

# Package ‘RCTS’

January 20, 2025

**Title** Clustering Time Series While Resisting Outliers

**Version** 0.2.4

**Description** Robust Clustering of Time Series (RCTS) has the functionality to cluster time series using both the classical and the robust interactive fixed effects framework.

The classical framework is devel-

oped in Ando & Bai (2017) <[doi:10.1080/01621459.2016.1195743](https://doi.org/10.1080/01621459.2016.1195743)>. The implementation within this package excludes the SCAD-penalty on the estimations of beta.

This robust framework is developed in Boudt & Heyn-

dels (2022) <[doi:10.1016/j.ecosta.2022.01.002](https://doi.org/10.1016/j.ecosta.2022.01.002)> and is made robust against different kinds of outliers.

The algorithm iteratively updates beta (the coefficients of the observable variables), group membership, and the latent factors (which can be common and/or group-specific) along with their loadings. The number of groups and factors can be estimated if they are unknown.

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**Imports** stats, magrittr, dplyr, purrr, stringr, tidyr, tibble, ggplot2, ncvreg, robustbase, cellWise, rlang, Rdpack

**Suggests** tsqn, doParallel, doSNOW, foreach, mclust, Matrix

**RdMacros** Rdpack

**Depends** R (>= 4.1.0)

**NeedsCompilation** no

**Author** Ewoud Heyndels [aut, cre] (<<https://orcid.org/0000-0003-4540-8571>>)

**Maintainer** Ewoud Heyndels <ewoud.heyndels@vub.be>

**Repository** CRAN

**Date/Publication** 2023-05-18 14:30:02 UTC

## Contents

adapt\_pic\_with\_sigma2maxmodel . . . . . 4

|   |    |
|---|----|
| adapt_X_estimating_less_variables . . . . .             | 5  |
| add_configuration . . . . .                             | 5  |
| add_metrics . . . . .                                   | 6  |
| add_pic . . . . .                                       | 7  |
| add_pic_parallel . . . . .                              | 8  |
| beta_true_heterogroups . . . . .                        | 10 |
| calculate_best_config . . . . .                         | 10 |
| calculate_errors_virtual_groups . . . . .               | 11 |
| calculate_error_term . . . . .                          | 12 |
| calculate_FL_group_estimated . . . . .                  | 14 |
| calculate_FL_group_true . . . . .                       | 15 |
| calculate_lambda . . . . .                              | 16 |
| calculate_lambda_group . . . . .                        | 17 |
| calculate_lgfg . . . . .                                | 18 |
| calculate_obj_for_g . . . . .                           | 19 |
| calculate_PIC . . . . .                                 | 20 |
| calculate_PIC_term1 . . . . .                           | 21 |
| calculate_sigma2 . . . . .                              | 22 |
| calculate_sigma2maxmodel . . . . .                      | 22 |
| calculate_TN_factor . . . . .                           | 23 |
| calculate_VCsquared . . . . .                           | 24 |
| calculate_virtual_factor_and_lambda_group . . . . .     | 25 |
| calculate_W . . . . .                                   | 26 |
| calculate_XB_estimated . . . . .                        | 27 |
| calculate_XB_true . . . . .                             | 27 |
| calculate_Z_common . . . . .                            | 28 |
| calculate_Z_group . . . . .                             | 29 |
| check_stopping_rules . . . . .                          | 30 |
| clustering_with_robust_distances . . . . .              | 31 |
| create_covMat_crosssectional_dependence . . . . .       | 31 |
| create_data_dgp2 . . . . .                              | 32 |
| create_true_beta . . . . .                              | 33 |
| define_configurations . . . . .                         | 34 |
| define_C_candidates . . . . .                           | 34 |
| define_kg_candidates . . . . .                          | 35 |
| define_number_subsets . . . . .                         | 35 |
| define_object_for_initial_clustering_macropca . . . . . | 36 |
| define_rho_parameters . . . . .                         | 37 |
| determine_beta . . . . .                                | 37 |
| determine_robust_lambda . . . . .                       | 39 |
| df_results_example . . . . .                            | 39 |
| do_we_estimate_common_factors . . . . .                 | 40 |
| do_we_estimate_group_factors . . . . .                  | 41 |
| estimate_algorithm . . . . .                            | 41 |
| estimate_beta . . . . .                                 | 42 |
| estimate_factor . . . . .                               | 44 |
| estimate_factor_group . . . . .                         | 45 |
| evade_crashes_macropca . . . . .                        | 47 |

|   |    |
|---|----|
| evade_floating_point_errors . . . . .               | 47 |
| factor_group_true_dgp3 . . . . .                    | 48 |
| fill_rc . . . . .                                   | 48 |
| fill_rcj . . . . .                                  | 49 |
| final_estimations_filter_kg . . . . .               | 49 |
| generate_grouped_factorstructure . . . . .          | 50 |
| generate_Y . . . . .                                | 51 |
| get_best_configuration . . . . .                    | 52 |
| get_convergence_speed . . . . .                     | 53 |
| get_final_estimation . . . . .                      | 53 |
| grid_add_variables . . . . .                        | 54 |
| g_true_dgp3 . . . . .                               | 55 |
| handleNA . . . . .                                  | 56 |
| handleNA_LG . . . . .                               | 56 |
| handle_macropca_errors . . . . .                    | 57 |
| initialise_beta . . . . .                           | 57 |
| initialise_clustering . . . . .                     | 59 |
| initialise_commonfactorstructure_macropca . . . . . | 60 |
| initialise_df_pic . . . . .                         | 61 |
| initialise_df_results . . . . .                     | 62 |
| initialise_rc . . . . .                             | 62 |
| initialise_rcj . . . . .                            | 63 |
| initialise_X . . . . .                              | 63 |
| iterate . . . . .                                   | 64 |
| kg_candidates_expand . . . . .                      | 66 |
| lambda_group_true_dgp3 . . . . .                    | 66 |
| LMROB . . . . .                                     | 67 |
| make_df_pic_parallel . . . . .                      | 67 |
| make_df_results_parallel . . . . .                  | 68 |
| make_subsamples . . . . .                           | 68 |
| matrixnorm . . . . .                                | 69 |
| OF_vectorized3 . . . . .                            | 69 |
| OF_vectorized_helpfunction3 . . . . .               | 71 |
| parallel_algorithm . . . . .                        | 72 |
| plot_VCsquared . . . . .                            | 73 |
| prepare_for_robpc . . . . .                         | 75 |
| RCTS . . . . .                                      | 75 |
| reassign_if_empty_groups . . . . .                  | 76 |
| restructure_X_to_order_slowN_fastT . . . . .        | 76 |
| return_robust_lambdaobject . . . . .                | 77 |
| robustpca . . . . .                                 | 78 |
| run_config . . . . .                                | 79 |
| scaling_X . . . . .                                 | 79 |
| solveFG . . . . .                                   | 80 |
| tabulate_potential_C . . . . .                      | 81 |
| update_g . . . . .                                  | 82 |
| X_dgp3 . . . . .                                    | 83 |
| Y_dgp3 . . . . .                                    | 84 |

---

adapt\_pic\_with\_sigma2maxmodel

*Adapts the object that contains PIC for all candidate C's and all subsamples with sigma2\_max\_model.*

---

### Description

The PIC is calculated with a sigma2 specific to the configuration (= number of groups and factors). Because the method to estimate the number of groups and factors requires sigma2 to be equal over all configurations (see proofs of different papers of Ando/Bai) we replace sigma2 by the sigma2 of the configuration with maximum number of groups and factors (this is the last one that was executed).

### Usage

```
adapt_pic_with_sigma2maxmodel(df, df_results, sigma2_max_model)
```

### Arguments

df                    contains PIC for all candidate C's and all subsamples  
df\_results           dataframe with results for each estimated configuration  
sigma2\_max\_model     sigma2 of model with maximum number of groups and factors

### Value

data.frame of same size as df

### Examples

```
set.seed(1)
df_pic <- data.frame(matrix(rnorm(4 * 50), nrow = 4)) #4 configuration / 50 candidate values for C
df_results <- data.frame(sigma2 = rnorm(4))
pic_sigma2 <- 3.945505
adapt_pic_with_sigma2maxmodel(df_pic, df_results, pic_sigma2)
```

---

adapt\_X\_estimating\_less\_variables

*When running the algorithm with a different number of observable variables then the number that is available, reformat X. (Mainly used for testing)*

---

### Description

When running the algorithm with a different number of observable variables then the number that is available, reformat X. (Mainly used for testing)

### Usage

```
adapt_X_estimating_less_variables(X, vars_est)
```

### Arguments

|          |  |
|----------|--|
| X        | dataframe with the observed variables  |
| vars_est | number of available observed variables for which a coefficient will be estimated |

### Value

Returns a 3D-array. If vars\_est is set to 0, it returns NA.

---

|                   |   |
|-------------------|---|
| add_configuration | <i>Adds the current configuration (number of groups and factors) to df_results.</i> |
|-------------------|---|

---

### Description

Adds the current configuration (number of groups and factors) to df\_results.

### Usage

```
add_configuration(df_results, S, k, kg)
```

### Arguments

|            |   |
|------------|---|
| df_results | dataframe with results for each estimated configuration   |
| S          | estimated number of groups in current configuration   |
| k          | estimated number of common factors in current configuration   |
| kg         | vector with the estimated number of group specific factors in current configuration (augmented with NA's to reach a length of 20) |

**Value**

data.frame

**Examples**

```
add_configuration(initialise_df_results(TRUE), 3, 0, c(3, 3, 3, rep(NA, 17)))
```

---

|             |  |
|-------------|--|
| add_metrics | <i>Adds several metrics to df_results.</i> |
|-------------|--|

---

**Description**

Adds several metrics to df\_results.

**Usage**

```
add_metrics(
  df_results,
  index_configuration,
  pic_sigma2,
  beta_est,
  g,
  comfactor,
  lambda,
  factor_group,
  lambda_group,
  iteration,
  g_true = NA,
  add_rand = FALSE
)
```

**Arguments**

|                     |   |
|---------------------|---|
| df_results          | dataframe with results for each estimated configuration                                   |
| index_configuration | index of the configuration of groups and factors  |
| pic_sigma2          | sum of squared errors divided by NT   |
| beta_est            | estimated values of beta  |
| g                   | Vector with estimated group membership for all individuals                                |
| comfactor           | estimated common factors  |
| lambda              | loadings of the estimated common factors  |
| factor_group        | estimated group specific factors  |
| lambda_group        | loadings of the estimated group specific factors  |
| iteration           | number of iteration   |
| g_true              | vector of length NN with true group memberships   |
| add_rand            | adds the adjusted randindex to the df (requires the mclust package); used for simulations |

**Value**

data.frame with final estimations of each configuration

**Examples**

```
df_results <- add_configuration(initialise_df_results(TRUE),
  3, 0, c(3, 3, 3, rep(NA, 17))) #data.frame with one configuration
add_metrics(df_results, 1, 3.94, NA, round(runif(30, 1, 3)), NA, NA, NA, NA, 9)
```

---

|         |   |
|---------|---|
| add_pic | <i>Fills in df_pic: adds a row with the calculated PIC for the current configuration.</i> |
|---------|---|

---

**Description**

Fills in df\_pic: adds a row with the calculated PIC for the current configuration.

**Usage**

```
add_pic(
  df,
  index_configuration,
  robust,
  Y,
  beta_est,
  g,
  S,
  k,
  kg,
  est_errors,
  C_candidates,
  method_estimate_beta = "individual",
  choice_pic = "pic2017"
)
```

**Arguments**

|                     |  |
|---------------------|--|
| df                  | input data frame   |
| index_configuration | index of the configuration of groups and factors           |
| robust              | robust or classical estimation                             |
| Y                   | Y: NxT dataframe with the panel data of interest           |
| beta_est            | estimated values of beta                                   |
| g                   | Vector with estimated group membership for all individuals |
| S                   | number of estimated groups                                 |

|                      |  |
|----------------------|--|
| k                    | estimated number of common factors   |
| kg                   | vector with the estimated number of group specific factors for each group  |
| est_errors           | NxT matrix with the error terms  |
| C_candidates         | candidates for C (parameter in PIC)  |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual".   |
| choice_pic           | parameter that defines which PIC is used to select the best configuration of groups and factors. Options are "pic2017" (uses the PIC of Ando and Bai (2017)), "pic2016" (Ando and Bai (2016)) weighs the fourth term with an extra factor relative to the size of the groups, and "pic2022". They differ in the penalty they perform on the number of group specific factors (and implicitly on the number of groups). They also differ in the sense that they have different NT-regions (where N is the number of time series and T is the length of the time series) where the estimated number of groups, and thus group specific factors will be wrong. Pic2022 is designed to shrink the problematic NT-region to very large N / very small T). |

**Value**

data.frame

**Examples**

```

set.seed(1)
original_data <- create_data_dgp2(30, 10)
Y <- original_data[[1]]
g <- original_data[[3]]
beta_est <- matrix(rnorm(4 * nrow(Y)), nrow = 4)
df_pic <- initialise_df_pic(1:5)
e <- matrix(rnorm(nrow(Y) * ncol(Y)), nrow(Y))
add_pic(df_pic, 1, TRUE, Y, beta_est, g, 3, 0, c(3, 3, 3), e, 1:5)

```

---

add\_pic\_parallel      *Calculates the PIC for the current configuration.*

---

**Description**

Calculates the PIC for the current configuration.

