Package 'LoTTA'

May 7, 2026

Type Package

Title Bayesian Inference in Regression Discontinuity Designs

Version 0.1.1

Description Implementation of the LoTTA (Local Trimmed Taylor Approximation) model described in ``Bayesian Regression Discontinuity Design with Unknown Cutoff'' by Kowalska, van de Wiel, van der Pas (2024) <doi:10.48550/arXiv.2406.11585>.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

Imports stats, bayestestR, utils

Depends R (>= 4.4.0), runjags, ggplot2, ggpubr

NeedsCompilation no

Author Julia Kowalska [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-6559-4354>)

Maintainer Julia Kowalska <j.m.kowalska@vu.nl>

Repository CRAN

Date/Publication 2025-07-22 11:20:48 UTC

Contents

LoTTA_fuzzy_BIN *LoTTA_fuzzy_BIN*

Description

Function that fits LoTTA model to the fuzzy RD data with binary outcomes with an either known or unknown/suspected cutoff. It supports two types of priors on the cutoff location: a scaled beta distribution of the form beta(alpha,beta)(cub-clb)+clb and a discrete distribution with the support of the form cstart+grid i for i=0,...,(cend-cstart)/grid. The score does NOT have to be normalized beforehand. We recommend NOT to transform the data before imputing it into the function, except for initial trimming of the score which should be done beforehand. The further trimming for the sensitivity analysis can be done through the function, which ensures that the data is normalized before the trimming.

Usage

```
LoTTA_fuzzy_BIN(
  x,
  t,
  y,
  c_prior,
  jlb = 0.2,
  ci = 0.95,
  trimmed = NULL,
  outcome_prior = list(pr = 1e-04),
  n_min = 25,
 param = c("c", "j", "kl", "kr", "eff", "a0l", "a1l", "a2l", "a3l", "a0r", "a1r", "a2r",
  "a3r", "b1lt", "a1lt", "a2lt", "b2lt", "b1rt", "a1rt", "a2rt", "b2rt", "k1t", "k2t"),
  normalize = TRUE,
  n.chains = 4,
  burnin = 10000,
  sample = 1000,
  adapt = 500,
  inits = NULL,
  method = "parallel",
  seed = NULL,
  ...
)
```

Arguments

x	• is the score data
t	• is the treatment allocation data
y	• is the binary outcome data
c_prior	• specifies the cutoff prior in case of the unknown cutoff or the cutoff point if the cutoff is known. Takes as value a number if the cutoff is known or a

list of values otherwise. For a continuous prior the list requires the following elements: clb - left end of the interval cub - right end of the interval in which the scaled and translated beta distribution is defined, alpha (optional) - shape parameter, default value = 1, beta (optional) - shape parameter, default value = 1. For a discrete prior the list requires the following elements: cstart - first point with positive prior mass, cend - last point with positive prior mass, grid - distance between the consecutive points in the support weights (optional) - vector of weights assigned to each point in the support, default is vector of 1's (uniform distribution)

`jlb` • minimum jump size

`ci` • specifies the probability level 1-alpha for the highest posterior density intervals; default is ci = 0.95

`trimmed` • takes as a value NULL or a vector of two values. It specifies potential trimming of the data. If set to NULL no trimming is applied to the data. If a list of two values is provided the data is trimmed to include data points with the score x in between those values; default is trimmed=NULL

`outcome_prior` • takes as a value a list with elements 'pr'. 'pr' specifies precision in the normal priors on the coefficients in the outcome function; default is list('pr'=0.0001)

`n_min` • specifies the minimum number of data points to which a cubic part of the outcome function is fit to ensure stability of the sampling procedure; default is n_min=25

`param` • takes as a value a vector with names of the parameters that are to be sampled; default is the list of all parameters

`normalize` • specifies if the data is to be normalized. The data is normalized as follows. If the prior is continuous: x_normalized=(x-d)/s, where d=(min(x)+max(x))*0.5 and s=max(x)-min(x). If the prior is discrete: x_normalized=x/s, where s=10^m, where m is chosen so that |max(abs(x))-1| is minimal. The priors inside the model are specified for the normalized data, in extreme cases not normalizing the data may lead to unreliable results; default is normalize=TRUE

`n.chains` • specifies the number of chains in the MCMC sampler; default is n.chains=4

`burnin` • specifies the number of burnin iterations without the adaptive iterations; default is burnin=5000

`sample` • specifies the number of samples per chain; default is samples=5000

`adapt` • specifies the number of adaptive iterations in the MCMC sampler; default is adapt=1000

`inits` • initial values for the sampler. By default the initial values are sampled inside the function. To run LoTTA with a method other than "parallel" inits must be set to NA or to a user defined value. If the user wants to provide its own values please refer to run.jags manual; default inits=NULL

`method` • set to default as 'parallel', which allows to sample the chains in parallel reducing computation time. To read more about possible method values type ?run.jags; default method='parallel'

`seed` • specifies the seed for replicability purposes; default is seed=NULL

`...` • other arguments of run.jags function. For more details type ?run.jags

Value

The function returns the list with the elements:

- Effect_estimate: contains a list with MAP estimate and HDI of the treatment effect, the cutoff location (if unknown) and the discontinuity size in the treatment probability function (compliance rate at c) on the original, unnormalized scale;
- JAGS_output: contains output of the run.jags function for the normalized data if normalize=TRUE, based on this output mixing of the chains can be assessed;
- Samples: contains posterior samples of the treatment effect (eff), cutoff location (c) if unknown, and compliance rate (j);
- Normalized_data: contains a list of the normalized data (if normalized=TRUE) and the parameters used to normalize the data (see arg normalize);
- Priors: contains a list of the priors' parameters ;
- Inits contains the list of initial values and .RNG.seed value

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(t)
}

funB <- function(x) {
  y = x
  x2 = x[x >= 0]
  x1 = x[x < 0]
  y[x < 0] = 1 / (1 + exp(-2 * x1)) - 0.5 + 0.4
  y[x >= 0] = (log(x2 * 2 + 1) - 0.15 * x2^2) * 0.6 - 0.20 + 0.4
  return(y)
}
```

