Package: intrinsicFRP (via r-universe)

October 27, 2026

Title An R Package for Factor Model Asset Pricing

Version 2.1.0

Date 2024-04-15

Maintainer Alberto Quaini <alberto91quaini@gmail.com>

Description Functions for evaluating and testing asset pricing models, including estimation and testing of factor risk premia, selection of ``strong'' risk factors (factors having nonzero population correlation with test asset returns), heteroskedasticity and autocorrelation robust covariance matrix estimation and testing for model misspecification and identification. The functions for estimating and testing factor risk premia implement the Fama-MachBeth (1973) <doi:10.1086/260061> two-pass approach, the misspecification-robust approaches of Kan-Robotti-Shanken (2013) <doi:10.1111/jofi.12035>, and the approaches based on tradable factor risk premia of Quaini-Trojani-Yuan (2023) <doi:10.2139/ssrn.4574683>. The functions for selecting the ``strong'' risk factors are based on the Oracle estimator of Quaini-Trojani-Yuan (2023) <doi:10.2139/ssrn.4574683> and the factor screening procedure of Gospodinov-Kan-Robotti (2014) <doi:10.2139/ssrn.2579821>. The functions for evaluating model misspecification implement the HJ model misspecification distance of Kan-Robotti (2008) <doi:10.1016/j.jempfin.2008.03.003>, which is a modification of the prominent Hansen-Jagannathan (1997) <doi:10.1111/j.1540-6261.1997.tb04813.x> distance. The functions for testing model identification specialize the Kleibergen-Paap (2006) <doi:10.1016/j.jeconom.2005.02.011> and the Chen-Fang (2019) <doi:10.1111/j.1540-6261.1997.tb04813.x> rank test to the regression coefficient matrix of test asset returns on risk factors. Finally, the function for heteroskedasticity and autocorrelation robust covariance estimation implements the Newey-West (1994) <doi:10.2307/2297912> covariance estimator.

License GPL (>= 3)

URL https://github.com/a91quaini/intrinsicFRP

BugReports https://github.com/a91quaini/intrinsicFRP/issues

Encoding UTF-8

RoxygenNote 7.3.1

LinkingTo Rcpp, RcppArmadillo

Imports glmnet, graphics, Rcpp

Depends R (>= 4.3.0)

LazyData true

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Alberto Quaini [aut, cre, cph]
(<https://orcid.org/0000-0002-1251-0599>)

Repository https://cran.r-universe.dev

Date/Publication 2024-04-16 02:33:13 UTC

RemoteUrl https://github.com/cran/intrinsicFRP

RemoteRef HEAD

RemoteSha 8ccb85100e9f9073f068221032c2fdc45b2508a6

Contents

ChenFang2019BetaRankTest . . . 3
factors . . . 4
FGXFactorsTest . . . 4
FRP . . . 6
GKRFactorScreening . . . 7
HACcovariance . . . 9
HJMisspecificationDistance . . . 10
IterativeKleibergenPaap2006BetaRankTest . . . 11
OracleTFRP . . . 12
returns . . . 15
risk_free . . . 15
SDFCoefficients . . . 16
TFRP . . . 17

Index **19**

ChenFang2019BetaRankTest

Asset Pricing Model Identification via Chen-Fang (2019) Beta Rank Test

Description

Tests the null hypothesis of reduced rank in the matrix of regression loadings for test asset excess returns on risk factors using the Chen-Fang (2019) doi:10.3982/QE1139 beta rank test. The test applies the Kleibergen-Paap (2006) doi:10.1016/j.jeconom.2005.02.011 iterative rank test for initial rank estimation when `target_level_kp2006_rank_test > 0`, with an adjustment to `level = target_level_kp2006_rank_test / n_factors`. When `target_level_kp2006_rank_test <= 0`, the number of singular values above `n_observations^(-1/4)` is used instead. It presumes that the number of factors is less than the number of returns (`n_factors < n_returns`). All the details can be found in Chen-Fang (2019) doi:10.3982/QE1139.

