

# Package: fastcpd (via r-universe)

October 24, 2024

**Type** Package

**Title** Fast Change Point Detection via Sequential Gradient Descent

**Version** 0.14.3

**Description** Implements fast change point detection algorithm based on the paper "Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis" by Xianyang Zhang, Trisha Dawn <<https://proceedings.mlr.press/v206/zhang23b.html>>. The algorithm is based on dynamic programming with pruning and sequential gradient descent. It is able to detect change points a magnitude faster than the vanilla Pruned Exact Linear Time(PELT). The package includes examples of linear regression, logistic regression, Poisson regression, penalized linear regression data, and whole lot more examples with custom cost function in case the user wants to use their own cost function.

**License** GPL (>= 3)

**URL** <https://fastcpd.xingchi.li>, <https://github.com/doccstat/fastcpd>

**BugReports** <https://github.com/doccstat/fastcpd/issues>

**Depends** R (>= 2.10)

**Imports** fastglm, forecast, glmnet, Matrix, methods, Rcpp (>= 0.11.0), stats, tseries

**Suggests** abind, breakfast, changepoint, cpm, CptNonPar, dplyr, fpop, ggplot2, gridExtra, jointseg, knitr, lubridate, matrixStats, mockthat, mvtnorm, not, numDeriv, RcppClock, reshape2, rmarkdown, segmented, stepR, testthat (>= 3.0.0), VARDetect, wbs, xml2, zoo

**LinkingTo** progress, Rcpp, RcppArmadillo, RcppClock, testthat

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Config/testthat/start-first** vignettes-comparison-pelt, examples-fastcpd\_arima, examples-fastcpd\_ts

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.1

**NeedsCompilation** yes

**Author** Xingchi Li [aut, cre, cph]

(<<https://orcid.org/0009-0006-2493-0853>>), Xianyang Zhang [aut, cph]

**Maintainer** Xingchi Li <[anthony.li@stat.tamu.edu](mailto:anthony.li@stat.tamu.edu)>

**Repository** CRAN

**Date/Publication** 2024-04-26 11:50:07 UTC

## Contents

bitcoin . . . . .	3
fastcpd . . . . .	4
fastcpd-class . . . . .	12
fastcpd_ar . . . . .	12
fastcpd_arima . . . . .	13
fastcpd_arma . . . . .	14
fastcpd_binomial . . . . .	16
fastcpd_garch . . . . .	17
fastcpd_lasso . . . . .	18
fastcpd_lm . . . . .	19
fastcpd_mean . . . . .	21
fastcpd_meanvariance . . . . .	22
fastcpd_poisson . . . . .	24
fastcpd_ts . . . . .	25
fastcpd_var . . . . .	26
fastcpd_variance . . . . .	27
occupancy . . . . .	28
plot.fastcpd . . . . .	29
print.fastcpd . . . . .	32
show.fastcpd . . . . .	32
summary.fastcpd . . . . .	33
transcriptome . . . . .	33
uk_seatbelts . . . . .	35
variance_arma . . . . .	37
variance_lm . . . . .	38
variance_mean . . . . .	39
variance_median . . . . .	40
well_log . . . . .	40

**Index** 42

---

bitcoin	<i>Bitcoin Market Price (USD)</i>
---------	-----------------------------------

---

### Description

The average USD market price across major bitcoin exchanges.

### Usage

```
bitcoin
```

### Format

A data frame with 1354 rows and 2 variables:

**date** POSIXct,POSIXt (TZ: "UTC") from 2019-01-02 to 2023-10-28

**price** The average USD market price across major bitcoin exchanges

### Source

<<https://www.blockchain.com/explorer/charts/market-price>>

### Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  p <- ggplot2::ggplot(bitcoin, ggplot2::aes(x = date, y = price)) +
    ggplot2::geom_line()
  print(p)

  result <- suppressWarnings(fastcpd.garch(
    diff(log(bitcoin$price[600:900])), c(1, 1),
    beta = "BIC", cost_adjustment = "BIC"
  ))
  summary(result)
  bitcoin$date[result@cp_set + 600]
  plot(result)

  cp_dates <- bitcoin[600 + result@cp_set + 1, "date"]
  ggplot2::ggplot(
    data = data.frame(
      x = bitcoin$date[600:900], y = bitcoin$price[600:900]
    ),
    ggplot2::aes(x = x, y = y)
  ) +
  ggplot2::geom_line(color = "steelblue") +
  ggplot2::geom_vline(
    xintercept = cp_dates,
    color = "red",
    linetype = "dotted",
```

```

    linewidth = 0.5,
    alpha = 0.7
  ) +
  ggplot2::labs(
    x = "Year",
    y = "Bitcoin price in USD"
  ) +
  ggplot2::annotate(
    "text",
    x = cp_dates,
    y = 2000,
    label = as.character(cp_dates),
    color = "steelblue"
  ) +
  ggplot2::theme_bw()
}

```

---

fastcpd

*Find change points efficiently*


---

## Description

`fastcpd()` takes in formulas, data, families and extra parameters and returns a `fastcpd` object.

## Usage

```

fastcpd(
  formula = y ~ . - 1,
  data,
  beta = "MBIC",
  cost_adjustment = "MBIC",
  family = NULL,
  cost = NULL,
  cost_gradient = NULL,
  cost_hessian = NULL,
  line_search = c(1),
  lower = rep(-Inf, p),
  upper = rep(Inf, p),
  pruning_coef = 0,
  segment_count = 10,
  trim = 0.02,
  momentum_coef = 0,
  multiple_epochs = function(x) 0,
  epsilon = 1e-10,
  order = c(0, 0, 0),
  p = ncol(data) - 1,
  cp_only = FALSE,

```

```

    vanilla_percentage = 0,
    warm_start = FALSE,
    ...
)

```

## Arguments

formula	A formula object specifying the model to be fitted. The (optional) response variable should be on the LHS of the formula, while the covariates should be on the RHS. The naming of variables used in the formula should be consistent with the column names in the data frame provided in <code>data</code> . The intercept term should be removed from the formula. The response variable is not needed for mean/variance change models and time series models. By default, an intercept column will be added to the data, similar to the <code>lm()</code> function. Thus, it is suggested that users should remove the intercept term by appending <code>- 1</code> to the formula. Note that the <code>fastcpd.family</code> functions do not require a formula input.
data	A data frame of dimension $T \times d$ containing the data to be segmented (where each row denotes a data point $z_t \in \mathbb{R}^d$ for $t = 1, \dots, T$ ) is required in the main function, while a matrix or a vector input is also accepted in the <code>fastcpd.family</code> functions.
beta	Penalty criterion for the number of change points. This parameter takes a string value of "BIC", "MBIC", "MDL" or a numeric value. If a numeric value is provided, the value will be used as the penalty. By default, the mBIC criterion is used, where $\beta = (p + 2) \log(T)/2$ . This parameter usage should be paired with <code>cost_adjustment</code> described below. Discussions about the penalty criterion can be found in the references.
cost_adjustment	Cost adjustment criterion. It can be "BIC", "MBIC", "MDL" or NULL. By default, the cost adjustment criterion is set to be "MBIC". The "MBIC" and "MDL" criteria modify the cost function by adding a negative adjustment term to the cost function. "BIC" or NULL does not modify the cost function. Details can in found in the references.
family	Family class of the change point model. It can be "mean" for mean change, "variance" for variance change, "meanvariance" for mean and/or variance change, "lm" for linear regression, "binomial" for logistic regression, "poisson" for Poisson regression, "lasso" for penalized linear regression, "ar" for AR( $p$ ) models, "arma" for ARMA( $p, q$ ) models, "arima" for ARIMA( $p, d, q$ ) models, "garch" for GARCH( $p, q$ ) models, "var" for VAR( $p$ ) models and "custom" for user-specified custom models. Omitting this parameter is the same as specifying the parameter to be "custom" or NULL, in which case, users must specify the custom cost function.
cost	Cost function to be used. <code>cost</code> , <code>cost_gradient</code> , and <code>cost_hessian</code> should not be specified at the same time with <code>family</code> as built-in families have cost functions implemented in C++ to provide better performance. If not specified, the default is the negative log-likelihood for the corresponding family. Custom cost functions can be provided in the following two formats: <ul style="list-style-type: none"> <li>• <code>cost = function(data) { ... }</code></li> </ul>

- `cost = function(data, theta) {...}`

Users can specify a loss function using the second format that will be used to calculate the cost value. In both formats, the input data is a subset of the original data frame in the form of a matrix (a matrix with a single column in the case of a univariate data set). In the first format, the specified cost function directly calculates the cost value. `fastcpd()` performs the vanilla PELT algorithm, and `cost_gradient` and `cost_hessian` should not be provided since no parameter updating is necessary for vanilla PELT. In the second format, the loss function  $\sum_{i=s}^t l(z_i, \theta)$  is provided, which has to be optimized over the parameter  $\theta$  to obtain the cost value. A detailed discussion about the custom cost function usage can be found in the references.

`cost_gradient` Gradient of the custom cost function. Example usage:

```
cost_gradient = function(data, theta) {
  ...
  return(gradient)
}
```

The gradient function takes two inputs, the first being a matrix representing a segment of the data, similar to the format used in the cost function, and the second being the parameter that needs to be optimized. The gradient function returns the value of the gradient of the loss function, i.e.,  $\sum_{i=s}^t \nabla l(z_i, \theta)$ .

`cost_hessian` Hessian of the custom loss function. The Hessian function takes two inputs, the first being a matrix representing a segment of the data, similar to the format used in the cost function, and the second being the parameter that needs to be optimized. The gradient function returns the Hessian of the loss function, i.e.,  $\sum_{i=s}^t \nabla^2 l(z_i, \theta)$ .

`line_search` If a vector of numeric values is provided, a line search will be performed to find the optimal step size for each update. Detailed usage of `line_search` can be found in the references.

`lower` Lower bound for the parameters. Used to specify the domain of the parameters after each gradient descent step. If not specified, the lower bound is set to be `-Inf` for all parameters. `lower` is especially useful when the estimated parameters take only positive values, such as the noise variance.

`upper` Upper bound for the parameters. Used to specify the domain of the parameters after each gradient descent step. If not specified, the upper bound is set to be `Inf` for all parameters.

`pruning_coef` Pruning coefficient `sc_0` used in the pruning step of the PELT algorithm with the default value 0. If `cost_adjustment` is specified as "MBIC", an adjustment term  $p \log(2)$  will be added to the pruning coefficient. If `cost_adjustment` is specified as "MDL", an adjustment term  $p \log_2(2)$  will be added to the pruning coefficient. Detailed discussion about the pruning coefficient can be found in the references.

`segment_count` An initial guess of the number of segments. If not specified, the initial guess of the number of segments is 10. The initial guess affects the initial estimates of the parameters in SeGD.

trim	Trimming for the boundary change points so that a change point close to the boundary will not be counted as a change point. This parameter also specifies the minimum distance between two change points. If several change points have mutual distances smaller than $\text{trim} * \text{nrow}(\text{data})$ , those change points will be merged into one single change point. The value of this parameter should be between 0 and 1.
momentum_coef	Momentum coefficient to be applied to each update. This parameter is used when the loss function is bad-shaped so that maintaining a momentum from previous update is desired. Default value is 0, meaning the algorithm doesn't maintain a momentum by default.
multiple_epochs	<p>A function can be specified such that an adaptive number of multiple epochs can be utilized to improve the algorithm's performance. <code>multiple_epochs</code> is a function of the length of the data segment. The function returns an integer indicating how many epochs should be performed apart from the default update. By default, the function returns zero, meaning no multiple epochs will be used to update the parameters. Example usage:</p> <pre>multiple_epochs = function(segment_length) {   if (segment_length &lt; 100) 1   else 0 }</pre> <p>This function will let SeGD perform parameter updates with an additional epoch for each segment with a length less than 100 and no additional epoch for segments with lengths greater or equal to 100.</p>
epsilon	Epsilon to avoid numerical issues. Only used for the Hessian computation in Logistic Regression and Poisson Regression.
order	Order of the $\text{AR}(p)$ , $\text{VAR}(p)$ or $\text{ARIMA}(p, d, q)$ model.
p	Number of covariates in the model. If not specified, the number of covariates will be inferred from the data, i.e., $p = \text{ncol}(\text{data}) - 1$ . This parameter is superseded by <code>order</code> in the case of time series models: "ar", "var", "arima".
cp_only	If TRUE, only the change points are returned. Otherwise, the cost function values together with the estimated parameters for each segment are also returned. By default the value is set to be FALSE so that <code>plot</code> can be used to visualize the results for a built-in model. <code>cp_only</code> has some performance impact on the algorithm, since the cost values and estimated parameters for each segment need to be calculated and stored. If the users are only interested in the change points, setting <code>cp_only</code> to be TRUE will help with the computational cost.
vanilla_percentage	The parameter $v$ is between zero and one. For each segment, when its length is no more than $vT$ , the cost value will be computed by performing an exact minimization of the loss function over the parameter. When its length is greater than $vT$ , the cost value is approximated through SeGD. Therefore, this parameter induces an algorithm that can be interpreted as an interpolation between dynamic programming with SeGD ( $v = 0$ ) and the vanilla PELT ( $v = 1$ ). The readers are referred to the references for more details.

warm_start	If TRUE, the algorithm will use the estimated parameters from the previous segment as the initial value for the current segment. This parameter is only used for the "glm" families.
...	Other parameters for specific models. <ul style="list-style-type: none"> <li>• <code>include.mean</code> is used to determine if a mean/intercept term should be included in the <math>ARIMA(p, d, q)</math> or <math>GARCH(p, q)</math> models.</li> <li>• <code>r.clock</code> is used to create an <code>RcppClock</code> object to record the time spent in the C++ code. Default is an empty string. If set to any non-empty string, an object with specified name will be created. Usage: <code>library(RcppClock); plot(VARIABLE_NAME)</code>.</li> <li>• <code>r.progress</code> is used to control the progress bar. By default the progress bar will be shown. To disable it, set <code>r.progress = FALSE</code>.</li> <li>• <code>p.response</code> is used to specify the number of response variables. This parameter is especially useful for linear models with multivariate responses.</li> </ul>

### Value

A `fastcpd` object.

### Gallery

<https://github.com/doccstat/fastcpd/tree/main/tests/testthat/examples>

### References

Xingchi Li, Xianyang Zhang (2024). "fastcpd: Fast Change Point Detection in R." *arXiv:2404.05933*, <https://arxiv.org/abs/2404.05933>.

Xianyang Zhang, Trisha Dawn (2023). "Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis." In Ruiz, Francisco, Dy, Jennifer, van de Meent, Jan-Willem (eds.), *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 series Proceedings of Machine Learning Research, 1129-1143.

### See Also

`fastcpd.family` for the family-specific function; `plot.fastcpd()` for plotting the results, `summary.fastcpd()` for summarizing the results.

### Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 200
  p <- 4
  d <- 2
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_1 <- matrix(runif(8, -3, -1), nrow = p)
  theta_2 <- matrix(runif(8, -1, 3), nrow = p)
  y <- rbind(
    x[1:125, ] %*% theta_1 + mvtnorm::rmvnorm(125, rep(0, d), 3 * diag(d)),
```



```

      x[126:n, ] %*% theta_2 + mvtnorm::rmvnorm(75, rep(0, d), 3 * diag(d))
    )
    result_mlm <- fastcpd(
      cbind(y.1, y.2) ~ . - 1, cbind.data.frame(y = y, x = x), family = "lm"
    )
    summary(result_mlm)
  }
  if (
    requireNamespace("mvtnorm", quietly = TRUE) &&
    requireNamespace("stats", quietly = TRUE)
  ) {
    set.seed(1)
    n <- 400 + 300 + 500
    p <- 5
    x <- mvtnorm::rmvnorm(n, mean = rep(0, p), sigma = diag(p))
    theta <- rbind(
      mvtnorm::rmvnorm(1, mean = rep(0, p - 3), sigma = diag(p - 3)),
      mvtnorm::rmvnorm(1, mean = rep(5, p - 3), sigma = diag(p - 3)),
      mvtnorm::rmvnorm(1, mean = rep(9, p - 3), sigma = diag(p - 3))
    )
    theta <- cbind(theta, matrix(0, 3, 3))
    theta <- theta[rep(seq_len(3), c(400, 300, 500)), ]
    y_true <- rowSums(x * theta)
    factor <- c(
      2 * stats::rbinom(400, size = 1, prob = 0.95) - 1,
      2 * stats::rbinom(300, size = 1, prob = 0.95) - 1,
      2 * stats::rbinom(500, size = 1, prob = 0.95) - 1
    )
    y <- factor * y_true + stats::rnorm(n)
    data <- cbind.data.frame(y, x)
    huber_threshold <- 1
    huber_loss <- function(data, theta) {
      residual <- data[, 1] - data[, -1, drop = FALSE] %*% theta
      indicator <- abs(residual) <= huber_threshold
      sum(
        residual^2 / 2 * indicator +
        huber_threshold * (
          abs(residual) - huber_threshold / 2
        ) * (1 - indicator)
      )
    }
    huber_loss_gradient <- function(data, theta) {
      residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
      if (abs(residual) <= huber_threshold) {
        -residual * data[nrow(data), -1]
      } else {
        -huber_threshold * sign(residual) * data[nrow(data), -1]
      }
    }
    huber_loss_hessian <- function(data, theta) {
      residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
      if (abs(residual) <= huber_threshold) {
        outer(data[nrow(data), -1], data[nrow(data), -1])
      }
    }
  }