**Usage**

```

add_pic_parallel(
  Y,
  beta_est,
  g,

```



```

    S,
    k,
    kg,
    robust,
    est_errors,
    C_candidates,
    choice_pic,
    method_estimate_beta = "individual"
)

```

### Arguments

|                      |  |
|----------------------|--|
| Y                    | Y: NxT dataframe with the panel data of interest   |
| beta_est             | estimated values of beta   |
| g                    | Vector with estimated group membership for all individuals   |
| S                    | number of estimated groups   |
| k                    | estimated number of common factors   |
| kg                   | vector with the estimated number of group specific factors for each group  |
| robust               | robust or classical estimation   |
| est_errors           | NxT matrix with the error terms  |
| C_candidates         | candidates for C (parameter in PIC)  |
| choice_pic           | parameter that defines which PIC is used to select the best configuration of groups and factors. Options are "pic2017" (uses the PIC of Ando and Bai (2017)), "pic2016" (Ando and Bai (2016)) weighs the fourth term with an extra factor relative to the size of the groups, and "pic2022". They differ in the penalty they perform on the number of group specific factors (and implicitly on the number of groups). They also differ in the sense that they have different NT-regions (where N is the number of time series and T is the length of the time series) where the estimated number of groups, and thus group specific factors will be wrong. Pic2022 is designed to shrink the problematic NT-region to very large N / very small T). |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual".   |

### Value

numeric vector with a value for each candidate C

---

beta\_true\_heterogroups

*Helpfunction in create\_true\_beta() for the option beta\_true\_heterogeneous\_groups. (This is the default option.)*

---

### Description

Helpfunction in create\_true\_beta() for the option beta\_true\_heterogeneous\_groups. (This is the default option.)

### Usage

```
beta_true_heterogroups(
  vars,
  S_true,
  extra_beta_factor = 1,
  limit_true_groups = 12
)
```

### Arguments

vars                    number of observable variables

S\_true                 true number of groups: should be at least 1 and maximum limit\_true\_groups

extra\_beta\_factor     option to multiply the coefficients in true beta; default = 1

limit\_true\_groups     Maximum number of true groups in a simulation-DGP for which the code in this package is implemented. Currently equals 12. For application on realworld data this parameter is not relevant.

### Value

matrix where the number of rows equals S\_true, and the number of columns equals max(1, vars)

---

calculate\_best\_config *Function that returns for each candidate C the best number of groups and factors, based on the PIC.*

---

### Description

Function that returns for each candidate C the best number of groups and factors, based on the PIC.

### Usage

```
calculate_best_config(df_results, df_pic, C_candidates, limit_est_groups = 20)
```



**Arguments**

|                                  |  |
|----------------------------------|--|
| group                            | group  |
| LF                               | NxT-matrix of the common factorstructure   |
| virtual_grouped_factor_structure | list with length the number of groups; every element of the list contains NxT-matrix   |
| NN                               | number of time series  |
| TT                               | length of time series  |
| k                                | number of common factors to be estimated   |
| kg                               | number of group specific factors to be estimated   |
| vars_est                         | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| method_estimate_beta             | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| Y                                | Y: NxT dataframe with the panel data of interest   |
| X                                | dataframe with the observed variables  |
| beta_est                         | estimated values of beta   |
| g                                | Vector with estimated group membership for all individuals   |

**Value**

NxT matrix with the errorterms (=Y minus the estimated factorstructure(s) and minus X\*beta)

---

calculate\_error\_term *Calculates the error term  $Y - X*beta\_est - LF - LgFg$ .*

---

**Description**

Calculates the error term  $Y - X*beta\_est - LF - LgFg$ .

**Usage**

```
calculate_error_term(
  Y,
  X,
  beta_est,
  g,
  factor_group,
  lambda_group,
  comfactor,
  lambda,
```

```

    S,
    k,
    kg,
    method_estimate_beta = "individual",
    no_common_factorstructure = FALSE,
    no_group_factorstructure = FALSE
  )

```

### Arguments

|                           |  |
|---------------------------|--|
| Y                         | Y: NxT dataframe with the panel data of interest   |
| X                         | X: NxTxp array containing the observable variables   |
| beta_est                  | estimated values of beta   |
| g                         | Vector with estimated group membership for all individuals   |
| factor_group              | estimated group specific factors   |
| lambda_group              | loadings of the estimated group specific factors   |
| comfactor                 | estimated common factors   |
| lambda                    | loadings of the estimated common factors   |
| S                         | number of estimated groups   |
| k                         | number of common factors to be estimated   |
| kg                        | number of group specific factors to be estimated   |
| method_estimate_beta      | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| no_common_factorstructure | if there is a common factorstructure being estimated   |
| no_group_factorstructure  | if there is a group factorstructure being estimated  |

### Value

NxT matrix

### Examples

```

X <- X_dgp3
Y <- Y_dgp3
# Set estimations for group factors and its loadings, and group membership
# to the true value for this example.
lambda_group <- lambda_group_true_dgp3
factor_group <- factor_group_true_dgp3
g <- g_true_dgp3
set.seed(1)
beta_est <- matrix(rnorm(nrow(Y) * 4), ncol = nrow(Y)) #random values for beta
comfactor <- matrix(0, ncol = ncol(Y))

```

```
lambda <- matrix(0, ncol = nrow(Y))
calculate_error_term(Y, X, beta_est, g, factor_group, lambda_group, comfactor, lambda,
  3, 0, c(3, 3, 3))
```

---

calculate\_FL\_group\_estimated

*Returns the estimated groupfactorstructure.*

---

### Description

Returns the estimated groupfactorstructure.

### Usage

```
calculate_FL_group_estimated(
  lg,
  fg,
  g,
  NN,
  TT,
  S,
  k,
  kg,
  num_factors_may_vary = TRUE
)
```

### Arguments

|                      |  |
|----------------------|--|
| lg                   | loadings of estimated group factors  |
| fg                   | estimated group factors  |
| g                    | Vector with estimated group membership for all individuals                   |
| NN                   | number of time series  |
| TT                   | length of time series  |
| S                    | number of estimated groups   |
| k                    | number of common factors to be estimated                                     |
| kg                   | number of group specific factors to be estimated                             |
| num_factors_may_vary | whether or not the number of groupfactors is constant over all groups or not |

### Value

list with NjxT matrices

---

`calculate_FL_group_true`*Calculate the true groupfactorstructure.*

---

**Description**

Calculate the true groupfactorstructure.

**Usage**

```
calculate_FL_group_true(  
  lgt,  
  fgt,  
  g_true,  
  NN,  
  TT,  
  S_true,  
  k_true,  
  kg_true,  
  num_factors_may_vary = TRUE,  
  dgp1_AB_local = FALSE  
)
```

**Arguments**

|                                   |  |
|-----------------------------------|--|
| <code>lgt</code>                  | true group factor loadings   |
| <code>fgt</code>                  | true group factors   |
| <code>g_true</code>               | vector of length NN with true group memberships                              |
| <code>NN</code>                   | number of time series  |
| <code>TT</code>                   | length of time series  |
| <code>S_true</code>               | true number of groups  |
| <code>k_true</code>               | true number of common factors  |
| <code>kg_true</code>              | true number of group factors for each group                                  |
| <code>num_factors_may_vary</code> | whether or not the number of groupfactors is constant over all groups or not |
| <code>dgp1_AB_local</code>        | gives information about which DGP we use; TRUE of FALSE                      |

**Value**

list with NjxT matrices

---

|                  |   |
|------------------|---|
| calculate_lambda | <i>calculates factor loadings of common factors</i> |
|------------------|---|

---

### Description

calculates factor loadings of common factors

### Usage

```
calculate_lambda(
  Y,
  X,
  beta_est,
  comfactor,
  factor_group,
  g,
  lgfg_list,
  k,
  kg,
  robust,
  method_estimate_beta,
  method_estimate_factors,
  verbose = FALSE,
  initialise = FALSE
)
```

### Arguments

|                         |  |
|-------------------------|--|
| Y                       | Y: NxT dataframe with the panel data of interest   |
| X                       | X: NxTxp array containing the observable variables   |
| beta_est                | estimated values of beta   |
| comfactor               | common factors   |
| factor_group            | estimated group specific factors   |
| g                       | Vector with group membership for all individuals   |
| lgfg_list               | This is a list (length number of groups) containing FgLg for every group.  |
| k                       | number of common factors to be estimated   |
| kg                      | number of group specific factors to be estimated   |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | defines method of robust estimation of the factors: "macro", "pertmm" or "cz"  |
| verbose                 | when TRUE, it prints messages  |
| initialise              | indicator of being in the initialisation phase   |



**Value**

Returns a matrix where each row contains a common factor. If the number of estimated common factors equals zero, it returns a matrix with 1 row, containing zero's.

---

calculate\_lambda\_group

*calculates factor loadings of groupfactors*

---

**Description**

returns object which includes group and id of the individuals

**Usage**

```
calculate_lambda_group(
  Y,
  X,
  beta_est,
  factor_group,
  g,
  lambda,
  comfactor,
  S,
  k,
  kg,
  robust,
  method_estimate_beta = "individual",
  method_estimate_factors = "macro",
  verbose = FALSE,
  initialise = FALSE
)
```

**Arguments**

|              |  |
|--------------|--|
| Y            | Y: NxT dataframe with the panel data of interest           |
| X            | X: NxTxp array containing the observable variables         |
| beta_est     | estimated values of beta                                   |
| factor_group | estimated group specific factors                           |
| g            | Vector with estimated group membership for all individuals |
| lambda       | loadings of the estimated common factors                   |
| comfactor    | estimated common factors                                   |
| S            | number of estimated groups                                 |
| k            | number of common factors to be estimated                   |
| kg           | number of group specific factors to be estimated           |

**robust** TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  
**method\_estimate\_beta** defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual".  
**method\_estimate\_factors** defines method of robust estimation of the factors: "macro", "pertmm" or "cz"  
**verbose** when TRUE, it prints messages  
**initialise** indicator of being in the initialisation phase

### Value

Returns a data.frame with a row for each time series. The first number of columns contain the individual loadings to the group specific factors. Furthermore "group" (group membership) and id (the order in which the time series appear in Y) are added.

### Examples

```

#' #example with data generated with DGP 2
data <- create_data_dgp2(30, 10)
Y <- data[[1]]
X <- data[[2]]
g <- data[[3]] #true group membership
set.seed(1)
beta_est <- matrix(rnorm(4 * nrow(Y)), nrow = 4)
factor_group <- data[[5]] #true values of group specific factors
comfactor <- matrix(0, nrow = 1, ncol = ncol(Y))
lambda <- matrix(0, nrow = 1, ncol = nrow(Y))
calculate_lambda_group(Y, X, beta_est, factor_group, g, lambda, comfactor,
3, 0, c(3, 3, 3), TRUE)

```

---

|                |   |
|----------------|---|
| calculate_lgfg | <i>Calculates the group factor structure: the matrix product of the group factors and their loadings.</i> |
|----------------|---|

---

### Description

Returns list (with as length the number of groups) with lgfg (product of grouploadings and group-factors). Each element of the list with the assumption that all individuals are in the same group k. This function is used to speed up code.

### Usage

```

calculate_lgfg(
  lambda_group,
  factor_group,
  S,

```

```

    k,
    kg,
    num_factors_may_vary,
    NN,
    TT
)

```

### Arguments

|                      |   |
|----------------------|---|
| lambda_group         | loadings of the estimated group specific factors                              |
| factor_group         | estimated group specific factors  |
| S                    | number of groups  |
| k                    | number of common factors  |
| kg                   | vector with the number of group specific factors for each group               |
| num_factors_may_vary | whether or not the number of group factors is constant over all groups or not |
| NN                   | number of time series   |
| TT                   | length of time series   |

### Value

list with S elements: each element contains a matrix with NN rows and TT columns with the estimated group factor structure of this particular group

---

|                     |  |
|---------------------|--|
| calculate_obj_for_g | <i>Calculates objective function for individual i and group k in order to estimate group membership.</i> |
|---------------------|--|

---

### Description

Helpfunction in update\_g(). Depends on an not yet established group k ( cannot use lgfg\_list)

### Usage

```
calculate_obj_for_g(i, k, errors_virtual, rho_parameters, robust, TT)
```

### Arguments

|                |  |
|----------------|--|
| i              | individual                                   |
| k              | group  |
| errors_virtual | list with errors for each possible group     |
| rho_parameters | median and madn of the calculated error term |
| robust         | robust or classical estimation               |
| TT             | length of time series                        |

**Value**

numeric value

---

|               |  |
|---------------|--|
| calculate_PIC | <i>Function to determine PIC (panel information criterium)</i> |
|---------------|--|

---

**Description**

This depends on kappa1 -> kappaN, through p (=number of nonzero elements of beta\_est). The parameter 'sigma2' is the non-robust sigma2. As it is only used in term 2 to 4, it does not actually matter what its value is (needs to be > 0). It could be set to 1 as well.