```
funB_sample_binary <- function(x) {
  y = x
  P = funB(x)
  for (j in 1:length(x)) {
    y[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(y)
}

## Toy example - for the function check only! ##
N=100
x = sort(runif(N, -1, 1))
t = sample_prob55(x)
y = funB_sample_binary(x)
c_prior=0 # known cutoff c=0

# running LoTTA model on fuzzy RDD with binary outcomes;
out = LoTTA_fuzzy_BIN(x,t,y,c_prior,burnin = 50,sample = 50,adapt=10,n.chains=1
,method = 'simple',inits = NA)

## Use case example ##

  N=500
  x = sort(runif(N, -1, 1))
  t = sample_prob55(x)
  y = funB_sample_binary(x)
  # comment out to try different priors:
   c_prior=list(clb=-0.25,cub=0.25) # uniform prior on the interval [-0.25,0.25]
  # c_prior=list(cstart=-0.25,cend=0.25,grid=0.05) # uniform discrete prior
  # on -0.25, -0.2, ..., 0.25
  # c_prior=0 # known cutoff c=0

  # running LoTTA model on fuzzy RDD with binary outcomes and unknown cutoff;
  # cutoff = 0, compliance rate = 0.55, treatment effect = -0.3636364
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_fuzzy_BIN(x,t,y,c_prior,burnin = 10000,sample = 5000,adapt=1000)

  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit of the outcome function
  LoTTA_plot_outcome(out,nbins = 60)
  # plot posterior fit of the treatment probablity function
  LoTTA_plot_treatment(out,nbins = 60)
  # plot dependence of the treatment effect on the cutoff location
  LoTTA_plot_effect(out)
```

LoTTA_fuzzy_CONT *LoTTA_fuzzy_CONT*

Description

Function that fits LoTTA model to the fuzzy RD data with continuous outcomes with an either known or unknown/suspected cutoff. It supports two types of priors on the cutoff location: a scaled beta distribution of the form beta(alpha,beta)(cub-clb)+clb and a discrete distribution with the support of the form cstart+grid i for i=0,...,(cend-cstart)/grid. The score does NOT have to be normalized beforehand. We recommend NOT to transform the data before imputing it into the function, except for initial trimming of the score which should be done beforehand. The further trimming for the sensitivity analysis can be done through the function, which ensures that the data is normalized before the trimming.

Usage

```
LoTTA_fuzzy_CONT(
  x,
  t,
  y,
  c_prior,
  jlb = 0.2,
  ci = 0.95,
  trimmed = NULL,
  outcome_prior = list(pr = 1e-04, shape = 0.01, scale = 0.01),
  n_min = 25,
 param = c("c", "j", "kl", "kr", "eff", "a0l", "a1l", "a2l", "a3l", "a0r", "a1r", "a2r",
  "a3r", "b1lt", "a1lt", "a2lt", "b2lt", "b1rt", "a1rt", "a2rt", "b2rt", "k1t", "k2t",
    "tau1r", "tau2r", "tau1l", "tau2l"),
  normalize = TRUE,
  n.chains = 4,
  burnin = 10000,
  sample = 1000,
  adapt = 500,
  inits = NULL,
  method = "parallel",
  seed = NULL,
  ...
)
```

Arguments

x	• is the score data
t	• is the treatment allocation data
y	• is the binary outcome data
c_prior	• specifies the cutoff prior in case of the unknown cutoff or the cutoff point if the cutoff is known. Takes as value a number if the cutoff is known or a

list of values otherwise. For a continuous prior the list requires the following elements: clb - left end of the interval cub - right end of the interval in which the scaled and translated beta distribution is defined, alpha (optional) - shape parameter, default value = 1, beta (optional) - shape parameter, default value = 1. For a discrete prior the list requires the following elements: cstart - first point with positive prior mass, cend - last point with positive prior mass, grid - distance between the consecutive points in the support weights (optional) - vector of weights assigned to each point in the support, default is vector of 1's (uniform distribution)

`jlb` • minimum jump size

`ci` • specifies the probability level 1-alpha for the highest posterior density intervals; default is ci = 0.95

`trimmed` • takes as a value NULL or a vector of two values. It specifies potential trimming of the data. If set to NULL no trimming is applied to the data. If a list of two values is provided the data is trimmed to include data points with the score x in between those values; default is trimmed=NULL

`outcome_prior` • takes as a value a list with elements 'pr' and 'shape', 'scale'. 'pr' specifies precision in the normal priors on the coefficients in the outcome function. 'shape' and 'scale' specify the shape and scale parameters in the gamma prior on the precision of the error terms; default is list('pr'= 0.0001,'shape'= 0.01,'scale'= 0.01)

`n_min` • specifies the minimum number of data points to which a cubic part of the outcome function is fit to ensure stability of the sampling procedure; default is n_min=25

`param` • takes as a value a vector with names of the parameters that are to be sampled; default is the list of all parameters

`normalize` • specifies if the data is to be normalized. The data is normalized as follows. If the prior is continuous: x_normalized=(x-d)/s, where d=(min(x)+max(x))*0.5 and s=max(x)-min(x), If the prior is discrete: x_normalized=x/s, where s=10^m, where m is chosen so that |max(abs(x))-1| is minimal. The outcome data is normalized as follows: y_normalized=(y-mu)/sd, where mu=mean(y) and sd=sd(y). The priors inside the model are specified for the normalized data, in extreme cases not normalizing the data may lead to unreliable results; default is normalize=TRUE

`n.chains` • specifies the number of chains in the MCMC sampler; default is n.chains=4

`burnin` • specifies the number of burnin iterations without the adaptive iterations; default is burnin=5000

`sample` • specifies the number of samples per chain; default is samples=5000

`adapt` • specifies the number of adaptive iterations in the MCMC sampler; default is adapt=1000

`inits` • initial values for the sampler. By default the initial values are sampled inside the function. To run LoTTA with a method other than "parallel" inits must be set to NA or to a user defined value. If the user wants to provide its own values please refer to run.jags manual; default inits=NULL

method	• set to default as 'parallel', which allows to sample the chains in parallel reducing computation time. To read more about possible method values type ?run.jags; default method='parallel'
seed	• specifies the seed for replicability purposes; default is seed=NULL
...	• other arguments of run.jags function. For more details type ?run.jags

Value

The function returns the list with the elements:

- Effect_estimate: contains a list with MAP estimate and HDI of the treatment effect, the cutoff location (if unknown) and the discontinuity size in the treatment probability function (compliance rate at c) on the original, unnormalized scale;
- JAGS_output: contains output of the run.jags function for the normalized data if normalize=TRUE, based on this output mixing of the chains can be assessed;
- Samples: contains posterior samples of the treatment effect (eff), cutoff location (c) if unknown, and compliance rate (j);
- Normalized_data: contains a list of the normalized data (if normalized=TRUE) and the parameters used to normalize the data (see arg normalize);
- Priors: contains a list of the priors' parameters ;
- Inits contains the list of initial values and .RNG.seed value

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(t)
}

funB <- function(x) {
```

```
  y = x
  x2 = x[x >= 0]
  x1 = x[x < 0]
  y[x < 0] = 1 / (1 + exp(-2 * x1)) - 0.5 + 0.4
  y[x >= 0] = (log(x2 * 2 + 1) - 0.15 * x2^2) * 0.6 - 0.20 + 0.4
  return(y)
}

funB_sample <- function(x) {
  y = funB(x)+ rnorm(length(x), 0, 0.1)
  return(y)
}

## Toy example - for the function check only! ##
N=100
x = sort(runif(N, -1, 1))
t = sample_prob55(x)
y = funB_sample(x)
c_prior=0 # known cutoff c=0

# running LoTTA model on fuzzy RDD with continuous outcomes;
out = LoTTA_fuzzy_CONT(x,t,y,c_prior,burnin = 50,sample = 50,adapt=10,n.chains=1
,method = 'simple',inits = NA)

## Use case example ##

  N=500
  x = sort(runif(N, -1, 1))
  t = sample_prob55(x)
  y = funB_sample(x)
  # comment out to try different priors:
  c_prior=list(clb=-0.25,cub=0.25) # uniform prior on the interval [-0.25,0.25]
  # c_prior=list(cstart=-0.25,cend=0.25,grid=0.05) # uniform discrete prior
  # on -0.25, -0.2, ..., 0.25
  # c_prior=0 # known cutoff c=0

  # running LoTTA model on fuzzy RDD with continuous outcomes and unknown cutoff;
  # cutoff = 0, compliance rate = 0.55, treatment effect = -0.3636364
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_fuzzy_CONT(x,t,y,c_prior,burnin = 10000,sample = 5000,adapt=1000)

  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit of the outcome function
  LoTTA_plot_outcome(out)
  # plot posterior fit of the treatment probablity function
  LoTTA_plot_treatment(out,nbins = 60)
  # plot dependence of the treatment effect on the cutoff location
  LoTTA_plot_effect(out,nbins = 5)
```