```

```

    } else {
      0.01 * diag(length(theta))
    }
  }
  huber_regression_result <- fastcpd(
    formula = y ~ . - 1,
    data = data,
    beta = (p + 1) * log(n) / 2,
    cost = huber_loss,
    cost_gradient = huber_loss_gradient,
    cost_hessian = huber_loss_hessian
  )
  summary(huber_regression_result)
}

set.seed(1)
p <- 5
x <- matrix(rnorm(375 * p, 0, 1), ncol = p)
theta <- rbind(rnorm(p, 0, 1), rnorm(p, 2, 1))
y <- c(
  rbinom(200, 1, 1 / (1 + exp(-x[1:200, ] %*% theta[1, ]))),
  rbinom(175, 1, 1 / (1 + exp(-x[201:375, ] %*% theta[2, ])))
)
data <- data.frame(y = y, x = x)
result_builtin <- suppressWarnings(fastcpd.binomial(data))
logistic_loss <- function(data, theta) {
  x <- data[, -1]
  y <- data[, 1]
  u <- x %*% theta
  nll <- -y * u + log(1 + exp(u))
  nll[u > 10] <- -y[u > 10] * u[u > 10] + u[u > 10]
  sum(nll)
}
logistic_loss_gradient <- function(data, theta) {
  x <- data[nrow(data), -1]
  y <- data[nrow(data), 1]
  c(-(y - 1 / (1 + exp(-x %*% theta)))) * x
}
logistic_loss_hessian <- function(data, theta) {
  x <- data[nrow(data), -1]
  prob <- 1 / (1 + exp(-x %*% theta))
  (x %*% x) * c((1 - prob) * prob)
}
result_custom <- fastcpd(
  formula = y ~ . - 1,
  data = data,
  epsilon = 1e-5,
  cost = logistic_loss,
  cost_gradient = logistic_loss_gradient,
  cost_hessian = logistic_loss_hessian
)
cat(
  "Change points detected by built-in logistic regression model: ",

```

```

    result_builtin@cp_set, "\n",
    "Change points detected by custom logistic regression model: ",
    result_custom@cp_set, "\n",
    sep = ""
  )
result_custom_two_epochs <- fastcpd(
  formula = y ~ . - 1,
  data = data,
  multiple_epochs = function(segment_length) 1,
  epsilon = 1e-5,
  cost = logistic_loss,
  cost_gradient = logistic_loss_gradient,
  cost_hessian = logistic_loss_hessian
)
summary(result_custom_two_epochs)

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 480
  p_true <- 6
  p <- 50
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(
    runif(p_true, -5, -2),
    runif(p_true, -3, 3),
    runif(p_true, 2, 5),
    runif(p_true, -5, 5)
  )
  theta_0 <- cbind(theta_0, matrix(0, ncol = p - p_true, nrow = 4))
  y <- c(
    x[1:80, ] %*% theta_0[1, ] + rnorm(80, 0, 1),
    x[81:200, ] %*% theta_0[2, ] + rnorm(120, 0, 1),
    x[201:320, ] %*% theta_0[3, ] + rnorm(120, 0, 1),
    x[321:n, ] %*% theta_0[4, ] + rnorm(160, 0, 1)
  )
  small_lasso_data <- cbind.data.frame(y, x)
  result_no_vp <- fastcpd.lasso(
    small_lasso_data,
    beta = "BIC",
    cost_adjustment = NULL,
    pruning_coef = 0
  )
  summary(result_no_vp)
  result_20_vp <- fastcpd.lasso(
    small_lasso_data,
    beta = "BIC",
    cost_adjustment = NULL,
    vanilla_percentage = 0.2,
    pruning_coef = 0
  )
  summary(result_20_vp)
}

```

---

fastcpd-class	<i>An S4 class to store the output created with <a href="#">fastcpd()</a></i>
---------------	---

---

### Description

This S4 class stores the output from [fastcpd\(\)](#) and [fastcpd.family](#). A fastcpd object consist of several slots including the call to [fastcpd\(\)](#), the data used, the family of the model, the change points, the cost values, the residuals, the estimated parameters and a boolean indicating whether the model was fitted with only change points or with change points and parameters, which you can select using @.

### Slots

call The call of the function.  
 data The data passed to the function.  
 order The order of the time series model.  
 family The family of the model.  
 cp\_set The set of change points.  
 cost\_values The cost function values for each segment.  
 residuals The residuals of the model with change points. Used only for built-in families.  
 thetas The estimated parameters for each segment. Used only for built-in families.  
 cp\_only A boolean indicating whether [fastcpd\(\)](#) was run to return only the change points or the change points with the estimated parameters and cost values for each segment.

---

fastcpd_ar	<i>Find change points efficiently in AR(p) models</i>
------------	---

---

### Description

[fastcpd\\_ar\(\)](#) and [fastcpd.ar\(\)](#) are wrapper functions of [fastcpd\(\)](#) to find change points in AR( $p$ ) models. The function is similar to [fastcpd\(\)](#) except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

### Usage

```
fastcpd_ar(data, order = 0, ...)
```

```
fastcpd.ar(data, order = 0, ...)
```

**Arguments**

data	A numeric vector, a matrix, a data frame or a time series object.
order	A positive integer specifying the order of the AR model.
...	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> . One special argument can be passed here is <code>include.mean</code> , which is a logical value indicating whether the mean should be included in the model. The default value is TRUE.

**Value**

A `fastcpd` object.

**See Also**

[fastcpd\(\)](#)

**Examples**

```
set.seed(1)
n <- 1000
x <- rep(0, n + 3)
for (i in 1:600) {
  x[i + 3] <- 0.6 * x[i + 2] - 0.2 * x[i + 1] + 0.1 * x[i] + rnorm(1, 0, 3)
}
for (i in 601:1000) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.4 * x[i + 1] + 0.2 * x[i] + rnorm(1, 0, 3)
}
result <- fastcpd.ar(x[3 + seq_len(n)], 3)
summary(result)
plot(result)
```

---

fastcpd\_arima

*Find change points efficiently in ARIMA(p, d, q) models*


---

**Description**

`fastcpd_arima()` and `fastcpd.arima()` are wrapper functions of `fastcpd()` to find change points in ARIMA( $p, d, q$ ) models. The function is similar to `fastcpd()` except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

**Usage**

```
fastcpd_arima(data, order = 0, ...)
```

```
fastcpd.arima(data, order = 0, ...)
```

**Arguments**

data	A numeric vector, a matrix, a data frame or a time series object.
order	A vector of length three specifying the order of the ARIMA model.
...	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> . One special argument can be passed here is <code>include.mean</code> , which is a logical value indicating whether the mean should be included in the model. The default value is TRUE.

