Package: markets (via r-universe)

October 26, 2024

Title Estimation Methods for Markets in Equilibrium and Disequilibrium

Version 1.1.5

Date 2024-02-17

Description Provides estimation methods for markets in equilibrium and disequilibrium. Supports the estimation of an equilibrium and four disequilibrium models with both correlated and independent shocks. Also provides post-estimation analysis tools, such as aggregation, marginal effect, and shortage calculations. See Karapanagiotis (2024) <doi:10.18637/jss.v108.i02> for an overview of the functionality and examples. The estimation methods are based on full information maximum likelihood techniques given in Maddala and Nelson (1974) <doi:10.2307/1914215>. They are implemented using the analytic derivative expressions calculated in Karapanagiotis (2020) <doi:10.2139/ssrn.3525622>. Standard errors can be estimated by adjusting for heteroscedasticity or clustering. The equilibrium estimation constitutes a case of a system of linear, simultaneous equations. Instead, the disequilibrium models replace the market-clearing condition with a non-linear, short-side rule and allow for different specifications of price dynamics.

Language en-US

URL `https://github.com/pi-kappa-devel/markets/`, `https://markets.pikappa.eu/`

BugReports `https://github.com/pi-kappa-devel/markets/issues`

Depends R (>= 4.1.0)

Imports dplyr (>= 0.7.6), Formula, MASS (>= 7.3-50), methods, rlang (>= 0.2.1), Rcpp, RcppGSL, RcppParallel, stats

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests ggplot2 (>= 3.0.0), knitr (>= 1.20), numDeriv (>= 2016.8.1.1), rmarkdown (>= 1.10), testthat (>= 2.0.0)

VignetteBuilder knitr

Collate 'data.R' 'equation_base.R' 'system_base.R' 'model_logger.R' 'market_model.R' 'disequilibrium_model.R' 'diseq_basic.R' 'diseq_deterministic_adjustment.R' 'diseq_directional.R' 'diseq_stochastic_adjustment.R' 'equation_basic.R' 'equation_deterministic_adjustment.R' 'equation_directional.R' 'equation_stochastic_adjustment.R' 'equilibrium_model.R' 'system_basic.R' 'gradient_basic.R' 'system_deterministic_adjustment.R' 'gradient_deterministic_adjustment.R' 'system_directional.R' 'gradient_directional.R' 'system_equilibrium.R' 'gradient_equilibrium.R' 'system_stochastic_adjustment.R' 'gradient_stochastic_adjustment.R' 'hessian_basic.R' 'hessian_directional.R' 'likelihood_basic.R' 'likelihood_deterministic_adjustment.R' 'likelihood_directional.R' 'likelihood_equilibrium.R' 'likelihood_stochastic_adjustment.R' 'market_fit.R' 'markets.R' 'model_simulation.R'

LinkingTo Rcpp, RcppGSL

NeedsCompilation yes

Author Pantelis Karapanagiotis [aut, cre] (<https://orcid.org/0000-0001-9871-1908>)

Maintainer Pantelis Karapanagiotis <pikappa.devel@gmail.com>

Repository CRAN

Date/Publication 2024-02-17 10:50:07 UTC

Contents

`coef` *Market fit coefficients*

Description

Returns the coefficients of the fitted market model.

Usage

```
## S4 method for signature 'market_fit'
coef(object)

## S4 method for signature 'market_fit'
coefficients(object)
```

Arguments

`object`	A fitted model object.

Value

A named vector of estimated model coefficients.

Methods (by class)

- `coef(market_fit)`: Estimated coefficients.
- `coefficients(market_fit)`: Estimated coefficients alias.

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+6))
)

# access the estimated coefficients
coef(fit)
coefficients(fit)
```

estimate *Model estimation*

Description

All models are estimated using full information maximum likelihood. The equilibrium_model can also be estimated using two-stage least squares. The maximum likelihood estimation is based on optim. If no starting values are provided, the function uses linear regression estimates as initializing values. The default optimization method is BFGS. For other alternatives see optim. The implementation of the two-stage least square estimation of the equilibrium_model is based on lm.

Usage

```
estimate(object, ...)

## S4 method for signature 'market_model'
estimate(
  object,
  gradient = "calculated",
  hessian = "calculated",
  standard_errors = "homoscedastic",
  ...
)

## S4 method for signature 'equilibrium_model'
estimate(object, method = "BFGS", optimizer = "optim", ...)
```

Arguments

object	A model object.
...	Additional parameter used in the model's estimation. These are passed further down to the optimization call. For the equilibrium_model model, the parameters are passed to lm, if the method is set to 2SLS, or to optim for any other method. For the rest of the models, the parameters are passed to optim.
gradient	One of two potential options: "numerical" and "calculated". By default, all the models are estimated using the analytic expressions of their likelihoods' gradients.
hessian	One of three potential options: "skip", "numerical", and "calculated". The default is to use the "calculated" Hessian for the models that expressions are available and the "numerical" Hessian in other cases. Calculated Hessian expressions are available for the basic and directional models.

`standard_errors`	One of three potential options: `"homoscedastic"`, `"heteroscedastic"`, or a vector with variables names for which standard error clusters are to be created. The default value is `"homoscedastic"`. If the option `"heteroscedastic"` is passed, the variance-covariance matrix is calculated using heteroscedasticity adjusted (Huber-White) standard errors. If a vector with variable names is supplied, the variance-covariance matrix is calculated by grouping the score matrix based on the passed variables.
`method`	A string specifying the estimation method. When the passed value is among `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"`, `"SANN"`, and `"Brent"`, the model is estimated using full information maximum likelihood based on `optim` functionality. When `"2SLS"` is supplied, the model is estimated using two-stage least squares via `lm`. In this case, the function returns a list containing the first and second stage estimates. The default value is `"BFGS"`.
`optimizer`	One of two options: `"optim"`, `"gsl"`. The default value is `"optim"`. If the option `"gsl"` is set, the equilibrium likelihood is maximized using `GSL`.

Details

The likelihood of the equilibrium model can be optimized either by using `optim` (the default option) or native `GSL` routines. The caller can override the default behavior by setting the `optimizer` argument equal to `"gsl"`, in which case `GSL` routines are used. This does not necessarily result to faster execution times. This functionality is primarily intended for advanced usage. The `optim` functionality is a fast, analysis-oriented alternative, which is more suitable for most use case.

When `optimizer = "gsl"` is used, the only available optimization method is BFGS. Additionally, the caller needs to specify in the control list values for the optimization step (`step`), the objective's optimization tolerance (`objective_tolerance`), the gradient's optimization tolerance (`gradient_tolerance`, and the maximum allowed number of iterations (`maxit`). If the `GSL` library is not available in the calling machine, the function returns a trivial result list with convergence status set equal to -1. If the C++17 execution policies are available, the implementation of the optimization is parallelized.

Value

A market fit object holding the estimation result.

Functions

- `estimate(market_model)`: Full information maximum likelihood estimation.
- `estimate(equilibrium_model)`: Equilibrium model estimation.

Examples