LoTTA_plot_effect *LoTTA_plot_effect*

Description

Function that visualizes the impact of the cutoff location on the treatment effect estimate. It plots too figures. The bottom figure depicts the posterior density of the cutoff location. The top figure depicts the box plot of the treatment effect given the cutoff point. If the prior on the cutoff location was discrete each box corresponds to a distinct cutoff point. If the prior was continuous each box correspond to an interval of cutoff values (the number of intervals can be changed through nbins).

Usage

```
LoTTA_plot_effect(
  LoTTA_posterior,
  nbins = 10,
  probs = c(0.025, 0.975),
  x_lab = "Cutoff location",
  y_lab1 = "Treatment effect",
  y_lab2 = "Density of cutoff location",
  title = "Cutoff location vs. Treatment effect",
  axis.text = element_text(family = "sans", size = 10),
  text = element_text(family = "serif"),
  plot.theme = theme_classic(base_size = 14),
  plot.title = element_text(hjust = 0.5),
  ...
)
```

Arguments

LoTTA_posterior	• output of one of the LoTTA functions (LoTTA_fuzzy_CONT, LoTTA_fuzzy_BIN) with all parameters sampled (the default option in those functions)
nbins	• number of bins to aggregate the values of the posterior cutoff location
probs	• list of two quantiles that limit the range of cutoff values displayed on the plots
x_lab	• label of the x-axis
y_lab1	• label of the y-axis of the bottom plot
y_lab2	• label of the y-axis of the top plot
title	• title of the plot
axis.text	• can be any value that is accepted in the argument *axis.text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one axis.text=element_text(family = "sans",size = 10)

text	• can be any value that is accepted in the argument *text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one text=element_text(family='serif')
plot.theme	• ggplot2 plot theme (see https://ggplot2.tidyverse.org/reference/ggtheme.html) possibly with additional arguments, it takes the default value plot.theme=theme_classic(base_size = 14),
plot.title	• can be any value that is accepted in the argument *plot.title* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot title plot.title = element_text(hjust = 0.5)
...	• other arguments of the *theme* function, refer to ggplot2 manual

Value

ggplot2 object

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(t)
}

funB <- function(x) {
  y = x
  x2 = x[x >= 0]
  x1 = x[x < 0]
  y[x < 0] = 1 / (1 + exp(-2 * x1)) - 0.5 + 0.4
  y[x >= 0] = (log(x2 * 2 + 1) - 0.15 * x2^2) * 0.6 - 0.20 + 0.4
  return(y)
}

funB_sample <- function(x) {
```

```
  y = funB(x)+ rnorm(length(x), 0, 0.1)
  return(y)
}

## Use case example ##

  N=500
  x = sort(runif(N, -1, 1))
  t = sample_prob55(x)
  y = funB_sample(x)
  # comment out to try different priors:
   c_prior=list(clb=-0.25,cub=0.25) # uniform prior on the interval [-0.25,0.25]
  # c_prior=list(cstart=-0.25,cend=0.25,grid=0.05) # uniform discrete prior
  # on -0.25, -0.2, ..., 0.25
  # c_prior=0 # known cutoff c=0