Usage

```
ChenFang2019BetaRankTest(
  returns,
  factors,
  n_bootstrap = 500,
  target_level_kp2006_rank_test = 0.05,
  check_arguments = TRUE
)
```

Arguments

`returns`	Matrix of test asset excess returns with dimensions `n_observations x n_returns`.
`factors`	Matrix of risk factors with dimensions `n_observations x n_factors`.
`n_bootstrap`	The number of bootstrap samples to use in the Chen-Fang (2019) test. Defaults to 500 if not specified.
`target_level_kp2006_rank_test`	The significance level for the Kleibergen-Paap (2006) rank test used for initial rank estimation. If set above 0, it indicates the level for this estimation within the Chen-Fang (2019) rank test. If set at 0 or negative, the initial rank estimator defaults to the count of singular values exceeding `n_observations^(-1/4)`. The default value is `0.05` to account for multiple testing.
`check_arguments`	Logical flag to determine if input arguments should be checked for validity. Default is `TRUE`.

Value

A list containing the Chen-Fang (2019) rank statistic and the associated p-value.

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# compute the model identification test
hj_test = ChenFang2019BetaRankTest(returns, factors)
```

`factors` *Factors - monthly observations from* `07/1963` *to* `02/2024`

Description

Monthly observations from `07/1963` to `02/2024` of the Fama-French `5` factors and the momentum factor.

Usage

```
factors
```

Format

`factors`**:**

A data frame with `624` rows and `7` columns:

Date Date in yyyymm format

... Factor observations

Source

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

---

`FGXFactorsTest` *Testing for the pricing contribution of new factors.*

Description

Computes the three-pass procedure of Feng Giglio and Xiu (2020) doi:org/10.1111/jofi.12883, which evaluates the contribution to cross-sectional pricing of any new factors on top of a set of control factors. The third step is a OLS regression of average returns on the covariances between asset returns and the new factors, as well as the control factors selected in either one of the first two steps. The first stwo steps consists in (i) a Lasso regression of average returns on the ovariances between asset returns and all control factors and (ii) a Lasso regression of the covariances between asset returns and the new factors on the ovariances between asset returns and all control

factors. The second selection aims at correcting for potential omitted variables in the first selection. Tuning of the penalty parameters in the Lasso regressions is performed via Cross Validation (CV). Standard errors are computed following Feng Giglio and Xiu (2020) using the Newey-West (1994) doi:10.2307/2297912 plug-in procedure to select the number of relevant lags, i.e., `n_lags = 4 * (n_observations/100)^(2/9)`. For the standard error computations, the function allows to internally pre-whiten the series by fitting a VAR(1), i.e., a vector autoregressive model of order 1.

Usage

```
FGXFactorsTest(
  gross_returns,
  control_factors,
  new_factors,
  n_folds = 5,
  check_arguments = TRUE
)
```

Arguments

`gross_returns`	A `n_observations x n_returns`-dimensional matrix of test asset gross returns.
`control_factors`	A `n_observations x n_control_factors`-dimensional matrix of control or benchmark factors.
`new_factors`	A `n_observations x n_new_factors`-dimensional matrix of new factors.
`n_folds`	An integer indicating the number of k-fold for cross validation. Default is `5`.
`check_arguments`	A boolean `TRUE` for internal check of all function arguments; `FALSE` otherwise. Default is `TRUE`.

Value

A list containing the `n_new_factors`-dimensional vector of SDF coefficients in `"sdf_coefficients"` and corresponding standard errors in `"standard_errors"`; it also returns the index of control factors that are selected by the two-step selection procedure.

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
control_factors = intrinsicFRP::factors[,2:4]
new_factors = intrinsicFRP::factors[,5:7]
returns = intrinsicFRP::returns[,-1]
RF = intrinsicFRP::risk_free[,-1]

gross_returns = returns + 1 + RF

output = FGXFactorsTest(
  gross_returns,
  control_factors,
```

```
  new_factors
)
```