```
# initialize the model using the houses dataset
model <- new(
  "diseq_deterministic_adjustment", # model type
  subject = ID, time = TREND, quantity = HS, price = RM,
  demand = RM + TREND + W + CSHS + L1RM + L2RM + MONTH,
  supply = RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(), # data
  correlated_shocks = FALSE # let shocks be independent
)

# estimate the model object (BFGS is used by default)
fit <- estimate(model)

# estimate the model by specifying the optimization details passed to the optimizer.
fit <- estimate(model, control = list(maxit = 1e+6))

# summarize results
summary(fit)


# simulate an equilibrium model
model <- simulate_model(
  "equilibrium_model", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -1.9, beta_d0 = 24.9, beta_d = c(2.3, -1.2), eta_d = c(2.0, -1.5),
    # supply coefficients
    alpha_s = .9, beta_s0 = 8.2, beta_s = c(3.3), eta_s = c(1.5, -2.2)
  ),
  seed = 99
)

# maximize the model's log-likelihood
fit <- estimate(
  model,
  optimizer = "gsl", control = list(
    step = 1e-2, objective_tolerance = 1e-8,
    gradient_tolerance = 1e-2, maxit = 1e+3
  )
)

summary(fit)
```

---

formula,market_model-method

Market model formula

Description

Market model formula

Usage

```
## S4 method for signature 'market_model'
formula(x)
```

Arguments

x	A market model object.

Details

Market model formulas adhere to the following specification:

`quantity | price | subject | time ~ demand | supply`

where

- quantity: The model's traded (observed) quantity variable.
- price: The model's price variable.
- quantity: The model's subject (e.g. firm) identification variable.
- quantity: The model's time identification variable.
- demand: The right hand side of the model's demand equation.
- supply: The right hand side of the model's supply equation.

The `diseq_stochastic_adjustment` additionally specify price dynamics by appending the right hand side of the equation at the end of the formula, i.e.

`quantity | price | subject | time ~ demand | supply | price_dynamics`

The left hand side part of the model formula specifies the elements that are needed for initializing the model. The market models of the package prepare the data based on these four variables using their respective identification assumptions. See market model classes for more details.

The function provides access to the formula used in model initialization.

Value

The model's formula

Examples

```
model <- simulate_model(
  "diseq_stochastic_adjustment", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -0.1, beta_d0 = 9.8, beta_d = c(0.3, -0.2), eta_d = c(0.6, -0.1),
    # supply coefficients
    alpha_s = 0.1, beta_s0 = 6.1, beta_s = c(0.9), eta_s = c(-0.5, 0.2),
    # price equation coefficients
    gamma = 1.2, beta_p0 = 3.1, beta_p = c(0.8)
  ),
  seed = 31
)

# access the model's formula
formula(model)
```

houses *Credit market data for US housing starts*

Description

Credit market data for US housing starts

Usage

```
data(houses)

fair_houses()
```

Format

A data frame with 138 rows and 7 columns

Details

The basic houses dataset (`houses`)**:**

A dataset containing the monthly mortgage rates and other attributes of the US market for new, non-farm houses from January 1958 to December 1969 (144 observations). The variables are as follows:

- `DATE` The date of the record.
- `HS` Private non-farm housing starts in thousands of units (Not seasonally adjusted).
- `RM` FHA Mortgage rate series on new homes in units of 100 (beginning-of-month Data).
- `DSLA` Savings capital (deposits) of savings and loan associations in millions of dollars.
- `DMSB` Deposits of mutual savings banks in millions of dollars.
- `DHLB` Advances of the federal home loan bank to savings and loan associations in million of dollars.
- `W` Number of working days in month.

Generate the variables of the Fair & Jaffee (1972) dataset. (`fair_houses`)**:**

Loads the `houses` dataset and creates the additional variables used by Fair & Jaffee (1972) doi:10.2307/1913181. These are

- `ID` A dummy entity identifier that is always equal to one since the houses data have only a time series component.
- `DSF` Flow of deposits in savings and loan associations and mutual savings banks in million of dollars. Equal to

$$DSLA_t + DMSB_t - (DSLA_{t-1} + DMSB_{t-1}).$$

- `DHF` Flow of advances of the federal home loan bank to savings and loan associations in million of dollars. Equal to

$$DHLB_t - DHLB_{t-1}.$$

- `MONTH` The month of the date of the observation.
- `L1RM` FHA Mortgage rate series on new homes in units of 100, lagged by one date.
- `L2RM` FHA Mortgage rate series on new homes in units of 100, lagged by two dates.
- `L1HS` Private non-farm housing starts in thousands of units (Not seasonally adjusted), lagged by one date.
- `CSHS` The cumulative sum of past housing starts. Used to proxy the stock of houses
- `MA6DSF` Moving average of order 6 of the flow of deposits in savings associations and loan associations and mutual savings banks.
- `MA3DHF` Moving average of order 3 of the flow of advances of the federal home loan bank to savings and loan associations.
- `TREND` A time trend variable.

Returns A modified version of the `houses` dataset.

Functions

- `fair_houses()`: Generate Fair & Jaffee (1972) dataset

Source

- `HS` Economic Reports of the President
- `RM` Fair (1971)
- `DSLA` Federal Reserve Bulletins
- `DMSB` Federal Reserve Bulletins
- `DHLB` Federal Reserve Bulletins
- `W` Manually calculated

References

- Fair, R. C. (1971). A short-run forecasting model of the United States economy. Heath Lexington Books.
- Fair, R. C., & Jaffee, D. M. (1972). Methods of Estimation for Markets in Disequilibrium. Econometrica, 40(3), 497. doi:10.2307/1913181
- Maddala, G. S., & Nelson, F. D. (1974). Maximum Likelihood Methods for Models of Markets in Disequilibrium. Econometrica, 42(6), 1013. doi:10.2307/1914215
- Hwang, H. (1980). A test of a disequilibrium model. Journal of Econometrics, 12(3), 319–333. doi:10.1016/03044076(80)900597

Examples

