density ratio, posterior contrasts, and cross-validation. 'mcp' is described in Lindeløv (submitted) <[doi:10.31219/osf.io/fzqxv](https://doi.org/10.31219/osf.io/fzqxv)> and generalizes the approach described in Carlin, Gelfand, & Smith (1992) <[doi:10.2307/2347570](https://doi.org/10.2307/2347570)> and Stephens (1994) <[doi:10.2307/2986119](https://doi.org/10.2307/2986119)>.

**License** GPL-2

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**RoxygenNote** 7.3.1

**Depends** R (>= 3.5.0)

**Imports** parallel, future (>= 1.16), future.apply (>= 1.4), rjags (>= 4.9), coda (>= 0.19.3), loo (>= 2.1.0), bayesplot (>= 1.7.0), tidybayes (>= 3.0.0), dplyr (>= 1.1.1), magrittr (>= 1.5), tidyr (>= 1.0.0), tidyselect (>= 0.2.5), tibble (>= 2.1.3), stringr (>= 1.4.0), ggplot2 (>= 3.2.1), patchwork (>= 1.0.0), stats, rlang (>= 0.4.1)

**Suggests** hexbin, testthat (>= 3.1.0), purrr (>= 0.3.4), knitr, rmarkdown

**NeedsCompilation** no

**Author** Jonas Kristoffer Lindeløv [aut, cre]

(<<https://orcid.org/0000-0003-4565-0595>>)

**Maintainer** Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

**Repository** CRAN

**Date/Publication** 2024-03-17 20:10:02 UTC

## R topics documented:

bernoulli . . . . .	2
criterion . . . . .	3
demo_fit . . . . .	4
exponential . . . . .	4
fitted.mcpfit . . . . .	5
hypothesis . . . . .	7
ilogit . . . . .	8
is.mcpfit . . . . .	9
logit . . . . .	9
mcp . . . . .	10
mcpfit-class . . . . .	13
mcp_example . . . . .	14
negbinomial . . . . .	15
phi . . . . .	16
plot.mcpfit . . . . .	16
plot_pars . . . . .	18
pp_check . . . . .	20
predict.mcpfit . . . . .	21
print.mcplist . . . . .	24
print.mcptext . . . . .	24
probit . . . . .	25
residuals.mcpfit . . . . .	25
sd_to_prec . . . . .	27
summary.mcpfit . . . . .	27
<b>Index</b>	<b>30</b>

---

bernoulli	<i>Bernoulli family for mcp</i>
-----------	---------------------------------

---

### Description

Bernoulli family for mcp

### Usage

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

### Arguments

link	Link function.
------	----------------

---

criterion	<i>Compute information criteria for model comparison</i>
-----------	--

---

## Description

Takes an `mcpfit` as input and computes information criteria using loo or WAIC. Compare models using `loo_compare` and `loo_model_weights`. more in `loo`.

## Usage

```
criterion(fit, criterion = "loo", ...)  
  
## S3 method for class 'mcpfit'  
loo(x, ...)  
  
## S3 method for class 'mcpfit'  
waic(x, ...)
```

## Arguments

<code>fit</code>	An <code>mcpfit</code> object.
<code>criterion</code>	One of "loo" (calls <code>loo</code> ) or "waic" (calls <code>waic</code> ).
<code>...</code>	Currently ignored
<code>x</code>	An <code>mcpfit</code> object.

## Value

a loo or `psis_loo` object.

## Functions

- `loo(mcpfit)`: Computes loo on `mcpfit` objects
- `waic(mcpfit)`: Computes WAIC on `mcpfit` objects

## Author(s)

Jonas Kristoffer Lindeløv <jonas@lindelov.dk>

## See Also

[criterion](#)  
[criterion](#)

**Examples**

```
# Define two models and sample them
# options(mc.cores = 3) # Speed up sampling
ex = mcp_example("intercepts") # Get some simulated data.
model1 = list(y ~ 1 + x, ~ 1)
model2 = list(y ~ 1 + x) # Without a change point
fit1 = mcp(model1, ex$data)
fit2 = mcp(model2, ex$data)

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

---

demo\_fit

*Example mcpfit for examples*


---

**Description**

This was generated using `mcp_examples("demo", sample = TRUE)`.

**Usage**

```
demo_fit
```

**Format**

An `mcpfit` object.

---

exponential

*Exponential family for mcp*


---

**Description**

Exponential family for mcp

**Usage**

```
exponential(link = "identity")
```

**Arguments**

link                    Link function (Character).

fitted.mcpfit

*Expected Values from the Posterior Predictive Distribution***Description**

Expected Values from the Posterior Predictive Distribution

**Usage**