**Value**

A `fastcpd` object.

**See Also**

[fastcpd\(\)](#)

**Examples**

```
set.seed(1)
n <- 271
w <- rnorm(n + 1, 0, 3)
dx <- rep(0, n + 1)
x <- rep(0, n + 1)
for (i in 1:180) {
  dx[i + 1] <- 0.8 * dx[i] + w[i + 1] - 0.5 * w[i]
  x[i + 1] <- x[i] + dx[i + 1]
}
for (i in 181:n) {
  dx[i + 1] <- -0.6 * dx[i] + w[i + 1] + 0.3 * w[i]
  x[i + 1] <- x[i] + dx[i + 1]
}
result <- fastcpd.arma(
  diff(x[1 + seq_len(n)]),
  c(1, 0, 1),
  segment_count = 3,
  include.mean = FALSE
)
summary(result)
plot(result)
```

---

fastcpd\_arma

*Find change points efficiently in ARMA(p, q) models*

---

**Description**

`fastcpd_arma()` and `fastcpd.arma()` are wrapper functions of `fastcpd()` to find change points in ARMA( $p, q$ ) models. The function is similar to `fastcpd()` except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

**Usage**

```
fastcpd_arma(data, order = c(0, 0), ...)
```

```
fastcpd.arma(data, order = c(0, 0), ...)
```

**Arguments**

`data` A numeric vector, a matrix, a data frame or a time series object.

`order` A vector of length two specifying the order of the ARMA model.

`...` Other arguments passed to `fastcpd()`, for example, `segment_count`.

**Value**

A `fastcpd` object.

**See Also**

[fastcpd\(\)](#)

**Examples**

```
set.seed(1)
n <- 300
w <- rnorm(n + 3, 0, 3)
x <- rep(0, n + 3)
for (i in 1:200) {
  x[i + 3] <- 0.1 * x[i + 2] - 0.3 * x[i + 1] + 0.1 * x[i] +
    0.1 * w[i + 2] + 0.5 * w[i + 1] + w[i + 3]
}
for (i in 201:n) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.1 * x[i + 1] - 0.3 * x[i] -
    0.6 * w[i + 2] - 0.1 * w[i + 1] + w[i + 3]
}
result <- suppressWarnings(
  fastcpd.arma(
    data = x[3 + seq_len(n)],
    order = c(3, 2),
    segment_count = 3,
    lower = c(rep(-1, 3 + 2), 1e-10),
    upper = c(rep(1, 3 + 2), Inf),
    line_search = c(1, 0.1, 1e-2),
    beta = "BIC",
    cost_adjustment = "BIC"
  )
)
summary(result)
plot(result)
```

---

fastcpd_binomial	<i>Find change points efficiently in logistic regression models</i>
------------------	---

---

### Description

`fastcpd_binomial()` and `fastcpd.binomial()` are wrapper functions of `fastcpd()` to find change points in logistic regression models. The function is similar to `fastcpd()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

### Usage

```
fastcpd_binomial(data, ...)
```

```
fastcpd.binomial(data, ...)
```

### Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>...</code>	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> .

### Value

A `fastcpd` object.

### See Also

[fastcpd\(\)](#)

### Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 500
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta <- rbind(rnorm(p, 0, 1), rnorm(p, 2, 1))
  y <- c(
    rbinom(300, 1, 1 / (1 + exp(-x[1:300, ] %*% theta[1, ]))),
    rbinom(200, 1, 1 / (1 + exp(-x[301:n, ] %*% theta[2, ])))
  )
  result <- suppressWarnings(fastcpd.binomial(cbind(y, x)))
  summary(result)
  plot(result)
}
```



---

fastcpd_garch	<i>Find change points efficiently in GARCH(<math>p</math>, <math>q</math>) models</i>
---------------	---

---

## Description

`fastcpd_garch()` and `fastcpd.garch()` are wrapper functions of `fastcpd()` to find change points in GARCH( $p$ ,  $q$ ) models. The function is similar to `fastcpd()` except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

## Usage

```
fastcpd_garch(data, order = c(0, 0), ...)
```

```
fastcpd.garch(data, order = c(0, 0), ...)
```

## Arguments

<code>data</code>	A numeric vector, a matrix, a data frame or a time series object.
<code>order</code>	A positive integer vector of length two specifying the order of the GARCH model.
<code>...</code>	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> .

## Value

A `fastcpd` object.

## See Also

[fastcpd\(\)](#)

## Examples

```
set.seed(1)
n <- 400
sigma_2 <- rep(1, n + 1)
x <- rep(0, n + 1)
for (i in seq_len(200)) {
  sigma_2[i + 1] <- 20 + 0.5 * x[i]^2 + 0.1 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
for (i in 201:400) {
  sigma_2[i + 1] <- 1 + 0.1 * x[i]^2 + 0.5 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
result <- suppressWarnings(
  fastcpd.garch(x[-1], c(1, 1), include.mean = FALSE)
)
summary(result)
```

```
plot(result)
```

---

```
fastcpd_lasso
```

```
Find change points efficiently in penalized linear regression models
```

---

## Description

`fastcpd_lasso()` and `fastcpd.lasso()` are wrapper functions of `fastcpd()` to find change points in penalized linear regression models. The function is similar to `fastcpd()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

## Usage

```
fastcpd_lasso(data, ...)
```

```
fastcpd.lasso(data, ...)
```

## Arguments

`data` A matrix or a data frame with the response variable as the first column.  
`...` Other arguments passed to `fastcpd()`, for example, `segment_count`.

## Value

A `fastcpd` object.

## See Also

[fastcpd\(\)](#)

## Examples

```
if (
  requireNamespace("dplyr", quietly = TRUE) &&
  requireNamespace("ggplot2", quietly = TRUE) &&
  requireNamespace("mvtnorm", quietly = TRUE) &&
  requireNamespace("reshape2", quietly = TRUE)
) {
  set.seed(1)
  n <- 480
  p_true <- 5
  p <- 50
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(
    runif(p_true, -5, -2),
    runif(p_true, -3, 3),
    runif(p_true, 2, 5),
  )
}
```

```

    runif(p_true, -5, 5)
  )
  theta_0 <- cbind(theta_0, matrix(0, ncol = p - p_true, nrow = 4))
  y <- c(
    x[1:80, ] %*% theta_0[1, ] + rnorm(80, 0, 1),
    x[81:200, ] %*% theta_0[2, ] + rnorm(120, 0, 1),
    x[201:320, ] %*% theta_0[3, ] + rnorm(120, 0, 1),
    x[321:n, ] %*% theta_0[4, ] + rnorm(160, 0, 1)
  )
  result <- fastcpd.lasso(
    cbind(y, x),
    multiple_epochs = function(segment_length) if (segment_length < 30) 1 else 0
  )
  summary(result)
  plot(result)

  thetas <- result@thetas
  thetas <- cbind.data.frame(thetas, t(theta_0))
  names(thetas) <- c(
    "segment 1", "segment 2", "segment 3", "segment 4",
    "segment 1 truth", "segment 2 truth", "segment 3 truth", "segment 4 truth"
  )
  thetas$coordinate <- c(seq_len(p_true), rep("rest", p - p_true))
  molten <- reshape2::melt(thetas, id.vars = "coordinate")
  molten <- dplyr::mutate(
    molten,
    segment = gsub("segment ", "", variable),
    segment = gsub(" truth", "", segment),
    height = as.numeric(gsub("segment.*", "", segment)) +
      0.2 * as.numeric(grepl("truth", variable)),
    parameter = ifelse(grepl("truth", variable), "truth", "estimated")
  )
  ggplot2::ggplot() +
    ggplot2::geom_point(
      data = molten,
      ggplot2::aes(
        x = value, y = height, shape = coordinate, color = parameter
      ),
      size = 4
    ) +
    ggplot2::ylim(0.8, 4.4) +
    ggplot2::ylab("segment") +
    ggplot2::theme_bw()
}

```

**Description**

`fastcpd_lm()` and `fastcpd.lm()` are wrapper functions of `fastcpd()` to find change points in linear regression models. The function is similar to `fastcpd()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

**Usage**