**Usage**

```
calculate_PIC(
  C,
  robust,
  S,
  k,
  kg,
  e2,
  sigma2,
  NN,
  TT,
  method_estimate_beta,
  beta_est,
  g,
  vars_est,
  choice_pic = "pic2017"
)
```

**Arguments**

|        |   |
|--------|---|
| C      | determines relative contribution of the penalty terms compared to the estimation error term |
| robust | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta             |
| S      | number of estimated groups  |
| k      | number of common factors to be estimated  |
| kg     | number of group specific factors to be estimated  |
| e2     | NxT matrix with error terms   |
| sigma2 | scalar: sum of squared error terms, scaled by NT  |
| NN     | number of time series   |
| TT     | length of time series   |

|                      |   |
|----------------------|---|
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual".  |
| beta_est             | estimated values of beta  |
| g                    | Vector with estimated group membership for all individuals  |
| vars_est             | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.   |
| choice_pic           | indicates which PIC to use to estimate the number of groups and factors: options are "pic2017" (uses the PIC of Ando and Bai (2017); works better for large N), "pic2016" (Ando and Bai (2016); works better for large T) weighs the fourth term with an extra factor relative to the size of the groups, and "pic2022" which shrinks the NT-space where the number of groups and factors would be over- or underestimated compared to pic2016 and pic2017. |

**Value**

numeric

**Examples**

```
set.seed(1)
NN <- 30
TT <- 10
e <- matrix(rnorm(NN * TT), nrow = NN)
beta_est <- matrix(rnorm(NN * 4), ncol = NN) #random values for beta
g <- round(runif(NN, 1, 3))
calculate_PIC(0.51, TRUE, 3, 0, c(3, 3, 3), e, e^2/(NN*TT), NN, TT, "individual", beta_est, g, 3)
```

---

calculate\_PIC\_term1    *Function to calculate the first term of PIC (panel information criterion)*

---

**Description**

This is used in calculate\_PIC()

**Usage**

```
calculate_PIC_term1(e, robust)
```

**Arguments**

|        |                                 |
|--------|---------------------------------|
| e      | NxT matrix with the error terms |
| robust | robust or classical estimation  |

**Value**

numeric

---

|                  |  |
|------------------|--|
| calculate_sigma2 | <i>Calculates sum of squared errors, divided by NT</i> |
|------------------|--|

---

**Description**

Calculates sum of squared errors, divided by NT

**Usage**

```
calculate_sigma2(e, NN = nrow(e), TT = ncol(e))
```

**Arguments**

|    |                         |
|----|-------------------------|
| e  | matrix with error terms |
| NN | N                       |
| TT | T                       |

**Value**

numeric

**Examples**

```
Y <- Y_dgp3
set.seed(1)
e <- matrix(rnorm(nrow(Y) * ncol(Y)), nrow = nrow(Y))
calculate_sigma2(e)
```

---

|                          |                                  |
|--------------------------|----------------------------------|
| calculate_sigma2maxmodel | <i>Calculates sigma2maxmodel</i> |
|--------------------------|----------------------------------|

---

**Description**

Sigma2 is the sum of the squared errors, divided by NT. We need the sigma2 of the maxmodel to use (in term 2,3,4 of the PIC) instead of the configuration-dependent sigma2. (See paper AndoBai 2016). sigma2\_max\_model could actually be set to 1 as well, as it can be absorbed in parameter C of the PIC.

**Usage**

```
calculate_sigma2maxmodel(e, kg_max, S, S_cand, kg, k, k_cand)
```

**Arguments**

|        |   |
|--------|---|
| e      | NxT-matrix containing the estimated error term                            |
| kg_max | scalar: maximum allowed number of estimated factors for any group         |
| S      | estimated number of groups  |
| S_cand | vector with candidate values for the number of groups                     |
| kg     | vector with the estimated number of group specific factors for each group |
| k      | estimated number of common factors  |
| k_cand | vector with candidate value for the number of common factors              |

**Value**

numeric

---

calculate\_TN\_factor    *Helpfunction. Calculates part of the 4th term of the PIC.*

---

**Description**

Helpfunction. Calculates part of the 4th term of the PIC.

**Usage**

```
calculate_TN_factor(TT, Nj)
```

**Arguments**

|    |                                  |
|----|----------------------------------|
| TT | length of time series            |
| Nj | number of time series in group j |

**Value**

numeric

---

calculate\_VCsquared     *Calculates VC<sup>2</sup>, to determine the stability of the found number of groups and factors over the subsamples.*

---

### Description

VC<sup>2</sup> depends on C (this is the scale parameter in PIC). When VC<sup>2</sup> is equal to zero, the found number of groups and factors are the same over the subsamples.

### Usage

```
calculate_VCsquared(
  rcj,
  rc,
  C_candidates,
  indices_subset,
  Smax,
  limit_est_groups = 20
)
```

### Arguments

|                  |  |
|------------------|--|
| rcj              | dataframe containing the number of groupfactors for all candidate C's and all subsamples   |
| rc               | dataframe containing the number of common factors for all candidate C's and all subsamples |
| C_candidates     | candidates for C (parameter in PIC)  |
| indices_subset   | all indices of the subsets   |
| Smax             | maximum allowed number of estimated groups   |
| limit_est_groups | maximum allowed number of groups that can be estimated                                     |

### Value

numeric vector with the VC<sup>2</sup>-value for each candidate C

### Examples

```
rcj <- data.frame(X1 = rep("3_3_3", 5), X2 = rep("3_2_1", 5))
rc <- data.frame(X1 = rep(1, 5), X2 = rep(0, 5))
calculate_VCsquared(rcj, rc, 1:5, 0:1, 3)
```



---

```
calculate_virtual_factor_and_lambda_group
    Helpfunction used in update_g()
```

---

### Description

This function calculates FgLg (the groupfactorstructure) for all possible groups where individual i can be placed. For each group were the groupfactors (Fg) estimated earlier. Now the grouploadings are needed for each group as well. In the classical case these are calculated by  $Fg*Y/T$ . In the robust case these are robust.

### Usage

```
calculate_virtual_factor_and_lambda_group(
    group,
    solve_FG_FG_times_FG,
    robust,
    NN_local,
    method_estimate_factors_local,
    g,
    vars_est,
    number_of_group_factors_local,
    number_of_common_factors_local,
    method_estimate_beta,
    factor_group,
    lambda,
    comfactor,
    Y,
    X,
    beta_est,
    verbose = FALSE
)
```

### Arguments

|                               |   |
|-------------------------------|---|
| group                         | number of groups  |
| solve_FG_FG_times_FG          | This is the same as groupfactor / T. It is only used in the Classical approach  |
| robust                        | robust or classical estimation of group membership  |
| NN_local                      | number of time series   |
| method_estimate_factors_local | specifies the robust algorithm to estimate factors: default is "macro"  |
| g                             | vector with estimated group membership for all individuals  |
| vars_est                      | number of variables that are included in the algorithm and have their coefficient estimated. This is usually equal to vars. |

|                                |  |
|--------------------------------|--|
| number_of_group_factors_local  | number of group factors to be estimated  |
| number_of_common_factors_local | number of common factors to be estimated   |
| method_estimate_beta           | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| factor_group                   | estimated group specific factors   |
| lambda                         | loadings of the estimated common factors   |
| comfactor                      | estimated common factors   |
| Y                              | Y: NxT dataframe with the panel data of interest   |
| X                              | X: NxTxp array containing the observable variables   |
| beta_est                       | estimated values of beta   |
| verbose                        | when TRUE, it prints messages  |

**Value**

NxT matrix containing the product of virtual groupfactors and virtual loadings

---

|             |   |
|-------------|---|
| calculate_W | <i>Calculates <math>W = Y - X*beta\_est</math>. It is used in the initialization step of the algorithm, to initialise the factorstructures.</i> |
|-------------|---|

---

**Description**

Calculates  $W = Y - X*beta\_est$ . It is used in the initialization step of the algorithm, to initialise the factorstructures.

**Usage**

```
calculate_W(Y, X, beta_est, g, vars_est, method_estimate_beta)
```

**Arguments**

|                      |  |
|----------------------|--|
| Y                    | Y: NxT dataframe with the panel data of interest   |
| X                    | X: NxTxp array containing the observable variables   |
| beta_est             | estimated values of beta   |
| g                    | Vector with group membership for all individuals   |
| vars_est             | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |

**Value**

NxT matrix

---

calculate\_XB\_estimated

*Calculates (the estimated value of) the matrix  $X*\beta_{est}$ .*

---

**Description**

Calculates (the estimated value of) the matrix  $X*\beta_{est}$ .

**Usage**

calculate\_XB\_estimated(X, beta\_est, g, vars\_est, method\_estimate\_beta, TT)

**Arguments**

|                      |  |
|----------------------|--|
| X                    | X: NxTxp array containing the observable variables   |
| beta_est             | estimated values of beta   |
| g                    | Vector with estimated group membership for all individuals   |
| vars_est             | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| TT                   | length of time series  |

**Value**

Returns a NxT matrix. If vars\_est is set to 0, it returns NA.

---

calculate\_XB\_true

*Calculates the product of  $X*\beta_{true}$ .*

---

**Description**

Calculates the product of  $X*\beta_{true}$ .

**Usage**

calculate\_XB\_true(X, beta\_true, g, g\_true, method\_estimate\_beta)

**Arguments**

|                      |  |
|----------------------|--|
| X                    | X: NxTxp array containing the observable variables   |
| beta_true            | true coefficients of the observable variables  |
| g                    | Vector with estimated group membership for all individuals   |
| g_true               | true group membership  |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |

**Value**

Returns a NxT matrix (if method\_estimate\_beta == "individual"), and otherwise NA.

---

|                    |   |
|--------------------|---|
| calculate_Z_common | <i>Calculates <math>Z = Y - X * \text{beta\_est} - LgFg</math>. It is used in the estimate of the common factorstructure.</i> |
|--------------------|---|

---

**Description**

Calculates  $Z = Y - X * \text{beta\_est} - LgFg$ . It is used in the estimate of the common factorstructure.

**Usage**

```
calculate_Z_common(
  Y,
  X,
  beta_est,
  g,
  lgfg_list,
  vars_est,
  kg,
  method_estimate_beta,
  method_estimate_factors,
  initialise = FALSE
)
```

**Arguments**

|           |   |
|-----------|---|
| Y         | Y: NxT dataframe with the panel data of interest                          |
| X         | X: NxTxp array containing the observable variables                        |
| beta_est  | estimated values of beta  |
| g         | Vector with group membership for all individuals                          |
| lgfg_list | This is a list (length number of groups) containing FgLg for every group. |

|                         |  |
|-------------------------|--|
| vars_est                | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| kg                      | number of group specific factors to be estimated   |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | defines method of robust estimation of the factors: "macro", "pertmm" or "cz"  |
| initialise              | indicator of being in the initialisation phase   |

**Value**

NxT matrix

---

|                   |  |
|-------------------|--|
| calculate_Z_group | <i>Calculates <math>Z = Y - X * \text{beta\_est} - LF</math>. It is used to estimate the groupfactorstructure.</i> |
|-------------------|--|

---

**Description**

Calculates  $Z = Y - X * \text{beta\_est} - LF$ . It is used to estimate the groupfactorstructure.

**Usage**

```
calculate_Z_group(
  Y,
  X,
  beta_est,
  g,
  lambda,
  comfactor,
  group,
  k,
  method_estimate_beta,
  initialise,
  vars_est
)
```

**Arguments**

|          |  |
|----------|--|
| Y        | Y: NxT dataframe with the panel data of interest   |
| X        | X: NxTxp array containing the observable variables |
| beta_est | estimated values of beta                           |
| g        | Vector with group membership for all individuals   |

|                      |  |
|----------------------|--|
| lambda               | loadings of the estimated common factors   |
| comfactor            | estimated common factors   |
| group                | indexnumber of the group   |
| k                    | number of common factors to be estimated   |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| initialise           | indicator of being in the initialisation phase   |
| vars_est             | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |

**Value**

NxT matrix

---

check\_stopping\_rules *Checks the rules for stopping the algorithm, based on its convergence speed.*

---

**Description**

Checks the rules for stopping the algorithm, based on its convergence speed.

**Usage**

```
check_stopping_rules(
  iteration,
  speed,
  all_OF_values,
  speedlimit = 0.01,
  verbose = FALSE
)
```

**Arguments**

|               |   |
|---------------|---|
| iteration     | number of iteration   |
| speed         | convergence speed   |
| all_OF_values | vector containing the values of the objective function from previous iterations |
| speedlimit    | if the convergence speed falls under this limit the algorithm stops             |
| verbose       | if TRUE, more information is printed  |

**Value**

logical

**Examples**

```
check_stopping_rules(4, 1.7, 5:1)
```

---

```
clustering_with_robust_distances
```

*Function that puts individuals in a separate "class zero", when their distance to all possible groups is bigger than a certain threshold.*

---

**Description**

It starts with defining a robust location and scatter (based on Ma & Genton (2000): Highly robust estimation of the autocovariance function).

**Usage**

```
clustering_with_robust_distances(g, number_of_groups, Y)
```

**Arguments**

|                  |  |
|------------------|--|
| g                | Vector with group membership for all individuals |
| number_of_groups | number of groups                                 |
| Y                | Y: the panel data of interest                    |

**Value**

numeric vector with the new clustering, now including class zero adjustments

---

```
create_covMat_crosssectional_dependence
```

*Function used in generating simulated data with non normal errors.*

---

**Description**

Used to include cross-sectional dependence or serial dependence into the simulated panel data.