  # running LoTTA model on fuzzy RDD with continuous outcomes and unknown cutoff;
  # cutoff = 0, compliance rate = 0.55, treatment effect = -0.3636364
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_fuzzy_CONT(x,t,y,c_prior,burnin = 10000,sample = 5000,adapt=1000)

  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit of the outcome function
  LoTTA_plot_outcome(out,nbins = 60)
  # plot posterior fit of the treatment probablity function
  LoTTA_plot_treatment(out,nbins = 60)
  # plot dependence of the treatment effect on the cutoff location
  LoTTA_plot_effect(out)
```

`LoTTA_plot_effect_DIS` *Function that visualizes the impact of the cutoff location on the treatment effect estimate. It plots too figures. The bottom figure depicts the posterior density of the cutoff location. The top figure depicts the box plot of the treatment effect given the cutoff point. If the prior on the cutoff location was discrete each box corresponds to a distinct cutoff point. If the prior was continuous each box correspond to an interval of cutoff values (the number of intervals can be changed through nbins).*

Description

Function that visualizes the impact of the cutoff location on the treatment effect estimate. It plots too figures. The bottom figure depicts the posterior density of the cutoff location. The top figure depicts the box plot of the treatment effect given the cutoff point. If the prior on the cutoff location was discrete each box corresponds to a distinct cutoff point. If the prior was continuous each box correspond to an interval of cutoff values (the number of intervals can be changed through nbins).

Usage

```
LoTTA_plot_effect_DIS(
  LoTTA_posterior,
  probs = c(0.025, 0.975),
  x_lab = "Cutoff location",
  y_lab1 = "Treatment effect",
  y_lab2 = "Density of cutoff location",
  title = "Cutoff location vs. Treatment effect",
  axis.text = element_text(family = "sans", size = 10),
  text = element_text(family = "serif"),
  plot.theme = theme_classic(base_size = 14),
  plot.title = element_text(hjust = 0.5),
  ...
)
```

Arguments

LoTTA_posterior	• output of one of the LoTTA functions (LoTTA_fuzzy_CONT, LoTTA_fuzzy_BIN) with all parameters sampled (the default option in those functions)
probs	• list of two quantiles that limit the range of cutoff values displayed on the plots
x_lab	• label of the x-axis
y_lab1	• label of the y-axis of the bottom plot
y_lab2	• label of the y-axis of the top plot
title	• title of the plot
axis.text	• can be any value that is accepted in the argument *axis.text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one axis.text=element_text(family = "sans",size = 10)
text	• can be any value that is accepted in the argument *text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one text=element_text(family='serif')
plot.theme	• ggplot2 plot theme (see https://ggplot2.tidyverse.org/reference/ggtheme.html) possibly with additional arguments, it takes the default value plot.theme=theme_classic(base_size = 14),
plot.title	• can be any value that is accepted in the argument *plot.title* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot title plot.title = element_text(hjust = 0.5)
...	• other arguments of the *theme* function, refer to ggplot2 manual

Value

ggplot2 object

LoTTA_plot_outcome *LoTTA_plot_outcome*

Description

Function that plots the median (or another quantile) of the LoTTA posterior outcome function along with the quanatile-based credible interval. The function is plotted on top of the binned input data. To bin the data, the score data is divided into bins of fixed length, then the average outcome in each bin is calculated. The average outcomes are plotted against the average values of the score in the corresponding bins. The data is binned separately on each side of the cutoff, the cutoff is marked on the plot with a dotted line. In case of an unknown cutoff, the MAP estimate is used.

Usage

```
LoTTA_plot_outcome(
  LoTTA_posterior,
  nbins = 100,
  probs = c(0.025, 0.5, 0.975),
  n_eval = 200,
  col_line = "#E69F00",
  size_line = 0.1,
  alpha_interval = 0.35,
  col_dots = "gray",
  size_dots = 3,
  alpha_dots = 0.6,
  col_cutoff = "black",
  title = "Outcome function",
  subtitle = NULL,
  y_lab = "",
  x_lab = expression(paste(italic("x"), " - score")),
  plot.theme = theme_classic(base_size = 14),
  legend.position = "none",
  name_legend = "Legend",
  labels_legend = "median outcome fun.",
  text = element_text(family = "serif"),
  legend.text = element_text(size = 14),
  plot.title = element_text(hjust = 0.5),
  plot.subtitle = element_text(hjust = 0.5),
  ...
)
```

Arguments

LoTTA_posterior

- output of one of the LoTTA functions (LoTTA_sharp_CONT, LoTTA_fuzzy_CONT, LoTTA_sharp_BIN, LoTTA_fuzzy_BIN) with all the parameters sampled (the default option in those functions)

nbins	• number of bins to aggregate the input data
probs	• list of three quantiles, the first and the last one define the quanatile-based credible interval, the middle value defines the quantile of the posterior function to plot; by default the quantiles correspond to the median posterior function and 95% credible interval probs=c(0.025,0.5,0.975)
n_eval	• n_eval*range(x) is the number of points at which each posterior function is evaluated, the higher number means slower computing time and a smoother plot; default n_eval=200
col_line	• the color of the line and the band
size_line	• thickness of the line
alpha_interval	• alpha value of the band, lower values correspond to a more transparent color
col_dots	• color of the dots that correspond to the binned data
size_dots	• size of the dots that correspond to the binned data
alpha_dots	• transparency of the dots that correspond to the binned data, lower values correspond to a more transparent color
col_cutoff	• color of the dotted line at the cutoff
title	• title of the plot
subtitle	• subtitle of the plot
y_lab	• label of the y-axis
x_lab	• label of the x-axis
plot.theme	• ggplot2 plot theme (see https://ggplot2.tidyverse.org/reference/ggtheme.html) possibly with additional arguments, it takes the default value plot.theme=theme_classic(base_size = 14),
legend.position	• position of the legend, refer to ggplot2 manual for the possible values; by default legend is not printed legend.position='none'
name_legend	• title of the legend
labels_legend	• the label of the plotted function in the legend
text	• can be any value that is accepted in the argument *text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one text=element_text(family='serif')
legend.text	• can be any value that is accepted in the argument *legend.text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font size to the legend to 14 legend.text=element_text(size = 14)
plot.title	• can be any value that is accepted in the argument *plot.title* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot title plot.title = element_text(hjust = 0.5)
plot.subtitle	• can be any value that is accepted in the argument *plot.subtitle* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot subtitle plot.title = element_text(hjust = 0.5)
...	• other arguments of the *theme* function, refer to ggplot2 manual

Value

ggplot2 object

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

funB <- function(x) {
  y = x
  x2 = x[x >= 0]
  x1 = x[x < 0]
  y[x < 0] = 1 / (1 + exp(-2 * x1)) - 0.5 + 0.4
  y[x >= 0] = (log(x2 * 2 + 1) - 0.15 * x2^2) * 0.6 - 0.20 + 0.4
  return(y)
}

funB_sample <- function(x) {
  y = funB(x)+ rnorm(length(x), 0, 0.1)
  return(y)
}
## Toy example - for the function check only! ##
# data generation
N=100
set.seed(1234)
x = sort(runif(N, -1, 1))
y = funB_sample(x)
c = 0

# running LoTTA function on sharp RDD with continuous outcomes;
out = LoTTA_sharp_CONT(x, y, c,normalize=FALSE, burnin = 100, sample = 100, adapt = 100,
n.chains=1, seed = NULL,method = 'simple',inits = NA)
# plot the outcome
LoTTA_plot_outcome(out,n_eval = 100)

## Use case example ##
# data generation
  N=500 # try different dataset size
  x = sort(runif(N, -1, 1))
  y = funB_sample(x)
  c = 0
  # plot the data
  plot(x,y)
  # running LoTTA function on sharp RDD with continuous outcomes;
  # cutoff = 0, treatment effect = -0.2
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_sharp_CONT(x, y, c, burnin = 10000, sample = 5000, adapt = 1000,n.chains=4)
  # print effect estimate:
  out$Effect_estimate
```

```
# print JAGS output to asses convergence (the output is for normalized data)
out$JAGS_output
# plot posterior fit
LoTTA_plot_outcome(out)
```