```
data(houses)
head(houses)
head(fair_houses())
```

`logLik.market_fit` *Log likelihood of a fitted market model*

Description

Specializes the `logLik` function for the market models of the package estimated with full information minimum likelihood. It returns `NULL` for the equilibrium model estimated with two stage least squares (`method = "2SLS"`).

Usage

```
## S3 method for class 'market_fit'
logLik(object, ...)

## S4 method for signature 'market_fit'
logLik(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	Additional arguments. Unused.

Value

A `logLik` object.

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+6))
)

# get the log likelihood object
logLik(fit)
```

marginal_effects *Marginal effects*

Description

Returns the estimated effect of a variable. The effect accounts for both sides of the market. If the given variable belongs only to the demand side, the name of result is prefixed by "D_". If the given variable belongs only to the supply side, the name of result is prefixed by "S_". If the variable can be found both sides, the result name is prefixed by "B_".

Usage

```
shortage_marginal(fit, variable, model, parameters)

shortage_probability_marginal(
  fit,
  variable,
  aggregate = "mean",
  model,
  parameters
)

## S4 method for signature 'missing,ANY,market_model,ANY'
shortage_marginal(variable, model, parameters)

## S4 method for signature 'missing,ANY,ANY,market_model,ANY'
shortage_probability_marginal(variable, aggregate, model, parameters)

## S4 method for signature 'missing,ANY,market_model,ANY'
shortage_marginal(variable, model, parameters)

## S4 method for signature 'market_fit,ANY,missing,missing'
shortage_marginal(fit, variable)

## S4 method for signature 'market_fit,ANY,ANY,missing,missing'
shortage_probability_marginal(fit, variable, aggregate)
```

Arguments

`fit`	A fitted market model.
`variable`	Variable name for which the effect is calculated.
`model`	A market model object.
`parameters`	A vector of parameters.
`aggregate`	Mode of aggregation. Valid options are "mean" (the default) and "at_the_mean".

Value

The estimated effect of the passed variable.

Functions

- `shortage_marginal()`: Marginal effect on market system

  Returns the estimated marginal effect of a variable on the market system. For a system variable x with demand coefficient $\beta_{d,x}$ and supply coefficient $\beta_{s,x}$, the marginal effect on the market system is given by
  $$M_x = \frac{\beta_{d,x} - \beta_{s,x}}{\sqrt{\sigma_d^2 + \sigma_s^2 - 2\rho_{ds}\sigma_d\sigma_s}}.$$

- `shortage_probability_marginal()`: Marginal effect on shortage probabilities

  Returns the estimated marginal effect of a variable on the probability of observing a shortage state. The mean marginal effect (`aggregate = "mean"`) on the shortage probability is given by
  $$M_x \mathrm{E} \phi \left(\frac{D - S}{\sqrt{\sigma_d^2 + \sigma_s^2 - 2rho\sigma_d\sigma_s}} \right)$$
  . and the marginal effect at the mean (`aggregate = "at_the_mean"`) by
  $$M_x \phi \left(\mathrm{E} \frac{D - S}{\sqrt{\sigma_d^2 + \sigma_s^2 - 2rho\sigma_d\sigma_s}} \right)$$
  where M_x is the marginal effect on the system, D is the demanded quantity, S the supplied quantity, and ϕ is the standard normal density.

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+5))
)

# mean marginal effect of variable "RM" on the shortage probabilities
#' shortage_probability_marginal(fit, "RM")

# marginal effect at the mean of variable "RM" on the shortage probabilities
shortage_probability_marginal(fit, "CSHS", aggregate = "at_the_mean")

# marginal effect of variable "RM" on the system
shortage_marginal(fit, "RM")
```

market_aggregation *Market side aggregation*

Description

Market side aggregation

Usage

```
aggregate_demand(fit, model, parameters)

## S4 method for signature 'missing,market_model,ANY'
aggregate_demand(model, parameters)

aggregate_supply(fit, model, parameters)

## S4 method for signature 'missing,market_model,ANY'
aggregate_supply(model, parameters)

## S4 method for signature 'market_fit,missing,missing'
aggregate_demand(fit)

## S4 method for signature 'market_fit,missing,missing'
aggregate_supply(fit)
```

Arguments

`fit`	A fitted market model object.
`model`	A model object.
`parameters`	A vector of model's parameters.

Details

Calculates the sample's aggregate demand or supply using the estimated coefficients of a fitted model. Alternatively, the function calculates aggregates using a model and a set of parameters passed separately. If the model's data have multiple distinct subjects at each date (e.g., panel data), aggregation is calculated over subjects per unique date. If the model has time series data, namely a single subject per time point, aggregation is calculated over all time points.

Value

The sum of the estimated demanded or supplied quantities evaluated at the given parameters.