```
## S3 method for class 'mcpfit'
fitted(
  object,
  newdata = NULL,
  summary = TRUE,
  probs = TRUE,
  rate = TRUE,
  prior = FALSE,
  which_y = "ct",
  varying = TRUE,
  arma = TRUE,
  nsamples = NULL,
  samples_format = "tidy",
  scale = "response",
  ...
)
```

**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.
summary	Summarise at each x-value
probs	Vector of quantiles. Only in effect when summary == TRUE.
rate	Boolean. For binomial models, plot on raw data (rate = FALSE) or response divided by number of trials (rate = TRUE). If FALSE, linear interpolation on trial number is used to infer trials at a particular x.
prior	TRUE/FALSE. Plot using prior samples? Useful for mcp(..., sample = "both")
which_y	What to plot on the y-axis. One of <ul style="list-style-type: none"> <li>"ct": The central tendency which is often the mean after applying the link function.</li> <li>"sigma": The variance</li> <li>"ar1", "ar2", etc. depending on which order of the autoregressive effects you want to plot.</li> </ul>
varying	One of:

	<ul style="list-style-type: none"> <li>• TRUE All varying effects (<code>fit\$pars\$varying</code>).</li> <li>• FALSE No varying effects (<code>c()</code>).</li> <li>• Character vector: Only include specified varying parameters - see <code>fit\$pars\$varying</code>.</li> </ul>
<code>arma</code>	Whether to include autoregressive effects. <ul style="list-style-type: none"> <li>• TRUE Compute autoregressive residuals. Requires the response variable in <code>newdata</code>.</li> <li>• FALSE Disregard the autoregressive effects. For <code>family = gaussian()</code>, <code>predict()</code> just use <code>sigma</code> for residuals.</li> </ul>
<code>nsamples</code>	Integer or NULL. Number of samples to return/summarise. If there are varying effects, this is the number of samples from each varying group. NULL means "all". Ignored if both are FALSE. More samples trade speed for accuracy.
<code>samples_format</code>	One of "tidy" or "matrix". Controls the output format when <code>summary == FALSE</code> . See more under "value"
<code>scale</code>	One of <ul style="list-style-type: none"> <li>• "response": return on the observed scale, i.e., after applying the inverse link function.</li> <li>• "linear": return on the parameter scale (where the linear trends are modelled).</li> </ul>
<code>...</code>	Currently unused

### Value

- If `summary = TRUE`: A tibble with the posterior mean for each row in `newdata`, If `newdata` is NULL, the data in `fit$data` is used.
- If `summary = FALSE` and `samples_format = "tidy"`: A tidybayes tibble with all the posterior samples (`Ns`) evaluated at each row in `newdata` (`Nn`), i.e., with `Ns x Nn` rows. If there are varying effects, the returned data is expanded with the relevant levels for each row. The return columns are:
  - Predictors from `newdata`.
  - Sample descriptors: ".chain", ".iter", ".draw" (see the tidybayes package for more), and "data\_row" (`newdata` rownumber)
  - Sample values: one column for each parameter in the model.
  - The estimate. Either "predict" or "fitted", i.e., the name of the `type` argument.
- If `summary = FALSE` and `samples_format = "matrix"`: An `N_draws X n_rows(newdata)` matrix with fitted/predicted values (depending on type). This format is used by `brms` and it's useful as `yrep` in `bayesplot::ppc_*` functions.

### Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

### See Also

[pp\\_eval](#) [predict.mcpfit](#) [residuals.mcpfit](#)

## Examples

```
fitted(demo_fit)
fitted(demo_fit, probs = c(0.1, 0.5, 0.9)) # With median and 80% credible interval.
fitted(demo_fit, summary = FALSE) # Samples instead of summary.
fitted(demo_fit,
       newdata = data.frame(time = c(-5, 20, 300)), # New data
       probs = c(0.025, 0.5, 0.975))
```

---

hypothesis	<i>Test hypotheses on mcp objects.</i>
------------	--

---

## 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` bayes`` environment and summarises the proportion of samples where the expression evaluates to TRUE. For equals-hypothesis, a Savage-Dickey ratio is computed. Savage-Dickey requires a prior too, so remember `mcp(..., sample = "both")`. This function is heavily inspired by the `'hypothesis'` function from the `'brms'` package.

## Usage

