Package 'fastcpd'

January 24, 2024

Type Package

Title Fast Change Point Detection via Sequential Gradient Descent

Version 0.10.3

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, utils

Suggests abind, breakfast, changepoint, cpm, ecp, fpop, ggplot2, gridExtra, knitr, lubridate, mcp, mockthat, mosum, mvtnorm, not, numDeriv, rmarkdown, segmented, stepR, strucchange, testthat (>= 3.0.0), wbs, xml2, zoo

LinkingTo progress, Rcpp, RcppArmadillo, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation yes

Author Xingchi Li [aut, cre, cph] (<https://orcid.org/0009-0006-2493-0853>),
Xianyang Zhang [aut, cph],
Trisha Dawn [aut, cph]

Maintainer Xingchi Li <anthony.li@stat.tamu.edu>

Repository CRAN

Date/Publication 2024-01-24 15:20:02 UTC

R topics documented:

bitcoin *Bitcoin Market Price (USD)*

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)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

ggplot2::ggplot(bitcoin, ggplot2::aes(x = date, y = price)) +
  ggplot2::geom_line()

result <- suppressWarnings(
  fastcpd.garch(diff(log(bitcoin$price[200:500])), c(1, 1))
)
summary(result)
bitcoin$date[result@cp_set + 200]
plot(result)
```

`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 = NULL,
  segment_count = 10,
  trim = 0.025,
  momentum_coef = 0,
  k = function(x) 0,
  family = NULL,
  epsilon = 1e-10,
  min_prob = 10^10,
  winsorise_minval = -20,
  winsorise_maxval = 20,
  p = ncol(data) - 1,
  order = c(0, 0, 0),
  cost = NULL,
  cost_gradient = NULL,
  cost_hessian = NULL,
  cp_only = FALSE,
  vanilla_percentage = 0,
  warm_start = FALSE,
  lower = NULL,
  upper = NULL,
  line_search = c(1),
  ...
)
```

Arguments

`formula`	A formula object specifying the model to be fitted. The optional response variable should be on the left hand side of the formula while the covariates should be on the right hand side. The intercept term should be removed from the formula. The response variable is not necessary if the data considered is not of regression type. For example, a mean or variance change model does not necessarily have response variables. By default an intercept column will be added to the data similar to the `lm` function in R. Thus it is suggested that user should remove the intercept term from the formula by appending `- 1` to the formula.

The default formula is suitable for regression data sets with one-dimensional response variable and the rest being covariates without intercept. The naming of variables used in the formula should be consistent with the column names in the data frame provided in `data`.

`data` A data frame containing the data to be segmented where each row denotes each data point. In one-dimensional response variable regression settings, the first column is the response variable while the rest are covariates. The response is not necessary in the case of mean change or variance change, in which case the formula will need to be adjusted accordingly.

`beta` Initial cost value specified in the algorithm in the paper. For the proper choice of a value, please refer to the paper. If not specified, BIC criterion is used to obtain a proper value, i.e., `beta = (p + 1) * log(nrow(data)) / 2`.

`segment_count` Number of segments for initial guess. If not specified, the initial guess on the number of segments is 10.

`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 `trim * nrow(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.

`k` Function on number of epochs in SGD. `k` should be a function taking only a parameter `x` meaning the current number of data points considered since last segmentaion. The return value of the function should be an integer indicating how many epochs should be performed apart from the default update. By default the function returns 0, meaning no multiple epochs will be used to update the parameters. Example usage:

```
k = function(x) {
  if (x < n / segment_count / 4 * 1) 3
  else if (x < n / segment_count / 4 * 2) 2
  else if (x < n / segment_count / 4 * 3) 1
  else 0
}
```

This function will perform 3 epochs for the first quarter of the data, 2 epochs for the second quarter of the data, 1 epoch for the third quarter of the data and no multiple epochs for the last quarter of the data. Experiments show that performing multiple epochs will significantly affect the performance of the algorithm. This parameter is left for the users to tune the performance of the algorithm if the result is not ideal. Details are discussed in the paper.

`family` Family of the model. Can be `"lm"`, `"binomial"`, `"poisson"`, `"lasso"`, `"custom"`, `"ar"`, `"var"`, `"ma"`, `"arima"`, `"garch"` or `NULL`. For simplicity, user can also omit this parameter, indicating that they will be using their own cost functions.