Functions

- `aggregate_demand()`: Demand aggregation.
- `aggregate_supply()`: Supply aggregation.

See Also

demanded_quantities, supplied_quantities

Examples

```
fit <- diseq_basic(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE
)

# get estimated aggregate demand
aggregate_demand(fit)

# simulate the deterministic adjustment model
model <- simulate_model(
  "diseq_deterministic_adjustment", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -0.6, beta_d0 = 9.8, beta_d = c(0.3, -0.2), eta_d = c(0.6, -0.1),
    # supply coefficients
    alpha_s = 0.2, beta_s0 = 4.1, beta_s = c(0.9), eta_s = c(-0.5, 0.2),
    # price equation coefficients
    gamma = 0.9
  ),
  seed = 1356
)

# estimate the model object
fit <- estimate(model)

# get estimated aggregate demand
aggregate_demand(fit)

# get estimated aggregate demand
aggregate_supply(fit)
```

---

market_descriptives *Market force data descriptive statistics*

Description

Market force data descriptive statistics

Usage

```
demand_descriptives(object)

supply_descriptives(object)

## S4 method for signature 'market_model'
demand_descriptives(object)

## S4 method for signature 'market_model'
supply_descriptives(object)
```

Arguments

`object`	A model object.

Details

Calculates and returns basic descriptive statistics for the model's demand or supply side data. Factor variables are excluded from the calculations. The function calculates and returns:

- `nobs` Number of observations.
- `nmval` Number of missing values.
- `min` Minimum observation.
- `max` Maximum observation.
- `range` Observations' range.
- `sum` Sum of observations.
- `median` Median observation.
- `mean` Mean observation.
- `mean_se` Mean squared error.
- `mean_ce` Confidence interval bound.
- `var` Variance.
- `sd` Standard deviation.
- `coef_var` Coefficient of variation.

Value

A data frame containing descriptive statistics.

Functions

- `demand_descriptives()`: Demand descriptive statistics.
- `supply_descriptives()`: Supply descriptive statistics.

Examples

```
# initialize the basic model using the houses dataset
model <- new(
  "diseq_basic", # model type
  subject = ID, time = TREND, quantity = HS, price = RM,
  demand = RM + TREND + W + CSHS + L1RM + L2RM + MONTH,
  supply = RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(), # data
  correlated_shocks = FALSE # allow shocks to be correlated
)

# get descriptive statistics of demand side variables
demand_descriptives(model)

# get descriptive statistics of supply side variables
supply_descriptives(model)
```

---

market_fit-class *Market model fit*

Description

This is the estimation output class for all market models of the package. It couples a market model object with estimation results. It provides a common user interface for accessing estimation results, irrespective of the underlying market model used. The estimation results are intended to be accessed by passing `market_fit` objects to methods such as `plot`, `summary`, and `logLik`.

Details

The `market_fit` class composes the `market_models` class with the estimation results obtained by `optim`, `lm` or `GSL`. All the public functionality of the underlying market model is also directly accessible from the output class.

Furthermore, the class is responsible for harmonizing the heterogeneous outputs resulting from different estimation methods of market models. For example, a `2SLS` estimation of the `equilibrium_model` returns a list of linear regression models (the first stage, demand, and supply models), while the maximum likelihood estimation of `diseq_basic` returns an `optim` list. In both cases, the `market_fit` stores the estimation output in the member `fit` of type `list` and produces additional harmonized list elements. Methods of the class examine the type of the `fit` and direct execution accordingly to different branches to produce a unified experience for the caller.

Slots

`model` The underlying market model object.

`fit` A list holding estimation outputs.

See Also

`market_models`

Examples

```
# estimate an equilibrium  model using the houses dataset
fit <- equilibrium_model(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  estimation_options = list(method = "2SLS")
)

# access an method of the underlying model
aggregate_demand(fit)

# summary of results
summary(fit)
```

`market_models` *Market model classes*

Description

Basic disequilibrium model with unknown sample separation (`diseq_basic`): The basic disequilibrium model consists of three equations. Two of them are the demand and supply equations. In addition, the model replaces the market clearing condition with the short side rule. The model is estimated using full information maximum likelihood.

$$D_{nt} = X'_{d,nt}\beta_d + u_{d,nt},$$
$$S_{nt} = X'_{s,nt}\beta_s + u_{s,nt},$$
$$Q_{nt} = \min\{D_{nt}, S_{nt}\}.$$

Disequilibrium model with deterministic price dynamics (`diseq_deterministic_adjustment`): The disequilibrium model with deterministic price adjustment consists of four equations. The two market equations, the short side rule and price evolution equation. The first two equations are stochastic. The price equation is deterministic. The sample is separated based on the sign of the price changes as in the `diseq_directional` model. The model is estimated using full information maximum likelihood.

$$D_{nt} = X'_{d,nt}\beta_d + P_{nt}\alpha_d + u_{d,nt},$$
$$S_{nt} = X'_{s,nt}\beta_s + P_{nt}\alpha_s + u_{s,nt},$$
$$Q_{nt} = \min\{D_{nt}, S_{nt}\},$$
$$\Delta P_{nt} = \frac{1}{\gamma}\left(D_{nt} - S_{nt}\right).$$

Directional disequilibrium model with sample separation (`diseq_directional`)**:** The directional disequilibrium model consists of three equations and a separation rule. The market is described by a linear demand, a linear supply equation and the short side rule. The separation rule splits the sample into states of excess supply and excess demand. If a price change is positive at the time point of the observation, then the observation is classified as being in an excess demand state. Otherwise, it is assumed that it represents an excess supply state. The model is estimated using full information maximum likelihood.