```
hypothesis(fit, hypotheses, width = 0.95, digits = 3)
```

## Arguments

<code>fit</code>	An <code>mcpfit</code> object.
<code>hypotheses</code>	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 <code>&lt;</code> , <code>&gt;</code> , <code>&lt;=</code> , or <code>&gt;=</code> . <code>hypothesis</code> returns the posterior probability and odds in favor of the stated hypothesis. The odds can be interpreted as a Bayes Factor. For example: <ul style="list-style-type: none"> <li>• <code>"cp_1 &gt; 30"</code>: the first change point is above 30.</li> <li>• <code>"int_1 &gt; int_2"</code>: the intercept is greater in segment 1 than 2.</li> <li>• <code>"x_2 - x_1 &lt;= 3"</code>: the difference between slope 1 and 2 is less than or equal to 3.</li> <li>• <code>"int_1 &gt; -2 &amp; int_1 &lt; 2"</code>: <code>int_1</code> is between -2 and 2 (an interval hypothesis). This can be useful as a Region Of Practical Equivalence test (ROPE).</li> <li>• <code>"cp_1^2 &lt; 30   (log(x_1) + log(x_2)) &gt; 5"</code>: be creative.</li> <li>• <code>"`cp_1_id[1]` &gt; `cp_1_id[2]`"</code>: <code>id1</code> is greater than <code>id2</code>, as estimated through the varying-by-"id" change point in segment 1. Note that `` required for varying effects.</li> </ul>

Hypotheses can also test equality using the equal sign (=). This runs a Savage-Dickey test, i.e., the proportion by which the probability density has increased from the prior to the posterior at a given value. Therefore, it requires `mcp(sample = "both")`. There are two requirements: First, there can only be one equal sign, so don't use and (&) or or (!). Second, the point to test has to be on the right, and the variables on the left.

- `"cp_1 = 30"`: is the first change point at 30? Or to be more precise: by what factor has the credence in `cp_1 = 30` risen/fallen when conditioning on the data, relative to the prior credence?
- `"int_1 + int_2 = 0"`: Is the sum of two intercepts zero?
- `"`cp_1_id[John]`/`cp_1_id[Erin]` = 2"`: is the varying change point for John (which is relative to `'cp_1'`) double that of Erin?

`width` Float. The width of the highest posterior density interval (between 0 and 1).  
`digits` a non-null value for `digits` specifies the minimum number of significant digits to be printed in values. The default, `NULL`, uses `getOption("digits")`. (For the interpretation for complex numbers see `signif`.) Non-integer values will be rounded down, and only values greater than or equal to 1 and no greater than 22 are accepted.

### Value

A data.frame 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.
- `lower` is the lower bound of the (two-sided) highest-density interval of width `width`.
- `upper` is the upper bound of ditto.
- `p` Posterior probability. For `"="` (Savage-Dickey), it is the BF converted to p. For directional hypotheses, it is the proportion of samples that returns `TRUE`.
- `BF` Bayes Factor in favor of the hypothesis. For `"="` it is the Savage-Dickey density ratio. For directional hypotheses, it is p converted to odds.

### Author(s)

Jonas Kristoffer Lindeløv <jonas@lindelov.dk>

---

ilogit

*Inverse logit function*

---

### Description

Inverse logit function

### Usage

`ilogit(eta)`



**Arguments**

eta                    A vector of logits

**Value**

A vector with same length as eta

---

*is.mcpfit*                    *Checks if argument is an mcpfit object*

---

**Description**

Checks if argument is an mcpfit object

**Usage**

`is.mcpfit(x)`

**Arguments**

x                    An R object.

---

*logit*                    *Logit function*

---

**Description**

Logit function

**Usage**

`logit(mu)`

**Arguments**

mu                    A vector of probabilities (0.0 to 1.0)

**Value**

A vector with same length as mu

## 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 more details and worked examples on the mcp website](#). All segments must regress on the same x-variable. Change points are forced to be ordered using truncation of the priors. You can run `fit = mcp(model, sample=FALSE)` to avoid sampling and the need for data if you just want to get the priors (`fit$prior`), the JAGS code `fit$jags_code`, or the R function to simulate data (`fit$simulate`).

## Usage