**Usage**

```
create_covMat_crosssectional_dependence(parameter, NN)
```

**Arguments**

|           |                                      |
|-----------|--------------------------------------|
| parameter | amount of cross-sectional dependence |
| NN        | number of time series                |

**Value**

NxN covariance matrix

---

|                  |   |
|------------------|---|
| create_data_dgp2 | <i>Creates an instance of DGP 2, as defined in Boudt and Heyndels (2022).</i> |
|------------------|---|

---

### Description

The default has 3 groups with each 3 group specific factors. Further it contains 0 common factors and 3 observed variables. The output is a list where the first element is the simulated panel dataset (a dataframe with N (amount of time series) rows and T (length of time series) columns). The second element contains the NxTxp array with the p observed variables. The third element contains the true group membership. The fourth element contains the true beta's (this has p+1 rows and one column for each group). The fifth element contains a list with the true group specific factors. The sixth element contains a dataframe with N rows where each row contains the group specific factor loadings that corresponds to the group specific factors. Further it contains the true group membership and an index (this corresponds to the rownumber in Y and X). The seventh and eighth elements contain the true common factor(s) and its loadings respectively.

### Usage

```
create_data_dgp2(N, TT, S_true = 3, vars = 3, k_true = 0, kg_true = c(3, 3, 3))
```

### Arguments

|         |   |
|---------|---|
| N       | number of time series                                       |
| TT      | length of time series                                       |
| S_true  | true number of groups                                       |
| vars    | number of available observed variables                      |
| k_true  | true number of common_factors                               |
| kg_true | vector with the true number of group factors for each group |

### Value

list

### Examples

```
create_data_dgp2(30, 10)
```



---

|                  |  |
|------------------|--|
| create_true_beta | <i>Creates beta_true, which contains the true values of beta (= the coefficients of X)</i> |
|------------------|--|

---

### Description

Creates beta\_true, which contains the true values of beta (= the coefficients of X)

### Usage

```
create_true_beta(
  vars,
  NN,
  S_true,
  method_true_beta = "heterogeneous_groups",
  limit_true_groups = 12,
  extra_beta_factor = 1
)
```

### Arguments

|                   |   |
|-------------------|---|
| vars              | number of observable variables  |
| NN                | number of time series   |
| S_true            | number of groups  |
| method_true_beta  | how the true values of beta are defined: "homogeneous" (equal for all individuals), "heterogeneous_groups" (equal within groups, and different between groups) or heterogeneous_individuals (different for all individuals) |
| limit_true_groups | Maximum number of true groups in a simulation-DGP for which the code in this package is implemented. Currently equals 12. For application on realworld data this parameter is not relevant.                                 |
| extra_beta_factor | multiplies coefficients in beta_est; default = 1  |

### Value

matrix with number of rows equal to number of observable variables + 1 (the first row contains the intercept) and number of columns equal to the true number of groups.

---

`define_configurations` *Constructs dataframe where the rows contains all configurations that are included and for which the estimators will be estimated.*

---

**Description**

Constructs dataframe where the rows contains all configurations that are included and for which the estimators will be estimated.

**Usage**

```
define_configurations(S_cand, k_cand, kg_cand)
```

**Arguments**

|                      |  |
|----------------------|--|
| <code>S_cand</code>  | candidates for S (number of groups)                  |
| <code>k_cand</code>  | candidates for k (number of common factors)          |
| <code>kg_cand</code> | candidates for kg (number of group specific factors) |

**Value**

data.frame

**Examples**

```
define_configurations(2:4, 0, 2:3)
```

---

`define_C_candidates` *Defines the candidate values for C.*

---

**Description**

Defines the candidate values for C.

**Usage**

```
define_C_candidates()
```

**Value**

numeric vector

**Examples**

```
define_C_candidates()
```

---

define\_kg\_candidates *Defines the set of combinations of group specific factors.*

---

### Description

Defines the set of combinations of group specific factors.

### Usage

```
define_kg_candidates(S, kg_min, kg_max, nfv = TRUE, limit_est_groups = 20)
```

### Arguments

|                  |   |
|------------------|---|
| S                | number of estimated groups  |
| kg_min           | minimum value for number of group specific factors  |
| kg_max           | minimum value for number of group specific factors  |
| nfv              | logical; whether the number of group specific factors is allowed to change among the groups |
| limit_est_groups | maximum allowed number of groups that can be estimated                                      |

### Value

Returns a data frame where each row contains the number of group specific factors for all the estimated groups. The number of columns is set to 20 (the current maximum amount of group that can be estimated)

### Examples

```
define_kg_candidates(3, 2, 4)
```

---

define\_number\_subsets *Returns a vector with the indices of the subsets. Must start with zero.*

---

### Description

Returns a vector with the indices of the subsets. Must start with zero.

### Usage

```
define_number_subsets(n)
```

### Arguments

|   |                   |
|---|-------------------|
| n | number of subsets |
|---|-------------------|

**Value**

numeric

**Examples**

```
define_number_subsets(3)
```

---

```
define_object_for_initial_clustering_macropca
```

*Defines the object that will be used to define a initial clustering.*

---

**Description**

This is a short version of `define_object_for_initial_clustering()` which only contains implementations for robust macropca case and classical case.

**Usage**

```
define_object_for_initial_clustering_macropca(
  Y,
  k,
  kg,
  comfactor,
  robust,
  method_estimate_beta = "individual",
  method_estimate_factors = "macro",
  verbose = FALSE
)
```

**Arguments**

|                         |  |
|-------------------------|--|
| Y                       | Y: NxT dataframe with the panel data of interest   |
| k                       | number of common factors to be estimated   |
| kg                      | number of group specific factors to be estimated   |
| comfactor               | estimated common factors   |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | specifies the robust algorithm to estimate factors: default is "macro". The value is not used when robust is set to FALSE.   |
| verbose                 | when TRUE, it prints messages  |

**Value**

matrix with N rows and 10 columns

---

define\_rho\_parameters *Determines parameters of rho-function.*

---

**Description**

Robust updating of group membership is based on a rho function (instead of the non-robust quadratic function) on the norm of the errors. This requires parameters of location and scale. They are defined here (currently as median and madn). This function is applied on the estimated errors:  $Y - XB - FL - FgLg$ . This function is used in `update_g()`.

**Usage**

```
define_rho_parameters(object = NULL)
```

**Arguments**

object           input

**Value**

list

---

determine\_beta           *Helpfunction in estimate\_beta() for estimating beta\_est.*

---

**Description**

Helpfunction in `estimate_beta()` for estimating `beta_est`.

**Usage**

```
determine_beta(  
  string,  
  X_special,  
  Y_special,  
  robust,  
  NN,  
  TT,  
  S,  
  method_estimate_beta,  
  initialisation = FALSE,  
  indices = NA,
```

```

vars_est,
sigma2,
nosetting_local = FALSE,
kappa_candidates = c(2^(-0:-20), 0)
)

```

### Arguments

|                      |  |
|----------------------|--|
| string               | can have values: "homogeneous" (when one beta_est is estimated for all individuals together) or "heterogeneous" (when beta_est is estimated either groupwise or elementwise)     |
| X_special            | preprocessed X (2-dimensional matrix with 'var_est' observable variables)  |
| Y_special            | preprocessed Y   |
| robust               | robust or classical estimation   |
| NN                   | number of time series  |
| TT                   | length of time series  |
| S                    | estimated number of groups   |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| initialisation       | indicator of being in the initialisation phase   |
| indices              | individuals for which beta_est is being estimated  |
| vars_est             | number of available observed variables for which a coefficient will be estimated. As default it is equal to the number of available observed variables.                          |
| sigma2               | sum of squared error terms, scaled by NT   |
| nosetting_local      | option to remove the recommended setting in lmrob(). It is much faster. Defaults to FALSE.   |
| kappa_candidates     | Defines the size of the SCAD-penalty used in the classical algorithm. This vector should contain more than 1 element.  |

### Value

The function returns a numeric vector (for the default setting: string == "heterogeneous") or a matrix with the estimated beta (if string == "homogeneous").

---

determine\_robust\_lambda

*Help-function for return\_robust\_lambdaobject().*


---

### Description

Uses the "almost classical lambda" (=matrix where the mean of each row is equal to the classical lambda) to create a robust lambda by using M estimation.

### Usage

```
determine_robust_lambda(
  almost_classical_lambda,
  fastoption = TRUE,
  fastoption2 = FALSE
)
```

### Arguments

|                         |  |
|-------------------------|--|
| almost_classical_lambda | matrix where the mean of each row is equal to the classical lambda                                 |
| fastoption              | Uses nlm() instead of optim(). This is faster.   |
| fastoption2             | experimental parameter: can speed nlm() up (10%), but loses accuracy. May benefit from finetuning. |

### Value

M-estimator of location of the parameter, by minimizing sum of rho()

---

|                    |  |
|--------------------|--|
| df_results_example | <i>An example for df_results. This dataframe contains the estimators for each configuration.</i> |
|--------------------|--|

---

### Description

An example for df\_results. This dataframe contains the estimators for each configuration.

### Usage

```
df_results_example
```

**Format**

Dataframe with 4 rows (one for each configuration) and 11 columns:

**S** number of groups

**k\_common** number of common factors

**k1** number of group specific factors in group 1

**k2** number of group specific factors in group 2

**k3** number of group specific factors in group 3

**g** estimated group membership

**beta\_est** estimated beta

**factor\_group** estimated group specific factors

**lambda\_group** estimated loadings to the group specific factors

**comfactor** estimated common factors

**lambda\_group** estimated loadings to the common factors

---

do\_we\_estimate\_common\_factors

*Helpfunction to shorten code: are common factors being estimated.*

---

**Description**

Helpfunction to shorten code: are common factors being estimated.

**Usage**

```
do_we_estimate_common_factors(k)
```

**Arguments**

k                    number of common factors

**Value**

numeric: 0 or 1



---

do\_we\_estimate\_group\_factors

*Helpfunction to shorten code: are group factors being estimated.*


---

**Description**

Helpfunction to shorten code: are group factors being estimated.

**Usage**

```
do_we_estimate_group_factors(kg)
```

**Arguments**

kg                    number of group factors to be estimated

**Value**

numeric: 0 or 1

---

|                    |  |
|--------------------|--|
| estimate_algorithm | <i>This function is a wrapper around the initialization and the estimation part of the algorithm, for one configuration. It is only used for the serialized algorithm.</i> |
|--------------------|--|

---

**Description**

This function is a wrapper around the initialization and the estimation part of the algorithm, for one configuration. It is only used for the serialized algorithm.

**Usage**

```
estimate_algorithm(Y, X, S, k, kg, maxit = 30, robust = TRUE)
```

**Arguments**

|        |   |
|--------|---|
| Y      | Y: NxT dataframe with the panel data of interest                                |
| X      | X: NxT <sub>p</sub> array containing the observable variables                   |
| S      | number of estimated groups  |
| k      | number of common factors to be estimated  |
| kg     | number of group specific factors to be estimated                                |
| maxit  | maximum limit for the number of iterations                                      |
| robust | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta |

**Value**

list with

1. estimated beta
2. vector with group membership
3. matrix with the common factor(s) (contains zero's if there are none estimated)
4. loadings to the common factor(s)
5. list with the group specific factors for each of the groups
6. data.frame with loadings to the group specific factors augmented with group membership and id (to have the order of the time series)

**Examples**

```
set.seed(1)
original_data <- create_data_dgp2(60, 30)
Y <- original_data[[1]]
X <- original_data[[2]]
estimate_algorithm(Y, X, 3, 0, c(3,3,3), maxit = 2, robust = TRUE)
```

---

estimate\_beta

*Estimates beta.*

---

**Description**

Update step of algorithm to obtain new estimation for beta. Note that we call it beta\_est because beta() exists in base R.

**Usage**

```
estimate_beta(
  Y,
  X,
  beta_est,
  g,
  lambda_group,
  factor_group,
  lambda,
  comfactor,
  method_estimate_beta = "individual",
  S,
  k,
  kg,
  vars_est,
  robust,
  num_factors_may_vary = TRUE,
```

```

    optimize_kappa = FALSE,
    nosetting = FALSE
  )

```

### Arguments

|                      |  |
|----------------------|--|
| Y                    | Y: NxT dataframe with the panel data of interest   |
| X                    | X: NxTxp array containing the observable variables   |
| beta_est             | estimated values of beta   |
| g                    | Vector with estimated group membership for all individuals   |
| lambda_group         | loadings of the estimated group specific factors   |
| factor_group         | estimated group specific factors   |
| lambda               | loadings of the estimated common factors   |
| comfactor            | estimated common factors   |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| S                    | number of estimated groups   |
| k                    | number of common factors to be estimated   |
| kg                   | number of group specific factors to be estimated   |
| vars_est             | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| robust               | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| num_factors_may_vary | whether or not the number of groupfactors is constant over all groups or not   |
| optimize_kappa       | indicates if kappa has to be optimized or not (only relevant for the classical algorithm)  |
| nosetting            | option to remove the recommended setting in lmrob(). It is much faster. Defaults to FALSE.   |

### Value

list: 1st element contains matrix (N columns: 1 for each time series of the panel data) with estimated beta\_est's. If vars\_est is set to 0, the list contains NA.