LoTTA_plot_treatment *Function that plots the median (or another quantile) of the LoTTA posterior treatment probability function along with the quanatile-based credible interval. The function is plotted on top of the binned input data. To bin the data, the score data is divided into bins of fixed length, then the proportion of treated is calculated in each bin. The proportions are plotted against the average values of the score in the corresponding bins. The data is binned separately on each side of the cutoff, the cutoff is marked on the plot with a dotted line. In case of an unknown cutoff, the MAP estimate is used.*

Description

Function that plots the median (or another quantile) of the LoTTA posterior treatment probability function along with the quanatile-based credible interval. The function is plotted on top of the binned input data. To bin the data, the score data is divided into bins of fixed length, then the proportion of treated is calculated in each bin. The proportions are plotted against the average values of the score in the corresponding bins. The data is binned separately on each side of the cutoff, the cutoff is marked on the plot with a dotted line. In case of an unknown cutoff, the MAP estimate is used.

Usage

```
LoTTA_plot_treatment(
  LoTTA_posterior,
  nbins = 100,
  probs = c(0.025, 0.5, 0.975),
  n_eval = 200,
  col_line = "#E69F00",
  size_line = 0.1,
  alpha_interval = 0.35,
  col_dots = "gray",
  size_dots = 3,
  alpha_dots = 0.6,
  col_cutoff = "black",
  title = "Treatment probability function",
  subtitle = NULL,
  y_lab = "",
  x_lab = expression(paste(italic("x"), " - score")),
  plot.theme = theme_classic(base_size = 14),
  legend.position = "none",
  name_legend = "Legend",
```

```
  labels_legend = "median treatment prob. fun.",
  text = element_text(family = "serif"),
  legend.text = element_text(size = 14),
  plot.title = element_text(hjust = 0.5),
  plot.subtitle = element_text(hjust = 0.5),
  ...
)
```

Arguments

LoTTA_posterior	• output of one of the LoTTA functions (LoTTA_fuzzy_CONT, LoTTA_fuzzy_BIN, LoTTA_treatment) with all the parameters sampled (the default option in those functions)
nbins	• number of bins to aggregate the input data
probs	• list of three quantiles, the first and the last one define the quanatile-based credible interval, the middle value defines the quantile of the posterior function to plot; by default the quantiles correspond to the median posterior function and 95% credible interval probs=c(0.025,0.5,0.975)
n_eval	• n_eval*range(x) is the number of points at which each posterior function is evaluated, the higher number means slower computing time and a smoother plot; default n_eval=200
col_line	• the color of the line and the band
size_line	• thickness of the line
alpha_interval	• alpha value of the band, lower values correspond to a more transparent color
col_dots	• color of the dots that correspond to the binned data
size_dots	• size of the dots that correspond to the binned data
alpha_dots	• transparency of the dots that correspond to the binned data, lower values correspond to a more transparent color
col_cutoff	• color of the dotted line at the cutoff
title	• title of the plot
subtitle	• subtitle of the plot
y_lab	• label of the y-axis
x_lab	• label of the x-axis
plot.theme	• ggplot2 plot theme (see https://ggplot2.tidyverse.org/reference/ggtheme.html) possibly with additional arguments, it takes the default value plot.theme=theme_classic(base_size = 14),
legend.position	• position of the legend, refer to ggplot2 manual for the possible values; by default legend is not printed legend.position='none'
name_legend	• title of the legend
labels_legend	• the label of the plotted function in the legend

text	• can be any value that is accepted in the argument *text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font to a serif one text=element_text(family='serif')
legend.text	• can be any value that is accepted in the argument *legend.text* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default is changes font size to the legend to 14 legend.text=element_text(size = 14)
plot.title	• can be any value that is accepted in the argument *plot.title* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot title plot.title = element_text(hjust = 0.5)
plot.subtitle	• can be any value that is accepted in the argument *plot.subtitle* in the *theme* function of ggplot2 package,refer to ggplot2 manual for the possible values; by default it centers the plot subtitle plot.title = element_text(hjust = 0.5)
...	• other arguments of the *theme* function, refer to ggplot2 manual