FRP *Factor risk premia.*

Description

Computes the Fama-MachBeth (1973) doi:10.1086/260061 factor risk premia: `FMFRP = (beta' * beta)^{-1} * beta' *` where `beta = Cov[R, F] * V[F]^{-1}` or the misspecification-robust factor risk premia of Kan-Robotti-Shanken (2013) doi:10.1111/jofi.12035: `KRSFRP = (beta' * V[R]^{-1} * beta)^{-1} * beta' * V[R]^{-1}` from data on factors `F` and test asset excess returns `R`. These notions of factor risk premia are by construction the negative covariance of factors `F` with candidate SDF `M = 1 - d' * (F - E[F])`, where SDF coefficients `d` are obtained by minimizing pricing errors: `argmin_{d} (E[R] - Cov[R,F] * d)' * (E[R] - Cov[R,F] *` and `argmin_{d} (E[R] - Cov[R,F] * d)' * V[R]^{-1} * (E[R] - Cov[R,F] * d)`, respectively. Optionally computes the corresponding heteroskedasticity and autocorrelation robust standard errors (accounting for a potential model misspecification) using the Newey-West (1994) doi:10.2307/2297912 plug-in procedure to select the number of relevant lags, i.e., `n_lags = 4 * (n_observations/100)^(2/9)`. For the standard error computations, the function allows to internally pre-whiten the series by fitting a VAR(1), i.e., a vector autoregressive model of order 1. All the details can be found in Kan-Robotti-Shanken (2013) doi:10.1111/jofi.12035.

Usage

```
FRP(
  returns,
  factors,
  misspecification_robust = TRUE,
  include_standard_errors = FALSE,
  hac_prewhite = FALSE,
  target_level_gkr2014_screening = 0,
  check_arguments = TRUE
)
```

Arguments

`returns`	A `n_observations x n_returns`-dimensional matrix of test asset excess returns.
`factors`	A `n_observations x n_factors`-dimensional matrix of factors.
`misspecification_robust`	A boolean: `TRUE` for the "misspecification-robust" Kan-Robotti-Shanken (2013) GLS approach using the inverse covariance matrix of returns; `FALSE` for standard Fama-MacBeth risk premia. Default is `TRUE`.
`include_standard_errors`	A boolean: `TRUE` if you want to compute the factor risk premia HAC standard errors; `FALSE` otherwise. Default is `FALSE`.

hac_prewhite A boolean indicating if the series needs pre-whitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is `false`.

target_level_gkr2014_screening
A number indicating the target level of the tests underlying the factor screening procedure in Gospodinov-Kan-Robotti (2014). If it is zero, then no factor screening procedure is implemented. Otherwise, it implements an iterative screening procedure based on the sequential removal of factors associated with the smallest insignificant t-test of a nonzero SDF coefficient. The threshold for the absolute t-test is `target_level_gkr2014_screening / n_factors`, where n_factors indicate the number of factors in the model at the current iteration. Default is `0.`, i.e., no factor screening.

check_arguments
A boolean: `TRUE` for internal check of all function arguments; `FALSE` otherwise. Default is `TRUE`.

Value

A list containing `n_factors`-dimensional vector of factor risk premia in `"risk_premia"`; if `include_standard_errors = TRUE`, then it further includes `n_factors`-dimensional vector of factor risk premia standard errors in `"standard_errors"`; if `target_level_gkr2014_screening >= 0`, it further includes the indices of the selected factors in `selected_factor_indices`.