```
mcp(
  model,
  data = NULL,
  prior = list(),
  family = gaussian(),
  par_x = NULL,
  sample = "post",
  cores = 1,
  chains = 3,
  iter = 3000,
  adapt = 1500,
  inits = NULL,
  jags_code = NULL
)
```

## Arguments

<code>model</code>	<p>A list of formulas - one for each segment. The first formula has the format <code>response ~ predictors</code> while the following formulas have the format <code>response ~ changepoint ~ predictors</code>. The response and change points can be omitted (<code>changepoint ~ predictors</code> assumes same response. <code>~ predictors</code> assumes an intercept-only change point). The following can be modeled:</p> <ul style="list-style-type: none"> <li>• <i>Regular formulas</i>: e.g., <code>~ 1 + x</code>. <a href="#">Read more</a>.</li> <li>• <i>Extended formulas</i>: e.g., <code>~ I(x^2) + exp(x) + sin(x)</code>. <a href="#">Read more</a>.</li> <li>• <i>Variance</i>: e.g., <code>~sigma(1)</code> for a simple variance change or <code>~sigma(rel(1) + I(x^2))</code> for more advanced variance structures. <a href="#">Read more</a></li> <li>• <i>Autoregression</i>: e.g., <code>~ar(1)</code> for a simple onset/change in AR(1) or <code>ar(2, 0 + x)</code> for an AR(2) increasing by x. <a href="#">Read more</a></li> </ul>
<code>data</code>	Data.frame or tibble in long format.
<code>prior</code>	Named list. Names are parameter names ( <code>cp_i</code> , <code>int_i</code> , <code>xvar_i</code> , 'sigma') and the values are either

	<ul style="list-style-type: none"> <li>• A JAGS distribution (e.g., <code>int_1 = "dnorm(0, 1) T(0,)"</code>) indicating a conventional prior distribution. Uninformative priors based on data properties are used where priors are not specified. This ensures good parameter estimations, but it is a questionable for hypothesis testing. <code>mcp</code> uses SD (not precision) for <code>dnorm</code>, <code>dt</code>, <code>dlogis</code>, etc. See details. Change points are forced to be ordered through the priors using truncation, except for uniform priors where the lower bound should be greater than the previous change point, <code>dunif(cp_1, MAXX)</code>.</li> <li>• A numerical value (e.g., <code>int_1 = -2.1</code>) indicating a fixed value.</li> <li>• A model parameter name (e.g., <code>int_2 = "int_1"</code>), indicating that this parameter is shared - typically between segments. If two varying effects are shared this way, they will need to have the same grouping variable.</li> <li>• A scaled Dirichlet prior is supported for change points if they are all set to <code>cp_i = "dirichlet(N)</code> where <code>N</code> is the alpha for this change point and <code>N = 1</code> is most often used. This prior is less informative about the location of the change points than the default uniform prior, but it samples less efficiently, so you will often need to set <code>iter</code> higher. It is recommended for hypothesis testing and for the estimation of more than 5 change points. <a href="#">Read more</a>.</li> </ul>
family	One of <code>gaussian()</code> , <code>binomial()</code> , <code>bernoulli()</code> , or <code>poission()</code> . Only default link functions are currently supported.
par_x	String (default: <code>NULL</code> ). Only relevant if no segments contains slope (no hint at what <code>x</code> is). Set this, e.g., <code>par_x = "time"</code> .
sample	One of <ul style="list-style-type: none"> <li>• <code>"post"</code>: Sample the posterior.</li> <li>• <code>"prior"</code>: Sample only the prior. Plots, summaries, etc. will use the prior. This is useful for prior predictive checks.</li> <li>• <code>"both"</code>: Sample both prior and posterior. Plots, summaries, etc. will default to using the posterior. The prior only has effect when doing Savage-Dickey density ratios in <a href="#">hypothesis</a>.</li> <li>• <code>"none"</code> or <code>FALSE</code>: Do not sample. Returns an <code>mcpfit</code> object without sample. This is useful if you only want to check prior strings (<code>fit\$prior</code>), the JAGS model (<code>fit\$jags_code</code>), etc.</li> </ul>
cores	Positive integer or <code>"all"</code> . Number of cores. <ul style="list-style-type: none"> <li>• <code>1</code>: serial sampling. <code>options(mc.cores = 3)</code> will dominate <code>cores = 1</code> but not larger values of <code>cores</code>.</li> <li>• <code>&gt;1</code>: parallel sampling on this number of cores. Ideally set <code>chains</code> to the same value. Note: <code>cores &gt; 1</code> takes a few extra seconds the first time it's called but subsequent calls will start sampling immediately.</li> <li>• <code>"all"</code>: use all cores but one and sets <code>chains</code> to the same value. This is a convenient way to maximally use your computer's power.</li> </ul>
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 <code>iter * chains</code> .
adapt	Positive integer. Also sometimes called <code>"burnin"</code> , this is the number of samples used to reach convergence. Set lower for greater speed. Set higher if the chains haven't converged yet or look at <a href="#">tips, tricks, and debugging</a> .

<code>inits</code>	A list of initial values for the parameters. This can be useful if a model fails to converge. Read more in <a href="#">jags.model</a> . Defaults to NULL, i.e., no inits.
<code>jags_code</code>	String. Pass JAGS code to <code>mcp</code> to use directly. This is useful if you want to tweak the code in <code>fit\$jags_code</code> and run it within the <code>mcp</code> framework.

## Details

Notes on priors:

- Order restriction is automatically applied to `cp_*` parameters using truncation (e.g., `T(cp_1, )`) so that they are in the correct order on the x-axis UNLESS you do it yourself. The one exception is for `dunif` distributions where you have to do it as above.
- In addition to the model parameters, `MINX` (minimum x-value), `MAXX` (maximum x-value), `SDX` (etc...), `MINY`, `MAXY`, and `SDY` are also available when you set priors. They are used to set uninformative default priors.
- Use `SD` when you specify priors for `dt`, `dlogis`, etc. JAGS uses precision but `mcp` converts to precision under the hood via the `sd_to_prec()` function. So you will see `SDs` in `fit$prior` but precision ( $1/SD^2$ ) in `fit$jags_code`

## Value

An `mcpfit` object.

## Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

## See Also

[get\\_segment\\_table](#)

## Examples

```
# Define the segments using formulas. A change point is estimated between each formula.
model = list(
  response ~ 1, # Plateau in the first segment (int_1)
  ~ 0 + time,  # Joined slope (time_2) at cp_1
  ~ 1 + time   # Disjoined slope (int_3, time_3) at cp_2
)

# Fit it and sample the prior too.
# options(mc.cores = 3) # Uncomment to speed up sampling
ex = mcp_example("demo") # Simulated data example
demo_fit = mcp(model, data = ex$data, sample = "both")

# 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 > 10")

# Make predictions
fitted(demo_fit)
predict(demo_fit)
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 = ex$data, par_x = "time") # fit another model here
demo_fit$loo = loo(demo_fit)
fit_null$loo = loo(fit_null)
loo::loo_compare(demo_fit$loo, fit_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
print(demo_fit$prior)

# Set priors and re-run
prior = list(
  int_1 = 15,
  time_2 = "dt(0, 2, 1) T(0, )", # t-dist slope. Truncated to positive.
  cp_2 = "dunif(cp_1, 80)", # change point to segment 2 > cp_1 and < 80.
  int_3 = "int_1" # Shared intercept between segment 1 and 3
)

fit3 = mcp(model, data = ex$data, prior = prior)

