Package ‘PINstimation’

December 17, 2025

Type Package

Version 0.2.0

Date 2025-12-16

Title Estimation of the Probability of Informed Trading

Maintainer Montasser Ghachem <montasser.ghachem@pinstimation.com>

Description A comprehensive bundle of utilities for the estimation of probability of informed trading models: original PIN in Easley and O'Hara (1992) and Easley et al. (1996); Multilayer PIN (MPIN) in Ersan (2016); Adjusted PIN (AdjPIN) in Duarte and Young (2009); and volume-synchronized PIN (VPIN) in Easley et al. (2011, 2012). Implementations of various estimation methods suggested in the literature are included. Additional compelling features comprise posterior probabilities, an implementation of an expectation-maximization (EM) algorithm, and PIN decomposition into layers, and into bad/good components. Versatile data simulation tools, and trade classification algorithms are among the supplementary utilities. The package provides fast, compact, and precise utilities to tackle the sophisticated, error-prone, and time-consuming estimation procedure of informed trading, and this solely using the raw trade-level data.

URL https://www.pinstimation.com,
https://github.com/monty-se/PINstimation

BugReports https://github.com/monty-se/PINstimation/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.2

VignetteBuilder knitr

Imports Rdpack, knitr, methods, skellam, nloptr, furrr, future, dplyr, rmarkdown, coda, magrittr, tidyr

RdMacros Rdpack

Depends R (>= 3.5.0)

Suggests fansi, htmltools

Language en-US

NeedsCompilation no

Author Montasser Ghachem [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-6991-3316>),
Oguz Ersan [aut] (ORCID: <https://orcid.org/0000-0003-3135-5317>),
Alexandre Borentain [ctb]

Repository CRAN

Date/Publication 2025-12-17 18:40:02 UTC

Contents

adjpin *Estimation of adjusted PIN model*

Description

Estimates the Adjusted Probability of Informed Trading (`adjPIN`) as well as the Probability of Symmetric Order-flow Shock (`PSOS`) from the `AdjPIN` model of Duarte and Young(2009).

Usage

```
adjpin(data, method = "ECM", initialsets = "GE", num_init = 20,
             restricted = list(), ..., verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`method` A character string referring to the method used to estimate the model of Duarte and Young (2009). It takes one of two values: `"ML"` refers to the standard maximum likelihood estimation, and `"ECM"` refers to the expectation-conditional maximization algorithm. The default value is `"ECM"`. Details of the ECM method, and comparative results can be found in Ghachem and Ersan (2025), and in Ghachem and Ersan (2023).

`initialsets` It can either be a character string referring to prebuilt algorithms generating initial parameter sets or a dataframe containing custom initial parameter sets. If `initialsets` is a character string, it refers to the method of generation of the initial parameter sets, and takes one of three values: `"GE"`, `"CL"`, or `"RANDOM"`. `"GE"` refers to initial parameter sets generated by the algorithm of Ersan and Ghachem (2025), and implemented in `initials_adjpin()`, `"CL"` refers to initial parameter sets generated by the algorithm of Cheng and Lai (2021), and implemented in `initials_adjpin_cl()`, while `"RANDOM"` generates random initial parameter sets as implemented in `initials_adjpin_rnd()`. The default value is `"GE"`. If `initialsets` is a dataframe, the function `adjpin()` will estimate the AdjPIN model using the provided initial parameter sets.

`num_init` An integer specifying the maximum number of initial parameter sets to be used in the estimation. If `initialsets="GE"`, the generation of initial parameter sets will stop when the number of initial parameter sets reaches `num_init`. It can stop earlier if the number of all possible generated initial parameter sets is lower than `num_init`. If `initialsets="RANDOM"`, exactly `num_init` initial parameter sets are returned. If `initialsets="CL"`: then `num_init` is ignored, and all `256` initial parameter sets are used. The default value is `20`. `[i]` The argument `num_init` is ignored when the argument `initialsets` is a dataframe.

`restricted` A binary list that allows estimating restricted AdjPIN models by specifying which model parameters are assumed to be equal. It contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the probability of liquidity shock in no-information days, and in information days is assumed to be the same (θ=θ'). If any of the remaining rate elements `{mu, eps, d}` is set to `TRUE`, (say `mu=TRUE`), then the rate is assumed to be the same on the buy side, and on the sell side (μ_b=μ_s). If more than one element is set to `TRUE`, then the restrictions are combined. For instance, if the argument `restricted` is set to `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the restricted AdjPIN model is estimated, where θ=θ', ε_b=ε_s, and Δ_b=Δ_s. If the value of the argument `restricted` is the empty list (`list()`), then all parameters of the model are assumed to be independent, and the unrestricted model is estimated. The default value is the empty list `list()`.

`...` Additional arguments passed on to the function `adjpin()`. The recognized arguments are `hyperparams`, and `fact`. The argument `hyperparams` consists of a list containing the hyperparameters of the `ECM` algorithm. When not empty, it contains one or more of the following elements: `maxeval`, and `tolerance`. It is used only when the `method` argument is set to `"ECM"`. The argument `fact` is a binary value that determines which likelihood functional form is used: A factorization of the likelihood function by Ersan and Ghachem (2025) when it is set to `TRUE`, otherwise, the original likelihood function of Duarte and Young (2009). The default value is `TRUE`. More about these arguments are in the Details section.

`verbose` A binary variable that determines whether detailed information about the steps of the estimation of the AdjPIN model is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

If `initialsets` is neither a dataframe, nor a character string from the set `{"GE", "CL", "RANDOM"}`, the estimation of the `AdjPIN` model is aborted. The default initial parameters (`"GE"`) for the estimation method are generated using a modified hierarchical agglomerative clustering. For more information, see `initials_adjpin()`.

The argument `hyperparams` contains the hyperparameters of the `ECM` algorithm. It is either empty or contains one or two of the following elements:

- `maxeval`: (`integer`) It stands for maximum number of iterations of the `ECM` algorithm for each initial parameter set. When missing, `maxeval` takes the default value of `100`.
- `tolerance` (`numeric`) The `ECM` algorithm is stopped when the (relative) change of log-likelihood is smaller than tolerance. When missing, `tolerance` takes the default value of `0.001`.

Value

Returns an object of class estimate.adjpin-class.

References

Cheng T, Lai H (2021). “Improvements in estimating the probability of informed trading models.” *Quantitative Finance*, **21**(5), 771-796.

Duarte J, Young L (2009). “Why is PIN priced?” *Journal of Financial Economics*, **91**(2), 119–138. ISSN 0304405X.

Ersan O, Ghachem M (2025). “A methodological approach to the computational problems in the estimation of adjusted PIN model.” *Quantitative Finance*, 1–13.

Ghachem M, Ersan O (2023). “PINstimation: An R Package for estimating probability of informed trading models.” *The R Journal*, **15**(2023), 145–168.

Ghachem M, Ersan O (2025). “Estimation of the probability of informed trading models via an expectation-conditional maximization algorithm.” *Financial Innovation*, **11**(1), 67.

Examples

```
# We use 'generatedata_adjpin()' to generate a S4 object of type 'dataset'
# with 60 observations.

sim_data <- generatedata_adjpin(days = 60)

# The actual dataset of 60 observations is stored in the slot 'data' of the
# S4 object 'sim_data'. Each observation corresponds to a day and contains
# the total number of buyer-initiated transactions ('B') and seller-
# initiated transactions ('S') on that day.

xdata <- sim_data@data

# ----------------------------------------------------------------------- #
# Compare the unrestricted AdjPIN model with various restricted models     #
# ----------------------------------------------------------------------- #

# Estimate the unrestricted AdjPIN model using the ECM algorithm (default),
# and show the estimation output

estimate.adjpin.0 <- adjpin(xdata, verbose = FALSE)

show(estimate.adjpin.0)

# Estimate the restricted AdjPIN model where mub=mus

estimate.adjpin.1 <- adjpin(xdata, restricted = list(mu = TRUE),
                                  verbose = FALSE)

# Estimate the restricted AdjPIN model where eps.b=eps.s

estimate.adjpin.2 <- adjpin(xdata, restricted = list(eps = TRUE),
                                  verbose = FALSE)
```

```
# Estimate the restricted AdjPIN model where d.b=d.s

estimate.adjpin.3 <- adjpin(xdata, restricted = list(d = TRUE),
                                  verbose = FALSE)

# Compare the different values of adjusted PIN

estimates <- list(estimate.adjpin.0, estimate.adjpin.1,
                  estimate.adjpin.2, estimate.adjpin.3)

adjpins <- sapply(estimates, function(x) x@adjpin)

psos <- sapply(estimates, function(x) x@psos)

summary <- cbind(adjpins, psos)
rownames(summary) <- c("unrestricted", "same.mu", "same.eps", "same.d")

show(round(summary, 5))
```

dailytrades *Example of quarterly data*

Description

An example dataset representative of quarterly data containing the aggregate numbers of buyer-initiated and seller-initiated trades for each trading day.

Usage

```
dailytrades
```

Format

A data frame with 60 observations and 2 variables:

- B: total number of buyer-initiated trades.
- S: total number of seller-initiated trades.

Source

Artificially created data set.

data.series-class *List of dataset objects*

Description

The class data.series is the blueprint of S4 objects that store a list of dataset objects.

Usage

```
## S4 method for signature 'data.series'
show(object)
```

Arguments

object	an object of class data.series

Slots

series (numeric) returns the number of dataset objects stored.

days (numeric) returns the length of the simulated data in days common to all dataset objects stored. The default value is 60.

model (character) returns a character string, either 'MPIN' or 'adjPIN'.

layers (numeric) returns the number of information layers in all dataset objects stored. It takes the value 1 for the adjusted PIN model, i.e. when model takes the value 'adjPIN'.

datasets (list) returns the list of the dataset objects stored.

restrictions (list) returns a binary list that contains the set of parameter restrictions on the original AdjPIN model in the estimated AdjPIN model. The restrictions are imposed equality constraints on model parameters. If the value of the parameter restricted is the empty list (list()), then the model has no restrictions, and the estimated model is the unrestricted, i.e., the original AdjPIN model. If not empty, the list contains one or multiple of the following four elements {theta, mu, eps, d}. For instance, If theta is set to TRUE, then the estimated model has assumed the equality of the probability of liquidity shocks in no-information, and information days, i.e., θ=θ'. If any of the remaining rate elements {mu, eps, d} is equal to TRUE, (say mu=TRUE), then the estimated model imposed equality of the concerned parameter on the buy side, and on the sell side (μ_b=μ_s). If more than one element is equal to TRUE, then the restrictions are combined. For instance, if the slot restrictions contains list(theta=TRUE, eps=TRUE, d=TRUE), then the estimated AdjPIN model has three restrictions θ=θ', ε_b=ε_s, and Δ_b=Δ_s, i.e., it has been estimated with just 7 parameters, in comparison to 10 in the original unrestricted model. [i] This slot only concerns datasets generated by the function generatedata_adjpin().

warnings (numeric) returns numbers referring to the warning errors caused by a conflict between the different arguments used to call the function generatedata_mpin().

runningtime (numeric) returns the running time of the data simulation in seconds.

`dataset-class` *Simulated data object*

Description

The class `dataset` is a blueprint of `S4` objects that store the result of simulation of the aggregate daily trading data.

Usage

```
## S4 method for signature 'dataset'
show(object)
```

Arguments

`object`	an object of class `dataset`

Details

`theoreticals` are the parameters used to generate the daily buys and sells. `empiricals` are computed from the generated daily buys and sells. If we generate data for a 60 days using α=0.1, the most likely outcome is to obtain 6 days (0.1 x 60) as information event days. In this case, the theoretical value of α=`0.1` is equal to the empirically estimated value of α=`6/60=0.1`. The number of generated information days can, however, be different from `6`; say `5`. In this case, empirical (actual) α parameter derived from the generated numbers would be `5/60=0.0833`, which differs from the theoretical α=`0.1`. The weak law of large numbers ensures the empirical parameters (`empiricals`) converge towards the theoretical parameters (`theoreticals`) when the number of days becomes very large. To detect the estimation biases from the models/methods, comparing the estimates with `empiricals` rather than `theoreticals` would yield more realistic results.

Slots

`model` (`character`) returns the model being simulated, either `"MPIN"`, or `"adjPIN"`.

`days` (`numeric`) returns the length of the generated data in days.

`layers` (`numeric`) returns the number of information layers in the simulated data. It takes the value `1` for the adjusted PIN model, i.e. when `model` takes the value `'adjPIN'`.

`theoreticals` (`list`) returns the list of the theoretical parameters used to generate the data.

`empiricals` (`list`) returns the list of the empirical parameters computed from the generated data.

`aggregates` (`numeric`) returns an aggregation of information layers' empirical parameters alongside with ε_b and ε_s. The aggregated parameters are calculated as follows: $\alpha_{agg} = \sum \alpha_j$ $\delta_{agg} = \sum \alpha_j \times \delta_j$, and $\mu_{agg} = \sum \alpha_j \times \mu_j$.

`emp.pin` (`numeric`) returns the `PIN/MPIN/AdjPIN` value derived from the empirically estimated parameters of the generated data.

`data` (`dataframe`) returns a dataframe containing the generated data.

likelihood (numeric) returns the value of the (log-)likelihood function evaluated at the empirical parameters.

warnings (character) stores warning messages for events that occurred during the data generation, such as conflict between two arguments.

restrictions (list) returns a binary list that contains the set of parameter restrictions on the original AdjPIN model in the estimated AdjPIN model. The restrictions are imposed equality constraints on model parameters. If the value of the parameter restricted is the empty list (list()), then the model has no restrictions, and the estimated model is the unrestricted, i.e., the original AdjPIN model. If not empty, the list contains one or multiple of the following four elements {theta, mu, eps, d}. For instance, If theta is set to TRUE, then the estimated model has assumed the equality of the probability of liquidity shocks in no-information, and information days, i.e., θ=θ'. If any of the remaining rate elements {mu, eps, d} is equal to TRUE, (say mu=TRUE), then the estimated model imposed equality of the concerned parameter on the buy side, and on the sell side (μ_b=μ_s). If more than one element is equal to TRUE, then the restrictions are combined. For instance, if the slot restrictions contains list(theta=TRUE, eps=TRUE, d=TRUE), then the estimated AdjPIN model has three restrictions θ=θ', ε_b=ε_s, and Δ_b=Δ_s, i.e., it has been estimated with just 7 parameters, in comparison to 10 in the original unrestricted model. [i] This slot only concerns datasets generated by the function generatedata_adjpin().

---

detecting-layers *Layer detection in trade-data*

Description

Detects the number of information layers present in trade-data using the algorithms in Ersan (2016), Ersan and Ghachem (2024), and Ghachem and Ersan (2025).

Usage