$$D_{nt} = X_{d,nt}'\beta_d + u_{d,nt},$$

$$S_{nt} = X_{s,nt}'\beta_s + u_{s,nt},$$

$$Q_{nt} = \min\{D_{nt}, S_{nt}\},$$

$$\Delta P_{nt} \geq 0 \Longrightarrow D_{nt} \geq S_{nt}.$$

Disequilibrium model with stochastic price dynamics (`diseq_stochastic_adjustment`)**:** The disequilibrium model with stochastic price adjustment is described by a system of four equations. Three of of them form a stochastic linear system of market equations equations coupled with a stochastic price evolution equation. The fourth equation is the short side rule. In contrast to the deterministic counterpart, the model does not impose any separation rule on the sample. It is estimated using full information maximum likelihood.

$$D_{nt} = X_{d,nt}'\beta_d + P_{nt}\alpha_d + u_{d,nt},$$

$$S_{nt} = X_{s,nt}'\beta_s + P_{nt}\alpha_s + u_{s,nt},$$

$$Q_{nt} = \min\{D_{nt}, S_{nt}\},$$

$$\Delta P_{nt} = \frac{1}{\gamma}\left(D_{nt} - S_{nt}\right) + X_{p,nt}'\beta_p + u_{p,nt}.$$

Equilibrium model (`equilibrium_model`)**:** The equilibrium model consists of thee equations. The demand, the supply and the market clearing equations. The model can be estimated using both full information maximum likelihood and two-stage least squares.

$$D_{nt} = X_{d,nt}'\beta_d + P_{nt}\alpha_d + u_{d,nt},$$

$$S_{nt} = X_{s,nt}'\beta_s + P_{nt}\alpha_s + u_{s,nt},$$

$$Q_{nt} = D_{nt} = S_{nt}.$$

A necessary identification condition is that there is at least one control that is exclusively part of the demand and one control that is exclusively part of the supply equation. In the first stage of the two-stage least square estimation, prices are regressed on remaining controls from both the demand and supply equations. In the second stage, the demand and supply equation is estimated using the fitted prices instead of the observed.

Slots

- `logger` Logger object.
- `subject_columns` Column name for the subject identifier.
- `time_column` Column name for the time point identifier.
- `explanatory_columns` Vector of explanatory column names for all model's equations.
- `data_columns` Vector of model's data column names. This is the union of the quantity, price and explanatory columns.
- `columns` Vector of primary key and data column names for all model's equations.
- `data` Model data frame.
- `model_name` Model name string.
- `market_type` Market type string.
- `system` Model's system of equations.

See Also

model_initialization

`market_quantities` *Estimated market quantities*

Description

Estimated market quantities

Usage

```
demanded_quantities(fit, model, parameters)

## S4 method for signature 'missing,market_model,ANY'
demanded_quantities(model, parameters)

supplied_quantities(fit, model, parameters)

## S4 method for signature 'missing,market_model,ANY'
supplied_quantities(model, parameters)

## S4 method for signature 'market_fit,missing,missing'
demanded_quantities(fit)

## S4 method for signature 'market_fit,missing,missing'
supplied_quantities(fit)
```

Arguments

`fit`	A fitted model object.
`model`	A model object.
`parameters`	A vector of model's parameters.

Details

Calculates and returns the estimated demanded or supplied quantities for each observation at the passed vector of parameters.

Value

A vector with the market quantities evaluated at the given parameter vector.

Functions

- `demanded_quantities()`: Estimated demanded quantities.
- `supplied_quantities()`: Estimated supplied quantities.

Examples

```
fit <- diseq_basic(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE
)

# get estimated demanded and supplied quantities
head(cbind(
  demanded_quantities(fit),
  supplied_quantities(fit)
))
```

`market_simulation` *Market model simulation*

Description

Market data and model simulation functionality based on the data generating process induced by the market model specifications.

`simulate_data`: Returns a data frame with simulated data from a generating process that matches the passed model string. By default, the simulated observations of the controls are drawn from a normal distribution.

`simulate_model`: Simulates a data frame based on the generating process of the passed model and uses it to initialize a model object. Data are simulated using the `simulate_data` function.

Usage

```
simulate_data(
  model_type_string,
  nobs = NA_integer_,
  tobs = NA_integer_,
  alpha_d = NA_real_,
  beta_d0 = NA_real_,
  beta_d = NA_real_,
  eta_d = NA_real_,
  alpha_s = NA_real_,
  beta_s0 = NA_real_,
  beta_s = NA_real_,
  eta_s = NA_real_,
  gamma = NA_real_,
  beta_p0 = NA_real_,
  beta_p = NA_real_,
  sigma_d = 1,
  sigma_s = 1,
  sigma_p = 1,
  rho_ds = 0,
  rho_dp = 0,
  rho_sp = 0,
  seed = NA_integer_,
  price_generator = function(n) stats::rnorm(n = n),
  control_generator = function(n) stats::rnorm(n = n),
  verbose = 0
)

## S4 method for signature 'ANY'
simulate_data(
  model_type_string,
  nobs = NA_integer_,
  tobs = NA_integer_,
  alpha_d = NA_real_,
  beta_d0 = NA_real_,
  beta_d = NA_real_,
  eta_d = NA_real_,
  alpha_s = NA_real_,
  beta_s0 = NA_real_,
  beta_s = NA_real_,
  eta_s = NA_real_,
  gamma = NA_real_,
  beta_p0 = NA_real_,
  beta_p = NA_real_,
  sigma_d = 1,
  sigma_s = 1,
  sigma_p = 1,
  rho_ds = 0,
  rho_dp = 0,
  rho_sp = 0,
  seed = NA_integer_,
  price_generator = function(n) stats::rnorm(n = n),
  control_generator = function(n) stats::rnorm(n = n),
  verbose = 0
)

simulate_model(
  model_type_string,
  simulation_parameters,
  seed = NA,
  verbose = 0,
  correlated_shocks = TRUE
)

## S4 method for signature 'ANY'
simulate_model(
  model_type_string,
  simulation_parameters,
  seed = NA,
  verbose = 0,
  correlated_shocks = TRUE
)
```

Arguments

`model_type_string`	Model type. It should be among `equilibrium_model`, `diseq_basic`, `diseq_directional`, `diseq_deterministic_adjustment`, and `diseq_stochastic_adjustment`.
`nobs`	Number of simulated entities.
`tobs`	Number of simulated dates.
`alpha_d`	Price coefficient of demand.
`beta_d0`	Constant coefficient of demand.
`beta_d`	Coefficients of exclusive demand controls.
`eta_d`	Demand coefficients of common controls.
`alpha_s`	Price coefficient of supply.
`beta_s0`	Constant coefficient of supply.
`beta_s`	Coefficients of exclusive supply controls.
`eta_s`	Supply coefficients of common controls.
`gamma`	Price equation’s stability factor.
`beta_p0`	Price equation’s constant coefficient.
`beta_p`	Price equation’s control coefficients.
`sigma_d`	Demand shock’s standard deviation.