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# compute KRS factor risk premia and their standard errors
frp = FRP(returns, factors, include_standard_errors = TRUE)
```

`GKRFactorScreening` *Factor screening procedure of Gospodinov-Kan-Robotti (2014)*

Description

Performs the factor screening procedure of Gospodinov-Kan-Robotti (2014) doi:10.2139/ssrn.2579821, which is an iterative model screening procedure based on the sequential removal of factors associated with the smallest insignificant t-test of a nonzero misspecification-robust SDF coefficient. The significance threshold for the absolute t-test is set to `target_level / n_factors`, where n_factors indicates the number of factors in the model at the current iteration; that is, it takes care of the multiple testing problem via a conservative Bonferroni correction. Standard errors are computed with the heteroskedasticity and autocorrelation using the Newey-West (1994) doi:10.2307/2297912 estimator, where the number of lags is selected using the Newey-West plug-in procedure: `n_lags = 4 * (n_observations/100)^(2/9)`. For the standard error computations, the function allows to internally pre-whiten the series by fitting a VAR(1), i.e., a vector

autoregressive model of order 1. All the details can be found in Gospodinov-Kan-Robotti (2014) doi:10.2139/ssrn.2579821.

Usage

```
GKRFactorScreening(
  returns,
  factors,
  target_level = 0.05,
  hac_prewhite = FALSE,
  check_arguments = TRUE
)
```

Arguments

returns	n_observations x n_returns-dimensional matrix of test asset excess returns.
factors	n_observations x n_factors-dimensional matrix of risk factors.
target_level	Number specifying the target significance threshold for the tests underlying the GKR factor screening procedure. To account for the multiple testing problem, the significance threshold for the absolute t-test is given by target_level_gkr2014_screening / n_factors, where n_factors indicate the number of factors in the model at the current iteration. Default is 0.05.
hac_prewhite	A boolean indicating if the series needs prewhitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is false.
check_arguments	boolean TRUE for internal check of all function arguments; FALSE otherwise. Default is TRUE.

Value

A list contaning the selected GKR SDF coefficients in SDF_coefficients, their standard errors in standard_errors, t-statistics in t_statistics and indices in the columns of the factor matrix factors supplied by the user in selected_factor_indices.

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# Perform the GKR factor screening procedure
screen = GKRFactorScreening(returns, factors)
```

HACcovariance *Heteroskedasticity and Autocorrelation robust covariance estimator*

Description

This function estimates the long-run covariance matrix of a multivariate centred time series accounting for heteroskedasticity and autocorrelation using the Newey-West (1994) doi:10.2307/2297912 estimator. The number is selected using the Newey-West plug-in procedure, where `n_lags = 4 * (n_observations/100)^(2/9)`. The function allows to internally prewhiten the series by fitting a VAR(1). All the details can be found in Newey-West (1994) doi:10.2307/2297912.

Usage

```
HACcovariance(series, prewhite = FALSE, check_arguments = TRUE)
```

Arguments

`series`	A matrix (or vector) of data where each column is a time series.
`prewhite`	A boolean indicating if the series needs prewhitening by fitting an AR(1). Default is `FALSE`
`check_arguments`	A boolean `TRUE` for internal check of all function arguments; `FALSE` otherwise. Default is `TRUE`.

Value

A symmetric matrix (or a scalar if only one column series is provided) representing the estimated HAC covariance.

Examples

```
# Import package data on 6 risk factors and 42 test asset excess returns
returns = intrinsicFRP::returns[,-1]
factors = intrinsicFRP::factors[,-1]

# Fit a linear model of returns on factors
fit = stats::lm(returns ~ factors)

# Extract residuals from the model
residuals = stats::residuals(fit)

# Compute the HAC covariance of the residuals
hac_covariance = HACcovariance(residuals)

# Compute the HAC covariance of the residuals imposing prewhitening
hac_covariance_pw = HACcovariance(residuals, prewhite = TRUE)
```

`HJMisspecificationDistance` *Compute the HJ asset pricing model misspecification distance.*

Description

Computes the Kan-Robotti (2008) <10.1016/j.jempfin.2008.03.003> squared model misspecification distance: `square_distance = min_{d} (E[R] - Cov[R,F] * d)' * V[R]^{-1} * (E[R] - Cov[R,F] * d)`, where `R` denotes test asset excess returns and `F` risk factors, and computes the associated confidence interval. This model misspecification distance is a modification of the prominent Hansen-Jagannathan (1997) doi:10.1111/j.1540-6261.1997.tb04813.x distance, adapted to the use of excess returns for the test asset, and a SDF that is a linear function of demeaned factors. Clearly, computation of the confidence interval is obtained by means of an asymptotic analysis under potentially misspecified models, i.e., without assuming correct model specification. Details can be found in Kan-Robotti (2008) <10.1016/j.jempfin.2008.03.003>.