# Show the JAGS model
demo_fit$jags_code

```

---

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 samples).

**Details**

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

User-provided information (see `mcp` for more details):

**Slots**

`model` A list of formulas, making up the model. Provided by user. See `mcp` for more details.

`data` A data frame. Provided by user. See `mcp` for more details.

`family` An `mcpfamily` object. Provided by user. See `mcp` for more details.

`prior` A named list. Provided by user. See `mcp` for more details.

`mcmc_post` An `mcmc.list` object with posterior samples.

`mcmc_prior` An `mcmc.list` object with prior samples.

`mcmc_loglik` An `mcmc.list` object with samples of log-likelihood.

`pars` A list of character vectors of model parameter names.

`jags_code` A string with jags code. Use `cat(fit$jags_code)` to show it.

`simulate` A method to simulate and predict data.

`.other` Information that is used internally by `mcp`.

---

`mcp_example`

*Get example models and data*

---

**Description**

Get example models and data

**Usage**

```
mcp_example(name, sample = FALSE)
```

**Arguments**

<code>name</code>	<p>Name of the example. One of:</p> <ul style="list-style-type: none"> <li>• <code>"demo"</code>: Two change points between intercepts and joined/disjoined slopes.</li> <li>• <code>"ar"</code>: One change point in autoregressive residuals.</li> <li>• <code>"binomial"</code>: Binomial with two change points. Much like <code>"demo"</code> on a logit scale.</li> <li>• <code>"intercepts"</code>: An intercept-only change point.</li> <li>• <code>rel_prior</code>: Relative parameterization and informative priors.</li> <li>• <code>"quadratic"</code>: A change point to a quadratic segment.</li> <li>• <code>"trigonometric"</code>: Trigonometric/seasonal data and model.</li> <li>• <code>"varying"</code>: Varying / hierarchical change points.</li> <li>• <code>"variance"</code>: A change in variance, including a variance slope.</li> </ul>
<code>sample</code>	TRUE (run <code>fit = mcp(model, data, ...)</code> ) or FALSE.

**Value**

List with

- `model`: A list of formulas
- `data`: The simulated data
- `simulated`: The parameters used for simulating the data.
- `fit`: an `mcpfit` if `sample = TRUE`,
- `call`: the code to run the above.

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

**Examples**

```
ex = mcp_example("demo")
plot(ex$data) # Plot data
print(ex$simulated) # See true parameters used to simulate
print(ex$call) # See how the data was simulated

# Fit the model. Either...
fit = mcp(ex$model, ex$data)
plot(fit)

ex_with_fit = mcp_example("demo", sample = TRUE)
plot(ex_with_fit$fit)
```

---

negbinomial

*Negative binomial for mcp*

---

**Description**

Parameterized as `mu` (mean; poisson lambda) and `size` (a shape parameter), so you can do `rnbinom(10, mu = 10, size = 1)`. Read more in the doc for `rnbinom`,

**Usage**

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

**Arguments**

`link` Link function (Character).

---

phi	<i>Inverse probit function</i>
-----	--------------------------------

---

**Description**

Inverse probit function

**Usage**

```
phi(eta)
```

**Arguments**

eta                    A vector of probits

**Value**

A vector with same length as mu

---

plot.mcpfit	<i>Plot full fits</i>
-------------	-----------------------

---

**Description**

Plot prior or posterior model draws on top of data. Use `plot_pars` to plot individual parameter estimates.

**Usage**

```
## S3 method for class 'mcpfit'  
plot(  
  x,  
  facet_by = NULL,  
  lines = 25,  
  geom_data = "point",  
  cp_dens = TRUE,  
  q_fit = FALSE,  
  q_predict = FALSE,  
  rate = TRUE,  
  prior = FALSE,  
  which_y = "ct",  
  arma = TRUE,  
  nsamples = 2000,  
  scale = "response",  
  ...  
)
```



**Arguments**

x	An <code>mcpfit</code> object
facet_by	String. Name of a varying group.
lines	Positive integer or FALSE. Number of lines (posterior draws). FALSE or lines = 0 plots no lines. Note that lines always plot fitted values - not predicted. For prediction intervals, see the <code>q_predict</code> argument.
geom_data	String. One of "point", "line" (good for time-series), or FALSE (don not plot).
cp_dens	TRUE/FALSE. Plot posterior densities of the change point(s)? Currently does not respect <code>facet_by</code> . This will be added in the future.
q_fit	Whether to plot quantiles of the posterior (fitted value). <ul style="list-style-type: none"> <li>• TRUE Add 2.5% and 97.5% quantiles. Corresponds to <code>q_fit = c(0.025, 0.975)</code>.</li> <li>• FALSE No quantiles</li> <li>• A vector of quantiles. For example, <code>quantiles = 0.5</code> plots the median and <code>quantiles = c(0.2, 0.8)</code> plots the 20% and 80% quantiles.</li> </ul>
q_predict	Same as <code>q_fit</code> , but for the prediction interval.
rate	Boolean. For binomial models, plot on raw data ( <code>rate = FALSE</code> ) or response divided by number of trials ( <code>rate = TRUE</code> ). If FALSE, linear interpolation on trial number is used to infer trials at a particular x.
prior	TRUE/FALSE. Plot using prior samples? Useful for <code>mcp(..., sample = "both")</code>
which_y	What to plot on the y-axis. One of <ul style="list-style-type: none"> <li>• "ct": The central tendency which is often the mean after applying the link function.</li> <li>• "sigma": The variance</li> <li>• "ar1", "ar2", etc. depending on which order of the autoregressive effects you want to plot.</li> </ul>
arma	Whether to include autoregressive effects. <ul style="list-style-type: none"> <li>• TRUE Compute autoregressive residuals. Requires the response variable in <code>newdata</code>.</li> <li>• FALSE Disregard the autoregressive effects. For <code>family = gaussian()</code>, <code>predict()</code> just use <code>sigma</code> for residuals.</li> </ul>
nsamples	Integer or NULL. Number of samples to return/summarise. If there are varying effects, this is the number of samples from each varying group. NULL means "all". Ignored if both are FALSE. More samples trade speed for accuracy.
scale	One of <ul style="list-style-type: none"> <li>• "response": return on the observed scale, i.e., after applying the inverse link function.</li> <li>• "linear": return on the parameter scale (where the linear trends are modelled).</li> </ul>
...	Currently ignored.

**Details**

plot() uses fit\$simulate() on posterior samples. These represent the (joint) posterior distribution.

**Value**

A **ggplot2** object.

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

**Examples**

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

plot(demo_fit, prior = TRUE) # The prior

plot(demo_fit, lines = 0, q_fit = TRUE) # 95% HDI without lines
plot(demo_fit, q_predict = c(0.1, 0.9)) # 80% prediction interval
plot(demo_fit, which_y = "sigma", lines = 100) # The variance parameter on y

# Show a panel for each varying 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
)
```