Omitting the parameter is the same as specifying the parameter to be `"custom"` or `NULL`, in which case, users must specify the cost function, with optional gradient and corresponding Hessian matrix functions.

`epsilon` Epsilon to avoid numerical issues. Only used for the Hessian computation in Logistic Regression and Poisson Regression.

`min_prob` Minimum probability to avoid numerical issues. Only used for Poisson Regression.

`winsorise_minval`

Minimum value for the parameter in Poisson Regression to be winsorised.

`winsorise_maxval`

Maximum value for the parameter in Poisson Regression to be winsorised.

`p` Number of covariates in the model. If not specified, the number of covariates will be inferred from the data, i.e., `p = ncol(data) - 1`. This parameter is superseded by `order` in the case of time series models: "ar", "var", "arima".

`order` Order of the AR(p), VAR(p) or ARIMA(p, d, q) model.

`cost` Cost function to be used. This and the following two parameters should not be specified at the same time with `family`. 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:

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

In both methods, users should implement the cost value calculation based on the data provided, where the data parameter can be considered as a segment of the original data frame in the form of a matrix. The first method is used when the cost function has an explicit solution, in which case the cost function value can be calculated directly from the data. The second method is used when the cost function does not have an explicit solution, in which case the cost function value can be calculated from the data and the estimated parameters. In the case of only one `data` argument is provided, `fastcpd` performs the vanilla PELT algorithm since no parameter updating is performed.

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

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

The gradient function should take two parameters, the first one being a segment of the data in the format of a matrix, the second one being the estimated parameters. The gradient function should return the gradient of the cost function with respect to the data and parameters.

`cost_hessian` Hessian function for the custom cost function. Similar to the gradient function, the Hessian function should take two parameters, the first one being a segment of the data in the format of a matrix, the second one being the estimated parameters. The Hessian function should return the Hessian matrix of the cost function with respect to the data and parameters.

`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 `plot` can be used to visualize the results for a built-in model. `cp_only` 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 `cp_only` to be `TRUE` will help with the computational cost.

`vanilla_percentage` How many of the data should be processed through vanilla PELT. Range should be between 0 and 1. The `fastcpd` algorithm is based on gradient descent and thus a starting estimate can be crucial. At the beginning of the algorithm, vanilla PELT can be performed to obtain a relatively accurate estimate of the parameters despite the small amount of the data being used. If set to be 0, all data will be processed through sequential gradient descnet. If set to be 1, all data will be processed through vaniall PELT. If the cost function have an explicit solution, i.e. does not depend on coefficients like the mean change case, this parameter will be set to be 1. If the value is set to be between 0 and 1, the first `vanilla_percentage * nrow(data)` data points will be processed through vanilla PELT and the rest will be processed through sequential gradient descent.

`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 `"glm"` families.

`lower` Lower bound for the parameters. Used to specify the domain of the parameter after each gradient descent step. If not specified, the lower bound will be set to be `-Inf` for all parameters.

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

`line_search` If a vector of numeric values are provided, line search will be performed to find the optimal step size for each update.

`...` Parameters specifically used for time series models. As of the current implementation, only `include.mean` will not be ignored and used in the ARIMA or GARCH model. `r.progress` now will not be ignored and used to control the progress bar. By default the progress bar will be shown and to disable it, set `r.progress = FALSE`.

Value

A class `fastcpd` object.

Gallery

https://fastcpd.xingchi.li/articles/gallery.html

References

Zhang X, Dawn T (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. https://proceedings.mlr.press/v206/zhang23b.html.

Examples

```
for (package in c("ggplot2", "mvtnorm")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = TRUE
  )
}

### linear regression with one-dimensional covariate
library(fastcpd)
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(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  family = "lm"
)
plot(result)
summary(result)

### custom logistic regression
library(fastcpd)
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(
  formula = y ~ . - 1,
  data = data,
  family = "binomial"
))
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 %o% 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,
  k = function(x) 1,
  epsilon = 1e-5,
  cost = logistic_loss,
  cost_gradient = logistic_loss_gradient,
  cost_hessian = logistic_loss_hessian
)
summary(result_custom_two_epochs)

### custom cost function huber regression
library(fastcpd)
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)
```

`fastcpd-class` *An S4 class to store the output created with fastcpd*

Description

This S4 class stores the output from fastcpd. 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.