```
detectlayers_e(data, confidence = 0.995, correction = TRUE)

detectlayers_eg(data, confidence = 0.995)

detectlayers_ecm(data, hyperparams = list())
```

Arguments

data	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
confidence	A number from (0.5,1), corresponding to the range of the confidence interval used to determine whether a given cluster is compact, and therefore can be considered an information layer. If all values of absolute order imbalances (AOI) within a given cluster are within the confidence interval of a Skellam distribution with level equal to 'confidence', and centered on the mean of AOI, then the cluster is considered compact, and, therefore, an information layer. If

some observations are outside the confidence interval, then the data is clustered further. The default value is `0.995`. `[i]` This is an argument of the functions `detectlayers_e()`, and `detectlayers_eg()`.

`correction` A binary variable that determines whether the data will be adjusted prior to implementing the algorithm of Ersan (2016). The default value is `TRUE`.

`hyperparams` A list containing the hyperparameters of the `ECM` algorithm. When not empty, it contains one or more of the following elements: `maxeval`, `tolerance`, `maxinit`, and `maxlayers`. More about these elements are found in the Details section. `[i]` This is an argument of the function `detectlayers_ecm()`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The argument `hyperparams` contains the hyperparameters of the `ECM` algorithm. It is either empty or contains one or more of the following elements:

- `maxeval`: `(integer)` It stands for maximum number of iterations of the `ECM` for each initial parameter set. When missing, `maxeval` takes the default value of `100`.
- `tolerance` `(numeric)` The `ECM` algorithm is stopped when the (relative) change of log-likelihood is smaller than tolerance. When missing, `tolerance` takes the default value of `0.001`.
- `maxinit`: `(integer)` It is the maximum number of initial parameter sets used for the `ECM` estimation per layer. When missing, `maxinit` takes the default value of `20`.
- `maxlayers` `(integer)` It is the upper limit of number of layers used in the ECM algorithm. To find the optimal number of layers, the ECM algorithm will estimate a model for each value of the number of layers between `1` and `maxlayers`, and then picks the model that has the lowest Bayes information criterion (BIC). When missing, `maxlayers` takes the default value of `8`.

Value

Returns an integer corresponding to the number of layers detected in the data.

References

Ersan O (2016). “Multilayer Probability of Informed Trading.” *Available at SSRN 2874420.*

Ersan O, Ghachem M (2024). “Identifying information types in the estimation of informed trading: an improved algorithm.” *Journal of Risk and Financial Management*, **17**(9), 409.

Ghachem M, Ersan O (2025). “Estimation of the probability of informed trading models via an expectation-conditional maximization algorithm.” *Financial Innovation*, **11**(1), 67.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Detect the number of layers present in the dataset 'dailytrades' using the
# different algorithms and display the results

e.layers <- detectlayers_e(xdata)
eg.layers <- detectlayers_eg(xdata)
em.layers <- detectlayers_ecm(xdata)

show(c(e = e.layers, eg = eg.layers, em = em.layers))
```

estimate.adjpin-class *AdjPIN estimation results*

Description

The class `estimate.adjpin` is a blueprint of the `S4` objects that store the results of the estimation of the `AdjPIN` model using `adjpin()`.

Usage

```
## S4 method for signature 'estimate.adjpin'
show(object)
```

Arguments

`object`	(estimate.adjpin-class)

Slots

`success` (`logical`) takes the value `TRUE` when the estimation has succeeded, `FALSE` otherwise.

`errorMessage` (`character`) contains an error message if the estimation of the `AdjPIN` model has failed, and is empty otherwise.

`convergent.sets` (`numeric`) returns the number of initial parameter sets, for which the likelihood maximization converged.

`method` (`character`) contains a reference to the estimation method: `"ECM"` for expectation-conditional maximization algorithm and '`"ML"`' for standard maximum likelihood estimation.

`factorization` (`character`) contains a reference to the factorization of the likelihood function used: `"GE"`for the factorization in Ersan and Ghachem (2025), and `"NONE"` for the original likelihood function in Duarte and Young (2009).

`restrictions` (`list`) returns a binary list that contains the set of parameter restrictions on the original AdjPIN model in the estimated AdjPIN model. The restrictions are imposed equality constraints on model parameters. If the value of the parameter `restricted` is the empty list (`list()`), then the model has no restrictions, and the estimated model is the unrestricted, i.e., the original AdjPIN model. If not empty, the list contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the estimated model has assumed the equality of the probability of liquidity shocks in no-information, and information days, i.e., θ=θ'. If any of the remaining rate elements `{mu, eps, d}` is equal to `TRUE`, (say `mu=TRUE`), then the estimated model imposed equality of the concerned parameter on the buy side, and on the sell side (μ_b=μ_s). If more than one element is equal to `TRUE`, then the restrictions are combined. For instance, if the slot `restrictions` contains `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the estimated AdjPIN model has three restrictions θ=θ', ε_b=ε_s, and Δ_b=Δ_s, i.e., it has been estimated with just `7` parameters, in comparison to `10` in the original unrestricted model.

`algorithm` (`character`) returns the implemented initial parameter set determination algorithm. `"GE"` is for Ersan and Ghachem (2025), `"CL"` is for Cheng and Lai (2021), `"RANDOM"` for random initial parameter sets, and `"CUSTOM"` for custom initial parameter sets.

`parameters` (`numeric`) returns the vector of the optimal maximum-likelihood estimates (α, δ, θ, θ', ε_b, ε_s, μ_b, μ_s, Δ_b, Δ_s).

`likelihood` (`numeric`) returns the value (of the factorization) of the likelihood function, as in Ersan and Ghachem (2025), evaluated at the set of optimal parameters.

`adjpin` (`numeric`) returns the value of the adjusted probability of informed trading (Duarte and Young 2009).

`psos` (`numeric`) returns the probability of symmetric order flow shock (Duarte and Young 2009).

`dataset` (`dataframe`) returns the dataset of buys and sells used in the estimation of the AdjPIN model.

`initialsets` (`dataframe`) returns the initial parameter sets used in the estimation of AdjPIN model.

`details` (`dataframe`) returns a dataframe containing the estimated parameters for each initial parameter set.

`hyperparams` (`list`) returns the hyperparameters of the `ECM` algorithm, which are `maxeval`, and `tolerance`.

`runningtime` (`numeric`) returns the running time of the `AdjPIN` estimation in seconds.

`estimate.mpin-class` *MPIN estimation results*

Description

The class `estimate.mpin` is the blueprint of `S4` objects that store the results of the estimation of the `MPIN` model, using the function `mpin_ml()`.

Usage

```
## S4 method for signature 'estimate.mpin'
show(object)
```

Arguments

`object`	an object of class `estimate.mpin`

Slots

`success` (`logical`) returns the value `TRUE` when the estimation has succeeded, `FALSE` otherwise.

`errorMessage` (`character`) returns an error message if the estimation of the `MPIN` model has failed, and is empty otherwise.

`convergent.sets` (`numeric`) returns the number of initial parameter sets at which the likelihood maximization converged.

`method` (`character`) returns the method of estimation used, and is equal to 'Maximum Likelihood Estimation'.

`layers` (`numeric`) returns the number of layers detected in the trading data, or provided by the user.

`detection` (logical) returns a reference to the layer-detection algorithm used (`"E"`, `"EG"`, `"ECM"`), if any algorithm is used. If the number of layers is provided by the user, detection takes the value `"USER"`.

`parameters` (`list`) returns the list of the maximum likelihood estimates (α, δ, μ, ε_b, ε_s), where α, δ, and μ are numeric vectors of length `layers`.

`aggregates` (`numeric`) returns an aggregation of information layers' estimated parameters alongside with ε_b, and ε_s. The aggregated parameters are calculated as follows: $\alpha_{agg} = \sum \alpha_j$ $\delta_{agg} = \sum \alpha_j \times \delta_j$, and $\mu_{agg} = \sum \alpha_j \times \mu_j$.

`likelihood` (`numeric`) returns the value of the (log-)likelihood function evaluated at the optimal set of parameters.

`mpinJ` (`numeric`) returns the values of the multilayer probability of informed trading per layer, calculated using the layer-specific estimated parameters.

`mpin` (`numeric`) returns the global value of the multilayer probability of informed trading. It is the sum of the multilayer probabilities of informed trading per layer stored in the slot `mpinJ`.

`mpin.goodbad` (`list`) returns a list containing a decomposition of `MPIN` into good-news, and bad-news `MPIN` components. The decomposition has been suggested for PIN measure in Brennan et al. (2016). The list has four elements: `mpinG`, and `mpinB` are the global good-news, and bad-news components of `MPIN`, while `mpinGj`, and `mpinBj` are two vectors containing the good-news (bad-news) components of `MPIN` computed per layer.

`dataset` (`dataframe`) returns the dataset of buys and sells used in the maximum likelihood estimation of the MPIN model.

`initialsets` (`dataframe`) returns the initial parameter sets used in the maximum likelihood estimation of the MPIN model.

`details` (`dataframe`) returns a dataframe containing the estimated parameters of the `MLE` method for each initial parameter set.

`runningtime` (`numeric`) returns the running time of the estimation of the `MPIN` model in seconds.

`estimate.mpin.ecm-class`

MPIN estimation results (ECM)

Description

The class `estimate.mpin.ecm` is the blueprint of `S4` objects that store the results of the estimation of the `MPIN` model using the Expectation-Conditional Maximization method, as implemented in the function `mpin_ecm()`.

Usage

```
## S4 method for signature 'estimate.mpin.ecm'
show(object)

selectModel(object, criterion)

## S4 method for signature 'estimate.mpin.ecm'
selectModel(object, criterion)

getSummary(object)

## S4 method for signature 'estimate.mpin.ecm'
getSummary(object)
```

Arguments

`object`	an object of class `estimate.mpin.ecm`.
`criterion`	a character string specifying the model selection criterion. `criterion` should take one of these values `{"BIC", "AIC", "AWE"}`. They stand for Bayesian Information Criterion, Akaike Information Criterion, and Approximate Weight of Evidence, respectively.

Functions

- `selectModel(estimate.mpin.ecm)`: returns the optimal model among the estimated models, i.e., the model having the lowest information criterion, provided by the user.
- `getSummary(estimate.mpin.ecm)`: returns a summary of the estimation of the `MPIN` model using the `ECM` algorithm for different values of the argument `layers`. For each estimation, the number of layers, the `MPIN` value, the log-likelihood value, as well as the values of the different information criteria, namely `AIC`, `BIC` and `AWE` are displayed.

Slots

`success` (`logical`) returns the value `TRUE` when the estimation has succeeded, `FALSE` otherwise.

`errorMessage` (`character`) returns an error message if the `MPIN` estimation has failed, and is empty otherwise.

`convergent.sets` (`numeric`) returns the number of initial parameter sets at which the likelihood maximization converged.

`method` (`character`) returns the method of estimation, and is equal to 'Expectation-Conditional Maximization Algorithm'.

`layers` (`numeric`) returns the number of layers estimated by the Expectation-Conditional Maximization algorithm, or provided by the user.

`optimal` (`logical`) returns whether the number of layers used for the estimation is provided by the user `(optimal=FALSE)`, or determined by the `ECM` algorithm `(optimal=TRUE)`.

`parameters` (`list`) returns the list of the maximum likelihood estimates (α, δ, μ, ε_b, ε_s), where α, δ, and μ are numeric vectors of length `layers`.

`aggregates` (`numeric`) returns an aggregation of information layers' parameters alongside with ε_b and ε_s. The aggregated parameters are calculated as follows: $\alpha_{agg} = \sum \alpha_j$ $\delta_{agg} = \sum \alpha_j \times \delta_j$, and $\mu_{agg} = \sum \alpha_j \times \mu_j$.

`likelihood` (`numeric`) returns the value of the (log-)likelihood function evaluated at the optimal set of parameters.

`mpinJ` (`numeric`) returns the values of the multilayer probability of informed trading per layer, calculated using the layer-specific estimated parameters.

`mpin` (`numeric`) returns the global value of the multilayer probability of informed trading. It is the sum of the multilayer probabilities of informed trading per layer stored in the slot `mpinJ`.

`mpin.goodbad` (`list`) returns a list containing a decomposition of `MPIN` into good-news, and bad-news `MPIN` components. The decomposition has been suggested for PIN measure in Brennan et al. (2016). The list has four elements: `mpinG`, and `mpinB` are the global good-news, and bad-news components of `MPIN`, while `mpinGj`, and `mpinBj` are two vectors containing the good-news (bad-news) components of `MPIN` computed per layer.

`dataset` (`dataframe`) returns the dataset of buys and sells used in the ECM estimation of the MPIN model.

`initialsets` (`dataframe`) returns the initial parameter sets used in the ECM estimation of the MPIN model.

`details` (`dataframe`) returns a dataframe containing the estimated parameters of the `ECM` method for each initial parameter set.

`models` (`list`) returns the list of `estimate.mpin.ecm` objects storing the results of estimation using the function `mpin_ecm()` for different values of the argument `layers`. It returns `NULL` when the argument `layers` of the function `mpin_ecm()` take a specific value.

`AIC` (`numeric`) returns the value of the Akaike Information Criterion (AIC).

`BIC` (`numeric`) returns the value of the Bayesian Information Criterion (BIC).

`AWE` (`numeric`) returns the value of the Approximate Weight of Evidence.

`criterion` (`character`) returns the model selection criterion used to find the optimal estimate for the `MPIN` model. It takes one of these values `'BIC'`, `'AIC'`, `'AWE'`; which stand for Bayesian Information Criterion, Akaike Information Criterion, and Approximate Weight of Evidence, respectively.

`hyperparams` (`list`) returns the hyperparameters of the `ECM` algorithm, which are `minalpha`, `maxeval`, `tolerance`, and `maxlayers`. Check the details section of `mpin_ecm()` to know more about these parameters.

`runningtime` (`numeric`) returns the running time of the estimation in seconds.

estimate.pin-class *PIN estimation results*

Description

The class estimate.pin is a blueprint of S4 objects that store the results of the different PIN functions: pin(), pin_yz(), pin_gwj(), and pin_ea().

Usage

```
## S4 method for signature 'estimate.pin'
show(object)
```

Arguments

object	an object of class estimate.pin

Slots

success (logical) takes the value TRUE when the estimation has succeeded, FALSE otherwise.

errorMessage (character) contains an error message if the PIN estimation has failed, and is empty otherwise.

convergent.sets (numeric) returns the number of initial parameter sets at which the likelihood maximization converged.

algorithm (character) returns the algorithm used to determine the set of initial parameter sets for the maximum likelihood estimation. It takes one of the following values:

- "YZ": Yan and Zhang (2012)
- "GWJ": Gan, Wei and Johnstone (2015)
- "YZ*": Yan and Zhang (2012) as modified by Ersan and Alici (2016)
- "EA": Ersan and Alici (2016)
- "CUSTOM": Custom initial parameter sets

factorization (character) returns the factorization of the PIN likelihood function as used in the maximum likelihood estimation. It takes one of the following values:

- "NONE": No factorization
- "EHO": Easley, Hvidkjaer and O'Hara (2010)
- "LK": Lin and Ke (2011)
- "E": Ersan (2016)

parameters (list) returns the list of the maximum likelihood estimates (α, δ, μ, ε_b, ε_s)

likelihood (numeric) returns the value of (the factorization of) the likelihood function evaluated at the optimal set of parameters.

pin (numeric) returns the value of the probability of informed trading.

`pin.goodbad` (`list`) returns a list containing a decomposition of `PIN` into good-news, and bad-news `PIN` components. The decomposition has been suggested in Brennan et al. (2016). The list has two elements: `pinG`, and `pinB` are the good-news, and bad-news components of `PIN`, respectively.

`dataset` (`dataframe`) returns the dataset of buys and sells used in the maximum likelihood estimation of the PIN model.

`initialsets` (`dataframe`) returns the initial parameter sets used in the maximum likelihood estimation of the PIN model.

`details` (`dataframe`) returns a dataframe containing the estimated parameters by the `MLE` method for each initial parameter set.

`runningtime` (`numeric`) returns the running time of the estimation of the `PIN` model in seconds.