Usage

```
HJMisspecificationDistance(
  returns,
  factors,
  ci_coverage = 0.95,
  hac_prewhite = FALSE,
  check_arguments = TRUE
)
```

Arguments

`returns`	A `n_observations x n_returns` matrix of test asset excess returns.
`factors`	A `n_observations x n_factors` matrix of risk factors.
`ci_coverage`	A number indicating the confidence interval coverage probability. Default is `0.95`.
`hac_prewhite`	A boolean indicating if the series needs pre-whitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is `false`.
`check_arguments`	A boolean: `TRUE` for internal check of all function arguments; `FALSE` otherwise. Default is `TRUE`.

Value

@return A list containing the squared misspecification-robust HJ distance in `squared_distance`, and the lower and upper confidence bounds in `lower_bound` and `upper_bound`, respectively.

Examples

```
# Import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# Compute the HJ model misspecification distance
hj_test = HJMisspecificationDistance(returns, factors)
```

`IterativeKleibergenPaap2006BetaRankTest`
Asset Pricing Model Identification via Iterative Kleibergen-Paap 2006 Beta Rank Test

---

Description

Evaluates the rank of regression loadings in an asset pricing model using the iterative Kleibergen-Paap (2006) doi:10.1016/j.jeconom.2005.02.011 beta rank test. It systematically tests the null hypothesis for each potential rank `q = 0, ..., n_factors - 1` and estimates the rank as the smallest `q` that has a p-value below the significance level, adjusted for the number of factors. The function presupposes more returns than factors (`n_factors < n_returns`). All the details can be found in Kleibergen-Paap (2006) doi:10.1016/j.jeconom.2005.02.011.

Usage

```
IterativeKleibergenPaap2006BetaRankTest(
  returns,
  factors,
  target_level = 0.05,
  check_arguments = TRUE
)
```

Arguments

`returns`	A matrix of test asset excess returns with dimensions `n_observations x n_returns`.
`factors`	A matrix of risk factors with dimensions `n_observations x n_factors`.
`target_level`	A numeric value specifying the significance level for the test. For each hypothesis test `H: rank(beta) = q`, the significance level is adjusted to `target_level / n_factors`. The default is `0.05`.
`check_arguments`	Logical flag indicating whether to perform internal checks of the function's arguments. Defaults to `TRUE`.

Value

A list containing estimates of the regression loading rank and the associated iterative Kleibergen-Paap 2006 beta rank statistics and p-values for each `q`.

Examples

```
# import package data on 15 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# compute the model identification test
hj_test = ChenFang2019BetaRankTest(returns, factors)
```