**Arguments**

fit	An <code>mcpfit</code> object.
pars	Character vector. One of: <ul style="list-style-type: none"> <li>• Vector of parameter names.</li> <li>• "population" plots all population parameters.</li> <li>• "varying" plots all varying effects. To plot a particular varying effect, use <code>regex_pars = "^name"</code>.</li> </ul>
regex_pars	Vector of regular expressions. This will typically just be the beginning of the parameter name(s), i.e., " <code>^cp_</code> " plots all change points, " <code>^my_varying</code> " plots all levels of a particular varying effect, and " <code>^cp_ ^my_varying</code> " plots both.
type	String or vector of strings. Calls <code>bayesplot::mcmc_&gt;&gt;type&lt;&lt;()</code> . Common calls are "combo", "trace", and "dens_overlay". Current options include 'acf', 'acf_bar', 'areas', 'areas_ridges', 'combo', 'dens', 'dens_chains', 'dens_overlay', 'hist', 'intervals', '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	TRUE/FALSE. Plot using prior samples? Useful for <code>mcp(..., sample = "both")</code>

**Details**

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

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

**Value**

A `ggplot2` object.

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindelov.dk>

**Examples**

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

## Not run:
# More options
plot_pars(demo_fit, regex_pars = "^cp_") # Plot only change points
plot_pars(demo_fit, pars = c("int_3", "time_3")) # Plot these parameters
plot_pars(demo_fit, type = c("trace", "violin")) # 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)

# Useful for varying effects:
# plot_pars(my_fit, pars = "varying", ncol = 3) # plot all varying effects
# plot_pars(my_fit, regex_pars = "my_varying", ncol = 3) # plot all levels of a particular varying

# Customize multi-column ggplots using "*" instead of "+" (patchwork)
library(ggplot2)
plot_pars(demo_fit, type = c("trace", "dens_overlay")) * theme_bw(10)

## End(Not run)

```

pp\_check

*Posterior Predictive Checks For Mcpfit Objects*

## Description

Plot posterior (default) or prior (prior = TRUE) predictive checks. This is convenience wrapper around the bayesplot::ppc\_\*() methods.

## Usage

```

pp_check(
  object,
  type = "dens_overlay",
  facet_by = NULL,
  newdata = NULL,
  prior = FALSE,
  varying = TRUE,
  arma = TRUE,
  nsamples = 100,
  ...
)

```

## Arguments

object	An mcpfit object.
type	One of bayesplot::available_ppc("grouped", invert = TRUE) %>% stringr::str_remove("ppc_")
facet_by	Name of a column in data modeled as varying effect(s).
newdata	A tibble or a data.frame containing predictors in the model. If NULL (default), the original data is used.
prior	TRUE/FALSE. Plot using prior samples? Useful for mcp(..., sample = "both")
varying	One of:

	<ul style="list-style-type: none"> <li>• TRUE All varying effects (<code>fit\$pars\$varying</code>).</li> <li>• FALSE No varying effects (<code>c()</code>).</li> <li>• Character vector: Only include specified varying parameters - see <code>fit\$pars\$varying</code>.</li> </ul>
arma	<p>Whether to include autoregressive effects.</p> <ul style="list-style-type: none"> <li>• TRUE Compute autoregressive residuals. Requires the response variable in <code>newdata</code>.</li> <li>• FALSE Disregard the autoregressive effects. For <code>family = gaussian()</code>, <code>predict()</code> just use <code>sigma</code> for residuals.</li> </ul>
nsamples	Number of draws. Note that you may want to use all data for summary geoms. e.g., <code>pp_check(fit, type = "ribbon", nsamples = NULL)</code> .
...	Further arguments passed to <code>bayesplot::ppc_type(y, yrep, ...)</code>

**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](#) [pp\\_eval](#)

**Examples**

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

---

predict.mcpfit

*Samples from the Posterior Predictive Distribution*

---

**Description**

Samples from the Posterior Predictive Distribution

**Usage**