`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` *Find change points efficiently in AR(p) models*

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 or a vector of length three with the last two elements being zeros specifying the order of the AR model.
`...`	Other arguments passed to `fastcpd`, for example, `segment_count`. One special argument can be passed here is `include.mean`, which is a logical value indicating whether the mean should be included in the model. The default value is `TRUE`.

Value

A class `fastcpd` object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

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 `fastcpd`, for example, `segment_count`. One special argument can be passed here is `include.mean`, which is a logical value indicating whether the mean should be included in the model. The default value is `TRUE`.

Value

A class `fastcpd` object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

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.arima(
  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 class `fastcpd` object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

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)
  )
)
summary(result)
plot(result)
```

`fastcpd.binomial` *Find change points efficiently in logistic regression models*

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

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 class fastcpd object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

set.seed(1)
x <- matrix(rnorm(1500, 0, 1), ncol = 5)
theta <- rbind(rnorm(5, 0, 1), rnorm(5, 2, 1))
y <- c(
  rbinom(125, 1, 1 / (1 + exp(-x[1:125, ] %*% theta[1, ]))),
  rbinom(175, 1, 1 / (1 + exp(-x[126:300, ] %*% theta[2, ])))
)
result <- suppressWarnings(fastcpd.binomial(cbind(y, x)))
summary(result)
plot(result)
```

---

fastcpd.garch *Find change points efficiently in GARCH(p, q) models*

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

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

Value

A class fastcpd object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

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 class fastcpd object.

Examples

```
for (package in c("ggplot2", "mvtnorm")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = 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)
)
result <- fastcpd.lasso(cbind(y, x), k = function(x) if (x < 30) 1 else 0)
summary(result)
plot(result)
```

`fastcpd.lm` *Find change points efficiently in linear regression models*

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 class `fastcpd` object.

Examples

```
for (package in c("ggplot2", "mvtnorm")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = 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[201:300, ] %*% theta_0[3, ] + rnorm(100, 0, 3)
)
result <- fastcpd.lm(cbind(y, x))
summary(result)
plot(result)
```

`fastcpd.ma` *Find change points efficiently in MA(q) models*

Description

`fastcpd_ma` and `fastcpd.ma` are wrapper functions of `fastcpd` to find change points in MA(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.ma(data, order = 0, ...)

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

Arguments

`data`	A numeric vector, a matrix, a data frame or a time series object.
`order`	A positive integer or a vector of length three with the first two elements being zeros specifying the order of the MA model.
`...`	Other arguments passed to `fastcpd`, for example, `segment_count`. One special argument can be passed here is `include.mean`, which is a logical value indicating whether the mean should be included in the model. The default value is `TRUE`.

Value

A class `fastcpd` object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

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 <- suppressMessages(
  fastcpd.ma(x, 4, include.mean = FALSE, trim = 0)
)
summary(result)
plot(result)
```

`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 class `fastcpd` object.

Examples

```
if (!requireNamespace("mvtnorm", quietly = TRUE)) utils::install.packages(
  "mvtnorm", repos = "https://cloud.r-project.org", quiet = 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)
```

`fastcpd.meanvariance` *Find change points efficiently in 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 class `fastcpd` object.

Examples

```
if (!requireNamespace("mvtnorm", quietly = TRUE)) utils::install.packages(
  "mvtnorm", repos = "https://cloud.r-project.org", quiet = TRUE
)

set.seed(1)
p <- 3
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(50, 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(50, p))
  )
)
summary(result)
```

`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

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 class fastcpd object.

Examples

```
for (package in c("ggplot2", "mvtnorm")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = 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` is a wrapper function for `fastcpd` to find change points in time series data. The function is similar to `fastcpd` except that the data is a time series data and the family is one of `"ar"`, `"var"`, `"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