`OracleTFRP` *Oracle tradable factor risk premia.*

Description

Computes Oracle tradable factor risk premia of Quaini-Trojani-Yuan (2023) doi:10.2139/ssrn.4574683 from data on K factors `F = [F_1,...,F_K]` and test asset excess returns R: `OTFRP = argmin_x ||TFRP - x||_2^2 +` where `TFRP` is the tradable factor risk premia estimator, `tau > 0` is a penalty parameter, and the Oracle weights are given by `w_k = 1 / ||corr[F_k, R]||_2^2`. This estimator is called "Oracle" in the sense that the probability that the index set of its nonzero estimated risk premia equals the index set of the true strong factors tends to 1 (Oracle selection), and that on the strong factors, the estimator reaches the optimal asymptotic Normal distribution. Here, strong factors are those that have a nonzero population marginal correlation with asset excess returns. Tuning of the penalty parameter `tau` is performed via Generalized Cross Validation (GCV), Cross Validation (CV) or Rolling Validation (RV). GCV tunes parameter `tau` by minimizing the criterium: `||PE(tau)||_2^2 / (1-df(tau)/T)^2` where `PE(tau) = E[R] - beta_{S(tau)} * OTFRP(tau)` are the pricing errors of the model for given tuning parameter `tau`, with `S(tau)` being the index set of the nonzero Oracle TFRP computed with tuning parameter `tau`, and `beta_{S(tau)} = Cov[R, F_{S(tau)}] * (Cov[F_{` the regression coefficients of the test assets excess returns on the factor mimicking portfolios, and `df(tau) = |S(tau)|` are the degrees of freedom of the model, given by the number of nonzero Oracle TFRP. CV and RV, instead, choose the value of `tau` that minimize the criterium: `PE(tau)' * V[PE(tau)]^{-1} PE(tau)` where `V[PE(tau)]` is the diagonal matrix collecting the marginal variances of pricing errors `PE(tau)`, and each of these components are aggregated over k-fold cross-validated data or over rolling windows of data, respectively. Oracle weights can be based on the correlation between factors and returns (suggested approach), on the regression coefficients of returns on factors or on the first-step tradable risk premia estimator. Optionally computes the corresponding heteroskedasticity and autocorrelation robust standard errors using the Newey-West (1994) doi:10.2307/2297912 plug-in procedure to select the number of relevant lags, i.e., `n_lags = 4 * (n_observations/100)^(2/9)`. For the standard error computations, the function allows to internally pre-whiten the series by fitting a VAR(1), i.e., a vector autoregressive model of order 1. All details are found in Quaini-Trojani-Yuan (2023) doi:10.2139/ssrn.4574683.

Usage

```
OracleTFRP(
  returns,
  factors,
  penalty_parameters,
```

```
  weighting_type = "c",
  tuning_type = "g",
  one_stddev_rule = TRUE,
  gcv_scaling_n_assets = FALSE,
  gcv_identification_check = FALSE,
  target_level_kp2006_rank_test = 0.05,
  n_folds = 5,
  n_train_observations = 120,
  n_test_observations = 12,
  roll_shift = 12,
  relaxed = FALSE,
  include_standard_errors = FALSE,
  hac_prewhite = FALSE,
  plot_score = TRUE,
  check_arguments = TRUE
)
```

Arguments

returns	A n_observations x n_returns-dimensional matrix of test asset excess returns.
factors	A n_observations x n_factors-dimensional matrix of factors.
penalty_parameters	A n_parameters-dimensional vector of penalty parameter values from smallest to largest.
weighting_type	A character specifying the type of adaptive weights: based on the correlation between factors and returns 'c'; based on the regression coefficients of returns on factors 'b'; based on the first-step tradable risk premia estimator 'a'; otherwise a vector of ones (any other character). Default is 'c'.
tuning_type	A character indicating the parameter tuning type: 'g' for generalized cross validation; 'c' for K-fold cross validation; 'r' for rolling validation. Default is 'g'.
one_stddev_rule	A boolean: TRUE for picking the most parsimonious model whose score is not higher than one standard error above the score of the best model; FALSE for picking the best model. Default is TRUE.
gcv_scaling_n_assets	(only relevant for tuning_type ='g') A boolean: TRUE for sqrt(n_assets) scaling (sqrt(n_assets) / n_observations); FALSE otherwise (1 / n_observations). Default is FALSE.
gcv_identification_check	(only relevant for tuning_type ='g') A boolean: TRUE for a loose check for model identification; FALSE otherwise. Default is FALSE.
target_level_kp2006_rank_test	(only relevant for tuning_type ='g' and if gcv_identification_check is TRUE) A number indicating the level of the Kleibergen Paap 2006 rank test. If it is strictly grater than zero, then the iterative Kleibergen Paap 2006 rank

test at level = target_level_kp2006_rank_test / n_factors (where division by the number of factors is performed as a Bonferroni correction for multiple testing) is used to compute an initial estimator of the rank of the factor loadings in the Chen Fang 2019 rank test. Otherwise, the initial rank estimator is taken to be the number of singular values above n_observations^(-1/4). Default is 0.05.