### Examples

```

X <- X_dgp3
Y <- Y_dgp3
# Set estimations for group factors and its loadings, and group membership to the true value
lambda_group <- lambda_group_true_dgp3
factor_group <- factor_group_true_dgp3
g <- g_true_dgp3
# There are no common factors to be estimated -> but needs placeholder

```

```

lambda <- matrix(0, nrow = 1, ncol = 300)
comfactor <- matrix(0, nrow = 1, ncol = 30)
#
# Choose how coefficients of the observable variables are estimated
method_estimate_beta <- "individual"
method_estimate_factors <- "macro"
beta_est <- estimate_beta(
  Y, X, NA, g, lambda_group, factor_group,
  lambda, comfactor,
  S = 3, k = 0, kg = c(3, 3, 3),
  vars_est = 3,
  robust = TRUE
)[[1]]

```

---

|                 |                                      |
|-----------------|--------------------------------------|
| estimate_factor | <i>Estimates common factor(s) F.</i> |
|-----------------|--------------------------------------|

---

### Description

The estimator for  $F$ , see Anderson (1984), is equal to the first  $k$  eigenvectors (multiplied by  $\sqrt{T}$ ) due to the restriction  $F'F/T = I$  associated with first  $r$  largest eigenvalues of the matrix  $WW'$  (which is of size  $T \times T$ ).

### Usage

```

estimate_factor(
  Y,
  X,
  beta_est,
  g,
  lgfg_list,
  k,
  kg,
  robust,
  method_estimate_beta,
  method_estimate_factors,
  initialise = FALSE,
  verbose = FALSE
)

```

### Arguments

|          |  |
|----------|--|
| Y        | Y: $N \times T$ dataframe with the panel data of interest          |
| X        | X: $N \times T \times p$ array containing the observable variables |
| beta_est | estimated values of beta   |
| g        | Vector with group membership for all individuals                   |

|                         |  |
|-------------------------|--|
| lgfg_list               | This is a list (length number of groups) containing FgLg for every group.  |
| k                       | number of common factors to be estimated   |
| kg                      | number of group specific factors to be estimated   |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | defines method of robust estimation of the factors: "macro", "pertmm" or "cz"  |
| initialise              | indicator of being in the initialisation phase   |
| verbose                 | when TRUE, it prints messages  |

**Value**

Return a list. The first element contains the  $k \times T$  matrix with the  $k$  estimated common factors. The second element contains either the robust MacroPCA-based loadings or NA.

---

estimate\_factor\_group *Estimates group factors Fg.*

---

**Description**

Estimates group factors Fg.

**Usage**

```
estimate_factor_group(
  Y,
  X,
  beta_est,
  g,
  lambda,
  comfactor,
  factor_group,
  S,
  k,
  kg,
  robust,
  method_estimate_beta = "individual",
  method_estimate_factors = "macro",
  initialise = FALSE,
  verbose = FALSE
)
```

**Arguments**

|                         |  |
|-------------------------|--|
| Y                       | Y: NxT dataframe with the panel data of interest   |
| X                       | X: NxTxp array containing the observable variables   |
| beta_est                | estimated values of beta   |
| g                       | Vector with group membership for all individuals   |
| lambda                  | loadings of the estimated common factors   |
| comfactor               | estimated common factors   |
| factor_group            | estimated group specific factors   |
| S                       | number of estimated groups   |
| k                       | number of common factors to be estimated   |
| kg                      | number of group specific factors to be estimated   |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | defines method of robust estimation of the factors: "macro", "pertmm" or "cz"  |
| initialise              | indicator of being in the initialisation phase   |
| verbose                 | when TRUE, it prints messages  |

**Value**

Returns a list with an element for each estimated group. Each element of the list is a matrix with the group specific factors as rows.

**Examples**

```
#example with data generated with DGP 2
data <- create_data_dgp2(30, 10)
Y <- data[[1]]
X <- data[[2]]
g <- data[[3]] #true group membership
set.seed(1)
beta_est <- matrix(rnorm(4 * nrow(Y)), nrow = 4)
factor_group <- data[[5]] #true values of group specific factors
comfactor <- matrix(0, nrow = 1, ncol = ncol(Y))
lambda <- matrix(0, nrow = 1, ncol = nrow(Y))
estimate_factor_group(Y, X, beta_est, g, lambda, comfactor, factor_group,
3, 0, c(3, 3, 3), TRUE)
```

---

`evade_crashes_macropca`*Solves a very specific issue with MacroPCA.*

---

**Description**

MacroPCA crashes Rstudio with certain dimensions of the input. Solve this by doubling every row. No information is added by this, so there is no influence on the end result, but crashes of Rstudio are evaded.

**Usage**

```
evade_crashes_macropca(object, verbose = FALSE)
```

**Arguments**

|         |                 |
|---------|-----------------|
| object  | input           |
| verbose | prints messages |

**Value**

matrix

---

`evade_floating_point_errors`*Function to evade floating point errors.*

---

**Description**

Sets values that should be zero but are  $>0$  (e.g.  $1e-13$ ) on zero.

**Usage**

```
evade_floating_point_errors(x, LIMIT = 1e-13)
```

**Arguments**

|       |                                     |
|-------|-------------------------------------|
| x     | numeric input                       |
| LIMIT | limit under which value is set to 0 |

**Value**

numeric

---

factor\_group\_true\_dgp3

*factor\_group\_true\_dgp3 contains the values of the true group factors on which Y\_dgp3 is based*

---

### Description

factor\_group\_true\_dgp3 contains the values of the true group factors on which Y\_dgp3 is based

### Usage

factor\_group\_true\_dgp3

### Format

list with length 3: each element has dimension 3 x 30

---

fill\_rc

*Fills in the optimized number of common factors for each C.*

---

### Description

Fills in the optimized number of common factors for each C.

### Usage

fill\_rc(df, all\_best\_values, subset)

### Arguments

|                 |   |
|-----------------|---|
| df              | input   |
| all_best_values | data frame with the optimal number of groups, common factors and group specific factors |
| subset          | index of the subsample  |

### Value

data.frame

### Examples

```
df_results <- add_configuration(initialise_df_results(TRUE),
  3, 0, c(3, 3, 3, rep(NA, 17))) #data.frame with one configuration
all_best_values <- calculate_best_config(df_results, data.frame(t(1:5)), 1:5)
rc <- fill_rc(initialise_rc(0:2, 1:5), all_best_values, 1)
```



---

|          |   |
|----------|---|
| fill_rcj | <i>Fills in the optimized number of groups and group specific factors for each C.</i> |
|----------|---|

---

**Description**

Fills in the optimized number of groups and group specific factors for each C.

**Usage**

```
fill_rcj(df, all_best_values, subset, S_cand, kg_cand)
```

**Arguments**

|                 |   |
|-----------------|---|
| df              | input   |
| all_best_values | data frame with the optimal number of groups, common factors and group specific factors |
| subset          | index of the subsample  |
| S_cand          | vector with candidate values for the number of estimated groups                         |
| kg_cand         | vector with candidate values for the number of estimated group specific factors         |

**Value**

data.frame

**Examples**

```
df_results <- add_configuration(initialise_df_results(TRUE),
  3, 0, c(3, 3, 3, rep(NA, 17))) #data.frame with one configuration
all_best_values <- calculate_best_config(df_results, data.frame(t(1:5)), 1:5)
rcj <- fill_rcj(initialise_rcj(0:2, 1:5) , all_best_values, 1, 2:4, 2:4)
```

---

|                             |   |
|-----------------------------|---|
| final_estimations_filter_kg | <i>Filters dataframe on the requested group specific factors configuration.</i> |
|-----------------------------|---|

---

**Description**

Filters dataframe on the requested group specific factors configuration.

**Usage**

```
final_estimations_filter_kg(df, kg)
```

**Arguments**

|    |  |
|----|--|
| df | input dataframe  |
| kg | vector with number of group specific factors for each group, on which should be filtered |

**Value**

data.frame

---

generate\_grouped\_factorstructure

*Generates the true groupfactorstructure, to use in simulations.*

---

**Description**

Loadings and factors are generated by: factors  $\sim N(j * \text{fgr\_factor\_mean}, \text{fgr\_factor\_sd})$  -> default case will be  $N(j, 1)$  loadings  $\sim N(\text{lgr\_factor\_mean}, j * \text{lgr\_factor\_sd})$  -> default case will be  $N(0, j)$

**Usage**

```
generate_grouped_factorstructure(
  S,
  kg_true,
  TT,
  g_true,
  lgr_factor_mean = 0,
  lgr_factor_sd = 1,
  fgr_factor_mean = 1,
  fgr_factor_sd = 1
)
```

**Arguments**

|                 |  |
|-----------------|--|
| S               | true number of groups  |
| kg_true         | vector with as length the number of groups, where each element is the true number of groupfactors of that group.                           |
| TT              | length of time series  |
| g_true          | vector of length NN with true group memberships  |
| lgr_factor_mean | mean of the normal distribution from which the loadings are generated  |
| lgr_factor_sd   | sd of the normal distribution from which the loadings are generated (multiplied by a coefficient for each different group)                 |
| fgr_factor_mean | mean of the normal distribution from which the group specific factors are generated (multiplied by a coefficient for each different group) |
| fgr_factor_sd   | sd of the normal distribution from which the group specific factors are generated  |

**Value**

list: first element contains the true group specific factors and the second element contains the corresponding loadings

---

|            |   |
|------------|---|
| generate_Y | <i>Generate panel data Y for simulations.</i> |
|------------|---|

---

**Description**

Generate panel data Y for simulations.

**Usage**

```
generate_Y(
  NN,
  TT,
  k_true,
  kg_true,
  g_true,
  beta_true,
  lambda_group_true,
  factor_group_true,
  lambda_true,
  comfactor_true,
  eps,
  X
)
```

**Arguments**

|                   |   |
|-------------------|---|
| NN                | number of time series   |
| TT                | length of time series   |
| k_true            | true number of common factors   |
| kg_true           | Vector of length the number of groups. Each element contains the true number of group factors for that group. |
| g_true            | vector of length NN with true group memberships   |
| beta_true         | true coefficients of the observable variables   |
| lambda_group_true | loadings of the true group specific factors   |
| factor_group_true | true group specific factors   |
| lambda_true       | loadings of the true common factors   |
| comfactor_true    | true common factors   |
| eps               | NN x TT-matrix containing the error term  |
| X                 | dataframe with the observed variables   |

**Value**

NN x TT matrix

---

get\_best\_configuration

*Finds the first stable interval after the first unstable point. It then defines the value for C for the begin, middle and end of this interval.*

---

**Description**

Finds the first stable interval after the first unstable point. It then defines the value for C for the begin, middle and end of this interval.

**Usage**

```
get_best_configuration(
  list_vc,
  list_rc,
  list_rcj,
  C_candidates,
  S_cand,
  return_short = FALSE,
  verbose = FALSE
)
```

**Arguments**

|              |   |
|--------------|---|
| list_vc      | list with resulting expression( $VC^2$ ) for each run   |
| list_rc      | list with resulting rc for each run   |
| list_rcj     | list with resulting rcj for each run  |
| C_candidates | candidates for C  |
| S_cand       | candidates for S (number of groups)   |
| return_short | if TRUE, the function returns the dataframe filtered for several specified potential candidates for C |
| verbose      | when TRUE, it prints messages   |

**Value**

data.frame with the optimized configuration for each candidate C (if return\_short is FALSE) and for each of the selected C's in the chosen stable interval (if return\_short is TRUE).

**Examples**

```

set.seed(1)
all_best_values <- calculate_best_config(add_configuration(initialise_df_results(TRUE),
  3, 0, c(3, 3, 3, rep(NA, 17))),
  data.frame(t(1:5)), 1:5)
rc <- fill_rc(initialise_rc(0:1, 1:5), all_best_values, 0)
rc <- fill_rc(rc, all_best_values, 1)
rcj <- fill_rcj(initialise_rcj(0:1, 1:5) , all_best_values, 0, 2:4, 2:4)
rcj <- fill_rcj(rcj, all_best_values, 1, 2:4, 2:4)
get_best_configuration(sort(runif(5)), rc, rcj, 1:5, 2:4, return_short = FALSE)

```

---

get\_convergence\_speed *Defines the convergence speed.*

---

**Description**

Defines the convergence speed.

**Usage**

```
get_convergence_speed(iteration, of)
```

**Arguments**

|           |                     |
|-----------|---------------------|
| iteration | number of iteration |
| of        | objective function  |

**Value**

numeric if iteration > 3, otherwise NA

**Examples**

```
get_convergence_speed(5, 10:1)
```

---

get\_final\_estimation *Function that returns the final clustering, based on the estimated number of groups and common and group specific factors.*

---

**Description**

Function that returns the final clustering, based on the estimated number of groups and common and group specific factors.

**Usage**

```
get_final_estimation(df, opt_groups, k, kg, type, limit_est_groups = 20)
```

**Arguments**

|                  |  |
|------------------|--|
| df               | input dataframe (this will be df_results_full)   |
| opt_groups       | the optimal number of groups   |
| k                | the optimal number of common factors   |
| kg               | vector with the optimal number of group specific factors   |
| type             | defines which estimation to return: options are "clustering", "beta", "fg" (group specific factors), "lg" (loadings corresponding to fg), "f" (common factors), "l" (loadings corresponding to f), |
| limit_est_groups | maximum allowed number of groups that can be estimated   |

**Value**

This function returns the estimations of the chosen configuration. If type is "clustering" it returns a numeric vector with the estimated group membership for all time series. If type is "beta", "lg" the function returns a data.frame. If type is "f" or "l" the function also returns a data.frame. If no common factors were estimated in the optimized configuration, then NA is returned. If type is "fg" the function returns a list.