`estimate.vpin-class` *VPIN estimation results*

Description

The class `estimate.vpin` is a blueprint for `S4` objects that store the results of the `VPIN` estimation method using the function `vpin()`.

The function show() displays a description of the estimate.vpin object: descriptive statistics of the `VPIN` variable, the set of relevant parameters, and the running time.

Usage

```
## S4 method for signature 'estimate.vpin'
show(object)
```

Arguments

`object`	an object of class `estimate.vpin`

Slots

`success` (`logical`) returns the value `TRUE` when the estimation has succeeded, `FALSE` otherwise.

`errorMessage` (`character`) returns an error message if the `VPIN` estimation has failed, and is empty otherwise.

`improved` (`logical`) returns the value `TRUE` when the model used is the improved volume-synchronized probability of informed trading of Ke and Lin (2017), and `FALSE` when the model used is the volume-synchronized probability of informed trading of Easley et al.(2011,2012).

`parameters` (`numeric`) returns a numeric vector of estimation parameters (tbSize, buckets, samplength, VBS, #days), where `tbSize` is the size of timebars (in seconds); `buckets` is the number of buckets per average volume day; `VBS` is Volume Bucket Size (daily average volume/number of buckets `buckets`); `samplength` is the length of the window used to estimate `VPIN`; and `#days` is the number of days in the dataset.

bucketdata (dataframe) returns the dataframe containing detailed information about buckets. Following the output of Abad and Yague (2012), we report for each bucket its identifier (bucket), the aggregate buy volume (agg.bVol), the aggregate sell volume (agg.sVol), the absolute order imbalance (AOI=|agg.bVol-agg.sVol|), the start time (starttime), the end time (endtime), the duration in seconds (duration) as well as the VPIN vector.

vpin (numeric) returns the vector of the volume-synchronized probabilities of informed trading.

ivpin (numeric) returns the vector of the improved volume- synchronized probabilities of informed trading as in Ke and Lin (2017).

dailyvpin (dataframe) returns the daily VPIN values. Two variants are provided for any given day: dvpin corresponds to the unweighted average of vpin values, and dvpin.weighted corresponds to the average of vpin values weighted by bucket duration.

runningtime (numeric) returns the running time of the VPIN estimation in seconds.

factorizations *Factorizations of the different PIN likelihood functions*

Description

The PIN likelihood function is derived from the original PIN model as developed by Easley and Ohara (1992) and Easley et al. (1996). The maximization of the likelihood function as is leads to computational problems, in particular, to floating point errors. To remedy to this issue, several log-transformations or factorizations of the different PIN likelihood functions have been suggested. The main factorizations in the literature are:

- fact_pin_eho(): factorization of Easley et al. (2010)
- fact_pin_lk(): factorization of Lin and Ke (2011)
- fact_pin_e(): factorization of Ersan (2016)

The factorization of the likelihood function of the multilayer PIN model, as developed in Ersan (2016).

- fact_mpin(): factorization of Ersan (2016)

The factorization of the likelihood function of the adjusted PIN model (Duarte and Young 2009), is derived, and presented in Ersan and Ghachem (2025).

- fact_adjpin(): factorization in Ersan and Ghachem (2025)

Usage

```
fact_pin_eho(data, parameters = NULL)

fact_pin_lk(data, parameters = NULL)

fact_pin_e(data, parameters = NULL)

fact_mpin(data, parameters = NULL)

fact_adjpin(data, parameters = NULL)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`parameters` In the case of the `PIN` likelihood factorization, it is an ordered numeric vector (α, δ, μ, ε_b, ε_s). In the case of the `MPIN` likelihood factorization, it is an ordered numeric vector (α, δ, μ, ε_b, ε_s), where α, δ, and μ are numeric vectors of size `J`, where `J` is the number of information layers in the data. In the case of the `AdjPIN` likelihood factorization, it is an ordered numeric vector (α, δ, θ, θ', ε_b, ε_s, μ_b, μ_s, Δ_b, Δ_s). The default value is `NULL`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

Our tests, in line with Lin and Ke (2011), and Ersan and Alici (2016), demonstrate very similar results for `fact_pin_lk()`, and `fact_pin_e()`, both having substantially better estimates than `fact_pin_eho()`.

Value

If the argument `parameters` is omitted, returns a function object that can be used with the optimization functions `optim()`, and `neldermead()`.

If the argument `parameters` is provided, returns a numeric value of the log-likelihood function evaluated at the dataset `data` and the parameters `parameters`, where `parameters` is a numeric vector following this order (α, δ, μ, ε_b, ε_s) for the factorizations of the `PIN` likelihood function, (α, δ, μ, ε_b, ε_s) for the factorization of the `MPIN` likelihood function, and (α, δ, θ, θ', ε_b, ε_s ,μ_b, μ_s, Δ_b, Δ_s) for the factorization of the `AdjPIN` likelihood function.

References

Duarte J, Young L (2009). "Why is PIN priced?" *Journal of Financial Economics*, **91**(2), 119–138. ISSN 0304405X.

Easley D, Hvidkjaer S, Ohara M (2010). "Factoring information into returns." *Journal of Financial and Quantitative Analysis*, **45**(2), 293–309. ISSN 00221090.

Easley D, Kiefer NM, Ohara M, Paperman JB (1996). "Liquidity, information, and infrequently traded stocks." *Journal of Finance*, **51**(4), 1405–1436. ISSN 00221082.

Easley D, Ohara M (1992). "Time and the Process of Security Price Adjustment." *The Journal of Finance*, **47**(2), 577–605. ISSN 15406261.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN).” *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Ersan O, Ghachem M (2025). “A methodological approach to the computational problems in the estimation of adjusted PIN model.” *Quantitative Finance*, 1–13.

Lin H, Ke W (2011). “A computing bias in estimating the probability of informed trading.” *Journal of Financial Markets*, **14**(4), 625-640. ISSN 1386-4181.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# ------------------------------------------------------------------------ #
# Using fact_pin_eho(), fact_pin_lk(), fact_pin_e() to find the likelihood #
# value as factorized by Easley(2010), Lin & Ke (2011), and Ersan(2016).   #
# ------------------------------------------------------------------------ #

# Choose a given parameter set to evaluate the likelihood function at a
# givenpoint  = (alpha, delta, mu, eps.b, eps.s)

givenpoint <- c(0.4, 0.1, 800, 300, 200)

# Use the ouput of fact_pin_e() with the optimization function optim() to
# find optimal estimates of the PIN model.

model <- suppressWarnings(optim(givenpoint, fact_pin_e(xdata)))

# Collect the model estimates from the variable model and display them.

varnames <- c("alpha", "delta", "mu", "eps.b", "eps.s")
estimates <- setNames(model$par, varnames)
show(estimates)

# Find the value of the log-likelihood function at givenpoint

lklValue <- fact_pin_lk(xdata, givenpoint)

show(lklValue)

# ------------------------------------------------------------------------ #
# Using fact_mpin() to find the value of the MPIN likelihood function as   #
# factorized by Ersan (2016).                                              #
# ------------------------------------------------------------------------ #

# Choose a given parameter set to evaluate the likelihood function at a
# givenpoint  = (alpha(), delta(), mu(), eps.b, eps.s) where alpha(), delta()
# and mu() are vectors of size 2.

givenpoint <- c(0.4, 0.5, 0.1, 0.6, 600, 1000, 300, 200)

# Use the output of fact_mpin() with the optimization function optim() to
# find optimal estimates of the PIN model.

model <- suppressWarnings(optim(givenpoint, fact_mpin(xdata)))

# Collect the model estimates from the variable model and display them.

varnames <- c(paste("alpha", 1:2, sep = ""), paste("delta", 1:2, sep = ""),
              paste("mu", 1:2, sep = ""), "eb", "es")
estimates <- setNames(model$par, varnames)
show(estimates)

# Find the value of the MPIN likelihood function at givenpoint

lklValue <- fact_mpin(xdata, givenpoint)

show(lklValue)

# ------------------------------------------------------------------------ #
# Using fact_adjpin() to find the value of the DY likelihood function as   #
# factorized by Ersan and Ghachem (2022b).                                 #
# ------------------------------------------------------------------------ #

# Choose a given parameter set to evaluate the likelihood function
# at a the initial parameter set givenpoint = (alpha, delta,
# theta, theta',eps.b, eps.s, muB, muS, db, ds)

givenpoint <- c(0.4, 0.1, 0.3, 0.7, 500, 600, 800, 1000, 300, 200)

# Use the output of fact_adjpin() with the optimization function
# neldermead() to find optimal estimates of the AdjPIN model.

low <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
up <- c(1, 1, 1, 1, Inf, Inf, Inf, Inf, Inf, Inf)
model <- nloptr::neldermead(
givenpoint, fact_adjpin(xdata), lower = low, upper = up)

# Collect the model estimates from the variable model and display them.

varnames <- c("alpha", "delta", "theta", "thetap", "eps.b", "eps.s",
              "muB", "muS", "db", "ds")
estimates <- setNames(model$par, varnames)
show(estimates)

# Find the value of the log-likelihood function at givenpoint

adjlklValue <- fact_adjpin(xdata, givenpoint)
show(adjlklValue)
```

`generatedata_adjpin` *Simulation of AdjPIN model data.*

Description

Generates a `dataset` object or a `data.series` object (a list of `dataset` objects) storing simulation parameters as well as aggregate daily buys and sells simulated following the assumption of the `AdjPIN` model of Duarte and Young (2009).

Usage

```
generatedata_adjpin(series=1, days = 60, parameters = NULL, ranges = list(),
restricted = list(), verbose = TRUE)
```

Arguments

`series` The number of datasets to generate.

`days` The number of trading days, for which aggregated buys and sells are generated. The default value is `60`.

`parameters` A vector of model parameters of size `10` and it has the following form {α, δ, θ, θ', ε_b, ε_s, μ_b, μ_s, Δ_b, Δ_s}.

`ranges` A list of ranges for the different simulation parameters having named elements `alpha` (α), `delta` (δ), `theta` (θ), `thetap` (θ'), `eps.b` (ε_b), `eps.s` (ε_s), `mu.b` (μ_b), `mu.s` (μ_s), `d.b` (Δ_b), `d.s` (Δ_s). The value of each element is a vector of two numbers: the first one is the minimal value `min_v` and the second one is the maximal value `max_v`. If the element corresponding to a given parameter is missing, the default range for that parameter is used, otherwise, the simulation parameters are uniformly drawn from the interval (`min_v`, `max_v`). The default value is `list()`.

`restricted` A binary list that allows estimating restricted AdjPIN models by specifying which model parameters are assumed to be equal. It contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the probability of liquidity shock in no-information days, and in information days is assumed to be the same (θ=θ'). If any of the remaining rate elements `{mu, eps, d}` is set to `TRUE`, (say `mu=TRUE`), then the rate is assumed to be the same on the buy side, and on the sell side (μ_b=μ_s). If more than one element is set to `TRUE`, then the restrictions are combined. For instance, if the argument `restricted` is set to `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the restricted AdjPIN model is estimated, where θ=θ', ε_b=ε_s, and Δ_b=Δ_s. If the value of the argument `restricted` is the empty list (`list()`), then all parameters of the model are assumed to be independent, and the unrestricted model is estimated. The default value is the empty list `list()`.

`verbose` A binary variable that determines whether detailed information about the progress of the data generation is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

If the argument `parameters` is missing, then the parameters are generated using the ranges specified in the argument `ranges`. If the argument `ranges` is set to `list()`, default ranges are used. Using the default ranges, the simulation parameters are obtained using the following procedure:

- α, δ: `(alpha, delta)` uniformly distributed on `(0, 1)`.
- θ, θ': `(theta,thetap)` uniformly distributed on `(0, 1)`.
- ε_b: `(eps.b)` an integer uniformly drawn from the interval `(100, 10000)` with step `50`.
- ε_s: `(eps.s)` an integer uniformly drawn from `((4/5)`ε_b, `(6/5)`ε_b`)` with step `50`.
- Δ_b: `(d.b)` an integer uniformly drawn from `((1/2)`ε_b, $2\varepsilon_b$`)`.
- Δ_s: `(d.s)` an integer uniformly drawn from `((4/5)`Δ_b, `(6/5)`Δ_b`)`.
- μ_b: `(mu.b)` uniformly distributed on the interval `((1/2) max(`ε_b, ε_s`), 5 max(`ε_b, ε_s`))`.
- μ_s: `(mu.s)` uniformly distributed on the interval `((4/5)`μ_b, `(6/5)`μ_b`)`..

Based on the simulation parameters `parameters`, daily buys and sells are generated by the assumption that buys and sells follow Poisson distributions with mean parameters:

- (ε_b, ε_s) in a day with no information and no liquidity shock;
- (ε_b+Δ_b, ε_s+Δ_s) in a day with no information and with liquidity shock;
- (ε_b+μ_b, ε_s) in a day with good information and no liquidity shock;
- (ε_b+μ_b+Δ_b, ε_s+Δ_s) in a day with good information and liquidity shock;
- (ε_b, ε_s+μ_s) in a day with bad information and no liquidity shock;
- (ε_b+Δ_s, ε_s+μ_s+Δ_s) in a day with bad information and liquidity shock;

Value

Returns an object of class `dataset` if `series=1`, and an object of class `data.series` if `series>1`.

References

Duarte J, Young L (2009). “Why is PIN priced?” *Journal of Financial Economics*, **91**(2), 119–138. ISSN 0304405X.

Examples

```
# ------------------------------------------------------------------------ #
# Generate data following the AdjPIN model using generatedata_adjpin()     #
# ------------------------------------------------------------------------ #

# With no arguments, the function generates one dataset object spanning
# 60 days, and where the parameters are chosen as described in the section
# 'Details'.

sdata <- generatedata_adjpin()

# Alternatively, simulation parameters can be provided. Recall the order of
# parameters (alpha, delta, theta, theta', eps.b, eps.s, mub, mus, db, ds).
```

```
givenpoint <- c(0.4, 0.1, 0.5, 0.6, 800, 1000, 2300, 4000, 500, 500)
sdata <- generatedata_adjpin(parameters = givenpoint)

# Data can be generated following restricted AdjPIN models, for example, with
# restrictions 'eps.b = eps.s', and 'mu.b = mu.s'.

sdata <- generatedata_adjpin(restricted = list(eps = TRUE, mu = TRUE))

# Data can be generated using provided ranges of simulation parameters as fed
# to the function using the argument 'ranges', where thetap corresponds to
# theta'.

sdata <- generatedata_adjpin(ranges = list(
  alpha = c(0.1, 0.15), delta = c(0.2, 0.2),
  theta = c(0.2, 0.6), thetap = c(0.2, 0.4)
))

# The value of a given simulation parameter can be set to a specific value by
# setting the range of the desired parameter takes a unique value, instead of
# a pair of values.

sdata <- generatedata_adjpin(ranges = list(
  alpha = 0.4, delta = c(0.2, 0.7),
  eps.b = c(100, 7000), mu.b = 8000
))

# Display the details of the generated simulation data

show(sdata)

# ------------------------------------------------------------------------ #
# Use generatedata_adjpin() to check the accuracy of adjpin()              #
# ------------------------------------------------------------------------ #

model <- adjpin(sdata@data, verbose = FALSE)

summary <- cbind(
  c(sdata@emp.pin['adjpin'], model@adjpin, abs(model@adjpin -
  sdata@emp.pin['adjpin'])),
  c(sdata@emp.pin['psos'], model@psos, abs(model@psos -
  sdata@emp.pin['psos']))
)
colnames(summary) <- c('adjpin', 'psos')
rownames(summary) <- c('Data', 'Model', 'Difference')

show(knitr::kable(summary, 'simple'))
```

generatedata_mpin *Simulation of MPIN model data*

Description

Generates a dataset object or a data.series object (a list of dataset objects) storing simulation parameters as well as aggregate daily buys and sells simulated following the assumption of the MPIN model of (Ersan 2016).