`data`	A numeric vector, a matrix, a data frame or a time series object.
`family`	A character string specifying the family of the time series. The value should be one of `"ar"`, `"var"`, `"arima"` or `"garch"`.
`order`	A positive integer or a vector of length less than four specifying the order of the time series. Possible combinations with `family` are: • ar, NUMERIC(1): AR(p) model using linear regression. • ar, NUMERIC(3): ARIMA(p, 0, 0) model using `forecast::Arima`, where p is the first element of the vector. • var, NUMERIC(1): VAR(p) model using linear regression. • ma, NUMERIC(1): MA(q) model using `forecast::Arima`. • ma, NUMERIC(3): ARIMA(0, 0, q) model using `forecast::Arima`, where q is the third element of the vector. • arima, NUMERIC(3): ARIMA(p, d, q) model using `forecast::Arima`. • garch, NUMERIC(2): GARCH(p, q) model using `tseries::garch`.
`...`	Other arguments passed to `fastcpd`, for example, `segment_count`. One special argument can be passed here is `include.mean`, which is a logical value indicating whether the mean should be included in the model. The default value is `TRUE`.

Value

A class `fastcpd` object.

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
```

```
)

set.seed(1)
n <- 600
x <- rep(0, n + 1)
for (i in 1:300) {
  x[i + 1] <- 0.8 * x[i] + rnorm(1, 0, 2)
}
for (i in 301:n) {
  x[i + 1] <- 0.1 * x[i] + rnorm(1, 0, 2)
}
result <- fastcpd.ts(
  x[1 + seq_len(n)],
  "ar",
  c(1, 0, 0),
  include.mean = FALSE,
  trim = 0,
  beta = (1 + 1 + 1) * log(n) / 2 * 3
)
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

`data`	A matrix, a data frame or a time series object.
`order`	A positive integer specifying the order of the VAR model.
`...`	Other arguments passed to `fastcpd`, for example, `segment_count`.

Value

A class `fastcpd` object.

Examples

```
set.seed(1)
n <- 800
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 1:500) {
  x[i + 2, ] <- theta_1 %*% c(x[i + 1, ], x[i, ]) + rnorm(p, 0, 1)
}
for (i in 501:800) {
  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 class `fastcpd` object.

Examples

```
if (!requireNamespace("mvtnorm", quietly = TRUE)) utils::install.packages(
  "mvtnorm", repos = "https://cloud.r-project.org", quiet = 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)
```

---

occupancy *Occupancy Detection Data 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, ...)

## S4 method for signature 'fastcpd,missing'
plot(x, y, ...)
```

Arguments

x	fastcpd object.
...	Ignored.
y	Ignored.

Value

No return value, called for plotting.

print.fastcpd *Print the call and the change points for a* fastcpd *object*

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	fastcpd object.
...	Ignored.

Value

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

---

show.fastcpd *Show the available methods for a* fastcpd *object*

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	fastcpd object.

Value

No return value, called for showing a list of available methods for a fastcpd object.

---

summary.fastcpd *Show the summary of a* fastcpd *object*

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	fastcpd 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` *Transcription Profiling of 57 Human Bladder Carcinoma Samples*

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

```
for (package in c("ggplot2", "gridExtra")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = 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 = "blue") +
      ggplot2::geom_vline(
        xintercept = result_all@cp_set,
        color = "red",
        linetype = "dotted",
        linewidth = 0.5,
        alpha = 0.7
      ) +
      ggplot2::theme_void()
  }
)

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

```
for (package in c("ggplot2", "lubridate", "zoo")) {
  if (!requireNamespace(package, quietly = TRUE)) utils::install.packages(
    package, repos = "https://cloud.r-project.org", quiet = TRUE
  )
}

result_ar <- fastcpd.ar(diff(uk_seatbelts[, "drivers"], lag = 12), 1)
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)

uk_seatbelts_df <- data.frame(
  dates = zoo::as.yearmon(time(uk_seatbelts)),
  drivers = c(uk_seatbelts[, "drivers"])
)

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

`well_log` *Well-log Dataset from Numerical Bayesian Methods Applied to Signal Processing*

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. Outliers with value less or equal to 1e5 are removed.

Usage

```
well_log
```

Format

A Time-Series of length 3989.

Source

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

Examples

```
if (!requireNamespace("ggplot2", quietly = TRUE)) utils::install.packages(
  "ggplot2", repos = "https://cloud.r-project.org", quiet = TRUE
)

result <- fastcpd.mean(well_log, trim = 0.002)
summary(result)
plot(result)
```

Index