```
fastcpd_lm(data, ...)
```

```
fastcpd.lm(data, ...)
```

**Arguments**

`data`                    A matrix or a data frame with the response variable as the first column.  
`...`                    Other arguments passed to `fastcpd()`, for example, `segment_count`.

**Value**

A `fastcpd` object.

**See Also**

[fastcpd\(\)](#)

**Examples**

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 300
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
  y <- c(
    x[1:100, ] %*% theta_0[1, ] + rnorm(100, 0, 3),
    x[101:200, ] %*% theta_0[2, ] + rnorm(100, 0, 3),
    x[201:n, ] %*% theta_0[3, ] + rnorm(100, 0, 3)
  )
  result_lm <- fastcpd.lm(cbind(y, x))
  summary(result_lm)
  plot(result_lm)

  set.seed(1)
  n <- 600
  p <- 4
  d <- 2
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_1 <- matrix(runif(8, -3, -1), nrow = p)
  theta_2 <- matrix(runif(8, -1, 3), nrow = p)
  y <- rbind(
    x[1:350, ] %*% theta_1 + mvtnorm::rmvnorm(350, rep(0, d), 3 * diag(d)),
```

```

      x[351:n, ] %*% theta_2 + mvtnorm::rmvnorm(250, rep(0, d), 3 * diag(d))
    )
    result_mlm <- fastcpd.lm(cbind.data.frame(y = y, x = x), p.response = 2)
    summary(result_mlm)
  }

```

---

fastcpd\_mean

*Find change points efficiently in mean change models*


---

### Description

`fastcpd_mean()` and `fastcpd.mean()` are wrapper functions of `fastcpd()` to find the mean change. The function is similar to `fastcpd()` except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

### Usage

```
fastcpd_mean(data, ...)
```

```
fastcpd.mean(data, ...)
```

### Arguments

`data`            A matrix, a data frame or a vector.  
`...`            Other arguments passed to `fastcpd()`, for example, `segment_count`.

### Value

A `fastcpd` object.

### See Also

[fastcpd\(\)](#)

### Examples

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 3
  data <- rbind(
    mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(400, mean = rep(50, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(300, mean = rep(2, p), sigma = diag(100, p))
  )
  result <- fastcpd.mean(data)
  summary(result)
}
set.seed(1)
data <- c(rnorm(10000), rnorm(1000) + 1)

```

```

(result_time <- system.time(
  result <- fastcpd.mean(data, r.progress = FALSE, cp_only = TRUE)
))
result@cp_set

set.seed(1)
data <- c(rnorm(10000), rnorm(10000, 1), rnorm(10000))
(result_time <- system.time(
  result <- fastcpd.mean(data, r.progress = FALSE, cp_only = TRUE)
))
result@cp_set

set.seed(1)
data <- c(rnorm(10000), rnorm(10000, 1), rnorm(10000))
(result_time <- system.time(
  result <- fastcpd.mean(
    data, beta = "BIC", cost_adjustment = "BIC",
    r.progress = FALSE, cp_only = TRUE
  )
))
result@cp_set

```

---

fastcpd\_meanvariance *Find change points efficiently in mean variance change models*

---

### Description

[fastcpd\\_meanvariance\(\)](#), [fastcpd.meanvariance\(\)](#), [fastcpd\\_mv\(\)](#), [fastcpd.mv\(\)](#) are wrapper functions of [fastcpd\(\)](#) to find the meanvariance change. The function is similar to [fastcpd\(\)](#) except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

### Usage

```
fastcpd_meanvariance(data, ...)
```

```
fastcpd.meanvariance(data, ...)
```

```
fastcpd_mv(data, ...)
```

```
fastcpd.mv(data, ...)
```

### Arguments

`data`            A matrix, a data frame or a vector.  
`...`            Other arguments passed to [fastcpd\(\)](#), for example, `segment_count`.

**Value**

A `fastcpd` object.

**See Also**

`fastcpd()`

**Examples**

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 1
  result <- fastcpd.mv(
    rbind(
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(300, mean = rep(10, p), sigma = diag(100, p))
    )
  )
  summary(result)
  plot(result)
}
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 4
  result <- fastcpd.mv(
    rbind(
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
      mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
      mvtnorm::rmvnorm(300, mean = rep(10, p), sigma = diag(100, p))
    )
  )
  summary(result)
}

set.seed(1)
data <- c(rnorm(2000, 0, 1), rnorm(2000, 1, 1), rnorm(2000, 1, 2))
(result_time <- system.time(
  result <- fastcpd.variance(data, r.progress = FALSE, cp_only = TRUE)
))
result@cp_set

set.seed(1)
data <- c(rnorm(2000, 0, 1), rnorm(2000, 1, 1), rnorm(2000, 1, 2))
(result_time <- system.time(

```

```

result <- fastcpd.variance(
  data, beta = "BIC", cost_adjustment = "BIC",
  r.progress = TRUE, cp_only = TRUE
)
))
result@cp_set

```

---

fastcpd\_poisson

*Find change points efficiently in Poisson regression models*


---

## Description

`fastcpd_poisson()` and `fastcpd.poisson()` are wrapper functions of `fastcpd()` to find change points in Poisson regression models. The function is similar to `fastcpd()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

## Usage

```
fastcpd_poisson(data, ...)
```

```
fastcpd.poisson(data, ...)
```

## Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>...</code>	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> .

## Value

A `fastcpd` object.

## See Also

[fastcpd\(\)](#)

## Examples

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 1100
  p <- 3
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  delta <- rnorm(p)
  theta_0 <- c(1, 0.3, -1)
  y <- c(
    rpois(500, exp(x[1:500, ] %*% theta_0)),
    rpois(300, exp(x[501:800, ] %*% (theta_0 + delta))),
  )
}

```



```

    rpois(200, exp(x[801:1000, ] ** theta_0)),
    rpois(100, exp(x[1001:1100, ] ** (theta_0 - delta)))
  )
  result <- fastcpd.poisson(cbind(y, x))
  summary(result)
  plot(result)
}

```

---

fastcpd\_ts

*Find change points efficiently in time series data*


---

## Description

`fastcpd_ts()` and `fastcpd.ts()` are wrapper functions for `fastcpd()` to find change points in time series data. The function is similar to `fastcpd()` except that the data is a time series and the family is one of "ar", "var", "arma", "arima" or "garch".

## Usage

```
fastcpd_ts(data, family = NULL, order = c(0, 0, 0), ...)
```

```
fastcpd.ts(data, family = NULL, order = c(0, 0, 0), ...)
```

## Arguments

<code>data</code>	A numeric vector, a matrix, a data frame or a time series object.
<code>family</code>	A character string specifying the family of the time series. The value should be one of "ar", "var", "arima" or "garch".
<code>order</code>	A positive integer or a vector of length less than four specifying the order of the time series. Possible combinations with <code>family</code> are: <ul style="list-style-type: none"> <li>"ar", NUMERIC(1): AR(<math>p</math>) model using linear regression.</li> <li>"var", NUMERIC(1): VAR(<math>p</math>) model using linear regression.</li> <li>"arima", NUMERIC(3): ARIMA(<math>p, d, q</math>) model using <code>forecast::Arima()</code>.</li> <li>"garch", NUMERIC(2): GARCH(<math>p, q</math>) model using <code>tseries::garch()</code>.</li> </ul>
<code>...</code>	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> . One special argument can be passed here is <code>include.mean</code> , which is a logical value indicating whether the mean should be included in the model. The default value is TRUE.

## Value

A `fastcpd` object.

## See Also

[fastcpd\(\)](#)

**Examples**