Usage

```
generatedata_mpin(series = 1, days = 60, layers = NULL,
                          parameters = NULL, ranges = list(), ...,
                          verbose = TRUE)
```

Arguments

series — The number of datasets to generate.

days — The number of trading days for which aggregated buys and sells are generated. Default value is 60.

layers — The number of information layers to be included in the simulated data. Default value is NULL. If layers is omitted or set to NULL, the number of layers is uniformly selected from the set {1, ..., maxlayers}.

parameters — A vector of model parameters of size 3J+2 where J is the number of information layers and it has the following form $\{\alpha_1$, ...,α_J, δ_1,..., δ_J, μ_1,..., μ_J, ε_b, $\varepsilon_s\}$.

ranges — A list of ranges for the different simulation parameters having named elements α, δ, ε_b, ε_s, and μ. The value of each element is a vector of two numbers: the first one is the minimal value min_v and the second one is the maximal value max_v. If the element corresponding to a given parameter is missing, the default range for that parameter is used. If the argument ranges is an empty list and parameters is NULL, the default ranges for the parameters are used. The simulation parameters are uniformly drawn from the interval (min_v, max_v) for the specified parameters. The default value is list().

... — Additional arguments passed on to the function generatedata_mpin(). The recognized arguments are confidence, maxlayers, eps_ratio, mu_ratio.

- confidence (numeric) denotes the range of the confidence interval associated with each layer such that all observations within the layer j lie in the theoretical confidence interval of the Skellam distribution centered on the mean order imbalance, at the level 'confidence'. The default value is 0.99.
- maxlayers (integer) denotes the upper limit of number of layers for the generated datasets. If the argument layers is missing, the layers of the simulated datasets will be uniformly drawn from {1,..., maxlayers}. When missing, maxlayers takes the default value of 5.
- eps_ratio (numeric) specifies the admissible range for the value of the ratio ε_s/ε_b, It can be a two-value vector or just a single value. If eps_ratio is a vector of two values: the first one is the minimal value and the second one is the maximal value; and the function tries to generate ε_s and ε_b satisfying that their ratios ε_s/ε_b lies within the interval eps_ratio. If eps_ratio is a single number, then the function tries to generate ε_s and ε_b satisfying

ε_s = ε_b x `eps_ratio`. If this range conflicts with other arguments such as `ranges`, a warning is displayed. The default value is `c(0.75, 1.25)`.

- `mu_ratio` (`numeric`) it is the minimal value of the ratio between two consecutive values of the vector `mu`. If `mu_ratio` = `1.25` e.g., then μ_{j+1} should be larger than `1.25`* μ_j for all `j = 1, .., J`. If `mu_ratio` conflicts with other arguments such as `ranges` or `confidence`, a warning is displayed. The default value is `NULL`.

`verbose` (`logical`) a binary variable that determines whether detailed information about the progress of the data generation is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

An information layer refers to a given type of information event existing in the data. The `PIN` model assumes a single type of information events characterized by three parameters for α, δ, and μ. The `MPIN` model relaxes the assumption, by relinquishing the restriction on the number of information event types. When `layers` = `1`, generated data fit the assumptions of the `PIN` model.

If the argument `parameters` is missing, then the simulation parameters are generated using the ranges specified in the argument `ranges`. If the argument `ranges` is `list()`, default ranges are used. Using the default ranges, the simulation parameters are obtained using the following procedure:

- $\alpha()$: a vector of length `layers`, where each α_j is uniformly distributed on `(0, 1)` subject to the condition:
  $\sum_j \alpha_j \leq 1$.
- $\delta()$: a vector of length `layers`, where each δ_j uniformly distributed on `(0, 1)`.
- $\mu()$: a vector of length `layers`, where each μ_j is uniformly distributed on the interval `(0.5 max(`ε_b`,` ε_s`), 5 max(`ε_b`,` ε_s`))`. The μ:s are then sorted so the excess trading increases in the information layers, subject to the condition that the ratio of two consecutive μ's should be at least `1.25`.
- ε_b: an integer drawn uniformly from the interval `(100, 10000)` with step `50`.
- ε_s: an integer uniformly drawn from `((3/4)`ε_b, `(5/4)`ε_b`)` with step `50`.

Based on the simulation parameters `parameters`, daily buys and sells are generated by the assumption that buys and sells follow Poisson distributions with mean parameters (ε_b, ε_s) on days with no information; with mean parameters (ε_b + μ_j, ε_s) on days with good information of layer j and (ε_b, ε_s + μ_j) on days with bad information of layer j.

Considerations for the ranges of simulation parameters: While `generatedata_mpin()` function enables the user to simulate data series with any set of theoretical parameters, we strongly recommend the use of parameter sets satisfying below conditions which are in line with the nature of empirical data and the theoretical models used within this package. When parameter values are not assigned by the user, the function, by default, simulates data series that are in line with these criteria.

- *Consideration 1*: any μ's value separable from ε_b and ε_s values, as well as other μ values. Otherwise, the `PIN` and `MPIN` estimation would not yield expected results.
  [x] Sharp example.1: ε_b= 1000; $\mu = 1$. In this case, no information layer can be captured in a healthy way by the use of the models which relies on Poisson distributions.

[x] Sharp example.2: ε_s= 1000, μ_1 $= 1000$, and μ_2 $= 1001$. Similarly, no distinction can be made on the two simulated layers of informed trading. In real life, this entails that there is only one type of information which would also be the estimate of the `MPIN` model. However, in the simulated data properties, there would be 2 layers which will lead the user to make a wrong evaluation of model performance.

- *Consideration 2*: ε_b and ε_s being relatively close to each other. When they are far from each other, that would indicate that there is substantial asymmetry between buyer and seller initiated trades, being a strong signal for informed trading. There is no theoretical evidence to indicate that the uninformed trading in buy and sell sides deviate much from each other in real life. Besides, numerous papers that work with `PIN` model provide close to each other uninformed intensities. when no parameter values are assigned by the user, the function generates data with the condition of sell side uninformed trading to be in the range of `(4/5):=80%` and `(6/5):=120%` of buy side uninformed rate.
  [x] Sharp example.3: ε_b= 1000, ε_s= 10000. In this case, the `PIN` and `MPIN` models would tend to consider some of the trading in sell side to be informed (which should be the actual case). Again, the estimation results would deviate much from the simulation parameters being a good news by itself but a misleading factor in model evaluation. See for example Cheng and Lai (2021) as a misinterpretation of comparative performances. The paper's findings highly rely on the simulations with extremely different ε_b and ε_s values (813-8124 pair and 8126-812).

Value

Returns an object of class dataset-class if `series=1`, and an object of class data.series-class if `series>1`.

References

Cheng T, Lai H (2021). "Improvements in estimating the probability of informed trading models." *Quantitative Finance*, **21**(5), 771-796.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Examples

```
# ------------------------------------------------------------------------ #
# There are different scenarios of using the function generatedata_mpin()  #
# ------------------------------------------------------------------------ #

# With no arguments, the function generates one dataset object spanning
# 60 days, containing a number of information layers uniformly selected
# from `{1, 2, 3, 4, 5}`, and where the parameters are chosen as
# described in the details.

sdata <- generatedata_mpin()

# The number of layers can be deduced from the simulation parameters, if
# fed directly to the function generatedata_mpin() through the argument
# 'parameters'. In this case, the output is a dataset object with one
# information layer.
```

```
givenpoint <- c(0.4, 0.1, 800, 300, 200)
sdata <- generatedata_mpin(parameters = givenpoint)

# The number of layers can alternatively be set directly through the
# argument 'layers'.

sdata <- generatedata_mpin(layers = 2)

# The simulation parameters can be randomly drawn from their corresponding
# ranges fed through the argument 'ranges'.

sdata <- generatedata_mpin(ranges = list(alpha = c(0.1, 0.7),
                                        delta = c(0.2, 0.7),
                                        mu = c(3000, 5000)))

# The value of a given simulation parameter can be set to a specific value by
# setting the range of the desired parameter takes a unique value, instead of
# a pair of values.

sdata <- generatedata_mpin(ranges = list(alpha = 0.4, delta = c(0.2, 0.7),
                                        eps.b = c(100, 7000),
                                        mu = c(8000, 12000)))

# If both arguments 'parameters', and 'layers' are simultaneously provided,
# and the number of layers detected from the length of the argument
# 'parameters' is different from the argument 'layers', the former is used
# and a warning is displayed.

sim.params <- c(0.4, 0.2, 0.9, 0.1, 400, 700, 300, 200)
sdata <- generatedata_mpin(days = 120, layers = 3, parameters = sim.params)

# Display the details of the generated data

show(sdata)

# ------------------------------------------------------------------------ #
# Use generatedata_mpin() to compare the accuracy of estimation methods    #
# ------------------------------------------------------------------------ #

# The example below illustrates the use of the function 'generatedata_mpin()'
# to compare the accuracy of the functions 'mpin_ml()', and 'mpin_ecm()'.

# The example will depend on three variables:
# n: the number of datasets used
# l: the number of layers in each simulated datasets
# xc : the number of extra clusters used in initials_mpin

# For consideration of speed, we will set n = 2, l = 2, and xc = 2
# These numbers can change to fit the user's preferences
n <- l <- xc <- 2

# We start by generating n datasets simulated according to the
# assumptions of the MPIN model.
```

```
dataseries <- generatedata_mpin(series = n, layers = l, verbose = FALSE)

# Store the estimates in two different lists: 'mllist', and 'ecmlist'

mllist <- lapply(dataseries@datasets, function(x)
  mpin_ml(x@data, xtraclusters = xc, layers = l, verbose = FALSE))

ecmlist <- lapply(dataseries@datasets, function(x)
  mpin_ecm(x@data, xtraclusters = xc, layers = l, verbose = FALSE))

# For each estimate, we calculate the absolute difference between the
# estimated mpin, and empirical mpin computed using dataset parameters.
# The absolute differences are stored in 'mldmpin' ('ecmdpin') for the
# ML (ECM) method,

mldpin <- sapply(1:n,
 function(x) abs(mllist[[x]]@mpin - dataseries@datasets[[x]]@emp.pin))

ecmdpin <- sapply(1:n,
 function(x) abs(ecmlist[[x]]@mpin - dataseries@datasets[[x]]@emp.pin))

# Similarly, we obtain vectors of running times for both estimation methods.
# They are stored in 'mltime' ('ecmtime') for the ML (ECM) method.

mltime <- sapply(mllist, function(x) x@runningtime)
ecmtime <- sapply(ecmlist, function(x) x@runningtime)

# Finally, we calculate the average absolute deviation from empirical PIN
# as well as the average running time for both methods. This allows us to
# compare them in terms of accuracy, and speed.

accuracy <- c(mean(mldpin), mean(ecmdpin))
timing <- c(mean(mltime), mean(ecmtime))
comparison <- as.data.frame(rbind(accuracy, timing))
colnames(comparison) <- c("ML", "ECM")
rownames(comparison) <- c("Accuracy", "Timing")

show(round(comparison, 6))
```

get_posteriors *Posterior probabilities for PIN and MPIN estimates*

Description

Computes, for each day in the sample, the posterior probability that the day is a no-information day, good-information day and bad-information day, respectively (Easley and Ohara (1992), Easley et al. (1996), Ersan (2016)).

Usage

```
get_posteriors(object)
```

Arguments

object (S4 object) an object of type estimate.pin, estimate.mpin, or estimate.mpin.ecm.

Value

If the argument object is of type estimate.pin, returns a dataframe of three variables post.N, post.G and post.B containing in each row the posterior probability that a given day is a no-information day (N), good-information day (G), or bad-information day (B) respectively.

If the argument object is of type estimate.mpin or estimate.mpin.ecm, with J layers, returns a dataframe of 2*J+1 variables Post.N, and Post.G[j] and Post.B[j] for each layer j containing in each row the posterior probability that a given day is a no-information day, good-information day in layer j or bad-information day in layer j, for each layer j respectively.

If the argument object is of any other type, an error is returned.

References

Easley D, Kiefer NM, Ohara M, Paperman JB (1996). "Liquidity, information, and infrequently traded stocks." *Journal of Finance*, **51**(4), 1405–1436. ISSN 00221082.

Easley D, Ohara M (1992). "Time and the Process of Security Price Adjustment." *The Journal of Finance*, **47**(2), 577–605. ISSN 15406261.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# ------------------------------------------------------------------------ #
# Posterior probabilities for PIN estimates                                #
# ------------------------------------------------------------------------ #

# Estimate PIN using the Ersan and Alici (2016) algorithm and the
# factorization Lin and Ke(2011).

estimate <- pin_ea(xdata, "LK", verbose = FALSE)

# Display the estimated PIN value

estimate@pin
```

```
# Store the posterior probabilities in a dataframe variable and display its
# first 6 rows.

modelposteriors <- get_posteriors(estimate)
show(round(head(modelposteriors), 3))

# ------------------------------------------------------------------------ #
# Posterior probabilities for MPIN estimates                               #
# ------------------------------------------------------------------------ #

# Estimate MPIN via the ECM algorithm, assuming that the dataset has 2
# information layers

estimate <- mpin_ecm(xdata, layers = 2, verbose = FALSE)

# Display the estimated Multilayer PIN value

show(estimate@mpin)

# Store the posterior probabilities in a dataframe variable and display its
# first six rows. The posterior probabilities are contained in a dataframe
# with 7 variables: one for no-information days, and two variables for each
# layer, one for good-information days and one for bad-information days.

modelposteriors <- get_posteriors(estimate)
show(round(head(modelposteriors), 3))
```

hfdata *High-frequency trade-data*

Description

A simulated dataset containing sample `timestamp`, `price`, `volume`, `bid` and `ask` for `100 000` high frequency transactions.

Usage

```
hfdata
```

Format

A data frame with `100 000` observations with 5 variables:

- `timestamp`: time of the trade.
- `price`: transaction price.
- `volume`: volume of the transactions, in asset units.
- `bid`: best bid price.
- `ask`: best ask price.

Source

Artificially created data set.

---

initials_adjpin *AdjPIN initial parameter sets of Ersan & Ghachem (2022b)*

Description

Based on the algorithm in Ersan and Ghachem (2025), generates sets of initial parameters to be used in the maximum likelihood estimation of `AdjPIN` model.

Usage

```
initials_adjpin(data, xtraclusters = 4, restricted = list(),
 verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`xtraclusters` An integer used to divide trading days into `#(4 + xtraclusters)` clusters, thereby resulting in `#comb(4 + xtraclusters - 1, 4 - 1)` initial parameter sets in line with Ersan and Alici (2016), and Ersan and Ghachem (2025).The default value is 4 as chosen in Ersan (2016).