n_folds (only relevant for tuning_type ='c') An integer indicating the number of k-fold for cross validation. Default is 5.

n_train_observations
(only relevant for tuning_type ='r') The number of observations in the rolling training set. Default is 120.

n_test_observations
(only relevant for tuning_type ='r') The number of observations in the test set. Default is 12.

roll_shift (only relevant for tuning_type ='r') The number of observation shift when moving from the rolling window to the next one. Default is 12.

relaxed A boolean: TRUE if you want to compute a post-selection unpenalized tradable factor risk premia to remove the bias due to shrinkage; FALSEotherwise. Default isFALSE‘.

include_standard_errors
A boolean TRUE if you want to compute the adaptive tradable factor risk premia HAC standard errors; FALSE otherwise. Default is FALSE.

hac_prewhite A boolean indicating if the series needs prewhitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is false.

plot_score A boolean: TRUE for plotting the model score; FALSE otherwise. Default is TRUE.

check_arguments
A boolean TRUE for internal check of all function arguments; FALSE otherwise. Default is TRUE.

Value

A list containing the n_factors-dimensional vector of adaptive tradable factor risk premia in "risk_premia"; the optimal penalty parameter value in "penalty_parameter"; the model score for each penalty parameter value in "model_score"; if include_standard_errors = TRUE, then it further includes n_factors-dimensional vector of tradable factor risk premia standard errors in "standard_errors".

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

penalty_parameters = seq(0., 1., length.out = 100)

# compute optimal adaptive tradable factor risk premia and their standard errors
oracle_tfrp = OracleTFRP(
returns,
```

```
factors,
penalty_parameters,
include_standard_errors = TRUE
)
```

`returns` *Test Asset Excess Returns - monthly observations from* `07/1963` *to* `02/2024`

Description

Monthly excess returns on the 25 Size/Book-to-Market double sorted portfolios and the 17 industry portfolios from `07/1963` to `02/2024`.

Usage

```
returns
```

Format

`returns`**:**

A data frame with 624 rows and 43 columns:

Date Date in yyyymm format

... Asset excess returns

Source

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

---

`risk_free` *Risk free - monthly observations from* `07/1963` *to* `02/2024`

Description

Monthly observations from `07/1963` to `02/2024` of the US risk free asset.

Usage

```
risk_free
```

Format

`risk_free`**:**

A data frame with 624 rows and 2 columns:

Date Date in yyyymm format

... risk free observations

Source

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

---

`SDFCoefficients` *SDF Coefficients*

Description

Computes the SDF coefficients of Fama-MachBeth (1973) doi:10.1086/260061 `FMSDFcoefficients = (C' * C)^{-1} * C`
or the misspecification-robust SDF coefficients of Gospodinov-Kan-Robotti (2014) doi:10.1093/rfs/hht135: `GKRSDFcoefficients = (C' * V[R]^{-1} * C)^{-1} * C' * V[R]^{-1} * E[R]` from data on factors `F` and test asset excess returns `R`. These notions of SDF coefficients minimize pricing errors: `argmin_{d} (E[R] - Cov[R,F] * d)' * W * (E[R] - Cov[R,F] * d)`, with `W=I`, i.e., the identity, and `W=V[R]^{-1}`, respectively. Optionally computes the corresponding heteroskedasticity and autocorrelation robust standard errors (accounting for a potential model misspecification) using the Newey-West (1994) doi:10.2307/2297912 plug-in procedure to select the number of relevant lags, i.e., `n_lags = 4 * (n_observations/100)^(2/9)`.

Usage