**Examples**

```
get_final_estimation(df_results_example, 3, 0, c(3, 3, 3), "clustering")
get_final_estimation(df_results_example, 3, 0, c(3, 3, 3), "beta")
get_final_estimation(df_results_example, 3, 0, c(3, 3, 3), "fg")
get_final_estimation(df_results_example, 3, 0, c(3, 3, 3), "lg")
```

---

|                    |  |
|--------------------|--|
| grid_add_variables | <i>Function which is used to have a dataframe (called "grid") with data (individualindex, timeindex, XT and LF) available.</i> |
|--------------------|--|

---

**Description**

It is used in iterate().

**Usage**

```
grid_add_variables(
  grid,
  Y,
  X,
  beta_est,
  g,
  lambda,
  comfactor,
  method_estimate_beta,
  vars_est,
```

```

    S,
    limit_est_groups_heterogroups = 15
  )

```

### Arguments

|                               |  |
|-------------------------------|--|
| grid                          | dataframe containing values for X*beta_est and LF (product of common factor and its loadings)  |
| Y                             | Y: NxT dataframe with the panel data of interest   |
| X                             | X: NxTxp array containing the observable variables   |
| beta_est                      | estimated values of beta   |
| g                             | Vector with estimated group membership for all individuals   |
| lambda                        | loadings of the estimated common factors   |
| comfactor                     | estimated common factors   |
| method_estimate_beta          | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| vars_est                      | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables.                    |
| S                             | number of estimated groups   |
| limit_est_groups_heterogroups | maximum amount of groups that can be estimated when method_estimate_beta is set to "group"   |

### Value

data.frame

---

|             |  |
|-------------|--|
| g_true_dgp3 | <i>g_true_dgp3 contains the true group memberships of the elements of Y_dgp3</i> |
|-------------|--|

---

### Description

g\_true\_dgp3 contains the true group memberships of the elements of Y\_dgp3

### Usage

```
g_true_dgp3
```

### Format

vector with 300 elements

### Examples

```
table(g_true_dgp3)
```

---

|          |   |
|----------|---|
| handleNA | <i>Function with as input a dataframe. (this will be "Y" or "to_divide") It filters out rows with NA.</i> |
|----------|---|

---

**Description**

Function with as input a dataframe. (this will be "Y" or "to\_divide") It filters out rows with NA.

**Usage**

```
handleNA(df)
```

**Arguments**

|    |       |
|----|-------|
| df | input |
|----|-------|

**Value**

list with a dataframe where the rows with NA are filtered out, and a dataframe with only those rows

---

|             |  |
|-------------|--|
| handleNA_LG | <i>Removes NA's in LG (in function calculate_virtual_factor_and_lambda_group() )</i> |
|-------------|--|

---

**Description**

Removes NA's in LG (in function calculate\_virtual\_factor\_and\_lambda\_group() )

**Usage**

```
handleNA_LG(df)
```

**Arguments**

|    |       |
|----|-------|
| df | input |
|----|-------|

**Value**

matrix



---

`handle_macropca_errors`*Helpfunction in robustpca().*

---

**Description**

It handles possible thrown errors in MacroPCA.

**Usage**

```
handle_macropca_errors(  
  object,  
  temp,  
  KMAX,  
  number_eigenvectors,  
  verbose = FALSE  
)
```

**Arguments**

|                                  |  |
|----------------------------------|--|
| <code>object</code>              | input  |
| <code>temp</code>                | this is the result of the trycatch block of using macropca on object |
| <code>KMAX</code>                | parameter kmax in MacroPCA   |
| <code>number_eigenvectors</code> | number of principal components that are needed                       |
| <code>verbose</code>             | when TRUE, it prints messages  |

**Value**

matrix of which the columns contain the chosen amount of eigenvectors of object

---

`initialise_beta`*Initialisation of estimation of beta (the coefficients with the observable variables)*

---

**Description**

Note: this needs to be called before the definition of grid.

**Usage**

```
initialise_beta(
  Y,
  X,
  S,
  robust,
  method_estimate_beta = "individual",
  nosetting_lmrob = FALSE
)
```

**Arguments**

|                      |  |
|----------------------|--|
| Y                    | Y: NxT dataframe with the panel data of interest   |
| X                    | dataframe with the observed variables  |
| S                    | estimated number of groups   |
| robust               | robust or classical estimation   |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| nosetting_lmrob      | option to remove the recommended setting in lmrob(). It is much faster. Defaults to FALSE.   |

**Value**

Matrix with number of rows equal to the number of estimated variables plus one. If `method_estimate_beta` is set to the default ("individual"), the number of columns is equal to the number of time series in Y. If `method_estimate_beta` is set to "group" or to "homogeneous" the number of columns is equal to the number of groups.

**Examples**

```
X <- X_dgp3
Y <- Y_dgp3
# Set estimations for group factors and its loadings, and group membership
# to the true value for this example.
lambda_group <- lambda_group_true_dgp3
factor_group <- factor_group_true_dgp3
beta_init <- initialise_beta(Y, X,
  S = 3, TRUE
)
```

---

`initialise_clustering` *Function that clusters time series in a dataframe with kmeans.*

---

### Description

If a time series contains NA's a random cluster will be assigned to that time series.

### Usage

```
initialise_clustering(  
  Y,  
  S,  
  k,  
  kg,  
  comfactor,  
  robust,  
  max_percent_outliers_tkmeans = 0,  
  verbose = FALSE  
)
```

### Arguments

|                              |   |
|------------------------------|---|
| Y                            | Y: NxT dataframe with the panel data of interest                                |
| S                            | the desired number of groups  |
| k                            | number of common factors to be estimated  |
| kg                           | number of group specific factors to be estimated                                |
| comfactor                    | estimated common factors  |
| robust                       | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta |
| max_percent_outliers_tkmeans | the proportion of observations to be trimmed                                    |
| verbose                      | when TRUE, it prints messages   |

### Value

numeric vector

### Examples

```
Y <- Y_dgp3  
comfactor <- matrix(0, nrow = ncol(Y))  
initialise_clustering(Y, 3, 0, c(3, 3, 3), comfactor, TRUE)
```

---

```
initialise_commonfactorstructure_macropca
```

*Initialises the estimation of the common factors and their loadings.*

---

### Description

This is a short version of `initialise_commonfactorstructure()` which only contains implementations for the robust macropca case and the classical case.

### Usage

```
initialise_commonfactorstructure_macropca(
  Y,
  X,
  beta_est,
  g,
  factor_group,
  k,
  kg,
  robust,
  method_estimate_beta = "individual",
  method_estimate_factors = "macro",
  verbose = FALSE
)
```

### Arguments

|                         |  |
|-------------------------|--|
| Y                       | Y: NxT dataframe with the panel data of interest   |
| X                       | dataframe with the observed variables  |
| beta_est                | estimated values of beta   |
| g                       | Vector with estimated group membership for all individuals   |
| factor_group            | estimated group specific factors   |
| k                       | number of estimated common factors   |
| kg                      | vector with the number of estimated group specific factors   |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | specifies the robust algorithm to estimate factors: default is "macro". The value is not used when robust is set to FALSE.   |
| verbose                 | when TRUE, it prints messages  |

**Value**

list: 1st element contains the common factor(s) and the second element contains the factor loadings

**Examples**

```
set.seed(1)
original_data <- create_data_dgp2(30, 20)
Y <- original_data[[1]]
X <- original_data[[2]]
g <- original_data[[3]]
beta_est <- matrix(rnorm(4 * ncol(Y)), nrow = 4)
initialise_commonfactorstructure_macropca(Y, X, beta_est, g, NA, 0, c(3, 3, 3), TRUE)
```

---

|                   |   |
|-------------------|---|
| initialise_df_pic | <i>Initialises a dataframe which will contain the PIC for each configuration and for each value of C.</i> |
|-------------------|---|

---

**Description**

Initialises a dataframe which will contain the PIC for each configuration and for each value of C.

**Usage**

```
initialise_df_pic(C_candidates)
```

**Arguments**

C\_candidates candidates for C (parameter in PIC)

**Value**

Returns an empty data.frame.

**Examples**

```
initialise_df_pic(1:10)
```

---

`initialise_df_results` *Initialises a dataframe that will contain an overview of metrics for each estimated configuration (for example adjusted randindex).*

---

### Description

Initialises a dataframe that will contain an overview of metrics for each estimated configuration (for example adjusted randindex).

### Usage

```
initialise_df_results(robust, limit_est_groups = 20)
```

### Arguments

`robust`                   robust or classical estimation  
`limit_est_groups`           maximum allowed number of groups that can be estimated

### Value

Returns an empty data.frame.

### Examples

```
initialise_df_results(TRUE)
```

---

`initialise_rc`           *Initialises rc.*

---

### Description

This function initialises a data frame which will eventually be filled with the optimized number of common factors for each C and for each subset of the original dataset.

### Usage

```
initialise_rc(indices_subset, C_candidates)
```

### Arguments

`indices_subset`   all indices of the subsets  
`C_candidates`     candidates for C (parameter in PIC)

### Value

data.frame

**Examples**

```
initialise_rc(0:2, 1:5)
```

---

|                |                         |
|----------------|-------------------------|
| initialise_rcj | <i>Initialises rcj.</i> |
|----------------|-------------------------|

---

**Description**

This function initialises a data frame which will eventually be filled with the optimized number of groups and group specific factors for each C and for each subset of the original dataset.

**Usage**

```
initialise_rcj(indices_subset, C_candidates)
```

**Arguments**

indices\_subset all indices of the subsets  
 C\_candidates candidates for C (parameter in PIC)

**Value**

data.frame

**Examples**

```
initialise_rcj(0:2, 1:5)
```

---

|              |  |
|--------------|--|
| initialise_X | <i>Creates X (the observable variables) to use in simulations.</i> |
|--------------|--|

---

**Description**

X is an array with dimensions N, T and number of observable variables. The variables are randomly generated with mean 0 and sd 1.

**Usage**

```
initialise_X(NN, TT, vars, scale_robust = TRUE)
```

**Arguments**

NN number of time series  
 TT length of time series  
 vars number of available observable variables  
 scale\_robust logical, defines if X will be scaled with robust metrics instead of with non-robust metrics

**Value**

array with dimensions  $N \times T \times$  number of observable variables

---

|         |   |
|---------|---|
| iterate | <i>Wrapper around estimate_beta(), update_g(), and estimating the factorstructures.</i> |
|---------|---|

---

**Description**

Wrapper around estimate\_beta(), update\_g(), and estimating the factorstructures.

**Usage**

```
iterate(
  Y,
  X,
  beta_est,
  g,
  lambda_group,
  factor_group,
  lambda,
  comfactor,
  S,
  k,
  kg,
  robust,
  method_estimate_beta = "individual",
  method_estimate_factors = "macro",
  verbose = FALSE
)
```

**Arguments**

|              |  |
|--------------|--|
| Y            | Y: $N \times T$ dataframe with the panel data of interest          |
| X            | X: $N \times T \times p$ array containing the observable variables |
| beta_est     | estimated values of beta   |
| g            | Vector with estimated group membership for all individuals         |
| lambda_group | loadings of the estimated group specific factors                   |
| factor_group | estimated group specific factors                                   |
| lambda       | loadings of the estimated common factors                           |
| comfactor    | estimated common factors   |
| S            | number of groups to estimate                                       |
| k            | number of common factors to estimate                               |



|                         |  |
|-------------------------|--|
| kg                      | vector with length S. Each element contains the number of group specific factors to estimate.  |
| robust                  | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta  |
| method_estimate_beta    | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| method_estimate_factors | specifies the robust algorithm to estimate factors: default is "macro". The value is not used when robust is set to FALSE.   |
| verbose                 | when TRUE, it prints messages  |

### Value

list with

1. estimated beta
2. vector with group membership
3. matrix with the common factor(s) (contains zero's if there are none estimated)
4. loadings to the common factor(s)
5. list with the group specific factors for each of the groups
6. data.frame with loadings to the group specific factors augmented with group membership and id (to have the order of the time series)
7. the value of the objective function

### Examples

```
set.seed(1)
original_data <- create_data_dgp2(30, 10)
Y <- original_data[[1]]
X <- original_data[[2]]
g <- original_data[[3]]
beta_est <- matrix(rnorm(4 * ncol(Y)), nrow = 4)
factor_group <- original_data[[5]]
lambda_group <- original_data[[6]]
comfactor <- matrix(0, nrow = 1, ncol = ncol(Y))
lambda <- matrix(0, nrow = 1, ncol = nrow(Y))
iterate(Y, X, beta_est, g, lambda_group, factor_group, lambda, comfactor, 3, 0, c(3, 3, 3), TRUE,
  verbose = FALSE)
```

---

kg\_candidates\_expand    *Function that returns the set of combinations of groupfactors for which the algorithm needs to run.*

---

### Description

Function that returns the set of combinations of groupfactors for which the algorithm needs to run.

### Usage

```
kg_candidates_expand(S, kg_min, kg_max, limit_est_groups = 20)
```

### Arguments

|                  |  |
|------------------|--|
| S                | number of groups                                       |
| kg_min           | minimum value for number of group specific factors     |
| kg_max           | minimum value for number of group specific factors     |
| limit_est_groups | maximum allowed number of groups that can be estimated |

### Value

data.frame where each row contains a possible combination of group specific factors for each of the groups

---

lambda\_group\_true\_dgp3

*lambda\_group\_true\_dgp3 contains the values of the loadings to the group factors on which Y\_dgp3 is based*

---

### Description

lambda\_group\_true\_dgp3 contains the values of the loadings to the group factors on which Y\_dgp3 is based

### Usage

```
lambda_group_true_dgp3
```

### Format

dataframe with 300 rows. The first 3 columns are the loadings to the factors. The 4th column contains group membership. The fifth column contains an id of the individuals.