`restricted` A binary list that allows estimating restricted AdjPIN models by specifying which model parameters are assumed to be equal. It contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the probability of liquidity shock in no-information days, and in information days is assumed to be the same (θ=θ'). If any of the remaining rate elements `{mu, eps, d}` is set to `TRUE`, (say `mu=TRUE`), then the rate is assumed to be the same on the buy side, and on the sell side (μ_b=μ_s). If more than one element is set to `TRUE`, then the restrictions are combined. For instance, if the argument `restricted` is set to `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the restricted AdjPIN model is estimated, where θ=θ', ε_b=ε_s, and Δ_b=Δ_s. If the value of the argument `restricted` is the empty list, then all parameters of the model are assumed to be independent, and the unrestricted model is estimated. The default value is the empty list `list()`.

`verbose` a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The function `initials_adjpin()` implements the algorithm suggested in Ersan and Ghachem (2025), and uses a hierarchical agglomerative clustering (HAC) to find initial parameter sets for the maximum likelihood estimation.

Value

Returns a dataframe of numerical vectors of ten elements $\{\alpha, \delta, \theta, \theta', \varepsilon_b, \varepsilon_s, \mu_b, \mu_s, \Delta_b, \Delta_s\}$.

References

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Ersan O, Ghachem M (2025). "A methodological approach to the computational problems in the estimation of adjusted PIN model." *Quantitative Finance*, 1–13.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Obtain a dataframe of initial parameter sets for the maximum likelihood
# estimation using the algorithm of Ersan and Ghachem (2022b).

init.sets <- initials_adjpin(xdata)

# Use the list to estimate adjpin using the adjpin() method
# Show the value of adjusted PIN

estimate <- adjpin(xdata, initialsets = init.sets, verbose = FALSE)
show(estimate@adjpin)
```

initials_adjpin_cl *AdjPIN initial parameter sets of Cheng and Lai (2021)*

Description

Based on an extension of the algorithm in Cheng and Lai (2021), generates sets of initial parameters to be used in the maximum likelihood estimation of `AdjPIN` model.

Usage

```
initials_adjpin_cl(data, restricted = list(), verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`restricted` A binary list that allows estimating restricted AdjPIN models by specifying which model parameters are assumed to be equal. It contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the probability of liquidity shock in no-information days, and in information days is assumed to be the same (θ=θ'). If any of the remaining rate elements `{mu, eps, d}` is set to `TRUE`, (say `mu=TRUE`), then the rate is assumed to be the same on the buy side, and on the sell side (μ_b=μ_s). If more than one element is set to `TRUE`, then the restrictions are combined. For instance, if the argument `restricted` is set to `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the restricted AdjPIN model is estimated, where θ=θ', ε_b=ε_s, and Δ_b=Δ_s. If the value of the argument `restricted` is the empty list, then all parameters of the model are assumed to be independent, and the unrestricted model is estimated. The default value is the empty list `list()`.

`verbose` a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The function implements an extension of the algorithm of Cheng and Lai (2021). In their paper, the authors assume that the probability of liquidity shock is the same in no-information, and information days, i.e., θ=θ', and use a procedure similar to that of Yan and Zhang (2012) to generate 64 initial parameter sets. The function implements an extension of their algorithm, by relaxing the assumption of equality of liquidity shock probabilities, and generates thereby `256` initial parameter sets for the unrestricted `AdjPIN` model.

Value

Returns a dataframe of numerical vectors of ten elements $\{\alpha, \delta, \theta, \theta', \varepsilon_b, \varepsilon_s, \mu_b, \mu_s, \Delta_b, \Delta_s\}$.

References

Cheng T, Lai H (2021). “Improvements in estimating the probability of informed trading models.” *Quantitative Finance*, **21**(5), 771-796.

Yan Y, Zhang S (2012). “An improved estimation method and empirical properties of the probability of informed trading.” *Journal of Banking and Finance*, **36**(2), 454–467. ISSN 03784266.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# The function adjpin(xdata, initialsets="CL") allows the user to directly
# estimate the AdjPIN model using the full set of initial parameter sets
# generated using the algorithm Cheng and Lai (2021)

estimate.1 <- adjpin(xdata,  initialsets="CL", verbose = FALSE)


# Obtaining the set of initial parameter sets using initials_adjpin_cl
# allows us to estimate the PIN model using a subset of these initial sets.

# Use initials_adjpin_cl() to generate 256 initial parameter sets using the
# algorithm of Cheng and Lai (2021).

initials_cl <- initials_adjpin_cl(xdata, verbose = FALSE)

# Use 20 randonly chosen initial sets from the dataframe 'initials_cl' in
# order to estimate the AdjPIN model using the function adjpin() with custom
# initial parameter sets

numberofsets <- nrow(initials_cl)
selectedsets <- initials_cl[sample(numberofsets, 20),]

estimate.2 <- adjpin(xdata, initialsets = selectedsets, verbose = FALSE)

# Compare the parameters and the pin values of both specifications

comparison <- rbind(
c(estimate.1@parameters, adjpin = estimate.1@adjpin, psos = estimate.1@psos),
c(estimate.2@parameters, estimate.2@adjpin, estimate.2@psos))

rownames(comparison) <- c("all", "50")
```

```
show(comparison)
```

`initials_adjpin_rnd` *AdjPIN random initial sets*

Description

Generates random initial parameter sets to be used in the estimation of the `AdjPIN` model of Duarte and Young (2009).

Usage

```
initials_adjpin_rnd(data, restricted = list(), num_init = 20,
 verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`restricted` A binary list that allows estimating restricted AdjPIN models by specifying which model parameters are assumed to be equal. It contains one or multiple of the following four elements `{theta, mu, eps, d}`. For instance, If `theta` is set to `TRUE`, then the probability of liquidity shock in no-information days, and in information days is assumed to be the same (θ=θ'). If any of the remaining rate elements `{mu, eps, d}` is set to `TRUE`, (say `mu=TRUE`), then the rate is assumed to be the same on the buy side, and on the sell side (μ_b=μ_s). If more than one element is set to `TRUE`, then the restrictions are combined. For instance, if the argument `restricted` is set to `list(theta=TRUE, eps=TRUE, d=TRUE)`, then the restricted AdjPIN model is estimated, where θ=θ', ε_b=ε_s, and Δ_b=Δ_s. If the value of the argument `restricted` is the empty list (`list()`), then all parameters of the model are assumed to be independent, and the unrestricted model is estimated. The default value is the empty list `list()`.

`num_init` An integer corresponds to the number of initial parameter sets to be generated. The default value is `20`.

`verbose` a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The buy rate parameters $\{\varepsilon_b,\ \mu_b,\ \Delta_b\}$ are randomly generated from the interval (`minB`, `maxB`), where `minB` (`maxB`) is the smallest (largest) value of buys in the dataset, under the condition that ε_b+μ_b+Δ_b< `maxB`. Analogously, the sell rate parameters $\{\varepsilon_s,\ \mu_s,\ \Delta_s\}$ are randomly generated from the interval (`minS`, `maxS`), where `minS` (`maxS`) is the smallest(largest) value of sells in the dataset, under the condition that ε_s+μ_s+Δ_s < `maxS`.

Value

Returns a dataframe of numerical vectors of ten elements $\{\alpha, \delta, \theta, \theta', \varepsilon_b, \varepsilon_s, \mu_b, \mu_s, \Delta_b, \Delta_s\}$.

References

Duarte J, Young L (2009). "Why is PIN priced?" *Journal of Financial Economics*, **91**(2), 119–138. ISSN 0304405X.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Obtain a dataframe of 20 random initial parameters for the MLE of
# the AdjPIN model using the initials_adjpin_rnd().

initial.sets <- initials_adjpin_rnd(xdata, num_init = 20)

# Use the dataframe to estimate the AdjPIN model using the adjpin()
# function.

estimate <- adjpin(xdata, initialsets = initial.sets, verbose = FALSE)

# Show the value of adjusted PIN

show(estimate@adjpin)
```

`initials_mpin` *MPIN initial parameter sets of Ersan (2016)*

Description

Based on the algorithm in Ersan (2016), generates initial parameter sets for the maximum likelihood estimation of the `MPIN` model.

Usage

```
initials_mpin(data, layers = NULL, detectlayers = "EG",
 xtraclusters = 4, verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`layers` An integer referring to the assumed number of information layers in the data. If the value of `layers` is `NULL`, then the number of layers is automatically determined by one of the following functions: `detectlayers_e()`, `detectlayers_eg()`, and `detectlayers_ecm()`. The default value is `NULL`.

`detectlayers` A character string referring to the layer detection algorithm used to determine the number of layers in the data. It takes one of three values: `"E"`, `"EG"`, and `"ECM"`. `"E"` refers to the algorithm in Ersan (2016), `"EG"` refers to the algorithm in Ersan and Ghachem (2024); while `"ECM"` refers to the algorithm in Ghachem and Ersan (2025). The default value is `"EG"`. Comparative results between the layer detection algorithms can be found in Ersan and Ghachem (2024).

`xtraclusters` An integer used to divide trading days into `#(1 + layers + xtraclusters)` clusters, thereby resulting in `#comb(layers + xtraclusters, layers)` initial parameter sets in line with Ersan and Alici (2016), and Ersan (2016). The default value is 4 as chosen in Ersan (2016).

`verbose` a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

Value

Returns a dataframe of initial parameter sets each consisting of `3J + 2` variables $\{\alpha, \delta, \mu, \varepsilon_b, \varepsilon_s\}$. α, δ, and μ are vectors of length `J` where `J` is the number of layers in the `MPIN` model.

References

Ersan O (2016). “Multilayer Probability of Informed Trading.” *Available at SSRN 2874420.*

Ersan O, Alici A (2016). “An unbiased computation methodology for estimating the probability of informed trading (PIN).” *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Ersan O, Ghachem M (2024). “Identifying information types in the estimation of informed trading:

an improved algorithm." *Journal of Risk and Financial Management*, **17**(9), 409.

Ghachem M, Ersan O (2025). "Estimation of the probability of informed trading models via an expectation-conditional maximization algorithm." *Financial Innovation*, **11**(1), 67.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Obtain a dataframe of initial parameter sets for estimation of the MPIN
# model using the algorithm of Ersan (2016) with 3 extra clusters.
# By default, the number of layers in the data is detected using the
# algorithm of Ersan and Ghachem (2022a).

initparams <- initials_mpin(xdata, xtraclusters = 3, verbose = FALSE)

# Show the six first initial parameter sets

print(round(t(head(initparams)), 3))

# Use 10 randomly selected initial parameter sets from initparams to
# estimate the probability of informed trading via mpin_ecm. The number
# of information layers will be detected from the initial parameter sets.

numberofsets <- nrow(initparams)
selectedsets <- initparams[sample(numberofsets, 10),]

estimate <- mpin_ecm(xdata, initialsets = selectedsets, verbose = FALSE)

# Display the estimated MPIN value

show(estimate@mpin)

# Display the estimated parameters as a numeric vector.

show(unlist(estimate@parameters))

# Store the posterior probabilities in a variable, and show the first 6 rows.

modelposteriors <- get_posteriors(estimate)
show(round(head(modelposteriors), 3))
```

initials_pin_ea *Initial parameter sets of Ersan & Alici (2016)*

Description

Based on the algorithm in Ersan and Alici (2016), generates initial parameter sets for the maximum likelihood estimation of the PIN model.

Usage

```
initials_pin_ea(data, xtraclusters = 4, verbose = TRUE)
```

Arguments

data	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
xtraclusters	An integer used to divide trading days into #(2 + xtraclusters) clusters, thereby resulting in #comb(1 + xtraclusters, 1) initial parameter sets in line with Ersan and Alici (2016). The default value is 4.
verbose	a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. NA values will be ignored.

The function initials_pin_ea() uses a hierarchical agglomerative clustering (HAC) to find initial parameter sets for the maximum likelihood estimation. The steps in Ersan and Alici (2016) algorithm differ from those used by Gan et al. (2015), and are summarized below.

Via the use of HAC, daily absolute order imbalances (AOIs) are grouped in 2+J (default J=4) clusters. After sorting the clusters based on AOIs, they are combined into two larger groups of days (event and no-event) by merging neighboring clusters with each other. Consequently, those groups are formed in #comb(5, 1) = 5 different ways. For each of the 5 configurations with which, days are grouped into two (event group and no-event group), the procedure below is applied to obtain initial parameter sets.

Days in the event group (the one with larger mean AOI) are distributed into two groups, i.e. good-event days (days with positive OI) and bad-event days (days with negative OI). Initial parameters are obtained from the frequencies, and average trade rates of three types of days. See Ersan and Alici (2016) for further details.

The higher the number of the additional clusters (xtraclusters), the better is the estimation. Ersan and Alici (2016), however, have shown the benefit of increasing this number beyond 4 is marginal, and statistically insignificant.

Value

Returns a dataframe of initial sets each consisting of five variables $\{\alpha, \delta, \mu, \varepsilon_b, \varepsilon_s\}$.

References

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Gan Q, Wei WC, Johnstone D (2015). "A faster estimation method for the probability of informed trading using hierarchical agglomerative clustering." *Quantitative Finance*, **15**(11), 1805–1821.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Obtain a dataframe of initial parameters for the maximum likelihood
# estimation using the algorithm of Ersan and Alici (2016).

init.sets <- initials_pin_ea(xdata)

# Use the obtained dataframe to estimate the PIN model using the function
# pin() with custom initial parameter sets

estimate.1 <- pin(xdata, initialsets = init.sets, verbose = FALSE)

# pin_ea() directly estimates the PIN model using initial parameter sets
# generated using the algorithm of Ersan & Alici (2016).

estimate.2 <- pin_ea(xdata, verbose = FALSE)

# Check that the obtained results are identical

show(estimate.1@parameters)
show(estimate.2@parameters)
```

initials_pin_gwj *Initial parameter set of Gan et al.(2015)*

Description

Based on the algorithm in Gan et al. (2015), generates an initial parameter set for the maximum likelihood estimation of the `PIN` model.

Usage

```
initials_pin_gwj(data, verbose = TRUE)
```

Arguments

`data`	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
`verbose`	a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

Value

Returns a dataframe containing numerical vector of five elements $\{\alpha, \delta, \mu, \varepsilon_b, \varepsilon_s\}$.