```
SDFCoefficients(
  returns,
  factors,
  misspecification_robust = TRUE,
  include_standard_errors = FALSE,
  hac_prewhite = FALSE,
  target_level_gkr2014_screening = 0,
  check_arguments = TRUE
)
```

Arguments

`returns`	A `n_observations x n_returns`-dimensional matrix of test asset excess returns.
`factors`	A `n_observations x n_factors`-dimensional matrix of factors.
`misspecification_robust`	A boolean: `TRUE` for the "misspecification-robust" Kan-Robotti-Shanken (2013) GLS approach using the inverse covariance matrix of returns; `FALSE` for standard Fama-MacBeth risk premia. Default is `TRUE`.
`include_standard_errors`	A boolean: `TRUE` if you want to compute the SDF coefficients' HAC standard errors; `FALSE` otherwise. Default is `FALSE`.
`hac_prewhite`	A boolean indicating if the series needs pre-whitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is `false`.

target_level_gkr2014_screening
: A number indicating the target level of the tests underlying the factor screening procedure in Gospodinov-Kan-Robotti (2014). If it is zero, then no factor screening procedure is implemented. Otherwise, it implements an iterative screening procedure based on the sequential removal of factors associated with the smallest insignificant t-test of a nonzero SDF coefficient. The threshold for the absolute t-test is target_level_gkr2014_screening / n_factors, where n_factors indicate the number of factors in the model at the current iteration. Default is 0., i.e., no factor screening.

check_arguments
: A boolean: TRUE for internal check of all function arguments; FALSE otherwise. Default is TRUE.

Value

A list containing n_factors-dimensional vector of SDF coefficients in "sdf_coefficients"; if include_standard_errors = TRUE, then it further includes n_factors-dimensional vector of SDF coefficients' standard errors in "standard_errors";

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# compute GKR SDF coefficients and their standard errors
frp = SDFCoefficients(returns, factors, include_standard_errors = TRUE)
```

TFRP *Tradable factor risk premia.*

Description

Computes tradable factor risk premia from data on factors F and test asset excess returns R: TFRP = Cov[F, R] * Var[R]^{-1} * E[R]; which are by construction the negative covariance of factors F with the SDF projection on asset returns, i.e., the minimum variance SDF. Optionally computes the corresponding heteroskedasticity and autocorrelation robust standard errors using the Newey-West (1994) doi:10.2307/2297912 plug-in procedure to select the number of relevant lags, i.e., n_lags = 4 * (n_observations/100)^(2/9). For the standard error computations, the function allows to internally pre-whiten the series by fitting a VAR(1), i.e., a vector autoregressive model of order 1. All details are found in Quaini-Trojani-Yuan (2023) doi:10.2139/ssrn.4574683.

Usage

```
TFRP(
  returns,
  factors,
```

```
  include_standard_errors = FALSE,
  hac_prewhite = FALSE,
  check_arguments = TRUE
)
```

Arguments

returns	A n_observations x n_returns-dimensional matrix of test asset excess returns.
factors	A n_observations x n_factors-dimensional matrix of factors.
include_standard_errors	A boolean: TRUE if you want to compute the tradable factor risk premia HAC standard errors; FALSE otherwise. Default is FALSE.
hac_prewhite	A boolean indicating if the series needs prewhitening by fitting an AR(1) in the internal heteroskedasticity and autocorrelation robust covariance (HAC) estimation. Default is false.
check_arguments	A boolean: TRUE for internal check of all function arguments; FALSE otherwise. Default is TRUE.

Value

A list containing n_factors-dimensional vector of tradable factor risk premia in "risk_premia"; if include_standard_errors=TRUE, then it further includes n_factors-dimensional vector of tradable factor risk premia standard errors in "standard_errors".

Examples

```
# import package data on 6 risk factors and 42 test asset excess returns
factors = intrinsicFRP::factors[,-1]
returns = intrinsicFRP::returns[,-1]

# compute tradable factor risk premia and their standard errors
tfrp = TFRP(returns, factors, include_standard_errors = TRUE)
```

Index

∗ **datasets**
factors, 4
returns, 15
risk_free, 15

ChenFang2019BetaRankTest, 3

factors, 4
FGXFactorsTest, 4
FRP, 6

GKRFactorScreening, 7

HACcovariance, 9
HJMisspecificationDistance, 10

IterativeKleibergenPaap2006BetaRankTest, 11

OracleTFRP, 12

returns, 15
risk_free, 15

SDFCoefficients, 16

TFRP, 17