---

|       |                              |
|-------|------------------------------|
| LMROB | <i>Wrapper around lmrob.</i> |
|-------|------------------------------|

---

**Description**

Designed to make sure the following error does not happen anymore: Error in if (init\$scale == 0) : missing value where TRUE/FALSE needed. KS2014 is the recommended setting (use "nosetting = FALSE").

**Usage**

```
LMROB(parameter_y, parameter_x, nointercept = FALSE, nosetting = FALSE)
```

**Arguments**

|             |  |
|-------------|--|
| parameter_y | dependent variable in regression   |
| parameter_x | independent variables in regression  |
| nointercept | if TRUE it performs regression without an intercept  |
| nosetting   | option to remove the recommended setting in lmrob(). It is much faster. Defaults to FALSE. |

**Value**

An object of class lmrob. If something went wrong it returns an object of class error.

---

|                      |  |
|----------------------|--|
| make_df_pic_parallel | <i>Makes a dataframe with the PIC for each configuration and each candidate C.</i> |
|----------------------|--|

---

**Description**

Makes a dataframe with the PIC for each configuration and each candidate C.

**Usage**

```
make_df_pic_parallel(x, C_candidates)
```

**Arguments**

|              |   |
|--------------|---|
| x            | output of the parallel version of the algorithm |
| C_candidates | candidates for C                                |

**Value**

data.frame

---

```
make_df_results_parallel
```

*Makes a dataframe with information on each configuration.*

---

### Description

Makes a dataframe with information on each configuration.

### Usage

```
make_df_results_parallel(x, limit_est_groups = 20)
```

### Arguments

`x` output of the parallel version of the algorithm  
`limit_est_groups` maximum allowed number of groups that can be estimated

### Value

data.frame

---

```
make_subsamples
```

*Selects a subsample of the time series, and of the length of the time series. Based on this it returns a list with a subsample of Y, the corresponding subsample of X and of the true group membership and factorstructures if applicable.*

---

### Description

Selects a subsample of the time series, and of the length of the time series. Based on this it returns a list with a subsample of Y, the corresponding subsample of X and of the true group membership and factorstructures if applicable.

### Usage

```
make_subsamples(original_data, subset, verbose = TRUE)
```

### Arguments

`original_data` list containing the true data: Y, X, g\_true, beta\_true, factor\_group\_true, lambda\_group\_true, comfactor\_true, lambda\_true  
`subset` index of the subsample: this defines how many times stepsize\_N is subtracted from the original N time series. Similar for stepsize\_T.  
`verbose` when TRUE, it prints messages

**Value**

Y, X, g\_true, comfactor\_true, lambda\_true, factor\_group\_true, lambda\_group\_true, sampleN, sampleT The output is a list where the first element is a subset of the panel dataset. The second element contains a subsetted 3D-array with the p observed variables. The third element contains the subsetted true group membership. The fourth and fifth elements contain the subsetted true common factor(s) and its loadings respectively. The sixth element contains a list with the subsetted true group specific factors. The seventh element contains a dataframe where each row contains the group specific factor loadings that corresponds to the group specific factors. The eighth and ninth element contain the indices of N and T respectively, which were used to create the subsets.

**Examples**

```
set.seed(1)
original_data <- create_data_dgp2(30, 10)
make_subsamples(original_data, 1)
```

---

matrixnorm

*Function to calculate the norm of a matrix.*


---

**Description**

Function to calculate the norm of a matrix.

**Usage**

```
matrixnorm(mat)
```

**Arguments**

mat                   input matrix

**Value**

numeric

---

OF\_vectorized3

*Calculates objective function for the classical algorithm: used in iterate() and in local\_search.*


---

**Description**

Calculates objective function for the classical algorithm: used in iterate() and in local\_search.

**Usage**

```
OF_vectorized3(
  NN,
  TT,
  g,
  grid,
  Y,
  beta_est,
  lc,
  fc,
  lg,
  fg,
  S,
  k,
  kg,
  method_estimate_beta,
  num_factors_may_vary = TRUE
)
```

**Arguments**

|                      |  |
|----------------------|--|
| NN                   | number of time series  |
| TT                   | length of time series  |
| g                    | Vector with group membership for all individuals   |
| grid                 | dataframe containing the matrix multiplications $XB$ , $FgLg$ and $FL$   |
| Y                    | Y: $N \times T$ dataframe with the panel data of interest  |
| beta_est             | estimated values of beta   |
| lc                   | loadings of estimated common factors   |
| fc                   | estimated common factors   |
| lg                   | estimated grouploadings  |
| fg                   | estimated groupfactors   |
| S                    | number of estimated groups   |
| k                    | number of common factors to be estimated   |
| kg                   | number of group specific factors to be estimated   |
| method_estimate_beta | defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual". |
| num_factors_may_vary | whether or not the number of groupfactors is constant over all groups or not   |

**Value**

numeric value of the objective function

---

OF\_vectorized\_helpfunction3  
*Helpfunction in OF\_vectorized3()*

---

**Description**

Helpfunction in OF\_vectorized3()

**Usage**

```
OF_vectorized_helpfunction3(  
  i,  
  t,  
  XBETA,  
  LF,  
  group_memberships,  
  lgfg_list,  
  Y,  
  kg  
)
```

**Arguments**

|                   |   |
|-------------------|---|
| i                 | index of individual   |
| t                 | index of time   |
| XBETA             | matrixproduct of X and beta_est   |
| LF                | matrixproduct of common factors and its loadings                                  |
| group_memberships | Vector with group membership for all individuals                                  |
| lgfg_list         | product of groupfactors and their loadings; list with length the number of groups |
| Y                 | Y: NxT dataframe with the panel data of interest                                  |
| kg                | vector containing the number of group factors to be estimated for all groups      |

**Value**

numeric: contains the contribution to the objective function of one timepoint for one time series

---

|                    |   |
|--------------------|---|
| parallel_algorithm | <i>Wrapper of the loop over the subsets which in turn use the parallelised algorithm.</i> |
|--------------------|---|

---

### Description

Wrapper of the loop over the subsets which in turn use the parallelised algorithm.

### Usage

```
parallel_algorithm(
  original_data,
  indices_subset,
  S_cand,
  k_cand,
  kg_cand,
  C_candidates,
  robust = TRUE,
  USE_DO = FALSE,
  choice_pic = "pic2022",
  maxit = 30
)
```

### Arguments

|                |  |
|----------------|--|
| original_data  | list containing the original data (1: Y, 2: X)   |
| indices_subset | vector with indices of the subsets; starts with zero   |
| S_cand         | candidates for S (number of groups)  |
| k_cand         | candidates for k (number of common factors)  |
| kg_cand        | candidates for kg (number of group specific factors)   |
| C_candidates   | candidates for C   |
| robust         | robust or classical estimation   |
| USE_DO         | (for testing purposes) if TRUE, then a serialized version is performed ("do" instead of "dopar")   |
| choice_pic     | indicates which PIC to use to estimate the number of groups and factors. Options are "pic2017" (PIC of Ando and Bai (2017); works better for large N), "pic2016" (Ando and Bai (2016); works better for large T) weighs the fourth term with an extra factor relative to the size of the groups, and "pic2022" which shrinks the NT-space where the number of groups and factors would be over- or underestimated compared to pic2016 and pic2017. This is the default. This parameter can also be a vector with multiple pic's. |
| maxit          | maximum limit for the number of iterations for each configuration; defaults to 30  |



**Value**

Returns a list with three elements.

1. Data.frame with the optimal number of common factors for each candidate C in the rows. Each column contains the results of one subset of the input data (the first row corresponds to the full dataset).
2. Data.frame with the optimal number of groups and group specific factors for each candidate C in the rows. The structure is the same as in the above. Each entry is of the form "1\_2\_3\_NA". This is to be interpreted as 3 groups (three non NA values) where group 1 contains 1 group specific factor, group 2 contains 2 and group 3 contains 3.
3. Data.frame with information about each configuration in the rows.

**Examples**

```
#Using a small dataset as an example; this will generate several warnings due to its small size.
#Note that this example is run sequentially instead of parallel,
# and consequently will print some intermediate information in the console.
#This example uses the classical algorithm instead of the robust algorithm
# to limit its running time.
set.seed(1)
original_data <- create_data_dgp2(30, 10)
#define the number of subsets used to estimate the optimal number of groups and factors
indices_subset <- define_number_subsets(2)
#define the candidate values for C (this is a parameter in the information criterium
# used to estimate the optimal number of groups and factors)
C_candidates <- define_C_candidates()

S_cand <- 3:3 # vector with candidate number of groups
k_cand <- 0:0 # vector with candidate number of common factors
kg_cand <- 1:2 # vector with candidate number of group specific factors

#excluding parallel part from this example
#c1 <- makeCluster(detectCores() - 1)
#registerDoSNOW(c1)
output <- parallel_algorithm(original_data, indices_subset, S_cand, k_cand, kg_cand,
  C_candidates, robust = FALSE, USE_DO = TRUE, maxit = 3)
#stopCluster(c1)
```

---

|                |  |
|----------------|--|
| plot_VCsquared | <i>Plots expression(<math>VC^2</math>) along with the corresponding number of groups (orange), common factors (darkblue) and group factors of the first group (lightblue).</i> |
|----------------|--|

---

**Description**

Plots expression( $VC^2$ ) along with the corresponding number of groups (orange), common factors (darkblue) and group factors of the first group (lightblue).

**Usage**

```
plot_VCsquared(
  VC_squared,
  rc,
  rcj,
  C_candidates,
  S_cand,
  xlim_min = 0.001,
  xlim_max = 100,
  add_true_lines = FALSE,
  verbose = FALSE
)
```

**Arguments**

|                |   |
|----------------|---|
| VC_squared     | measure of variability in the optimal configuration between the subsets   |
| rc             | dataframe containing the number of common factors for all candidate C's and all subsamples  |
| rcj            | dataframe containing the number of groupfactors for all candidate C's and all subsamples  |
| C_candidates   | candidates for C (parameter in PIC)   |
| S_cand         | candidate numbers for the number of groups  |
| xlim_min       | starting point of the plot  |
| xlim_max       | end point of the plot   |
| add_true_lines | if set to TRUE, for each C the true number of groups, common factors, and group specific factors of group 1 will be added to the plot |
| verbose        | if TRUE, more details are printed   |

**Value**

A ggplot object.

**Examples**

```
set.seed(1)
#requires filled in dataframes rc and rcj
all_best_values <- calculate_best_config(add_configuration(initialise_df_results(TRUE),
  3, 0, c(3, 3, 3, rep(NA, 17))),
  data.frame(t(1:20)), 1:20)
rc <- fill_rc(initialise_rc(0:1, 1:20), all_best_values, 0)
rc <- fill_rc(rc, all_best_values, 1)
rcj <- fill_rcj(initialise_rcj(0:1, 1:20) , all_best_values, 0, 2:4, 2:4)
rcj <- fill_rcj(rcj, all_best_values, 1, 2:4, 2:4)
plot_VCsquared(c(runif(9), 0, 0, runif(9)), rc, rcj, 1:20, 2:4)
```

---

prepare\_for\_robtpca      *Helpfunction: prepares object to perform robust PCA on.*

---

### Description

It contains options to use the classical or robust covmatrix or no covariance matrix at all.

### Usage

```
prepare_for_robtpca(object, NN, TT, option = 3)
```

### Arguments

|        |   |
|--------|---|
| object | this is the object of which we may take the covariance matrix and then to perform robust PCA on |
| NN     | N   |
| TT     | T   |
| option | 1 (robust covmatrix), 2 (classical covmatrix), 3 (no covmatrix)                                 |

### Value

matrix

---

RCTS

*RCTS*

---

### Description

This package is about clustering time series in a robust manner. The method of Ando & Bai (Clustering Huge Number of Financial Time Series: A Panel Data Approach With High-Dimensional Predictors and Factor Structures) is extended to make it robust against contamination, a common issue with real world data. In this package the core functions for the robust approach are included. It also contains a simulated dataset (dataset\_Y\_dgp3).

---

reassign\_if\_empty\_groups

*Randomly reassign individual(s) if there are empty groups. This can happen if the total number of time series is low compared to the number of desired groups.*

---

### Description

Randomly reassign individual(s) if there are empty groups. This can happen if the total number of time series is low compared to the number of desired groups.

### Usage

```
reassign_if_empty_groups(g, S_true, NN)
```

### Arguments

|        |  |
|--------|--|
| g      | Vector with group membership for all individuals |
| S_true | true number of groups                            |
| NN     | number of time series                            |

### Value

numeric vector with the estimated group membership for all time series

---

restructure\_X\_to\_order\_slowN\_fastT

*Restructures X (which is an 3D-array of dimensions (N,T,p) to a 2D-matrix of dimension (NxT,p).*

---

### Description

Restructures X (which is an 3D-array of dimensions (N,T,p) to a 2D-matrix of dimension (NxT,p).

### Usage

```
restructure_X_to_order_slowN_fastT(X, vars_est)
```

### Arguments

|          |   |
|----------|---|
| X        | input   |
| vars_est | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables. |

**Value**

The function returns a 2D-array, unless the input  $X$  is NA, in which case the output will be NA as well.