References

Gan Q, Wei WC, Johnstone D (2015). “A faster estimation method for the probability of informed trading using hierarchical agglomerative clustering.” *Quantitative Finance*, **15**(11), 1805–1821.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Obtain the initial parameter set for the maximum likelihood estimation
# using the algorithm of Gan et al.(2015).

initparams <- initials_pin_gwj(xdata)

# Use the obtained dataframe to estimate the PIN model using the function
# pin() with custom initial parameter sets

estimate.1 <- pin(xdata, initialsets = initparams, verbose = FALSE)

# pin_gwj() directly estimates the PIN model using an initial parameter set
# generated using the algorithm of Gan et al.(2015).

estimate.2 <- pin_gwj(xdata, "E", verbose = FALSE)

# Check that the obtained results are identical

show(estimate.1@parameters)
show(estimate.2@parameters)
```

initials_pin_yz *Initial parameter sets of Yan and Zhang (2012)*

Description

Based on the grid search algorithm of Yan and Zhang (2012), generates initial parameter sets for the maximum likelihood estimation of the PIN model.

Usage

```
initials_pin_yz(data, grid_size = 5, ea_correction = FALSE,
 verbose = TRUE)
```

Arguments

data	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
grid_size	An integer between 1, and 20; representing the size of the grid. The default value is 5. See more in details.
ea_correction	A binary variable determining whether the modifications of the algorithm of Yan and Zhang (2012) suggested by Ersan and Alici (2016) are implemented. The default value is FALSE.
verbose	a binary variable that determines whether information messages about the initial parameter sets, including the number of the initial parameter sets generated. No message is shown when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. NA values will be ignored.

The argument grid_size determines the size of the grid of the variables: alpha, delta, and eps.b. If grid_size is set to a given value m, the algorithm creates a sequence starting from 1/2m, and ending in 1 - 1/2m, with a step of 1/m. The default value of 5 corresponds to the size of the grid in Yan and Zhang (2012). In that case, the sequence starts at 0.1 = 1/(2 x 5), and ends in 0.9 = 1 - 1/(2 x 5) with a step of 0.2 = 1/m.

The function initials_pin_yz() implements, by default, the original Yan and Zhang (2012) algorithm as the default value of ea_correction takes the value FALSE. When the value of ea_correction is set to TRUE; then, sets with irrelevant mu values are excluded, and sets with boundary values are reintegrated in the initial parameter sets.

Value

Returns a dataframe of initial sets each consisting of five variables $\{\alpha, \delta, \mu, \varepsilon_b, \varepsilon_s\}$.

References

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Yan Y, Zhang S (2012). "An improved estimation method and empirical properties of the probability of informed trading." *Journal of Banking and Finance*, **36**(2), 454–467. ISSN 03784266.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# The function pin_yz() allows the user to directly estimate the PIN model
# using the full set of initial parameter sets generated using the algorithm
# of Yan and # Zhang (2012).

estimate.1 <- pin_yz(xdata, verbose = FALSE)

# Obtaining the set of initial parameter sets using initials_pin_yz allows
# us to estimate the PIN model using a subset of these initial sets.

initparams <- initials_pin_yz(xdata, verbose = FALSE)

# Use 10 randonly chosen initial sets from the dataframe 'initparams' in
# order to estimate the PIN model using the function pin() with custom
# initial parameter sets

numberofsets <- nrow(initparams)
selectedsets <- initparams[sample(numberofsets, 10),]

estimate.2 <- pin(xdata, initialsets = selectedsets, verbose = FALSE)

# Compare the parameters and the pin values of both specifications

comparison <- rbind(c(estimate.1@parameters, pin = estimate.1@pin),
                    c(estimate.2@parameters, estimate.2@pin))

rownames(comparison) <- c("all", "10")

show(comparison)
```

mpin_ecm *MPIN model estimation via an ECM algorithm*

Description

Estimates the multilayer probability of informed trading (MPIN) using an Expectation Conditional Maximization algorithm, as in Ghachem and Ersan (2025).

Usage

```
mpin_ecm(data, layers = NULL, xtraclusters = 4, initialsets = NULL,
                            ..., verbose = TRUE)
```

Arguments

data A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

layers An integer referring to the assumed number of information layers in the data. If the argument layers is given, then the ECM algorithm will use the number of layers provided. If layers is omitted, the function mpin_ecm() will simultaneously optimize the number of layers as well as the parameters of the MPIN model.

xtraclusters An integer used to divide trading days into #(1 + layers + xtraclusters) clusters, thereby resulting in #comb((layers + xtraclusters, layers) initial parameter sets in line with Ersan and Alici (2016), and Ersan (2016). The default value is 4 as chosen in Ersan (2016).

initialsets A dataframe containing initial parameter sets for estimation of the MPIN model. The default value is NULL. If initialsets is NULL, the initial parameter sets are provided by the function initials_mpin().

... Additional arguments passed on to the function mpin_ecm. The recognized arguments are hyperparams, and is_parallel.

- hyperparams is a list containing the hyperparameters of the ECM algorithm. When not empty, it contains one or more of the following elements: minalpha, maxeval, tolerance, criterion, and maxlayers. More about these elements are in the details section.
- is_parallel is a logical variable that specifies whether the computation is performed using parallel or sequential processing. The default value is FALSE. For more details, please refer to the vignette 'Parallel processing' in the package, or online.

verbose (logical) a binary variable that determines whether detailed information about the steps of the estimation of the MPIN model is displayed. No output is produced when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The initial parameters for the expectation-conditional maximization algorithm are computed using the function `initials_mpin()` with default settings. The factorization of the `MPIN` likelihood function used is developed by Ersan (2016), and is implemented in `fact_mpin()`.

The argument `hyperparams` contains the hyperparameters of the ECM algorithm. It is either empty or contains one or more of the following elements:

- `minalpha` (`numeric`) It stands for the minimum share of days belonging to a given layer, i.e., layers falling below this threshold are removed during the iteration, and the model is estimated with a lower number of layers. When missing, `minalpha` takes the default value of `0.001`.
- `maxeval`: (`integer`) It stands for maximum number of iterations of the ECM algorithm for each initial parameter set. When missing, `maxeval` takes the default value of `100`.
- `tolerance` (`numeric`) The ECM algorithm is stopped when the (relative) change of log-likelihood is smaller than tolerance. When missing, `tolerance` takes the default value of `0.001`.
- `criterion` (`character`) It is the model selection criterion used to find the optimal estimate for the `MPIN` model. It take one of these values `"BIC"`, `"AIC"` and `"AWE"`; which stand for Bayesian Information Criterion, Akaike Information Criterion and Approximate Weight of Evidence, respectively (Akogul and Erisoglu 2016). When missing, `criterion` takes the default value of `"BIC"`.
- `maxlayers` (`integer`) It is the upper limit of number of layers used for estimation in the ECM algorithm. If the argument `layers` is missing, the ECM algorithm will estimate `MPIN` models for all layers in the integer set from `1` to `maxlayers`. When missing, `maxlayers` takes the default value of `8`.
- `maxinit` (`integer`) It is the maximum number of initial sets used for each individual estimation in the ECM algorithm. When missing, `maxinit` takes the default value of `100`.

If the argument `layers` is given, then the Expectation Conditional Maximization algorithm will use the number of layers provided. If `layers` is omitted, the function `mpin_ecm()` will simultaneously optimize the number of layers as well as the parameters of the `MPIN` model. Practically, the function `mpin_ecm()` uses the ECM algorithm to optimize the `MPIN` model parameters for each number of layers within the integer set from `1` to `8` (or to `maxlayers` if specified in the argument `hyperparams`); and returns the optimal model with the lowest Bayesian information criterion (BIC) (or the lowest information criterion `criterion` if specified in the argument `hyperparams`).

Value

Returns an object of class estimate.mpin.ecm-class.

References

Akogul S, Erisoglu M (2016). “A comparison of information criteria in clustering based on mixture of multivariate normal distributions.” *Mathematical and Computational Applications*, **21**(3), 34.

Ersan O (2016). “Multilayer Probability of Informed Trading.” *Available at SSRN 2874420*.

Ersan O, Alici A (2016). “An unbiased computation methodology for estimating the probability of informed trading (PIN).” *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Ghachem M, Ersan O (2025). “Estimation of the probability of informed trading models via an expectation-conditional maximization algorithm.” *Financial Innovation*, **11**(1), 67.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Estimate the MPIN model using the expectation-conditional maximization
# (ECM) algorithm.

# ------------------------------------------------------------------------ #
# Estimate the MPIN model, assuming that there exists 2 information layers #
# in the dataset                                                           #
# ------------------------------------------------------------------------ #

estimate <- mpin_ecm(xdata, layers = 2, verbose = FALSE)

# Show the estimation output

show(estimate)

# Display the optimal parameters from the Expectation Conditional
# Maximization algorithm

show(estimate@parameters)

# Display the global multilayer probability of informed trading

show(estimate@mpin)

# Display the multilayer probability of informed trading per layer

show(estimate@mpinJ)

# Display the first five rows of the initial parameter sets used in the
# expectation-conditional maximization estimation
```

```
show(round(head(estimate@initialsets, 5), 4))

# ------------------------------------------------------------------------ #
# Omit the argument 'layers', so the ECM algorithm optimizes both the      #
# number of layers and the MPIN model parameters.                          #
# ------------------------------------------------------------------------ #

estimate <- mpin_ecm(xdata, verbose = FALSE)

# Show the estimation output

show(estimate)

# Display the optimal parameters from the estimation of the MPIN model using
# the expectation-conditional maximization (ECM) algorithm

show(estimate@parameters)

# Display the multilayer probability of informed trading

show(estimate@mpin)

# Display the multilayer probability of informed trading per layer

show(estimate@mpinJ)

# Display the first five rows of the initial parameter sets used in the
# expectation-conditional maximization estimation.

show(round(head(estimate@initialsets, 5), 4))

# ------------------------------------------------------------------------ #
# Tweak in the hyperparameters of the ECM algorithm                        #
# ------------------------------------------------------------------------ #

# Create a variable ecm.params containing the hyperparameters of the ECM
# algorithm. This will surely make the ECM algorithm take more time to give
# results

ecm.params <- list(tolerance = 0.0000001)

# If we suspect that the data contains more than eight information layers, we
# can raise the number of models to be estimated to 10 as an example, i.e.,
# maxlayers = 10.

ecm.params$maxlayers <- 10

# We can also choose Approximate Weight of Evidence (AWE) for model
# selection instead of the default Bayesian Information Criterion (BIC)

ecm.params$criterion <- 'AWE'
```

```
# We can also increase the maximum number of initial sets to 200, in
# order to obtain higher level of accuracy for models with high number of
# layers.  We set the sub-argument 'maxinit' to `200`. Remember that its
# default value is `100`.

ecm.params$maxinit <- 200

estimate <- mpin_ecm(xdata, xtraclusters = 2, hyperparams = ecm.params,
                                                      verbose = FALSE)

# We can change the model selection criterion by calling selectModel()

estimate <- selectModel(estimate, ”AIC”)

# We get the mpin_ecm estimation results for the MPIN model with 2 layers
# using the slot models. We then show the first five rows of the
# corresponding slot details.

models <- estimate@models
show(round(head(models[[2]]@details, 5), 4))

# We can also use the function getSummary to get an idea about the change in
# the estimation parameters as a function of the number of layers in the
# MPIN model. The function getSummary returns a dataframe that contains,
# among others, the number of layers of the model, the number of layers in
# the optimal model,the MPIN value, and the values of the different
# information criteria, namely AIC, BIC and AWE.

summary <- getSummary(estimate)

# We can plot the MPIN value and the layers at the optimal model as a
# function of the number of layers to see whether additional layers in the
# model actually contribute to a better precision in the probability of
# informed trading. Remember that the hyperparameter 'minalpha' is
# responsible for dropping layers with ”frequency” lower than 'minalpha'.

plot(summary$layers, summary$MPIN,
   type = ”o”, col = ”red”,
   xlab = ”MPIN model layers”, ylab = ”MPIN value”
 )

plot(summary$layers, summary$em.layers,
   type = ”o”, col = ”blue”,
   xlab = ”MPIN model layers”, ylab = ”layers at the optimal model”
)
```

mpin_ml *MPIN model estimation via standard ML methods*

Description

Estimates the multilayer probability of informed trading (`MPIN`) using the standard Maximum Likelihood method.

Usage

```
mpin_ml(data, layers = NULL, xtraclusters = 4, initialsets = NULL,
detectlayers = "EG", ..., verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`layers` An integer referring to the assumed number of information layers in the data. If the argument `layers` is given, then the maximum likelihood estimation will use the number of layers provided. If `layers` is omitted, the function `mpin_ml()` will find the optimal number of layers using the algorithm developed in Ersan and Ghachem (2024) (as default).

`xtraclusters` An integer used to divide trading days into `(1 + layers + xtraclusters)` clusters, thereby resulting in `#comb(layers + xtraclusters, layers)` initial parameter sets in line with Ersan and Alici (2016), and Ersan (2016). The default value is `4` as chosen in Ersan (2016).

`initialsets` A dataframe containing initial parameter sets for the estimation of the `MPIN` model. The default value is `NULL`. If `initialsets` is `NULL`, the initial parameter sets are determined by the function `initials_mpin()`.

`detectlayers` A character string referring to the layer detection algorithm used to determine the number of layer in the data. It takes one of three values: `"E"`, `"EG"`, and `"ECM"`. `"E"` refers to the algorithm in Ersan (2016), `"EG"` refers to the algorithm in Ersan and Ghachem (2024); while `"ECM"` refers to the algorithm in Ghachem and Ersan (2025). The default value is `"EG"`. Comparative results between the layer detection algorithms can be found in Ersan and Ghachem (2024).

`...` Additional arguments passed on to the function `mpin_ml`. The recognized argument is `is_parallel`. `is_parallel` is a logical variable that specifies whether the computation is performed using parallel processing. The default value is `FALSE`.

`verbose` A binary variable that determines whether detailed information about the steps of the estimation of the MPIN model is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

Value

Returns an object of class estimate.mpin-class

References

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Ersan O, Ghachem M (2024). "Identifying information types in the estimation of informed trading: an improved algorithm." *Journal of Risk and Financial Management*, **17**(9), 409.

Ghachem M, Ersan O (2025). "Estimation of the probability of informed trading models via an expectation-conditional maximization algorithm." *Financial Innovation*, **11**(1), 67.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# ------------------------------------------------------------------------ #
# Estimate MPIN model using the standard ML method                         #
# ------------------------------------------------------------------------ #

# Estimate the MPIN model using mpin_ml() assuming that there is a single
# information layer in the data. The model is then equivalent to the PIN
# model. The argument 'layers' takes the value '1'.
# We use two extra clusters to generate the initial parameter sets.

estimate <- mpin_ml(xdata, layers = 1, xtraclusters = 2, verbose = FALSE)

# Show the estimation output

show(estimate)

# Estimate the MPIN model using the function mpin_ml(), without specifying
# the number of layers. The number of layers is then detected using Ersan and
# Ghachem (2022a).
# --------------------------------------------------------------

estimate <- mpin_ml(xdata, xtraclusters = 2, verbose = FALSE)

# Show the estimation output

show(estimate)
```

```
# Display the likelihood-maximizing parameters

show(estimate@parameters)

# Display the global multilayer probability of informed trading

show(estimate@mpin)

# Display the multilayer probabilities of informed trading per layer

show(estimate@mpinJ)

# Display the first five initial parameters sets used in the maximum
# likelihood estimation

show(round(head(estimate@initialsets, 5), 4))
```

---

pin *PIN estimation - custom initial parameter sets*

Description

Estimates the Probability of Informed Trading (`PIN`) using custom initial parameter sets

Usage

```
pin(data, initialsets, factorization = "E", verbose = TRUE)
```

Arguments

`data`	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
`initialsets`	A dataframe with the following variables in this order (α, δ, μ, ε_b, ε_s).
`factorization`	A character string from {`"EHO"`, `"LK"`, `"E"`, `"NONE"`} referring to a given factorization. The default value is set to `"E"`.
`verbose`	A binary variable that determines whether detailed information about the steps of the estimation of the PIN model is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The factorization variable takes one of four values:

- "EHO" refers to the factorization in Easley et al. (2010)
- "LK" refers to the factorization in Lin and Ke (2011)
- "E" refers to the factorization in Ersan (2016)
- "NONE" refers to the original likelihood function - with no factorization

Value

Returns an object of class estimate.pin-class

References

Easley D, Hvidkjaer S, Ohara M (2010). “Factoring information into returns.” *Journal of Financial and Quantitative Analysis*, **45**(2), 293–309. ISSN 00221090.

Ersan O (2016). “Multilayer Probability of Informed Trading.” *Available at SSRN 2874420*.

Lin H, Ke W (2011). “A computing bias in estimating the probability of informed trading.” *Journal of Financial Markets*, **14**(4), 625-640. ISSN 1386-4181.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

#----------------------------------------------------------------
# Using generic function pin()
#----------------------------------------------------------------

# Define initial parameters:
# initialset = (alpha, delta, mu, eps.b, eps.s)

initialset <- c(0.3, 0.1, 800, 300, 200)

# Estimate the PIN model using the factorization of the PIN likelihood
# function by Ersan (2016)

estimate <- pin(xdata, initialsets = initialset, verbose = FALSE)

# Display the estimated PIN value

show(estimate@pin)

# Display the estimated parameters

show(estimate@parameters)

# Store the initial parameter sets used for MLE in a dataframe variable,
# and display its first five rows

initialsets <- estimate@initialsets
show(head(initialsets, 5))
```

`pin_bayes` *PIN estimation - Bayesian approach*

Description

Estimates the Probability of Informed Trading (PIN) using Bayesian Gibbs sampling as in Griffin et al. (2021) and the initial sets from the algorithm in Ersan and Alici (2016).