```

set.seed(1)
n <- 400
w <- rnorm(n + 4, 0, 0.1)
x <- rep(NA, n)
for (i in 1:200) {
  x[i] <- w[i + 4] - 5 / 3 * w[i + 3] + 11 / 12 * w[i + 2] - 5 / 12 * w[i + 1] +
    1 / 6 * w[i]
}
for (i in 201:n) {
  x[i] <- w[i + 4] - 4 / 3 * w[i + 3] + 7 / 9 * w[i + 2] - 16 / 27 * w[i + 1] +
    4 / 27 * w[i]
}
result <- fastcpd.ts(
  x,
  "arma",
  c(0, 4),
  lower = c(-2, -2, -2, -2, 1e-10),
  upper = c(2, 2, 2, 2, Inf),
  line_search = c(1, 0.1, 1e-2),
  trim = 0.05
)
summary(result)
plot(result)

```

---

fastcpd\_var

*Find change points efficiently in VAR(p) models*


---

**Description**

`fastcpd_var()` and `fastcpd.var()` are wrapper functions of `fastcpd_ts()` to find change points in VAR( $p$ ) models. The function is similar to `fastcpd_ts()` except that the data is by default a matrix with row as an observation and thus a formula is not required here.

**Usage**

```
fastcpd_var(data, order = 0, ...)
```

```
fastcpd.var(data, order = 0, ...)
```

**Arguments**

<code>data</code>	A matrix, a data frame or a time series object.
<code>order</code>	A positive integer specifying the order of the VAR model.
<code>...</code>	Other arguments passed to <code>fastcpd()</code> , for example, <code>segment_count</code> .

**Value**

A [fastcpd](#) object.

**See Also**

[fastcpd\(\)](#)

**Examples**

```
set.seed(1)
n <- 300
p <- 2
theta_1 <- matrix(c(-0.3, 0.6, -0.5, 0.4, 0.2, 0.2, 0.2, -0.2), nrow = p)
theta_2 <- matrix(c(0.3, -0.4, 0.1, -0.5, -0.5, -0.2, -0.5, 0.2), nrow = p)
x <- matrix(0, n + 2, p)
for (i in 1:200) {
  x[i + 2, ] <- theta_1 %*% c(x[i + 1, ], x[i, ]) + rnorm(p, 0, 1)
}
for (i in 201:n) {
  x[i + 2, ] <- theta_2 %*% c(x[i + 1, ], x[i, ]) + rnorm(p, 0, 1)
}
result <- fastcpd.var(x, 2)
summary(result)
```

---

fastcpd\_variance

*Find change points efficiently in variance change models*

---

**Description**

[fastcpd\\_variance\(\)](#) and [fastcpd.variance\(\)](#) are wrapper functions of [fastcpd\(\)](#) to find the variance change. The function is similar to [fastcpd\(\)](#) except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

**Usage**

```
fastcpd_variance(data, ...)
```

```
fastcpd.variance(data, ...)
```

**Arguments**

`data` A matrix, a data frame or a vector.  
`...` Other arguments passed to [fastcpd\(\)](#), for example, `segment_count`.

**Value**

A [fastcpd](#) object.

**See Also**[fastcpd\(\)](#)**Examples**

```

set.seed(1)
data <- c(rnorm(300, 0, 1), rnorm(400, 0, 100), rnorm(300, 0, 1))
result <- fastcpd.variance(data)
summary(result)
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 3
  result <- fastcpd.variance(
    rbind(
      mvtnorm::rmvnorm(
        300, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
      ),
      mvtnorm::rmvnorm(
        400, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
      ),
      mvtnorm::rmvnorm(
        300, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
      )
    )
  )
  summary(result)
}

set.seed(1)
data <- c(rnorm(3000, 0, 1), rnorm(3000, 0, 2), rnorm(3000, 0, 1))
(result_time <- system.time(
  result <- fastcpd.variance(data, r.progress = FALSE, cp_only = TRUE)
))
result@cp_set

set.seed(1)
data <- c(rnorm(3000, 0, 1), rnorm(3000, 0, 2), rnorm(3000, 0, 1))
(result_time <- system.time(
  result <- fastcpd.variance(
    data, beta = "BIC", cost_adjustment = "BIC",
    r.progress = FALSE, cp_only = TRUE
  )
))
result@cp_set

```

**Description**

Data set for binary classification of room occupancy from temperature, humidity, light and CO2 measurements. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.

**Usage**

```
occupancy
```

**Format**

A data frame with 9752 rows and 7 variables:

**date** Character in the format "YYYY-MM-DD hh:mm:ss" from 2015-02-11 14:48:00 to 2015-02-18 09:19:00

**Temperature** Temperature in Celsius

**Humidity** Humidity

**Light** Light

**CO2** CO2

**HumidityRatio** Humidity Ratio

**Occupancy** Binary variable with values 0 (unoccupied) and 1

**Source**

<<https://github.com/LuisM78/Occupancy-detection-data>>

---

plot.fastcpd

*Plot the data and the change points for a `fastcpd` object*

---

**Description**

Plot the data and the change points for a `fastcpd` object

**Usage**

```
## S3 method for class 'fastcpd'
plot(
  x,
  color_max_count = Inf,
  data_point_alpha = 0.8,
  data_point_linewidth = 0.5,
  data_point_size = 1,
  legend_position = "none",
  panel_background = ggplot2::element_blank(),
  panel_border = ggplot2::element_rect(fill = NA, colour = "grey20"),
  panel_grid_major = ggplot2::element_line(colour = "grey98"),
```

```

panel_grid_minor = ggplot2::element_line(colour = "grey98"),
segment_separator_alpha = 0.8,
segment_separator_color = "grey",
segment_separator_linetype = "dashed",
strip_background = ggplot2::element_rect(fill = "grey85", colour = "grey20"),
xlab = NULL,
ylab = NULL,
...
)

## S4 method for signature 'fastcpd,missing'
plot(
  x,
  color_max_count = Inf,
  data_point_alpha = 0.8,
  data_point_linewidth = 0.5,
  data_point_size = 1,
  legend_position = "none",
  panel_background = ggplot2::element_blank(),
  panel_border = ggplot2::element_rect(fill = NA, colour = "grey20"),
  panel_grid_major = ggplot2::element_line(colour = "grey98"),
  panel_grid_minor = ggplot2::element_line(colour = "grey98"),
  segment_separator_alpha = 0.8,
  segment_separator_color = "grey",
  segment_separator_linetype = "dashed",
  strip_background = ggplot2::element_rect(fill = "grey85", colour = "grey20"),
  xlab = NULL,
  ylab = NULL,
  ...
)

```

### Arguments

x	A <a href="#">fastcpd</a> object.
color_max_count	Maximum number of colors to use for the plotting of segments.
data_point_alpha	Alpha of the data points.
data_point_linewidth	Linewidth of the data points.
data_point_size	Size of the data points.
legend_position	Position of the legend.
panel_background	Background of the panel.
panel_border	Border of the panel.

panel_grid_major	Major grid lines of the panel.
panel_grid_minor	Minor grid lines of the panel.
segment_separator_alpha	Alpha of the segment separator lines.
segment_separator_color	Color of the segment separator lines.
segment_separator_linetype	Linetype of the segment separator lines.
strip_background	Background of the strip.
xlab	Label for the x-axis.
ylab	Label for the y-axis.
...	Ignored.

**Value**

No return value, called for plotting.

**Examples**

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 1
  x <- mvtnorm::rmvnorm(300, rep(0, p), diag(p))
  theta_0 <- matrix(c(1, -1, 0.5))
  y <- c(
    x[1:100, ] * theta_0[1, ] + rnorm(100, 0, 1),
    x[101:200, ] * theta_0[2, ] + rnorm(100, 0, 1),
    x[201:300, ] * theta_0[3, ] + rnorm(100, 0, 1)
  )
  result <- fastcpd.lm(cbind(y, x), r.clock = "fastcpd_profiler")
  summary(result)
  plot(result)

  if (requireNamespace("RcppClock", quietly = TRUE)) {
    library(RcppClock)
    plot(fastcpd_profiler)
  }
}

```

---

print.fastcpd	<i>Print the call and the change points for a <a href="#">fastcpd</a> object</i>
---------------	--

---

**Description**

Print the call and the change points for a [fastcpd](#) object

**Usage**

```
## S3 method for class 'fastcpd'  
print(x, ...)  
  
## S4 method for signature 'fastcpd'  
print(x, ...)
```

**Arguments**

x	A <a href="#">fastcpd</a> object.
...	Ignored.

