Package 'rACMEMEEV'

December 12, 2025

Title Multi-Variate Measurement Error Adjustment

Version 1.0.0

Description A methodology to perform multivariate measurement error adjustment using external validation data. Allows users to remove the attenuating effect of measurement error by incorporating a distribution of external validation data, and allows for plotting of all resultant adjustments. Sensitivity analyses can also be run through this package to test how different ranges of validity coefficients can impact the effect of the measurement error adjustment. The methods implemented in this package are based on the work by Muoka, A., Agogo, G., Ngesa, O., Mwambi, H. (2020): <doi:10.12688/f1000research.27892.1

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0), BH, glmnet

Config/testthat/edition 3

Imports rjags, MCMCpack, ggplot2, reshape2, rstan, stats, utils, ggpubr, patchwork, bayesplot, doSNOW, foreach, snow, parallelly, dplyr

VignetteBuilder knitr

URL `https://github.com/westford14/rACME-MEEV`

BugReports `https://github.com/westford14/rACME-MEEV/issues`

NeedsCompilation no

Author Alexander Lee [aut, cre, cph]

Maintainer Alexander Lee <`westford14@gmail.com`>

Repository CRAN

Date/Publication 2025-12-12 21:30:13 UTC

Contents

`acf_plots` *Create Auto Correlation Plots for Models*

Description

Create auto correlation plots for the models to test if there is any high autocorrelation in the sampling space, which would imply that sampler should be adjusted or the data needs to be adjusted.

Usage

```
acf_plots(input, columns, pre_model = FALSE, stan = FALSE)
```

Arguments

`input`	list List of the model output
`columns`	vector The target columns to assess
`pre_model`	bool If you are passing the pre-model
`stan`	bool If you are using the stan model

Value

plotted objects

Examples

```
columns <- c("fruit", "veg", "tobacco")
fruit_v_coef <- generate_coefficient(100, 0.3, 0.8, 0.95)
veg_v_coef <- generate_coefficient(100, 0.25, 0.75, 0.95)
tob_v_coef <- generate_coefficient(100, 0.4, 0.7, 0.95)
validity_coefficients <- c(fruit_v_coef, veg_v_coef, tob_v_coef)
data <- data.frame(
 list(
    "BMI" = rnorm(100, mean = 0, sd = 1),
    "fruit" = rnorm(100, mean = 0, sd = 1),
    "veg" = rnorm(100, mean = 0, sd = 1),
    "tobacco" = rnorm(100, mean = 0, sd = 1)
  )
)
output <- acme_model(data, columns)
lambda <- attenuation_matrix(
  output,
  columns,
  validity_coefficients,
)
model_output <- multivariate_model(
  "BMI ~ fruit + veg + tobacco",
  data = data,
  columns = columns,
  a_c_matrix = lambda$matrix,
  sds = lambda$sds,
  variances = lambda$variances,
  univariate = TRUE
)
acf_plots(model_output$naive, columns)
```

`acme_model` *Model the Data*

Description

Using the methodology generated by Muoko et. al (see README for full citation), run the modelling with the JAGS sampler. This function accepts a number of different arguments, but defaulted arguments are assumed to be best for *most* systems. Obviously, if there is prior knowledge, this can be adjusted at the user's own discretion.

Usage

```
acme_model(
  data,
  columns,
  n_chains = 1,
  n_adapt_steps = 500,
  n_burn = 1000,
  n_thin = 1,
  n_steps = 10000,
  seed = 42,
  stan = FALSE
)
```

Arguments

`data`	data.frame The data to format
`columns`	vector The columns to target
`n_chains`	numeric Number of chains to run, default = 1
`n_adapt_steps`	numeric Number of adapt steps to run, default = 500
`n_burn`	numeric Number of draws to burn at the start, default = 1000
`n_thin`	numeric Thinning factor for the draws, default = 1
`n_steps`	numeric The total number of draws to run, default = 10,000
`seed`	numeric The random seed to set
`stan`	boolean If you would like to use the experimental Stan backend

Value

List with all the appropriate things needed for modelling with JAGS

Examples

```
data <- data.frame(list("fruit" = c(1, 2), "veg" = c(3, 4)))
acme_model(data, names(data))
```

`attenuation_matrix` *Create Attenuation Contamination Matrix*

Description

From the output JAGS model, create the needed attenuation contamination matrix for further analysis. Please be aware that this is multivariate so a minimum of 3 columns is needed.