Usage

```
pin_bayes(data, xtraclusters = 4, sweeps = 1000, burnin = 500,
                 prior.a = 1, prior.b = 2, verbose = TRUE)
```

Arguments

`data` A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

`xtraclusters` An integer used to divide trading days into #(2 + xtraclusters) clusters, thereby resulting in #comb(1 + xtraclusters, 1) initial parameter sets in line with Ersan and Alici (2016). The default value is 4.

`sweeps` An integer referring to the number of iterations for the Gibbs Sampler. This has to be large enough to ensure convergence of the Markov chain. The default value is 1000.

`burnin` An integer referring to the number of initial iterations for which the parameter draws should be discarded. This is to ensure that we keep the draws at the point where the MCMC has converged to the parameter space in which the parameter estimate is likely to fall. This figure must always be less than the sweeps. The default value is 500.

`prior.a` An integer controlling the mean number of informed trades, such as the prior of informed buys and sells is the Gamma density function with μ ~ Ga(prior.a, η). The default value is 1. For more details, please refer to Griffin et al. (2021).

`prior.b` An integer controlling the mean number of uninformed trades, such as the prior of uninformed buys and sells is the Gamma density function with ε_b ~ Ga(prior.b, η), and ε_s ~ Ga(prior.b, η). The default value is 2. For more details, please refer to Griffin et al. (2021).

`verbose` A binary variable that determines whether detailed information about the steps of the estimation of the PIN model is displayed. No output is produced when verbose is set to FALSE. The default value is TRUE.

Details

The argument ’data’ should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The function `pin_bayes()` implements the algorithm detailed in Ersan and Alici (2016). The higher the number of the additional clusters (`xtraclusters`), the better is the estimation. Ersan and Alici (2016), however, have shown the benefit of increasing this number beyond 5 is marginal, and statistically insignificant.

The function `initials_pin_ea()` provides the initial parameter sets obtained through the implementation of the Ersan and Alici (2016) algorithm. For further information on the initial parameter set determination, see `initials_pin_ea()`.

Value

Returns an object of class estimate.pin-class

References

Ersan O, Alici A (2016). “An unbiased computation methodology for estimating the probability of informed trading (PIN).” *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Griffin J, Oberoi J, Oduro SD (2021). “Estimating the probability of informed trading: A Bayesian approach.” *Journal of Banking & Finance*, **125**, 106045.

Examples

```
# Use the function generatedata_mpin() to generate a dataset of
# 60 days according to the assumptions of the original PIN model.

sdata <- generatedata_mpin(layers = 1)
xdata <- sdata@data

# Estimate the PIN model using the Bayesian approach developed in
# Griffin et al. (2021), and initial parameter sets generated using the
# algorithm of Ersan and Alici (2016). The argument xtraclusters is
# set to 1. We also leave the arguments 'sweeps' and 'burnin' at their
# default values.

estimate <- pin_bayes(xdata, xtraclusters = 1, verbose = FALSE)


# Display the empirical PIN value at the data, and the PIN value
# estimated using the bayesian approach

setNames(c(sdata@emp.pin, estimate@pin), c("data", "estimate"))
```

```
# Display the empirial and the estimated parameters

show(unlist(sdata@empiricals))
show(estimate@parameters)

# Find the initial set that leads to the optimal estimate
optimal <- which.max(estimate@details$likelihood)

# Store the matrix of Monte Carlo simulation for the optimal
# estimate, and display its last five rows

mcmatrix <- estimate@details$markovmatrix[[optimal]]
show(tail(mcmatrix, 5))

# Display the summary of Geweke test for the Monte Carlo matrix above.
show(estimate@details$summary[[optimal]])
```

pin_ea *PIN estimation - initial parameter sets of Ersan & Alici (2016)*

Description

Estimates the Probability of Informed Trading (PIN) using the initial sets from the algorithm in Ersan and Alici (2016).

Usage

```
pin_ea(data, factorization, xtraclusters = 4, verbose = TRUE)
```

Arguments

data
: A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).

factorization
: A character string from {"E", "EHO", "LK", "NONE"} referring to a given factorization. The default value is "E".

xtraclusters
: An integer used to divide trading days into #(2 + xtraclusters) clusters, thereby resulting in #comb(1 + xtraclusters, 1) initial parameter sets in line with Ersan and Alici (2016). The default value is 4.

verbose
: A binary variable that determines whether detailed information about the steps of the estimation of the PIN model is displayed. No output is produced when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. `NA` values will be ignored.

The factorization variable takes one of four values:

- `"EHO"` refers to the factorization in Easley et al. (2010)
- `"LK"` refers to the factorization in Lin and Ke (2011)
- `"E"` refers to the factorization in Ersan (2016)
- `"NONE"` refers to the original likelihood function - with no factorization

The function `pin_ea()` implements the algorithm detailed in Ersan and Alici (2016). The higher the number of the additional layers (`xtraclusters`), the better is the estimation. Ersan and Alici (2016), however, have shown the benefit of increasing this number beyond 5 is marginal, and statistically insignificant.

The function `initials_pin_ea()` provides the initial parameter sets obtained through the implementation of the Ersan and Alici (2016) algorithm. For further information on the initial parameter set determination, see `initials_pin_ea()`.

Value

Returns an object of class estimate.pin-class

References

Easley D, Hvidkjaer S, Ohara M (2010). "Factoring information into returns." *Journal of Financial and Quantitative Analysis*, **45**(2), 293–309. ISSN 00221090.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Lin H, Ke W (2011). "A computing bias in estimating the probability of informed trading." *Journal of Financial Markets*, **14**(4), 625-640. ISSN 1386-4181.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades
```

```
# Estimate the PIN model using the factorization of Ersan (2016), and initial
# parameter sets generated using the algorithm of Ersan and Alici (2016).
# The argument xtraclusters is omitted so will take its default value 4.

estimate <- pin_ea(xdata, verbose = FALSE)

# Display the estimated PIN value

show(estimate@pin)

# Display the estimated parameters

show(estimate@parameters)

# Store the initial parameter sets used for MLE in a dataframe variable,
# and display its first five rows

initialsets <- estimate@initialsets
show(head(initialsets, 5))
```

---

pin_gwj *PIN estimation - initial parameter set of Gan et al. (2015)*

Description

Estimates the Probability of Informed Trading (PIN) using the initial set from the algorithm in Gan et al.(2015).

Usage

```
pin_gwj(data, factorization = "E", verbose = TRUE)
```

Arguments

data	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
factorization	A character string from {"EHO", "LK", "E", "NONE"} referring to a given factorization. The default value is set to "E".
verbose	A binary variable that determines whether detailed information about the steps of the estimation of the PIN model is displayed. No output is produced when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. NA values will be ignored.

The factorization variable takes one of four values:

- "EHO" refers to the factorization in Easley et al. (2010)
- "LK" refers to the factorization in Lin and Ke (2011)
- "E" refers to the factorization in Ersan (2016)
- "NONE" refers to the original likelihood function - with no factorization

The function pin_gwj() implements the algorithm detailed in Gan et al. (2015). You can use the function initials_pin_gwj() in order to get the initial parameter set.

Value

Returns an object of class estimate.pin-class

References

Easley D, Hvidkjaer S, Ohara M (2010). "Factoring information into returns." *Journal of Financial and Quantitative Analysis*, **45**(2), 293–309. ISSN 00221090.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Gan Q, Wei WC, Johnstone D (2015). "A faster estimation method for the probability of informed trading using hierarchical agglomerative clustering." *Quantitative Finance*, **15**(11), 1805–1821.

Lin H, Ke W (2011). "A computing bias in estimating the probability of informed trading." *Journal of Financial Markets*, **14**(4), 625-640. ISSN 1386-4181.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Estimate the PIN model using the factorization of Ersan (2016), and initial
# parameter sets generated using the algorithm of Gan et al. (2015).
# The argument xtraclusters is omitted so will take its default value 4.

estimate <- pin_gwj(xdata, verbose = FALSE)

# Display the estimated PIN value

show(estimate@pin)

# Display the estimated parameters
```

```
show(estimate@parameters)

# Store the initial parameter sets used for MLE in a dataframe variable,
# and display its first five rows

initialsets <- estimate@initialsets
show(head(initialsets, 5))
```

pin_yz *PIN estimation - initial parameter sets of Yan & Zhang (2012)*

Description

Estimates the Probability of Informed Trading (PIN) using the initial parameter sets generated using the grid search algorithm of Yan and Zhang (2012).

Usage

```
pin_yz(data, factorization, ea_correction = FALSE, grid_size = 5,
                                                   verbose = TRUE)
```

Arguments

data	A dataframe with 2 variables: the first corresponds to buyer-initiated trades (buys), and the second corresponds to seller-initiated trades (sells).
factorization	A character string from {"EHO", "LK", "E", "NONE"} referring to a given factorization. The default value is "E".
ea_correction	A binary variable determining whether the modifications of the algorithm of Yan and Zhang (2012) suggested by Ersan and Alici (2016) are implemented. The default value is FALSE.
grid_size	An integer between 1, and 20; representing the size of the grid. The default value is 5. See more in details.
verbose	A binary variable that determines whether detailed information about the steps of the estimation of the PIN model is displayed. No output is produced when verbose is set to FALSE. The default value is TRUE.

Details

The argument 'data' should be a numeric dataframe, and contain at least two variables. Only the first two variables will be considered: The first variable is assumed to correspond to the total number of buyer-initiated trades, while the second variable is assumed to correspond to the total number of seller-initiated trades. Each row or observation correspond to a trading day. NA values will be ignored.

The factorization variable takes one of four values:

- "EHO" refers to the factorization in Easley et al. (2010)

- "LK" refers to the factorization in Lin and Ke (2011)
- "E" refers to the factorization in Ersan (2016)
- "NONE" refers to the original likelihood function - with no factorization

The argument grid_size determines the size of the grid of the variables: alpha, delta, and eps.b. If grid_size is set to a given value m, the algorithm creates a sequence starting from 1/2m, and ending in 1 - 1/2m, with a step of 1/m. The default value of 5 corresponds to the size of the grid in Yan and Zhang (2012). In that case, the sequence starts at 0.1 = 1/(2 x 5), and ends in 0.9 = 1 - 1/(2 x 5) with a step of 0.2 = 1/m.

The function pin_yz() implements, by default, the original Yan and Zhang (2012) algorithm as the default value of ea_correction takes the value FALSE. When the value of ea_correction is set to TRUE; then, sets with irrelevant mu values are excluded, and sets with boundary values are reintegrated in the initial parameter sets.

Value

Returns an object of class estimate.pin-class

References

Easley D, Hvidkjaer S, Ohara M (2010). "Factoring information into returns." *Journal of Financial and Quantitative Analysis*, **45**(2), 293–309. ISSN 00221090.

Ersan O (2016). "Multilayer Probability of Informed Trading." *Available at SSRN 2874420*.

Ersan O, Alici A (2016). "An unbiased computation methodology for estimating the probability of informed trading (PIN)." *Journal of International Financial Markets, Institutions and Money*, **43**, 74–94. ISSN 10424431.

Lin H, Ke W (2011). "A computing bias in estimating the probability of informed trading." *Journal of Financial Markets*, **14**(4), 625-640. ISSN 1386-4181.

Yan Y, Zhang S (2012). "An improved estimation method and empirical properties of the probability of informed trading." *Journal of Banking and Finance*, **36**(2), 454–467. ISSN 03784266.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# Estimate the PIN model using the factorization of Lin and Ke(2011), and
# initial parameter sets generated using the algorithm of Yan & Zhang (2012).
# In contrast to the original algorithm, we set the grid size for the grid
# search algorithm at 3. The original algorithm assumes a grid of size 5.

estimate <- pin_yz(xdata, "LK", grid_size = 3, verbose = FALSE)
```

```
# Display the estimated PIN value

show(estimate@pin)

# Display the estimated parameters

show(estimate@parameters)

# Store the initial parameter sets used for MLE in a dataframe variable,
# and display its first five rows

initialsets <- estimate@initialsets
show(head(initialsets, 5))
```

---

set_display_digits *Package-wide number of digits*

Description

Sets the number of digits to display in the output of the different package functions.

Usage

```
set_display_digits(digits = list())
```

Arguments

digits	A list of numbers corresponding to the different display digits. The default value is list().

Details

The parameter digits is a named list. It will be containing:

- d1: contains the number of display digits for the values of probability estimates such as α, δ, pin, mpin, mpin(j), adjpin, psos, θ, and θ'.
- d2: contains the number of display digits for the values of μ, ε_b and ε_s, as well as information criteria: AIC, BIC, and AWE.
- d3: contains the number of display digits for the remaining values such as vpin statistics and likelihood value .

If the function is called with no arguments, the display digits will be reset to the default values, i.e., list(d1 = 6, d2 = 2, d3 = 3)). If the argument digits is not omitted, the function will only accept a list containing exactly three numerical values, each ranging between 0 and 10. The list can be named or unnamed. If the numbers in the argument digits are not integers, they will be rounded.

Value

No return value, called for side effects.

Examples

```
# There is a preloaded quarterly dataset called 'dailytrades' with 60
# observations. Each observation corresponds to a day and contains the
# total number of buyer-initiated trades ('B') and seller-initiated
# trades ('S') on that day. To know more, type ?dailytrades

xdata <- dailytrades

# We show the output of the function pin_ea() using the default values
# of display digits. We then change these values using the function
# set_display_digits(), before displaying the same estimate.pin object
# again to see the difference.

model <- pin_ea(xdata, verbose = FALSE)
show(model)

# Change the number of digits for d1 to 3, of d2 to 0 and of d3 to 2

set_display_digits(list(3, 0, 2))

# No need to run the function mpin_ml() again to update the display of an
# estimate.mpin object.This holds for all estimate* S4 objects.

show(model)
```

trade_classification *Classification and aggregation of high-frequency data*

Description

`classify_trades()` classifies high-frequency trading data into buyer-initiated and seller-initiated trades using different algorithms, and different time lags (or leads).
`aggregate_trades()` aggregates high-frequency trading data into aggregated data for provided frequency of aggregation. The aggregation is preceded by a trade classification step which classifies trades using different trade classification algorithms and time lags (or leads).

Usage