**Value**

Return a (temporarily) invisible copy of the [fastcpd](#) object. Called primarily for printing the change points in the model.

---

---

show.fastcpd	<i>Show the available methods for a <a href="#">fastcpd</a> object</i>
--------------	--

---

**Description**

Show the available methods for a [fastcpd](#) object

**Usage**

```
## S3 method for class 'fastcpd'  
show(object)  
  
## S4 method for signature 'fastcpd'  
show(object)
```

**Arguments**

object	A <a href="#">fastcpd</a> object.
--------	-----------------------------------

**Value**

No return value, called for showing a list of available methods for a [fastcpd](#) object.



---

summary.fastcpd	<i>Show the summary of a <a href="#">fastcpd</a> object</i>
-----------------	---

---

**Description**

Show the summary of a [fastcpd](#) object

**Usage**

```
## S3 method for class 'fastcpd'
summary(object, ...)

## S4 method for signature 'fastcpd'
summary(object, ...)
```

**Arguments**

object	A <a href="#">fastcpd</a> object.
...	Ignored.

**Value**

Return a (temporarily) invisible copy of the [fastcpd](#) object. Called primarily for printing the summary of the model including the call, the change points, the cost values and the estimated parameters.

---

transcriptome	<i>Transcription Profiling of 57 Human Bladder Carcinoma Samples</i>
---------------	--

---

**Description**

Transcriptome analysis of 57 bladder carcinomas on Affymetrix HG-U95A and HG-U95Av2 microarrays

**Usage**

```
transcriptome
```

**Format**

A data frame with 2215 rows and 43 variables:

- 3** Individual 3
- 4** Individual 4
- 5** Individual 5

**6** Individual 6  
**7** Individual 7  
**8** Individual 8  
**9** Individual 9  
**10** Individual 10  
**14** Individual 14  
**15** Individual 15  
**16** Individual 16  
**17** Individual 17  
**18** Individual 18  
**19** Individual 19  
**21** Individual 21  
**22** Individual 22  
**24** Individual 24  
**26** Individual 26  
**28** Individual 28  
**30** Individual 30  
**31** Individual 31  
**33** Individual 33  
**34** Individual 34  
**35** Individual 35  
**36** Individual 36  
**37** Individual 37  
**38** Individual 38  
**39** Individual 39  
**40** Individual 40  
**41** Individual 41  
**42** Individual 42  
**43** Individual 43  
**44** Individual 44  
**45** Individual 45  
**46** Individual 46  
**47** Individual 47  
**48** Individual 48  
**49** Individual 49  
**50** Individual 50  
**51** Individual 51  
**53** Individual 53  
**54** Individual 54  
**57** Individual 57

**Source**

<<https://www.ebi.ac.uk/biostudies/arrayexpress/studies/E-TABM-147>>  
 <<https://github.com/cran/ecp/tree/master/data>>

**Examples**

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  result <- fastcpd.mean(transcriptome$"10", trim = 0.005)
  summary(result)
  plot(result)

  result_all <- fastcpd.mean(
    transcriptome,
    beta = (ncol(transcriptome) + 1) * log(nrow(transcriptome)) / 2 * 5,
    trim = 0
  )

  plots <- lapply(
    seq_len(ncol(transcriptome)), function(i) {
      ggplot2::ggplot(
        data = data.frame(
          x = seq_along(transcriptome[, i]), y = transcriptome[, i]
        ),
        ggplot2::aes(x = x, y = y)
      ) +
      ggplot2::geom_line(color = "steelblue") +
      ggplot2::geom_vline(
        xintercept = result_all@cp_set,
        color = "red",
        linetype = "dotted",
        linewidth = 0.5,
        alpha = 0.7
      ) +
      ggplot2::theme_void()
    }
  )

  if (requireNamespace("gridExtra", quietly = TRUE)) {
    gridExtra::grid.arrange(grobs = plots, ncol = 1, nrow = ncol(transcriptome))
  }
}
```

---

 uk\_seatbelts

*UK Seatbelts Data*


---

**Description**

Road Casualties in Great Britain 1969–84.

**Usage**

```
uk_seatbelts
```

**Format**

uk\_seatbelts is a multiple time series, with columns

**DriversKilled** car drivers killed.

**front** front-seat passengers killed or seriously injured.

**rear** rear-seat passengers killed or seriously injured.

**kms** distance driven.

**PetrolPrice** petrol price.

**VanKilled** number of van ('light goods vehicle') drivers.

**law** 0/1: was the law in effect that month?

**Source**

R package **datasets**

**Examples**

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
  requireNamespace("lubridate", quietly = TRUE) &&
  requireNamespace("zoo", quietly = TRUE)
) {
  result_ar <- fastcpd.ar(diff(uk_seatbelts[, "drivers"], 12), 1, beta = "BIC")
  summary(result_ar)
  plot(result_ar)

  result_lm <- suppressMessages(fastcpd.lm(
    diff(uk_seatbelts[, c("drivers", "kms", "PetrolPrice", "law")], lag = 12)
  ))
  cp_dates <- as.Date("1969-01-01", format = "%Y-%m-%d")
  cp_dates <- cp_dates + lubridate::period(month = 1 + result_lm@cp_set + 12)
  cp_dates <- zoo::as.yearmon(cp_dates)

  dates <- zoo::as.yearmon(time(uk_seatbelts))

  uk_seatbelts_df <- data.frame(
    dates = dates,
    drivers = c(uk_seatbelts[, "drivers"]),
    color = as.factor((dates < cp_dates[1]) + (dates < cp_dates[2]))
  )

  ggplot2::ggplot() +
    ggplot2::geom_line(
      data = uk_seatbelts_df,
      mapping = ggplot2::aes(x = dates, y = drivers, color = color)
    ) +

```

```

ggplot2::geom_vline(
  xintercept = cp_dates,
  linetype = "dashed",
  color = "red"
) +
zoo::scale_x_yearmon() +
ggplot2::annotate(
  "text",
  x = cp_dates,
  y = 1025,
  label = as.character(cp_dates),
  color = "blue"
) +
ggplot2::theme_bw() +
ggplot2::theme(legend.position = "none")
}

```

---

variance\_arma

*Variance estimation for ARMA model with change points*


---

### Description

Estimate the variance for each block and then take the average.

### Usage

```
variance_arma(data, p, q, max_order = p * q)
```

```
variance.arma(data, p, q, max_order = p * q)
```

### Arguments

data	A one-column matrix or a vector.
p	The order of the autoregressive part.
q	The order of the moving average part.
max_order	The maximum order of the AR model to consider.

### Value

A numeric value representing the variance.

### Examples

```

set.seed(1)
n <- 300
w <- rnorm(n + 3, 0, 10)
x <- rep(0, n + 3)
for (i in 1:200) {

```

```

x[i + 3] <- 0.1 * x[i + 2] - 0.3 * x[i + 1] + 0.1 * x[i] +
  0.1 * w[i + 2] + 0.5 * w[i + 1] + w[i + 3]
}
for (i in 201:n) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.1 * x[i + 1] - 0.3 * x[i] -
    0.6 * w[i + 2] - 0.1 * w[i + 1] + w[i + 3]
}
(result <- variance.arma(x[-seq_len(3)], p = 3, q = 2))

```

---

variance\_lm

*Variance estimation for linear models with change points*


---

### Description

Estimate the variance for each block and then take the average.