Value

ggplot2 object

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(t)
}



## Toy example - for the function check only! ##
N=100
x = sort(runif(N, -1, 1))
```

```
t = sample_prob55(x)
c_prior=0 # known cutoff

# running LoTTA treatment-only model;
out = LoTTA_treatment(x,t,c_prior,burnin = 50, sample = 50, adapt = 10,n.chains=1
                      ,method = 'simple',inits = NA)
# plot posterior fit of the treatment probablity function
LoTTA_plot_treatment(out,nbins = 60)

## Use case example ##

  N=500
  x = sort(runif(N, -1, 1))
  t = sample_prob55(x)

  # comment out to try different priors:
   c_prior=list(clb=-0.25,cub=0.25) # uniform prior on the interval [-0.25,0.25]
  #  c_prior=list(cstart=-0.25,cend=0.25,grid=0.05) # uniform discrete prior
  # on -0.25, -0.2, ..., 0.25
  # c_prior=0 # known cutoff c=0

  # running LoTTA treatment-only model;
  # cutoff = 0, compliance rate = 0.55
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_treatment(x,t,c_prior,burnin = 10000,sample = 5000,adapt=1000)

  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit of the treatment probablity function
  LoTTA_plot_treatment(out,nbins = 60)
```

LoTTA_sharp_BIN *LoTTA_sharp_BIN*

Description

Function that fits LoTTA model to the sharp RD data with binary outcomes. The score does NOT have to be normalized beforehand. We recommend NOT to transform the data before imputing it into the function, except for initial trimming of the score which should be done beforehand. The further trimming for the sensitivity analysis can be done through the function, which ensures that the data is normalized before the trimming.

Usage

```
LoTTA_sharp_BIN(
  x,
```

```
  y,
  c,
  ci = 0.95,
  trimmed = NULL,
  outcome_prior = list(pr = 1e-04),
  n_min = 25,
 param = c("eff", "a0l", "a1l", "a2l", "a3l", "a0r", "a1r", "a2r", "a3r", "kl", "kr"),
  normalize = TRUE,
  n.chains = 4,
  burnin = 5000,
  sample = 5000,
  adapt = 1000,
  inits = NULL,
  method = "parallel",
  seed = NULL,
  ...
)
```

Arguments

x	• is the score data
y	• is the binary outcome data
c	• specifies the cutoff point
ci	• specifies the probability level 1-alpha for the highest posterior density intervals; default is ci = 0.95
trimmed	• takes as a value NULL or a vector of two values. It specifies potential trimming of the data. If set to NULL no trimming is applied to the data. If a list of two values is provided the data is trimmed to include data points with the score x in between those values; deafult is trimmed=NULL
outcome_prior	• takes as a value a list with elements 'pr'. 'pr' specifies precision in the normal priors on the coefficients in the outcome function; default is list('pr'=0.0001)
n_min	• specifies the minimum number of data points to which a cubic part of the outcome function is fit to ensure stability of the sampling procedure; default is n_min=25
param	• takes as a value a vector with names of the parameters that are to be sampled; default is the list of all parameters
normalize	• specifies if the data is to be normalized. The data is normalized as follows. x_normalized=(x-d)/s, where d=(min(x)+max(x))*0.5 and s=max(x)-min(x). The priors inside the model are specified for the normalized data, in extreme cases not normalizing the data may lead to unreliable results; default is normalize=TRUE
n.chains	• specifies the number of chains in the MCMC sampler; default is n.chains=4
burnin	• specifies the number of burnin iterations without the adaptive iterations; default is burnin=5000
sample	• specifies the number of samples per chain; default is samples=5000

adapt	• specifies the number of adaptive iterations in the MCMC sampler; default is adapt=1000
inits	• initial values for the sampler. By default the initial values are sampled inside the function. To run LoTTA with a method other than "parallel" inits must be set to NA or to a user defined value. If the user wants to provide its own values please refer to run.jags manual; default inits=NULL
method	• set to default as 'parallel', which allows to sample the chains in parallel reducing computation time. To read more about possible method values type ?run.jags; default method='parallel'
seed	• specifies the seed for replicability purposes; default is seed=NULL
...	• other arguments of run.jags function. For more details type ?run.jags

Value

The function returns the list with the elements:

- Effect_estimate: contains a list with MAP estimate and HDI of the treatment effect on the original, unnormalized scale;
- JAGS_output: contains output of the run.jags function for the normalized data if normalize=TRUE, based on this output mixing of the chains can be assessed;
- Samples: contains posterior samples of the treatment effect (eff);
- Normalized_data: contains a list of the normalized data (if normalized=TRUE) and the parameters used to normalize the data (see arg normalize);
- Priors: contains a list of the outcome prior parameters ;
- Inits contains the list of initial values and .RNG.seed value

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
```

```
  }
  return(t)
}

## Toy example - for the function check only! ##
# data generation
N=100
x = sort(runif(N, -1, 1))
y = sample_prob55(x)
c = 0

# running LoTTA model on a sharp RDD with a binary outcome
out = LoTTA_sharp_BIN(x, y, c, burnin = 50, sample = 50, adapt = 10,n.chains=1
                      ,method = 'simple',inits = NA)

## Use case example ##

  # data generation
  N=1000 # try different dataset size
  x = sort(runif(N, -1, 1))
  y = sample_prob55(x)
  c = 0

  # running LoTTA function on sharp RDD with binary outcomes;
  # cutoff = 0, treatment effect = 0.55
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_sharp_BIN(x, y, c, burnin = 10000, sample = 5000, adapt = 1000,n.chains=4)
  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit
  LoTTA_plot_outcome(out,nbins = 60)
```

`LoTTA_sharp_CONT` *LoTTA_sharp_CONT*

Description

Function that fits LoTTA model to the sharp RD data with continuous outcomes. The data does NOT have to be normalized beforehand. We recommend NOT to transform the data before imputing it into the function, except for initial trimming which should be done beforehand. The further trimming for the sensitivity analysis can be done through the function, which ensures that the data is normalized before the trimming.