`sigma_s`	Supply shock's standard deviation.
`sigma_p`	Price equation shock's standard deviation.
`rho_ds`	Demand and supply shocks' correlation coefficient.
`rho_dp`	Demand and price shocks' correlation coefficient.
`rho_sp`	Supply and price shocks' correlation coefficient.
`seed`	Pseudo random number generator seed.
`price_generator`	Pseudo random number generator callback for prices. The default generator is $N(0, 1)$.
`control_generator`	Pseudo random number generator callback for non-price controls. The default generator is $N(0, 1)$.
`verbose`	Verbosity level.
`simulation_parameters`	List of parameters used in model simulation. See the `simulate_data` function for details.
`correlated_shocks`	Should the model be estimated using correlated shocks?

Value

`simulate_data`**:** The simulated data.

`simulate_model`**:** The simulated model

.

Functions

- `simulate_data()`: Simulate model data.
- `simulate_model()`: Simulate model.

Examples

```
model <- simulate_model(
  "diseq_stochastic_adjustment", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -0.1, beta_d0 = 9.8, beta_d = c(0.3, -0.2), eta_d = c(0.6, -0.1),
    # supply coefficients
    alpha_s = 0.1, beta_s0 = 6.1, beta_s = c(0.9), eta_s = c(-0.5, 0.2),
    # price equation coefficients
    gamma = 1.2, beta_p0 = 3.1, beta_p = c(0.8)
  ),
  seed = 31
)
```

model_description *Short model and market descriptions*

Description

`name`**:** A unique identifying string for the model.

`describe`**:** A short (one-liner) description of the market model.

`market_type`**:** A market type string (equilibrium or disequilibrium) for a given model.

Usage

```
name(object)

describe(object)

market_type(object)

## S4 method for signature 'market_model'
name(object)

## S4 method for signature 'market_model'
describe(object)

## S4 method for signature 'market_model'
market_type(object)

## S4 method for signature 'market_fit'
name(object)

## S4 method for signature 'market_fit'
describe(object)

## S4 method for signature 'market_fit'
market_type(object)
```

Arguments

`object`	A model object.

Value

`name`**:** The model's name.

`describe`**:** The model's description.

`market_type`**:** The model's market type.

Functions

- `name()`: Model name
- `describe()`: Model description
- `market_type()`: Market type

Examples

```
# initialize the equilibrium using the houses dataset
model <- new(
  "diseq_basic", # model type
  subject = ID, time = TREND, quantity = HS, price = RM,
  demand = RM + TREND + W + CSHS + L1RM + L2RM + MONTH,
  supply = RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses()
)

# model name
name(model)
# model description
describe(model)
# market type
market_type(model)
```

`model_initialization` *Model initialization*

Description

Model initialization

Usage

```
## S4 method for signature 'diseq_basic'
initialize(
  .Object,
  quantity,
  price,
  demand,
  supply,
  subject,
  time,
  data,
  correlated_shocks = TRUE,
  verbose = 0
)

## S4 method for signature 'diseq_deterministic_adjustment'
```

```
initialize(
  .Object,
  quantity,
  price,
  demand,
  supply,
  subject,
  time,
  data,
  correlated_shocks = TRUE,
  verbose = 0
)

## S4 method for signature 'diseq_directional'
initialize(
  .Object,
  quantity,
  price,
  demand,
  supply,
  subject,
  time,
  data,
  correlated_shocks = TRUE,
  verbose = 0
)

## S4 method for signature 'diseq_stochastic_adjustment'
initialize(
  .Object,
  quantity,
  price,
  demand,
  supply,
  price_dynamics,
  subject,
  time,
  data,
  correlated_shocks = TRUE,
  verbose = 0
)

## S4 method for signature 'equilibrium_model'
initialize(
  .Object,
  quantity,
  price,
  demand,
  supply,
  subject,
  time,
  data,
  correlated_shocks = TRUE,
  verbose = 0
)
```

Arguments

`.Object`	The object to be Constructed.
`quantity`	The quantity variable of the system.
`price`	The price variable of the system.
`demand`	A formula representation of the right hand side of the demand equation.
`supply`	A formula representation of the right hand side of the supply equation.
`subject`	The subject identifier of the data set.
`time`	The time identifier of the data set.
`data`	The data set.
`correlated_shocks`	Should the model be estimated using correlated shocks?
`verbose`	Verbosity level.
`price_dynamics`	A formula representation of the price equation.

Details

The following two subsections describe the common initialization steps of all market model classes.

Variable construction: The constructor prepares the model's variables using the passed specifications. The specification variables are expected to be of type `language`. The right hand side specifications of the system are expected to follow the syntax of `formula`. The construction of the model's data uses the variables extracted by these specification. The demand variables are extracted by a formula that uses the `quantity` on the left hand side and the `demand` on the right hand side of the formula. The supply variables are constructed by the `quantity` and the `supply` inputs. In the case of the `diseq_stochastic_adjustment` model, the price dynamics' variables are extracted using the `price dynamics` input. The `price dynamics` for the `diseq_stochastic_adjustment` should contain only terms other than that of excess demand. The excess demand term of the price equation is automatically generated by the constructor.

Data preparation: 1. If the passed data set contains rows with NA values, they are dropped. If the verbosity level allows warnings, a warning is emitted reporting how many rows were dropped.