Usage

```
attenuation_matrix(model_output, columns, validity_coefficients, stan = FALSE)
```

Arguments

`model_output`	list List of the mcmc summary and covariance matrix
`columns`	vector Vector of the column names that are being assessed
`validity_coefficients`	vector Vector of the validity coefficients
`stan`	boolean If you are passing in a Stan backend pre-model

Value

List with the attenuation-contamination matrix and the standard deviations

Examples

```
columns <- c("fruit", "veg", "tobacco")
fruit_v_coef <- generate_coefficient(100, 0.3, 0.8, 0.95)
veg_v_coef <- generate_coefficient(100, 0.25, 0.75, 0.95)
tob_v_coef <- generate_coefficient(100, 0.4, 0.7, 0.95)
validity_coefficients <- c(fruit_v_coef, veg_v_coef, tob_v_coef)
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
output <- acme_model(data, columns)
attenuation_matrix(
  output,
  columns,
  validity_coefficients,
)
```

`create_modelling_data` *Data Preparation and Formatting for Modelling*

Description

Create the needed list that gets based to the JAGS backend for modelling. This should be a standard data.frame object that can then be standardized and reshaped into a format appropriate for modeling. Errors will be thrown if the data is not a class, or inherit from, a data.frame or if the specified columns do not exist in the data.frame.

Usage

```
create_modelling_data(data, columns)
```

Arguments

`data`	data.frame The data to format
`columns`	vector The columns to target

Value

List with all the appropriate things needed for modelling with JAGS

Examples

```
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
create_modelling_data(data, c("BMI", "fruit", "veg", "tobacco"))
```

`create_model_string` *Define a Model that is JAGS Usable*

Description

Create the model string, save it to a temporary folder, and return back the location of the temporary model file for usage by JAGS later.

Usage

```
create_model_string()
```

Value

Full path to the model specification

Examples

```
create_model_string()
```

`create_stan_model_string` *Define the Pre-Model Using Stan*

Description

Create the model string, save it to a temporary folder, and return back the location of the temporary model file for usage by Stan later. This is experimental and has been validated, but caution should be used when utilizing the Stan backend as the pre-model.

Usage

```
create_stan_model_string()
```

Value

Full path to the model specification

Examples

```
create_stan_model_string()
```

`fisher_z_transform` *Perform the Fisher Z Transformation*

Description

Take a validity coefficient and perform the Fisher Z Transformation to approximate a normal distribution

Usage

```
fisher_z_transform(coefficient)
```

Arguments

`coefficient`	numeric The validity coefficient to transform.

Value

Transformed validity coefficient

Examples

```
coef <- 0.3
fisher_z_transform(coef)
```

`generate_coefficient` *Generate updated validity coefficient using Fisher Z Transformation*

Description

Using the Fisher Z Transformation, generate new validity coefficients based on proposed upper and lower boundaries. This new validity coefficient is based on transformed upper and lower boundaries as well as a fitted normal distribution to the se new proposed upper and lower boundaries.

Usage

```
generate_coefficient(n, lower_bound, upper_bound, interval = 0.95, seed = 42)
```

Arguments

`n`	the samples for the normal distribution
`lower_bound`	numeric the lower boundary of the validity coefficient
`upper_bound`	numeric the upper boudnary of the validity coefficient
`interval`	numeric the confidence interval to use (default 0.95)
`seed`	numeric the random seed to use

Value

Validity coefficient

Examples

```
n <- 1000
coef_upper <- 0.9
coef_lower <- 0.3
ci <- 0.95
generate_coefficient(n, coef_lower, coef_upper, ci)
```

`multivariate_model` *Model the Data with Multivariate Adjustment*

Description

Using the methodology generated by Muoko et. al (see README for full citation), run the modelling with the JAGS sampler. This model uses the calculated attenuation-contamination matrix to adjust the various covariates needed in the model. The function accepts the computed pre-model, the attenuation-contamination coefficient,, a model string, and standard deviations. There is an optional argument for also fitting the univariate model to run further comparisons between the naive, univariate, and multivariate models in later steps.