Usage

```
LoTTA_sharp_CONT(
  x,
```

```
  y,
  c,
  ci = 0.95,
  trimmed = NULL,
  outcome_prior = list(pr = 1e-04, shape = 0.01, scale = 0.01),
  n_min = 25,
 param = c("eff", "a0l", "a1l", "a2l", "a3l", "a0r", "a1r", "a2r", "a3r", "tau1r",
    "tau2r", "tau1l", "tau2l", "kl", "kr"),
  normalize = TRUE,
  n.chains = 4,
  burnin = 10000,
  sample = 5000,
  adapt = 1000,
  inits = NULL,
  method = "parallel",
  seed = NULL,
  ...
)
```

Arguments

x	• is the score data
y	• is the continuous outcome data
c	• specifies the cutoff point
ci	• specifies the probability level 1-alpha for the highest posterior density intervals; default is ci = 0.95
trimmed	• takes as a value NULL or a vector of two values. It specifies potential trimming of the data. If set to NULL no trimming is applied to the data. If a list of two values is provided the data is trimmed to include data points with the score x in between those values; deafult is trimmed=NULL
outcome_prior	• takes as a value a list with elements 'pr' and 'shape', 'scale'. 'pr' specifies precision in the normal priors on the coefficients in the outcome function. 'shape' and 'scale' specify the shape and scale parameters in the gamma prior on the precision of the error terms; default is list('pr'= 0.0001,'shape'= 0.01,'scale'= 0.01)
n_min	• specifies the minimum number of data points to which a cubic part of the outcome function is fit to ensure stability of the sampling procedure; default is n_min=25
param	• takes as a value a vector with names of the parameters that are to be sampled; default is the list of all parameters
normalize	• specifies if the data is to be normalized. The data is normalized as follows. x_normalized=(x-d)/s, where d=(min(x)+max(x))*0.5 and s=max(x)-min(x). y_normalized=(y-mu)/sd, where mu=mean(y) and sd=sd(y). The priors inside the model are specified for the normalized data, in extreme cases not normalizing the data may lead to unreliable results; default is normalize=TRUE

n.chains	• specifies the number of chains in the MCMC sampler; default is n.chains=4
burnin	• specifies the number of burnin iterations without the adaptive iterations; default is burnin=5000
sample	• specifies the number of samples per chain; default is samples=5000
adapt	• specifies the number of adaptive iterations in the MCMC sampler; default is adapt=1000
inits	• initial values for the sampler. By default the initial values are sampled inside the function. To run LoTTA with a method other than "parallel" inits must be set to NA or to a user defined value. If the user wants to provide its own values please refer to run.jags manual; default inits=NULL
method	• set to default as 'parallel', which allows to sample the chains in parallel reducing computation time. To read more about possible method values type ?run.jags; default method='parallel'
seed	• specifies the seed for replicability purposes; default is seed=NULL
...	• other arguments of run.jags function. For more details type ?run.jags

Value

The function returns the list with the elements:

- Effect_estimate: contains a list with MAP estimate and HDI of the treatment effect on the original, unnormalized scale;
- JAGS_output: contains output of the run.jags function for the normalized data if normalize=TRUE, based on this output mixing of the chains can be assessed;
- Samples: contains posterior samples of the treatment effect (eff);
- Normalized_data: contains a list of the normalized data (if normalized=TRUE) and the parameters used to normalize the data (see arg normalize);
- Priors: contains a list of the outcome prior parameters ;
- Inits contains the list of initial values and .RNG.seed value

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

funB <- function(x) {
  y = x
  x2 = x[x >= 0]
  x1 = x[x < 0]
  y[x < 0] = 1 / (1 + exp(-2 * x1)) - 0.5 + 0.4
  y[x >= 0] = (log(x2 * 2 + 1) - 0.15 * x2^2) * 0.6 - 0.20 + 0.4
  return(y)
}
```

```
funB_sample <- function(x) {
  y = funB(x)+ rnorm(length(x), 0, 0.1)
  return(y)
}

## Toy example - for the function check only! ##
# data generation
N=100
set.seed(1234)
x = sort(runif(N, -1, 1))
y = funB_sample(x)
c = 0

# running LoTTA function on sharp RDD with continuous outcomes;
out = LoTTA_sharp_CONT(x, y, c,normalize=FALSE, burnin = 50, sample = 50, adapt = 10,
n.chains=1, seed = NULL,method = 'simple',inits = NA)

## Use case example ##
# data generation
  N=500 # try different dataset size
  x = sort(runif(N, -1, 1))
  y = funB_sample(x)
  c = 0
  # plot the data
  plot(x,y)
  # running LoTTA function on sharp RDD with continuous outcomes;
  # cutoff = 0, treatment effect = -0.2
  # remember to check convergence and adjust burnin, sample and adapt if needed
  out = LoTTA_sharp_CONT(x, y, c, burnin = 10000, sample = 5000, adapt = 1000,n.chains=4)
  # print effect estimate:
  out$Effect_estimate
  # print JAGS output to asses convergence (the output is for normalized data)
  out$JAGS_output
  # plot posterior fit
  LoTTA_plot_outcome(out)
```

LoTTA_treatment *LoTTA_treatment*

Description

Function that fits LoTTA treatment model to the fuzzy RD treatment data with an either known or unknown/suspected cutoff. It supports two types of priors on the cutoff location: a scaled beta distribution of the form beta(alpha,beta)(cub-clb)+clb and a discrete distribution with the support of the form cstart+grid i for i=0,...,(cend-cstart)/grid. The score does NOT have to be normalized beforehand. We recommend NOT to transform the data before imputing it into the function, except for initial trimming of the score which should be done beforehand. The further trimming for the sensitivity analysis can be done through the function, which ensures that the data is normalized before the trimming.

Usage