```
## S3 method for class 'mcpfit'
predict(
  object,
  newdata = NULL,
  summary = TRUE,
  probs = TRUE,
  rate = TRUE,
  prior = FALSE,
  which_y = "ct",
  varying = TRUE,
  arma = TRUE,
  nsamples = NULL,
  samples_format = "tidy",
  ...
)
```

**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.
summary	Summarise at each x-value
probs	Vector of quantiles. Only in effect when summary == TRUE.
rate	Boolean. For binomial models, plot on raw data (rate = FALSE) or response divided by number of trials (rate = TRUE). If FALSE, linear interpolation on trial number is used to infer trials at a particular x.
prior	TRUE/FALSE. Plot using prior samples? Useful for mcp(..., sample = "both")
which_y	What to plot on the y-axis. One of <ul style="list-style-type: none"> <li>• "ct": The central tendency which is often the mean after applying the link function.</li> <li>• "sigma": The variance</li> <li>• "ar1", "ar2", etc. depending on which order of the autoregressive effects you want to plot.</li> </ul>
varying	One of: <ul style="list-style-type: none"> <li>• TRUE All varying effects (fit\$pars\$varying).</li> <li>• FALSE No varying effects (c()).</li> <li>• Character vector: Only include specified varying parameters - see fit\$pars\$varying.</li> </ul>
arma	Whether to include autoregressive effects. <ul style="list-style-type: none"> <li>• TRUE Compute autoregressive residuals. Requires the response variable in newdata.</li> <li>• FALSE Disregard the autoregressive effects. For family = gaussian(), predict() just use sigma for residuals.</li> </ul>

nsamples	Integer or NULL. Number of samples to return/summarise. If there are varying effects, this is the number of samples from each varying group. NULL means "all". Ignored if both are FALSE. More samples trade speed for accuracy.
samples_format	One of "tidy" or "matrix". Controls the output format when summary == FALSE. See more under "value"
...	Currently unused

### Value

- If `summary = TRUE`: A tibble with the posterior mean for each row in `newdata`, If `newdata` is NULL, the data in `fit$data` is used.
- If `summary = FALSE` and `samples_format = "tidy"`: A tidybayes tibble with all the posterior samples (Ns) evaluated at each row in `newdata` (Nn), i.e., with Ns x Nn rows. If there are varying effects, the returned data is expanded with the relevant levels for each row. The return columns are:
  - Predictors from `newdata`.
  - Sample descriptors: ".chain", ".iter", ".draw" (see the tidybayes package for more), and "data\_row" (`newdata` rownumber)
  - Sample values: one column for each parameter in the model.
  - The estimate. Either "predict" or "fitted", i.e., the name of the type argument.
- If `summary = FALSE` and `samples_format = "matrix"`: An `N_draws X n_rows(newdata)` matrix with fitted/predicted values (depending on type). This format is used by `brms` and it's useful as `yrep` in `bayesplot::ppc_*` functions.

### Author(s)

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

### See Also

[pp\\_eval\\_fitted.mcpfit](#) [residuals.mcpfit](#)

### Examples

```
predict(demo_fit) # Evaluate at each demo_fit$data
predict(demo_fit, probs = c(0.1, 0.5, 0.9)) # With median and 80% credible interval.
predict(demo_fit, summary = FALSE) # Samples instead of summary.
predict(
  demo_fit,
  newdata = data.frame(time = c(-5, 20, 300)), # Evaluate
  probs = c(0.025, 0.5, 0.975)
)
```

---

```
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.
...        Currently ignored
```

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

---

```
print.mcptext          Nice printing texts
```

---

**Description**

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

**Usage**

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

**Arguments**

```
x          Character, often with newlines.
...        Currently ignored.
```

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

**Examples**

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



---

probit	<i>Probit function</i>
--------	------------------------

---

**Description**

Probit function

**Usage**

```
probit(mu)
```

**Arguments**

mu                    A vector of probabilities (0.0 to 1.0)

**Value**

A vector with same length as mu

---

residuals.mcpfit	<i>Compute Residuals From Mcpfit Objects</i>
------------------	--

---

**Description**

Equivalent to `fitted(fit, ...) - fit$data[, fit$data$yvar]` (or `fitted(fit, ...) - newdata[, fit$data$yvar]`), but with fixed arguments for fitted: `rate = FALSE`, `which_y = 'ct'`, `samples_format = 'tidy'`.

**Usage**

```
## S3 method for class 'mcpfit'
residuals(
  object,
  newdata = NULL,
  summary = TRUE,
  probs = TRUE,
  prior = FALSE,
  varying = TRUE,
  arma = TRUE,
  nsamples = NULL,
  ...
)
```

**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.
summary	Summarise at each x-value
probs	Vector of quantiles. Only in effect when summary == TRUE.
prior	TRUE/FALSE. Plot using prior samples? Useful for mcp(..., sample = "both")
varying	One of: <ul style="list-style-type: none"> <li>• TRUE All varying effects (fit\$pars\$varying).</li> <li>• FALSE No varying effects (c()).</li> <li>• Character vector: Only include specified varying parameters - see fit\$pars\$varying.</li> </ul>
arma	Whether to include autoregressive effects. <ul style="list-style-type: none"> <li>• TRUE Compute autoregressive residuals. Requires the response variable in newdata.</li> <li>• FALSE Disregard the autoregressive effects. For family = gaussian(), predict() just use sigma for residuals.</li> </ul>
nsamples	Integer or NULL. Number of samples to return/summarise. If there are varying effects, this is the number of samples from each varying group. NULL means "all". Ignored if both are FALSE. More samples trade speed for accuracy.
...	Currently unused

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindelov.dk>

**See Also**

[pp\\_eval](#) [fitted.mcpfit](#) [predict.mcpfit](#)

**Examples**