Usage

```
multivariate_model(
  formula,
  data,
  columns,
  a_c_matrix,
  n_burn = 1000,
  n_thin = 1,
  n_steps = 10000,
  seed = 42,
  b0 = 0,
  capital_b_0 = 1e-06,
  sampler = "Metropolis",
  c0 = 0.001,
  d0 = 0.001,
  univariate = FALSE,
  sds = NULL,
  variances = NULL
)
```

Arguments

`formula`	character The model formula
`data`	data.frame The data to model
`columns`	vector The columns that are relevant
`a_c_matrix`	matrix The attenuation-contamination matrix
`n_burn`	numeric Number of draws to burn at the start, default = 1000
`n_thin`	numeric Thinning factor for the draws, default = 1
`n_steps`	numeric The total number of draws to run, default = 10,000
`seed`	numeric The random seed to set
`b0`	numeric The prior mean of the beta
`capital_b_0`	numeric The precision of the beta
`sampler`	character The sampler to use for the model
`c0`	numeric Shape parameter for gamma
`d0`	numeric Scale parameter for gamm
`univariate`	bool Whether or not to run the univariate model, default = TRUE
`sds`	list If you are running the univariate model, the listing of standard deviations needs to be applied
`variances`	list If you are running the univariate model, the listing of variances need to be applied

Value

List with fitted models ready for further analysis

Examples

```
columns <- c("fruit", "veg", "tobacco")
fruit_v_coef <- generate_coefficient(100, 0.3, 0.8, 0.95)
veg_v_coef <- generate_coefficient(100, 0.25, 0.75, 0.95)
tob_v_coef <- generate_coefficient(100, 0.4, 0.7, 0.95)
validity_coefficients <- c(fruit_v_coef, veg_v_coef, tob_v_coef)
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
```

```
output <- acme_model(data, columns)
lambda <- attenuation_matrix(
  output,
  columns,
  validity_coefficients,
)
model_output <- multivariate_model(
  "BMI ~ fruit + veg + tobacco",
  data = data,
  columns = columns,
  a_c_matrix = lambda$matrix,
  sds = lambda$sds,
  variances = lambda$variances,
  univariate = TRUE
)
```

---

pipeline *Pipeline used for running a model start to finish.*

Description

This is an unexported function that can run the entire pre-model, attenuation-contamination matrix, and model fitting from start to finish.

Usage

```
pipeline(data, formula, parameters, columns, stan = FALSE, seed = 42)
```

Arguments

`data`	data.frame The input data
`formula`	character The formula for the multivariate model
`parameters`	vector The validity coefficients
`columns`	vector The columns of the covariates
`stan`	bool If you would like to run with the Stan backend
`seed`	numeric The random seed to use

Value

list The output parameters

Examples

```
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
parameters <- list(
  fruit = c(0.3, 0.55, 0.8),
  veg = c(0.25, 0.5, 0.75),
  tobacco = c(0.4, 0.55, 0.7)
)
grid <- expand.grid(parameters)
param_grid <- list()
for (i in seq_len(nrow(grid))) {
  name <- paste0("iteration_", i)
  param_grid[[name]] <- list(
    parameters = grid[i, ]
  )
}
output <- pipeline(
   data,
   "BMI ~ fruit + veg + tobacco",
   as.numeric(param_grid[[i]][["parameters"]]),
   c("fruit", "veg", "tobacco")
)
```

`plot_a_list` *Custom Function to Use Patchwork*

Description

Uses patchwork to make gridding easier for plotting the traceplots and acf plots for the Stan pre-model.

Usage

```
plot_a_list(plot_list, rows, columns)
```

Arguments

`plot_list`	list List of plots
`rows`	numeric Number of rows
`columns`	numeric Number of columns

`plot_covariates` *Diagnostics for Models*

Description

Using the previously fit models, add the diagnostics for assessing the adjusted covariates based on the naive, univariate, and multivariate models.

Usage

```
plot_covariates(model_output, columns)
```

Arguments

`model_output`	list List of the model output
`columns`	vector The target columns to assess

Value

plotted objects

Examples

```
columns <- c("fruit", "veg", "tobacco")
fruit_v_coef <- generate_coefficient(100, 0.3, 0.8, 0.95)
veg_v_coef <- generate_coefficient(100, 0.25, 0.75, 0.95)
tob_v_coef <- generate_coefficient(100, 0.4, 0.7, 0.95)
validity_coefficients <- c(fruit_v_coef, veg_v_coef, tob_v_coef)
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
output <- acme_model(data, columns)
lambda <- attenuation_matrix(
  output,
  columns,
  validity_coefficients,
)
model_output <- multivariate_model(
  "BMI ~ fruit + veg + tobacco",
  data = data,
  columns = columns,
  a_c_matrix = lambda$matrix,
  sds = lambda$sds,
  variances = lambda$variances,
  univariate = TRUE
)
plot_covariates(model_output, columns)
```