2. After dropping the rows, factor levels may be invalidated. If needed, the constructor readjusts the factor variables by removing the unobserved levels. Factor indicators and interaction terms are automatically created.

3. The primary key column is constructed by pasting the values of the columns of the subject and time variables.

4. In the cases of the diseq_directional, diseq_deterministic_adjustment, and the diseq_stochastic_adjustmen models, a column with lagged prices is constructed. Since lagged prices are unavailable for the observations of the first time point, these observations are dropped. If the verbosity level allows the emission of information messages, the constructor prints the number of dropped observations.
5. In the cases of the diseq_directional and the diseq_stochastic_adjustment models, a column with price differences is created.

Value

The initialized model.

Functions

- initialize(diseq_basic): Basic disequilibrium model base constructor
- initialize(diseq_deterministic_adjustment): Disequilibrium model with deterministic price adjustment constructor
- initialize(diseq_directional): Directional disequilibrium model base constructor
- initialize(diseq_stochastic_adjustment): Disequilibrium model with stochastic price adjustment constructor
- initialize(equilibrium_model): Equilibrium model constructor

Examples

```
simulated_data <- simulate_data(
  "diseq_basic", 500, 3, # model type, observed entities, observed time points
  -0.9, 8.9, c(0.3, -0.2), c(-0.03, -0.01), # demand coefficients
  0.9, 6.2, c(0.03), c(-0.05, 0.02) # supply coefficients
)

# initialize the model
model <- new(
  "diseq_basic", # model type
  subject = id, time = date, quantity = Q, price = P,
  demand = P + Xd1 + Xd2 + X1 + X2, supply = P + Xs1 + X1 + X2,
  simulated_data, # data
  correlated_shocks = FALSE # use independent shocks
)

show(model)
simulated_data <- simulate_data(
  # model type, observed entities and time points
  "diseq_deterministic_adjustment", 500, 3,
  # demand coefficients
  -0.9, 8.9, c(0.03, -0.02), c(-0.03, -0.01),
  # supply coefficients
  0.9, 4.2, c(0.03), c(0.05, 0.02),
  # price adjustment coefficient
  1.4
)
```

```
# initialize the model
model <- new(
  "diseq_deterministic_adjustment", # model type
  subject = id, time = date, quantity = Q, price = P,
  demand = P + Xd1 + Xd2 + X1 + X2, supply = P + Xs1 + X1 + X2,
  simulated_data, # data
  correlated_shocks = TRUE # allow shocks to be correlated
)

show(model)

simulated_data <- simulate_data(
  "diseq_directional", 500, 3, # model type, observed entities, observed time points
  -0.2, 4.3, c(0.03, 0.02), c(0.03, 0.01), # demand coefficients
  0.0, 4.0, c(0.03), c(0.05, 0.02) # supply coefficients
)

# in the directional model prices cannot be included in both demand and supply
model <- new(
  "diseq_directional", # model type
  subject = id, time = date, quantity = Q, price = P,
  demand = P + Xd1 + Xd2 + X1 + X2, supply = Xs1 + X1 + X2,
  simulated_data, # data
  correlated_shocks = TRUE # allow shocks to be correlated
)

show(model)

simulated_data <- simulate_data(
  # model type, observed entities and time points
  "diseq_stochastic_adjustment", 500, 3,
  # demand coefficients
  -0.1, 9.8, c(0.3, -0.2), c(0.6, 0.1),
  # supply coefficients
  0.1, 7.1, c(0.9), c(-0.5, 0.2),
  # price adjustment coefficient
  1.4, 3.1, c(0.8)
)

# initialize the model
model <- new(
  "diseq_stochastic_adjustment", # model type
  subject = id, time = date, quantity = Q, price = P,
  demand = P + Xd1 + Xd2 + X1 + X2, supply = P + Xs1 + X1 + X2,
  price_dynamics = Xp1,
  simulated_data, # data
  correlated_shocks = TRUE # allow shocks to be correlated
)

show(model)
simulated_data <- simulate_data(
  "equilibrium_model", 500, 3, # model type, observed entities and time points
  -0.9, 14.9, c(0.3, -0.2), c(-0.03, -0.01), # demand coefficients
  0.9, 3.2, c(0.3), c(0.5, 0.02) # supply coefficients
)

# initialize the model
model <- new(
  "equilibrium_model", # model type
  subject = id, time = date, quantity = Q, price = P,
  demand = P + Xd1 + Xd2 + X1 + X2, supply = P + Xs1 + X1 + X2,
  simulated_data, # data
  correlated_shocks = TRUE # allow shocks to be correlated
)

show(model)
```

`model_likelihoods` *Model likelihoods and derivatives*

Description

Methods that calculate the likelihoods, scores, gradients, and Hessians of market models. The likelihood functions are based on Maddala and Nelson (1974) doi:10.2307/1914215. The likelihoods, gradient, and Hessian expressions that the function uses are derived in Karapanagiotis (2020) doi:10.2139/ssrn.3525622.

`log_likelihood`: Returns the log-likelihood. The function calculates the model's log likelihood by evaluating the log likelihood of each observation in the sample and summing the evaluation results.

`gradient`: Returns the gradient of the log-likelihood evaluated at the passed parameters.

`hessian`: Returns the hessian of the log-likelihood evaluated at the passed parameters.

`scores`: It calculates the gradient of the likelihood at the given parameter point for each observation in the sample. It, therefore, returns an n x k matrix, where n denotes the number of observations in the sample and k the number of estimated parameters. The ordering of the parameters is the same as the one that is used in the summary of the results. The method can be called either using directly a fitted model object, or by separately providing a model object and a parameter vector.