```
classify_trades(data, algorithm = "Tick", timelag = 0, ..., verbose = TRUE)

aggregate_trades(
  data,
  algorithm = "Tick",
  timelag = 0,
  frequency = "day",
  unit = 1,
  ...,
  verbose = TRUE
)
```

Arguments

`data` A dataframe with 4 variables in the following order (`timestamp`, `price`, `bid`, `ask`).

`algorithm` A character string refers to the algorithm used to determine the trade initiator, a buyer or a seller. It takes one of four values (`"Tick"`, `"Quote"`, `"LR"`, `"EMO"`). The default value is `"Tick"`. For more information about the different algorithms, check the Details section.

`timelag` Numeric scalar. Time offset in microseconds used to select the quote matched to each trade for the `"Quote"`, `"EMO"` and `"LR"` algorithms. Interpreted in seconds as `timelag / 1e6`. See **Time lags vs. leads** in `@details` for the exact matching rule and edge cases (start/end of sample).

Examples: `timelag = 5000` is a 5-millisecond lag; `timelag = -500000` is a 0.5-second lead.

`...` Additional arguments passed to the functions `classify_trades()` `aggregate_trades()`. The recognized arguments are `fullreport`, and `is_parallel`. Other arguments will be ignored.

- `fullreport` is binary variable passed to `aggregate_trades()` that specifies whether the variable `freq` is returned. The default value is `FALSE`.
- `is_parallel` is a logical variable passed to `classify_trades()` that specifies whether the computation is performed using parallel or sequential processing. #' The default value is `TRUE`. For more details, please refer to the vignette 'Parallel processing' in the package, or online.

`verbose` A binary variable that determines whether detailed information about the progress of the trade classification is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

`frequency` The frequency used to aggregate intraday data. It takes one of the following values: `"sec"`, `"min"`, `"hour"`, `"day"`, `"week"`, `"month"`. The default value is `"day"`.

`unit` An integer referring to the size of the aggregation window used to aggregate intraday data. The default value is `1`. For example, when the parameter `frequency` is set to `"min"`, and the parameter `unit` is set to 15, then the intraday data is aggregated every 15 minutes.

Details

Trade classification algorithms

The argument `algorithm` takes one of four values:

- `"Tick"` refers to the tick algorithm: Trade is classified as a buy (sell) if the price of the trade to be classified is above (below) the closest different price of a previous trade.

- "Quote" refers to the quote algorithm: it classifies a trade as a buy (sell) if the trade price of the trade to be classified is above (below) the mid-point of the bid and ask spread. Trades executed at the mid-spread are not classified.
- "LR" refers to LR algorithm as in Lee and Ready (1991). It classifies a trade as a buy (sell) if its price is above (below) the mid-spread (quote algorithm), and uses the tick algorithm if the trade price is at the mid-spread.
- "EMO" refers to EMO algorithm as in Ellis et al. (2000). It classifies trades at the bid (ask) as sells (buys) and uses the tick algorithm to classify trades within the then prevailing bid-ask spread.

Time lags vs. leads (timelag)

For the "Quote", "LR" and "EMO" algorithms, classification relies on a quote (bid, ask or midquote) matched to each trade. The argument timelag controls *when* that quote is taken relative to the trade time:

- *Positive lags* (timelag > 0): for a trade at time t, the algorithm uses the quote corresponding to the last trade observed at or before t - |timelag| seconds. If no such past trade exists, the trade has no matched quote.
- *Zero lag* (timelag = 0): for a trade at time t, the algorithm uses the quote attached to that trade itself, which in the data setup corresponds to the bid–ask spread just before the trade is executed.
- *Negative lags / leads* (timelag < 0): for a trade at time t, the algorithm uses the quote corresponding to the last trade observed at or before t + |timelag| seconds (a future quote). If no such future trade exists, the trade has no matched quote.

In all cases the time offset is interpreted in seconds as timelag/1e6.

For example, timelag = 500000 corresponds to 0.5 seconds lag, and timelag = -2000000 corresponds to a 2-second lead.

Trades for which no suitable lagged/leading quote exists within the requested window are handled as follows:

- For "Quote", the corresponding trades receive NA classifications.
- For "LR", the quote-based classification is still used where available; trades exactly at the (lagged/leading) midquote fall back to the tick rule. When no midquote exists within the window, the result is NA.
- For "EMO", the bid/ask from the lagged/leading quote is used when available. If no such quote exists, the EMO quote-based step is skipped and the tick rule classification is retained.

LR recommend the use of mid-spread five-seconds earlier ('5-second' rule) mitigating trade misclassifications for many of the 150 NYSE stocks they analyze. On the other hand, in more recent studies such as Piwowar and Wei (2006) and Aktas and Kryzanowski (2014), the use of 1-second lagged midquotes are shown to yield lower rates of misclassifications. The default value is set to 0 seconds (no time-lag). Considering the ultra-fast nature of today's financial markets, time-lag is in the unit of milliseconds. Shorter than 1-second lags can also be implemented by entering values such as 100 or 500.

Value

The function classify_trades() returns a dataframe of five variables. The first four variables are obtained from the argument `data`: `timestamp`, `price`, `bid`, `ask`. The fifth variable is `isbuy`, which takes the value `TRUE`, when the trade is classified as a buyer-initiated trade, and `FALSE` when the trade is classified as a seller-initiated trade.

The function aggregate_trades() returns a dataframe of two (or three) variables. If `fullreport` is set to `TRUE`, then the returned dataframe has three variables `{freq, b, s}`. If `fullreport` is set to `FALSE`, then the returned dataframe has two variables `{b, s}`, and, therefore, can be #'directly used for the estimation of the `PIN` and `MPIN` models.

References

Aktas OU, Kryzanowski L (2014). "Trade classification accuracy for the BIST." *Journal of International Financial Markets, Institutions and Money*, **33**, 259-282. ISSN 1042-4431.

Ellis K, Michaely R, Ohara M (2000). "The Accuracy of Trade Classification Rules: Evidence from Nasdaq." *The Journal of Financial and Quantitative Analysis*, **35**(4), 529–551.

Lee CMC, Ready MJ (1991). "Inferring Trade Direction from Intraday Data." *The Journal of Finance*, **46**(2), 733–746. ISSN 00221082, 15406261.

Piwowar MS, Wei L (2006). "The Sensitivity of Effective Spread Estimates to Trade-Quote Matching Algorithms." *Electronic Markets*, **16**(2), 112-129.

Examples

```
# There is a preloaded dataset called 'hfdata' contained in the package.
# It is an artificially created high-frequency trading data. The dataset
# contains  100 000 trades and five variables 'timestamp', 'price',
# 'volume', 'bid', and 'ask'. For more information, type ?hfdata.

xdata <- hfdata
xdata$volume <- NULL

# Use the LR algorithm with a timelag of 0.5 seconds i.e. 500000
# microseconds to classify high-frequency trades in the dataset 'xdata'

lgtrades <- classify_trades(xdata, "LR", timelag = 500000, verbose = FALSE)

# LR algorithm with a 0.5-second lead (-500000 microseconds)

ldtrades <- classify_trades(xdata, "LR", timelag = -500000, verbose = FALSE)

# Compare the number of buyer- and seller-initiated trades between the
# lagged and leading LR classifications.

comparison_tbl <- rbind(
transform(lgtrades, version = "lag of 0.5s"),
transform(ldtrades, version = "lead of 0.5s")
)
```

```
comparison_tbl <- with(comparison_tbl,
  aggregate(list(Buys = as.logical(isbuy), Sells = !as.logical(isbuy)),
  by = list(version = version),
  FUN = sum, na.rm = TRUE)
)

show(comparison_tbl)

# Use the EMO algorithm with a timelag of 1 second, i.e. 1000000 microseconds
# to aggregate intraday data in 'xdata' at a frequency of 15 minutes.

emotrades <- aggregate_trades(xdata, algorithm = "EMO", timelag = 1000000,
frequency = "min", unit = 15, verbose = FALSE)

# Use the Quote algorithm with a timelag of 1 second to aggregate intraday
# data in the dataset 'xdata' at a daily frequency.

qtrades <- aggregate_trades(xdata, algorithm = "Quote", timelag = 1000000,
frequency = "day", unit = 1, verbose = FALSE)

# Since the argument 'fullreport' is set to FALSE by default, then the
# output 'qtrades' can be used directly for the estimation of the PIN
# model, namely using pin_ea().

estimate <- pin_ea(qtrades, verbose = FALSE)

# Show the estimate

show(estimate)
```

vpin_measures *Estimation of Volume-Synchronized PIN model (vpin) and the improved volume-synchronized PIN model (ivpin)*

Description

Estimates the Volume-Synchronized Probability of Informed Trading as developed in Easley et al. (2011) and Easley et al. (2012).
Estimates the improved Volume-Synchronized Probability of Informed Trading as developed in Ke et al. (2017).

Usage

```
vpin(
  data,
  timebarsize = 60,
  buckets = 50,
  samplength = 50,
  tradinghours = 24,
  verbose = TRUE
)

ivpin(
  data,
  timebarsize = 60,
  buckets = 50,
  samplength = 50,
  tradinghours = 24,
  grid_size = 5,
  verbose = TRUE
)
```

Arguments

`data` A dataframe with 3 variables: `{timestamp, price, volume}`.

`timebarsize` An integer referring to the size of timebars in seconds. The default value is `60`.

`buckets` An integer referring to the number of buckets in a daily average volume. The default value is `50`.

`samplength` An integer referring to the sample length or the window size used to calculate the `VPIN` vector. The default value is `50`.

`tradinghours` An integer referring to the length of daily trading sessions in hours. The default value is `24`.

`verbose` A logical variable that determines whether detailed information about the steps of the estimation of the VPIN (IVPIN) model is displayed. No output is produced when `verbose` is set to `FALSE`. The default value is `TRUE`.

`grid_size` An integer between `1`, and `20`; representing the size of the grid used in the estimation of IVPIN. The default value is `5`. See more in details.

Details

The dataframe data should contain at least three variables. Only the first three variables will be considered and in the following order `{timestamp, price, volume}`.

The argument `timebarsize` is in seconds enabling the user to implement shorter than `1` minute intervals. The default value is set to `1` minute (`60` seconds) following Easley et al. (2011, 2012).

The argument `tradinghours` is used to correct the duration per bucket if the market trading session does not cover a full day `(24 hours)`. The duration of a given bucket is the difference between the timestamp of the last trade `endtime` and the timestamp of the first trade `stime` in the bucket. If the first and last trades in a bucket occur on different days, and the market trading session is shorter than `24 hours`, the bucket's duration will be inflated. For example, if the daily trading session is 8 hours `(tradinghours = 8)`, and the start time of a bucket is `2018-10-12 17:06:40` and its end time is `2018-10-13 09:36:00`, the straightforward calculation gives a duration of `59,360 secs`. However, this duration includes 16 hours when the market is closed. The corrected duration considers only the market activity time: `duration = 59,360 - 16 * 3600 = 1,760 secs`, approximately `30 minutes`.

The argument `grid_size` determines the size of the grid for the variables `alpha` and `delta`, used to generate the initial parameter sets that prime the maximum-likelihood estimation step of the algorithm by Ke et al. (2017) for estimating IVPIN. If grid_size is set to a value m, the algorithm creates a sequence starting from 1 / (2m) and ending at 1 - 1 / (2m), with a step of 1 / m. The default value of 5 corresponds to the grid size used by Yan and Zhang (2012), where the sequence starts at 0.1 = 1 / (2 * 5) and ends at 0.9 = 1 - 1 / (2 * 5) with a step of 0.2 = 1 / 5. Increasing the value of grid_size increases the running time and may marginally improve the accuracy of the IVPIN estimates

Value

Returns an object of class estimate.vpin-class, which contains the following slots:

@improved A logical variable that takes the value FALSE when the classical VPIN model is estimated (using vpin()), and TRUE when the improved VPIN model is estimated (using ivpin()).

@bucketdata A data frame created as in Abad and Yague (2012).

@dailyvpin A data frame with calendar–day aggregates of VPIN. For each trading day, it contains three variables: day (Date), dvpin (simple daily average of per–bucket VPIN), and dwvpin (duration–weighted daily VPIN, i.e. the weighted average of bucket VPINs with weights proportional to the effective bucket durations).

@vpin A vector of VPIN values.

@ivpin A vector of IVPIN values, which remains empty when the function vpin() is called.

References

Abad D, Yague J (2012). "From PIN to VPIN: An introduction to order flow toxicity." *The Spanish Review of Financial Economics*, **10**(2), 74–83.

Easley D, De Prado MML, Ohara M (2011). "The microstructure of the \" flash crash\": flow toxicity, liquidity crashes, and the probability of informed trading." *The Journal of Portfolio Management*, **37**(2), 118–128.

Easley D, Lopez De Prado MM, OHara M (2012). "Flow toxicity and liquidity in a high-frequency world." *Review of Financial Studies*, **25**(5), 1457–1493. ISSN 08939454.

Ke W, Lin HW, others (2017). "An improved version of the volume-synchronized probability of informed trading." *Critical Finance Review*, **6**(2), 357–376.

Yan Y, Zhang S (2012). "An improved estimation method and empirical properties of the probability of informed trading." *Journal of Banking and Finance*, **36**(2), 454–467. ISSN 03784266.

Examples

```
# The package includes a preloaded dataset called 'hfdata'.
# This dataset is an artificially created high-frequency trading data
# containing 100,000 trades and five variables: 'timestamp', 'price',
# 'volume', 'bid', and 'ask'. For more information, type ?hfdata.

xdata <- hfdata

### Estimation of the VPIN model ###
```

```
# Estimate the VPIN model using the following parameters:
# - timebarsize: 5 minutes (300 seconds)
# - buckets: 50 buckets per average daily volume
# - samplength: 250 for the VPIN calculation

estimate <- vpin(xdata, timebarsize = 300, buckets = 50,
 samplength = 250)

# Display a description of the VPIN estimate

show(estimate)

# Display the parameters of the VPIN estimates

show(estimate@parameters)

# Display the summary statistics of the VPIN vector

summary(estimate@vpin)

# Store the computed data of the different buckets in a dataframe 'buckets'
# and display the first 10 rows of the dataframe.

buckets <- estimate@bucketdata
show(head(buckets, 10))

# Display the first 10 rows of the dataframe containing daily vpin values.

dayvpin <- estimate@dailyvpin
show(head(dayvpin, 10))


### Estimation of the IVPIN model ###

# Estimate the IVPIN model using the same parameters as above.
# The grid_size parameter is unspecified and will default to 5.

iestimate <- ivpin(xdata[1:50000,], timebarsize = 300, samplength = 50, verbose = FALSE)

# Display the summary statistics of the IVPIN vector

summary(iestimate@ivpin)

# The output of ivpin() also contains the VPIN vector in the @vpin slot.
# Plot the VPIN and IVPIN vectors in the same plot using the iestimate object.

# Define the range for the VPIN and IVPIN vectors, removing NAs.

vpin_range <- range(c(iestimate@vpin, iestimate@ivpin), na.rm = TRUE)

# Plot the VPIN vector in blue
```

```
plot(iestimate@vpin, type = "l", col = "blue", ylim = vpin_range,
     ylab = "VPIN/iVPIN", xlab = "Bucket", main = "Plot of VPIN and IVPIN")

# Add the IVPIN vector in red
lines(iestimate@ivpin, type = "l", col = "red")

# Add a legend to the plot
legend("topright", legend = c("VPIN", "IVPIN"), col = c("blue", "red"),
 lty = 1,
 cex = 0.6,  # Adjust the text size
 x.intersp = 1.2,  # Adjust the horizontal spacing
 y.intersp = 2,  # Adjust the vertical spacing
 inset = c(0.05, 0.05))  # Adjust the position slightly
```

Index