---

```
return_robust_lambdaobject
      Calculates robust loadings
```

---

**Description**

Uses the almost classical lambda (this is an object of which the mean equals to the classical lambda) to create a robust lambda by using M estimation

**Usage**

```
return_robust_lambdaobject(
  Y_like_object,
  group,
  type,
  g,
  NN,
  k,
  kg,
  comfactor_rrn,
  factor_group_rrn,
  verbose = FALSE
)
```

**Arguments**

|                  |   |
|------------------|---|
| Y_like_object    | this is Y_ster or W or W_j                                |
| group            | index of group  |
| type             | scalar which shows in which setting this function is used |
| g                | vector with group memberships                             |
| NN               | number of time series                                     |
| k                | number of common factors                                  |
| kg               | number of group factors                                   |
| comfactor_rrn    | estimated common factors                                  |
| factor_group_rrn | estimated group specific factors                          |
| verbose          | when TRUE, it prints messages                             |

**Value**

Nxk dataframe

---

|           |   |
|-----------|---|
| robustpca | <i>Function that uses robust PCA and estimates robust factors and loadings.</i> |
|-----------|---|

---

### Description

Contains call to MacroPCA()

### Usage

```
robustpca(object, number_eigenvectors, KMAX = 20, verbose_robustpca = FALSE)
```

### Arguments

|                     |  |
|---------------------|--|
| object              | input  |
| number_eigenvectors | number of eigenvectors to extract  |
| KMAX                | The maximal number of principal components to compute. This is a parameter in cellWise::MacroPCA() |
| verbose_robustpca   | when TRUE, it prints messages: used for testing (requires Matrix-package when set to TRUE)         |

### Details

Notes:

Different values for kmax give different factors, but the product *lambdafactor stays constant*. Note that this number needs to be big enough, otherwise *eigen()* will be used. Variation in *k* does give different results for *lambdafactor*

MacroPCA() crashes with specific values of dim(object). For example when dim(object) = c(193,27). This is solved with *evade\_crashes\_macropca()*, for those problematic dimensions that are already encountered during tests.

### Value

list with as the first element the robust factors and as the second element the robust factor loadings

---

|            |  |
|------------|--|
| run_config | <i>Wrapper around the non-parallel algorithm, to estimate beta, group membership and the factorstructures.</i> |
|------------|--|

---

**Description**

The function estimates beta, group membership and the common and group specific factorstructures for one configuration.

**Usage**

```
run_config(robust, config, C_candidates, Y, X, choice_pic, maxit = 30)
```

**Arguments**

|              |   |
|--------------|---|
| robust       | TRUE or FALSE: defines using the classical or robust algorithm to estimate beta   |
| config       | contains one configuration of groups and factors  |
| C_candidates | candidates for C (parameter in PIC)   |
| Y            | Y: NxT dataframe with the panel data of interest  |
| X            | X: NxTxp array containing the observable variables  |
| choice_pic   | indicates which PIC to use to estimate the number of groups and factors: options are "pic2017" (uses the PIC of Ando and Bai (2017); works better for large N), "pic2016" (Ando and Bai (2016); works better for large T) weighs the fourth term with an extra factor relative to the size of the groups, and "pic2022" which shrinks the NT-space where the number of groups and factors would be over- or underestimated compared to pic2016 and pic2017. |
| maxit        | maximum limit for the number of iterations  |

**Value**

list with the estimators and metrics for this configuration

---

|           |                      |
|-----------|----------------------|
| scaling_X | <i>Scaling of X.</i> |
|-----------|----------------------|

---

**Description**

Scaling of X.

**Usage**

```
scaling_X(X, firsttime, robust, vars)
```

**Arguments**

|           |  |
|-----------|--|
| X         | input  |
| firsttime | Scaling before generating Y and before adding outliers: this is always with mean and sd. If this is FALSE, it indicates that we are using the function for a second time, after adding the outliers. In the robust case it uses median and MAD, otherwise again mean and sd. |
| robust    | logical, scaling with robust metrics instead of with non-robust measures   |
| vars      | number of observable variables   |

**Value**

3D-array with the same dimensions as X

---

|         |  |
|---------|--|
| solveFG | <i>Helpfunction in update_g(), to calculate solve(FG x t(FG)) x FG</i> |
|---------|--|

---

**Description**

Helpfunction in update\_g(), to calculate solve(FG x t(FG)) x FG

**Usage**

```
solveFG(TT, S, kg, factor_group, testing = FALSE)
```

**Arguments**

|              |   |
|--------------|---|
| TT           | length of time series   |
| S            | number of groups  |
| kg           | vector with the estimated number of group specific factors for each group   |
| factor_group | estimated group specific factors  |
| testing      | variable that determines if we are in 'testing phase'; defaults to FALSE (requires Matrix-package if set to TRUE) |

**Value**

list: the number of elements in this list is equal to S (the number of groups). Each of the elements in this list has a number rows equal to the number of group specific factors, and TT columns.



---

tabulate\_potential\_C *Shows the configurations for potential C's of the first stable interval (beginpoint, midpoint and endpoint)*

---

### Description

Shows the configurations for potential C's of the first stable interval (beginpoint, midpoint and endpoint)

### Usage

```
tabulate_potential_C(
  df,
  runs,
  beginpoint,
  midpoint_log,
  midpoint,
  endpoint,
  S_cand
)
```

### Arguments

|              |   |
|--------------|---|
| df           | input dataframe   |
| runs         | number of panel data sets for which the algorithm has run. If larger than one, the median VC2 is used to determine C. |
| beginpoint   | first C of the chosen stable interval   |
| midpoint_log | middle C (on a logscale) of the chosen stable interval  |
| midpoint     | middle C of the chosen stable interval  |
| endpoint     | last C of the chosen stable interval  |
| S_cand       | candidate number for the number of groups   |

### Value

data.frame

---

update\_g                      *Function that estimates group membership.*

---

### Description

Function that estimates group membership.

### Usage

```
update_g(
  Y,
  X,
  beta_est,
  g,
  factor_group,
  lambda,
  comfactor,
  S,
  k,
  kg,
  vars_est,
  robust,
  method_estimate_factors,
  method_estimate_beta,
  verbose = FALSE
)
```

### Arguments

|                         |   |
|-------------------------|---|
| Y                       | Y: NxT dataframe with the panel data of interest  |
| X                       | X: NxTxp array containing the observable variables  |
| beta_est                | estimated values of beta  |
| g                       | Vector with estimated group membership for all individuals  |
| factor_group            | estimated group specific factors  |
| lambda                  | loadings of the estimated common factors  |
| comfactor               | estimated common factors  |
| S                       | number of estimated groups  |
| k                       | number of common factors to be estimated  |
| kg                      | number of group specific factors to be estimated  |
| vars_est                | number of variables that will be included in the algorithm and have their coefficient estimated. This is usually equal to the number of observable variables. |
| robust                  | robust or classical estimation of group membership  |
| method_estimate_factors | defines method of robust estimation of the factors: "macro", "pertmm" or "cz"   |

`method_estimate_beta` defines how beta is estimated. Default case is an estimated beta for each individual. Default value is "individual." Possible values are "homogeneous", "group" or "individual".

`verbose` when TRUE, it prints messages

### Value

Returns a list. The first element contains a vector with the estimated group membership for all time series. The second element contains the values which were used to determine the group membership. The third element is only relevant if `method_estimate_factors` is set to "cz" (non-default) and contains the group membership before moving some of the time series to class zero.

### Examples

```
X <- X_dgp3
Y <- Y_dgp3
# Set estimations for group factors and its loadings, and group membership to the true value
lambda_group <- lambda_group_true_dgp3
factor_group <- factor_group_true_dgp3
g_true <- g_true_dgp3 # true values of group membership
g <- g_true # estimated values of group membership; set in this example to be equal to true values
# There are no common factors to be estimated -> use placeholder with values set to zero
lambda <- matrix(0, nrow = 1, ncol = 300)
comfactor <- matrix(0, nrow = 1, ncol = 30)
# Choose how coefficients of the observable are estimated
beta_est <- estimate_beta(
  Y, X, NA, g, lambda_group, factor_group,
  lambda, comfactor,
  S = 3, k = 0, kg = c(3, 3, 3),
  vars_est = 3, robust = TRUE
)[[1]]
g_new <- update_g(
  Y, X, beta_est, g,
  factor_group, lambda, comfactor,
  S = 3,
  k = 0,
  kg = c(3, 3, 3),
  vars_est = 3,
  robust = TRUE,
  "macro", "individual"
)[[1]]
```

---

X\_dgp3

*The dataset X\_dgp3 contains the values of the 3 observable variables on which Y\_dgp3 is based.*

---

**Description**

The dataset X\_dgp3 contains the values of the 3 observable variables on which Y\_dgp3 is based.

**Usage**

```
X_dgp3
```

**Format**

array with 300 x 30 x 3 elements

**Examples**

```
head(X_dgp3[, , 1])
hist(X_dgp3[, , 1])
```

---

Y\_dgp3

*Y\_dgp3 contains a simulated dataset for DGP 3.*

---

**Description**

$Y = XB + LgFg$ . It has 3 groups and each group has 3 groupfactors. At last there were 3 observable variables generated into it.

**Usage**

```
Y_dgp3
```

**Format**

300 x 30 matrix. Each row is one time series.

**Examples**

```
plot(Y_dgp3[, 1:2], col = g_true_dgp3, xlab = "First column of Y", ylab = "Second column of Y",
     main = "Plot of the first two columns of the dataset Y. \nColors are the true groups.")
```

# Index

## \* datasets

- df\_results\_example, 39
  - factor\_group\_true\_dgp3, 48
  - g\_true\_dgp3, 55
  - lambda\_group\_true\_dgp3, 66
  - X\_dgp3, 83
  - Y\_dgp3, 84
- adapt\_pic\_with\_sigma2maxmodel, 4
- adapt\_X\_estimating\_less\_variables, 5
- add\_configuration, 5
- add\_metrics, 6
- add\_pic, 7
- add\_pic\_parallel, 8
- beta\_true\_heterogroups, 10
- calculate\_best\_config, 10
- calculate\_error\_term, 12
- calculate\_errors\_virtual\_groups, 11
- calculate\_FL\_group\_estimated, 14
- calculate\_FL\_group\_true, 15
- calculate\_lambda, 16
- calculate\_lambda\_group, 17
- calculate\_lgfg, 18
- calculate\_obj\_for\_g, 19
- calculate\_PIC, 20
- calculate\_PIC\_term1, 21
- calculate\_sigma2, 22
- calculate\_sigma2maxmodel, 22
- calculate\_TN\_factor, 23
- calculate\_VCsquared, 24
- calculate\_virtual\_factor\_and\_lambda\_group, 25
- calculate\_W, 26
- calculate\_XB\_estimated, 27
- calculate\_XB\_true, 27
- calculate\_Z\_common, 28
- calculate\_Z\_group, 29
- check\_stopping\_rules, 30
- clustering\_with\_robust\_distances, 31
- create\_covMat\_crosssectional\_dependence, 31
- create\_data\_dgp2, 32
- create\_true\_beta, 33
- define\_C\_candidates, 34
- define\_configurations, 34
- define\_kg\_candidates, 35
- define\_number\_subsets, 35
- define\_object\_for\_initial\_clustering\_macropca, 36
- define\_rho\_parameters, 37
- determine\_beta, 37
- determine\_robust\_lambda, 39
- df\_results\_example, 39
- do\_we\_estimate\_common\_factors, 40
- do\_we\_estimate\_group\_factors, 41
- estimate\_algorithm, 41
- estimate\_beta, 42
- estimate\_factor, 44
- estimate\_factor\_group, 45
- evade\_crashes\_macropca, 47
- evade\_floating\_point\_errors, 47
- factor\_group\_true\_dgp3, 48
- fill\_rc, 48
- fill\_rcj, 49
- final\_estimations\_filter\_kg, 49
- g\_true\_dgp3, 55
- generate\_grouped\_factorstructure, 50
- generate\_Y, 51
- get\_best\_configuration, 52
- get\_convergence\_speed, 53
- get\_final\_estimation, 53
- grid\_add\_variables, 54
- handle\_macropca\_errors, 57
- handleNA, 56

handleNA\_LG, 56

initialise\_beta, 57  
initialise\_clustering, 59  
initialise\_commonfactorstructure\_macropca,  
60  
initialise\_df\_pic, 61  
initialise\_df\_results, 62  
initialise\_rc, 62  
initialise\_rcj, 63  
initialise\_X, 63  
iterate, 64

kg\_candidates\_expand, 66

lambda\_group\_true\_dgp3, 66  
LMROB, 67

make\_df\_pic\_parallel, 67  
make\_df\_results\_parallel, 68  
make\_subsamples, 68  
matrixnorm, 69

OF\_vectorized3, 69  
OF\_vectorized\_helpfunction3, 71

parallel\_algorithm, 72  
plot\_VCsquared, 73  
prepare\_for\_robtpca, 75

RCTS, 75  
reassign\_if\_empty\_groups, 76  
restructure\_X\_to\_order\_slowN\_fastT, 76  
return\_robust\_lambdaobject, 77  
robustpca, 78  
run\_config, 79

scaling\_X, 79  
solveFG, 80

tabulate\_potential\_C, 81

update\_g, 82

X\_dgp3, 83

Y\_dgp3, 84