`sensitivity_analysis` *Run a sensitivity analysis on the error adjustment*

Description

Create a sensitivity analysis based on a grid of validity coefficient parameters. The input of the parameter space should be a list with each key being the name of one of the covariates to vary and a vector of the parameter space to test the sensitivity. Be very careful because all combinations of the parameters will be tested so you can very easily run this for too long. Also please note that the parameters should be be bound between 0 and 1, the theoretical limits of the validity coefficients. An example of this parameter grid is:

Usage

```
sensitivity_analysis(
  parameters,
  data,
  formula,
  columns,
  stan = FALSE,
  seed = 42
)
```

Arguments

`parameters`	list As described above
`data`	data.frame The data to use for the sensitivity analysis
`formula`	character The formula for the model
`columns`	vector The columns within the data for the covariates
`stan`	boolean Whether or not to run with the Stan backend
`seed`	numeric The random seed to set

Details

params <- list(fruit = c(0.1, 0.2, 0.3), veg = c(0.1, 0.2, 0.3), tobacco = c(0.1, 0.2, 0.3))

But, again please note that this will then fit 3 * 3 * 3 different models so the run time here can explode. Also parallel computation will be utilized here, but it is much more difficult to debug should errors arise. All this to say is: buyer beware.

Value

list with the means and the SDs of the parameters

Examples

```
columns <- c("fruit", "veg", "tobacco")
data <- data.frame(
 list(
   "BMI" = rnorm(5, mean = 0, sd = 1),
   "fruit" = rnorm(5, mean = 0, sd = 1),
   "veg" = rnorm(5, mean = 0, sd = 1),
   "tobacco" = rnorm(5, mean = 0, sd = 1)
 )
)
parameters <- list(
  fruit = c(0.3),
  veg = c(0.25),
  tobacco = c(0.4)
)
output_jags <- sensitivity_analysis(
  parameters,
  data,
  "BMI ~ fruit + veg + tobacco",
  columns
)
```

`standardize_with_return`
Standardize the data of 1-D vector

Description

Perform standardization of data on a 2-D dataframe type object. Standardization in this case refers to (x - mean(x)) / sd(x) where X is a 1-dimensional vector.

Usage

```
standardize_with_return(data)
```

Arguments

`data`	vector The vector to standardize

Value

List with the original standard deviation, mean, and the standardized data

Examples

```
data <- rnorm(100, mean = 0, sd = 1)
standardize_with_return(data)
```

traceplots *Create Traceplots for the Parameters*

Description

Create sampler traceplots to run diagnostics on whether or not the sampler converged when creating the posterior. Columns should always be in the order that they appear in the model.

Usage

```
traceplots(input, columns, pre_model = FALSE, stan = FALSE)
```

Arguments

`input`	list List of the model output
`columns`	vector The target columns to assess
`pre_model`	bool If you are passing the pre-model
`stan`	bool If the model is stan based

Value

plotted objects

Examples

```
columns <- c("fruit", "veg", "tobacco")
fruit_v_coef <- generate_coefficient(100, 0.3, 0.8, 0.95)
veg_v_coef <- generate_coefficient(100, 0.25, 0.75, 0.95)
tob_v_coef <- generate_coefficient(100, 0.4, 0.7, 0.95)
validity_coefficients <- c(fruit_v_coef, veg_v_coef, tob_v_coef)
data <- data.frame(
 list(
   "BMI" = rnorm(100, mean = 0, sd = 1),
   "fruit" = rnorm(100, mean = 0, sd = 1),
   "veg" = rnorm(100, mean = 0, sd = 1),
   "tobacco" = rnorm(100, mean = 0, sd = 1)
 )
)
output <- acme_model(data, columns)
lambda <- attenuation_matrix(
  output,
  columns,
  validity_coefficients,
)
model_output <- multivariate_model(
  "BMI ~ fruit + veg + tobacco",
  data = data,
  columns = columns,
  a_c_matrix = lambda$matrix,
  sds = lambda$sds,
  variances = lambda$variances,
  univariate = TRUE
)
traceplots(model_output$naive, columns)
```

Index