```
residuals(demo_fit)
residuals(demo_fit, probs = c(0.1, 0.5, 0.9)) # With median and 80% credible interval.
residuals(demo_fit, summary = FALSE) # Samples instead of summary.
```

---

sd_to_prec	<i>Transform a prior from SD to precision.</i>
------------	--

---

**Description**

JAGS uses precision rather than SD. This function converts `dnorm(4.2, 1.3)` into `dnorm(4.2, 1/1.3^2)`. It allows users to specify priors using SD and then it's transformed for the JAGS code. It works for the following distributions: `dnorm`, `dt`, `dcauchy`, `ddexp`, `dlogis`, `dnorm`. In all of these, `tau/sd` is the second parameter.

**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, )`.

**Value**

A string

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

---

summary.mcpfit	<i>Summarise mcpfit objects</i>
----------------	---------------------------------

---

**Description**

Summarise parameter estimates and model diagnostics.

**Usage**

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

fixef(object, width = 0.95, prior = FALSE, ...)

ranef(object, width = 0.95, prior = FALSE, ...)

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

**Arguments**

object	An <code>mcpfit</code> object.
width	Float. The width of the highest posterior density interval (between 0 and 1).
digits	a non-null value for digits specifies the minimum number of significant digits to be printed in values. The default, NULL, uses <code>getOption("digits")</code> . (For the interpretation for complex numbers see <code>signif</code> .) Non-integer values will be rounded down, and only values greater than or equal to 1 and no greater than 22 are accepted.
prior	TRUE/FALSE. Summarise prior instead of posterior?
...	Currently ignored
x	An <code>mcpfit</code> object.

**Value**

A data frame with parameter estimates and MCMC diagnostics. OBS: The change point distributions are often not unimodal and symmetric so the intervals can be deceiving Plot them using `plot_pars(fit)`.

- mean is the posterior mean
- lower is the lower quantile of the highest-density interval (HDI) given in width.
- upper is the upper quantile.
- Rhat is the Gelman-Rubin convergence diagnostic which is often taken to be acceptable if < 1.1. It is computed using `gelman.diag`.
- n.eff is the effective sample size computed using `effectiveSize`. Low effective sample sizes are also obvious as poor mixing in trace plots (see `plot_pars(fit)`). Read how to deal with such problems [here](#)
- ts\_err is the time-series error, taking autoregressive correlation into account. It is computed using `spectrum0.ar`.

For simulated data, the summary contains two additional columns so that it is easy to inspect whether the model can recover the parameters. 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 HDI interval (lower to upper).

**Functions**

- `fixef()`: Get population-level ("fixed") effects of an `mcpfit` object.
- `ranef()`: Get varying ("random") effects of an `mcpfit` object.
- `print(mcpfit)`: Print the posterior summary of an `mcpfit` object.

**Author(s)**

Jonas Kristoffer Lindeløv <jonas@lindeloev.dk>

### **Examples**

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

# Get the results as a data frame
results = summary(demo_fit)

# Varying (random) effects
# ranef(my_fit)

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

# Index

## \* datasets

demo\_fit, 4

bernoulli, 2

criterion, 3, 3

demo\_fit, 4

effectiveSize, 28

exponential, 4

fitted(fitted.mcpfit), 5

fitted.mcpfit, 5, 23, 26

fixed.effects(summary.mcpfit), 27

fixef(summary.mcpfit), 27

gelman.diag, 28

get\_segment\_table, 12

hypothesis, 7, 11

ilogit, 8

is.mcpfit, 9

jags.model, 12

logit, 9

L00(criterion), 3

loo, 3

loo(criterion), 3

loo\_compare, 3

loo\_model\_weights, 3

mcmc.list, 14

mcp, 10, 13, 14, 19

mcp\_example, 14

mcpfit, 3, 4, 7, 12, 17, 19, 24, 28

mcpfit(mcpfit-class), 13

mcpfit-class, 13

negbinomial, 15

phi, 16

plot(plot.mcpfit), 16

plot.mcpfit, 16, 21

plot\_pars, 18

pp\_check, 20

pp\_eval, 6, 21, 23, 26

predict(predict.mcpfit), 21

predict.mcpfit, 6, 21, 26

print(summary.mcpfit), 27

print.mcplist, 24

print.mcptext, 24

probit, 25

random.effects(summary.mcpfit), 27

ranef(summary.mcpfit), 27

resid(residuals.mcpfit), 25

residuals(residuals.mcpfit), 25

residuals.mcpfit, 6, 23, 25

sd\_to\_prec, 27

spectrum0.ar, 28

summary(summary.mcpfit), 27

summary.mcpfit, 27

WAIC(criterion), 3

waic, 3

waic(criterion), 3