```
LoTTA_treatment(
  x,
  t,
  c_prior,
  jlb = 0.2,
  ci = 0.95,
  trimmed = NULL,
  n_min = 25,
 param = c("c", "j", "b1lt", "a1lt", "a2lt", "b2lt", "b1rt", "a1rt", "a2rt", "b2rt",
    "k1t", "k2t"),
  normalize = TRUE,
  n.chains = 4,
  burnin = 5000,
  sample = 1000,
  adapt = 500,
  inits = NULL,
  method = "parallel",
  seed = NULL,
  ...
)
```

Arguments

x	• is the score data
t	• is the treatment allocation data
c_prior	• specifies the cutoff prior in case of the unknown cutoff or the cutoff point if the cutoff is known. Takes as value a number if the cutoff is known or a list of values otherwise. For a continuous prior the list requires the following elements: clb - left end of the interval cub - right end of the interval in which the scaled and translated beta distribution is defined, alpha (optional) - shape parameter, default value = 1, beta (optional) - shape parameter, default value = 1. For a discrete prior the list requires the following elements: cstart - first point with positive prior mass, cend - last point with positive prior mass, grid - distance between the consecutive points in the support weights (optional) - vector of weights assigned to each point in the support, default is vector of 1's (uniform distribution)
jlb	• minimum jump size
ci	• specifies the probability level 1-alpha for the highest posterior density intervals; default is ci = 0.95
trimmed	• takes as a value NULL or a vector of two values. It specifies potential trimming of the data. If set to NULL no trimming is applied to the data. If a list of two values is provided the data is trimmed to include data points with the score x in between those values; default is trimmed=NULL
n_min	• specifies the minimum number of data points to which a cubic part of the outcome function is fit to ensure stability of the sampling procedure; default is n_min=25

`param`	• takes as a value a vector with names of the parameters that are to be sampled; default is the list of all parameters
`normalize`	• specifies if the data is to be normalized. The data is normalized as follows. If the prior is continuous: x_normalized=(x-d)/s, where d=(min(x)+max(x))*0.5 and s=max(x)-min(x), If the prior is discrete: x_normalized=x/s, where s=10^m, where m is chosen so that |max(abs(x))-1| is minimal. The outcome data is normalized as follows: y_normalized=(y-mu)/sd, where mu=mean(y) and sd=sd(y). The priors inside the model are specified for the normalized data, in extreme cases not normalizing the data may lead to unreliable results; default is normalize=TRUE
`n.chains`	• specifies the number of chains in the MCMC sampler; default is n.chains=4
`burnin`	• specifies the number of burnin iterations without the adaptive iterations; default is burnin=5000
`sample`	• specifies the number of samples per chain; default is samples=5000
`adapt`	• specifies the number of adaptive iterations in the MCMC sampler; default is adapt=1000
`inits`	• initial values for the sampler. By default the initial values are sampled inside the function. To run LoTTA with a method other than "parallel" inits must be set to NA or to a user defined value. If the user wants to provide its own values please refer to run.jags manual; default inits=NULL
`method`	• set to default as 'parallel', which allows to sample the chains in parallel reducing computation time. To read more about possible method values type ?run.jags; default method='parallel'
`seed`	• specifies the seed for replicability purposes; default is seed=NULL
`...`	• other arguments of run.jags function. For more details type ?run.jags

Value

The function returns the list with the elements:

- Effect_estimate: contains a list with MAP estimate and HDI of the cutoff location (if unknown) and the discontinuity size in the treatment probability function (compliance rate at c) on the original, unnormalized scale;
- JAGS_output: contains output of the run.jags function for the normalized data if normalize=TRUE, based on this output mixing of the chains can be assessed;
- Samples: contains posterior samples of the cutoff location (c) if unknown, and compliance rate (j);
- Normalized_data: contains a list of the normalized data (if normalized=TRUE) and the parameters used to normalize the data (see arg normalize);
- Priors: contains a list of the priors' parameters ;
- Inits contains the list of initial values and .RNG.seed value

Examples

```
# functions to generate the data

ilogit <- function(x) {
  return(1 / (1 + exp(-x)))
}

fun_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  return(P)
}

sample_prob55 <- function(x) {
  P = rep(0, length(x))
  P[x >= 0.] = ilogit((8.5 * x[x >= 0.] - 1.5)) / 10.5 + 0.65 - 0.0007072
  P[x < 0.] = (x[x < 0.] + 1)^4 / 15 + 0.05
  t = rep(0, length(x))
  for (j in 1:length(x)) {
    t[j] = sample(c(1, 0), 1, prob = c(P[j], 1 - P[j]))
  }
  return(t)
}



## Toy example - for the function check only! ##
N=100
x = sort(runif(N, -1, 1))
t = sample_prob55(x)
c_prior=0 # known cutoff

# running LoTTA treatment-only model;
out = LoTTA_treatment(x,t,c_prior,burnin = 50, sample = 50, adapt = 10,n.chains=1
                      ,method = 'simple',inits = NA)

## Use case example ##

  N=500
  x = sort(runif(N, -1, 1))
  t = sample_prob55(x)

  # comment out to try different priors:
   c_prior=list(clb=-0.25,cub=0.25) # uniform prior on the interval [-0.25,0.25]
  #  c_prior=list(cstart=-0.25,cend=0.25,grid=0.05) # uniform discrete prior
  # on -0.25, -0.2, ..., 0.25
  # c_prior=0 # known cutoff c=0

  # running LoTTA treatment-only model;
  # cutoff = 0, compliance rate = 0.55
  # remember to check convergence and adjust burnin, sample and adapt if needed
```

```
out = LoTTA_treatment(x,t,c_prior,burnin = 10000,sample = 5000,adapt=1000)

# print effect estimate:
out$Effect_estimate
# print JAGS output to asses convergence (the output is for normalized data)
out$JAGS_output
# plot posterior fit of the treatment probablity function
LoTTA_plot_treatment(out,nbins = 60)
```

Index