### Usage

```
variance_lm(data, d = 1, block_size = ncol(data) - d + 1, outlier_iqr = Inf)
```

```
variance.lm(data, d = 1, block_size = ncol(data) - d + 1, outlier_iqr = Inf)
```

### Arguments

data	A matrix or a data frame with the response variable as the first column.
d	The dimension of the response variable.
block_size	The size of the blocks to use for variance estimation.
outlier_iqr	The number of interquartile ranges to use as a threshold for outlier detection.

### Value

A numeric value representing the variance.

### Examples

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 300
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta <- rbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
  y <- c(
    x[1:100, ] %*% theta[1, ] + rnorm(100, 0, 3),
    x[101:200, ] %*% theta[2, ] + rnorm(100, 0, 3),
    x[201:n, ] %*% theta[3, ] + rnorm(100, 0, 3)
  )
  (sigma2 <- variance.lm(cbind(y, x)))
}

```

```

set.seed(1)
n <- 300
p <- 4
d <- 2
x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
theta <- cbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
theta <- cbind(theta, theta)
y <- rbind(
  x[1:100, ] %*% theta[, 1:2] +
    mvtnorm::rmvnorm(100, rep(0, d), diag(3, d)),
  x[101:200, ] %*% theta[, 3:4] +
    mvtnorm::rmvnorm(100, rep(0, d), diag(3, d)),
  x[201:n, ] %*% theta[, 5:6] +
    mvtnorm::rmvnorm(100, rep(0, d), diag(3, d))
)
(sigma <- variance.lm(cbind(y, x), d = d))
}

```

variance\_mean

*Variance estimation for mean change models***Description**

Implement Rice estimator for variance in mean change models.

**Usage**

```
variance_mean(data)
```

```
variance.mean(data)
```

**Arguments**

`data` A matrix or a data frame with data points as each row.

**Value**

A matrix representing the variance-covariance matrix or a numeric value representing the variance.

**Examples**

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 3
  data <- rbind(
    mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(400, mean = rep(50, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(300, mean = rep(2, p), sigma = diag(100, p))
  )
  (sigma <- variance.mean(data))
}

```

---

variance_median	<i>Variance estimation for median change models</i>
-----------------	---

---

**Description**

Implement Rice estimator.

**Usage**

```
variance_median(data)
```

```
variance.median(data)
```

**Arguments**

data            A vector of data points.

**Value**

A numeric value representing the variance.

**Examples**

```
(sigma2 <- variance.median(well_log))
```

---

well_log	<i>Well-log Dataset from Numerical Bayesian Methods Applied to Signal Processing</i>
----------	--

---

**Description**

This is the well-known well-log dataset used in many changepoint papers obtained from Alan Turing Institute GitHub repository and licensed under the MIT license.

**Usage**

```
well_log
```

**Format**

A Time-Series of length 4050.

**Source**

<<https://github.com/alan-turing-institute/TCPD>>



**Examples**

```
result <- fastcpd.mean(well_log, trim = 0.001)
summary(result)
plot(result)

if (requireNamespace("matrixStats", quietly = TRUE)) {
  sigma2 <- variance.median(well_log)
  median_loss <- function(data) {
    sum(abs(data - matrixStats::colMedians(data))) / sqrt(sigma2) / 2
  }
  result <- fastcpd(
    formula = ~ x - 1,
    data = cbind.data.frame(x = well_log),
    cost = median_loss,
    trim = 0.002
  )
  summary(result)

  segment_starts <- c(1, result@cp_set)
  segment_ends <- c(result@cp_set - 1, length(well_log))
  residual <- NULL
  for (segment_index in seq_along(segment_starts)) {
    segment <-
      well_log[segment_starts[segment_index]:segment_ends[segment_index]]
    residual <- c(residual, segment - median(segment))
  }

  result@residuals <- matrix(residual)
  result@family <- "mean"
  result@data <- data.frame(x = c(well_log))
  plot(result)
}
```

# Index

## \* datasets

- bitcoin, 3
  - occupancy, 28
  - transcriptome, 33
  - uk\_seatbelts, 35
  - well\_log, 40
- bitcoin, 3
- fastcpd, 4, 4, 8, 13–18, 20, 21, 23–25, 27, 29, 30, 32, 33
- fastcpd(), 4, 6, 12–18, 20–28
- fastcpd-class, 12
- fastcpd.ar (fastcpd\_ar), 12
- fastcpd.ar(), 12
- fastcpd.arima (fastcpd\_arima), 13
- fastcpd.arima(), 13
- fastcpd.arma (fastcpd\_arma), 14
- fastcpd.arma(), 14
- fastcpd.binomial (fastcpd\_binomial), 16
- fastcpd.binomial(), 16
- fastcpd.family, 5, 8, 12
- fastcpd.garch (fastcpd\_garch), 17
- fastcpd.garch(), 17
- fastcpd.lasso (fastcpd\_lasso), 18
- fastcpd.lasso(), 18
- fastcpd.lm (fastcpd\_lm), 19
- fastcpd.lm(), 20
- fastcpd.mean (fastcpd\_mean), 21
- fastcpd.mean(), 21
- fastcpd.meanvariance (fastcpd\_meanvariance), 22
- fastcpd.meanvariance(), 22
- fastcpd.mv (fastcpd\_meanvariance), 22
- fastcpd.mv(), 22
- fastcpd.poisson (fastcpd\_poisson), 24
- fastcpd.poisson(), 24
- fastcpd.ts (fastcpd\_ts), 25
- fastcpd.ts(), 25
- fastcpd.var (fastcpd\_var), 26
- fastcpd.var(), 26
- fastcpd.variance (fastcpd\_variance), 27
- fastcpd.variance(), 27
- fastcpd\_ar, 12
- fastcpd\_ar(), 12
- fastcpd\_arima, 13
- fastcpd\_arima(), 13
- fastcpd\_arma, 14
- fastcpd\_arma(), 14
- fastcpd\_binomial, 16
- fastcpd\_binomial(), 16
- fastcpd\_garch, 17
- fastcpd\_garch(), 17
- fastcpd\_lasso, 18
- fastcpd\_lasso(), 18
- fastcpd\_lm, 19
- fastcpd\_lm(), 20
- fastcpd\_mean, 21
- fastcpd\_mean(), 21
- fastcpd\_meanvariance, 22
- fastcpd\_meanvariance(), 22
- fastcpd\_mv (fastcpd\_meanvariance), 22
- fastcpd\_mv(), 22
- fastcpd\_poisson, 24
- fastcpd\_poisson(), 24
- fastcpd\_ts, 25
- fastcpd\_ts(), 25, 26
- fastcpd\_var, 26
- fastcpd\_var(), 26
- fastcpd\_variance, 27
- fastcpd\_variance(), 27
- forecast::Arima(), 25
- lm(), 5
- occupancy, 28
- plot, fastcpd, missing-method (plot.fastcpd), 29
- plot.fastcpd, 29

`plot.fastcpd()`, 8  
`print, fastcpd-method (print.fastcpd)`, 32  
`print.fastcpd`, 32

`show, fastcpd-method (show.fastcpd)`, 32  
`show.fastcpd`, 32  
`summary, fastcpd-method (summary.fastcpd)`, 33  
`summary.fastcpd`, 33  
`summary.fastcpd()`, 8

`transcriptome`, 33  
`tseries::garch()`, 25

`uk_seatbelts`, 35

`variance.arma (variance_arma)`, 37  
`variance.lm (variance_lm)`, 38  
`variance.mean (variance_mean)`, 39  
`variance.median (variance_median)`, 40  
`variance_arma`, 37  
`variance_lm`, 38  
`variance_mean`, 39  
`variance_median`, 40

`well_log`, 40