Usage

```
log_likelihood(object, parameters)

gradient(object, parameters)

hessian(object, parameters)

scores(object, parameters, fit)
```

```
## S4 method for signature 'diseq_basic'
log_likelihood(object, parameters)

## S4 method for signature 'diseq_basic'
gradient(object, parameters)

## S4 method for signature 'diseq_basic,ANY,ANY'
scores(object, parameters)

## S4 method for signature 'diseq_deterministic_adjustment'
log_likelihood(object, parameters)

## S4 method for signature 'diseq_deterministic_adjustment'
gradient(object, parameters)

## S4 method for signature 'diseq_deterministic_adjustment,ANY,ANY'
scores(object, parameters)

## S4 method for signature 'diseq_directional'
log_likelihood(object, parameters)

## S4 method for signature 'diseq_directional'
gradient(object, parameters)

## S4 method for signature 'diseq_directional,ANY,ANY'
scores(object, parameters)

## S4 method for signature 'diseq_stochastic_adjustment'
log_likelihood(object, parameters)

## S4 method for signature 'diseq_stochastic_adjustment'
gradient(object, parameters)

## S4 method for signature 'diseq_stochastic_adjustment,ANY,ANY'
scores(object, parameters)

## S4 method for signature 'equilibrium_model'
log_likelihood(object, parameters)

## S4 method for signature 'equilibrium_model'
gradient(object, parameters)

## S4 method for signature 'equilibrium_model,ANY,ANY'
scores(object, parameters)

## S4 method for signature 'diseq_basic'
hessian(object, parameters)
```

```
## S4 method for signature 'diseq_directional'
hessian(object, parameters)

## S4 method for signature 'missing,missing,market_fit'
scores(fit)
```

Arguments

`object`	A model object.
`parameters`	A vector of parameters at which the function is to be evaluated.
`fit`	A fitted model object.

Value

`log_likelihood`**:** The sum of the likelihoods evaluated for each observation.

`gradient`**:** The log likelihood's gradient.

`hessian`**:** The log likelihood's hessian.

`scores`**:** The score matrix.

Examples

```
model <- simulate_model(
  "diseq_basic", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -0.9, beta_d0 = 8.9, beta_d = c(0.6), eta_d = c(-0.2),
    # supply coefficients
    alpha_s = 0.9, beta_s0 = 7.9, beta_s = c(0.03, 1.2), eta_s = c(0.1)
  ),
  seed = 7523
)

# estimate the model object (BFGS is used by default)
fit <- estimate(model)

# Calculate the score matrix
head(scores(model, coef(fit)))
```

ncoef *Number of coefficients*

Description

Returns the number of model's coefficients. This is the sum of demand, supply, price equation, and the variance-covariance matrix coefficients.

Usage

```
ncoef(object)

## S4 method for signature 'market_model'
ncoef(object)

## S4 method for signature 'market_fit'
ncoef(object)
```

Arguments

`object`	A model object.

Value

The number of model coefficients.

Examples

```
model <- new(
  "diseq_basic", # model type
  subject = ID, time = TREND, quantity = HS, price = RM,
  demand = RM + TREND + W + CSHS + L1RM + L2RM + MONTH,
  supply = RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses()
)

# get the number of model coefficients
ncoef(model)
```

nobs,market_model-method

Number of observations

Description

Returns the number of observations that are used by an initialized model. If there are missing values, the number of used observations may differ from the numbers of observations of the data set that was passed to the model's initialization.

Usage

```
## S4 method for signature 'market_model'
nobs(object)

## S4 method for signature 'market_fit'
nobs(object)
```

Arguments

object	A model object.

Value

The number of used observations.

Examples

```
model <- new(
  "diseq_basic", # model type
  subject = ID, time = TREND, quantity = HS, price = RM,
  demand = RM + TREND + W + CSHS + L1RM + L2RM + MONTH,
  supply = RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses()
)

# get the number observations
nobs(model)
```

plot,market_fit-method

Plots the fitted model

Description

Displays a graphical illustration of the passed fitted model object. The function creates a scatter plot of quantity-price pairs for the records corresponding to the given subject and time identifiers. Then, it plots the average fitted demand and supply quantities for the same data subset letting prices vary between the minimum and maximum price points observed in the data subset.

Usage

```
## S4 method for signature 'market_fit'
plot(x, subject, time, show_scatter = TRUE, ...)
```

Arguments

`x`	A model object.
`subject`	A vector of subject identifiers to be used in the visualization.
`time`	A vector of time identifiers to be used in the visualization.
`show_scatter`	Should the price-quantity scatter be plotted? By default `TRUE`.
`...`	Additional parameters to be used for styling the figure. Specifically `xlab`, `ylab`, and `main` are currently handled by the function.

Details

If the `subject` argument is missing, all subjects are used. If the `time` argument is missing, all time points are used. The scatter plot of the quantity-price data can be suppressed by setting `show_scatter = FALSE`.

Value

No return value, called for for side effects (visualization).

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+6))
)
```

```
# show model's illustration plot
plot(fit)
```

---

shortage_analysis *Analysis of shortages*

Description

The following methods offer functionality for analyzing estimated shortages of the market models. The methods can be called either using directly a fitted model object, or by separately providing a model object and a parameter vector.

Usage

```
shortages(fit, model, parameters)

normalized_shortages(fit, model, parameters)

relative_shortages(fit, model, parameters)

shortage_probabilities(fit, model, parameters)

shortage_indicators(fit, model, parameters)

shortage_standard_deviation(fit, model, parameters)

## S4 method for signature 'missing,market_model,ANY'
shortages(model, parameters)

## S4 method for signature 'missing,market_model,ANY'
normalized_shortages(model, parameters)

## S4 method for signature 'missing,market_model,ANY'
relative_shortages(model, parameters)

## S4 method for signature 'missing,market_model,ANY'
shortage_probabilities(model, parameters)

## S4 method for signature 'missing,market_model,ANY'
shortage_indicators(model, parameters)

## S4 method for signature 'missing,market_model,ANY'
shortage_standard_deviation(model, parameters)

## S4 method for signature 'missing,diseq_stochastic_adjustment,ANY'
shortage_standard_deviation(model, parameters)
```

```
## S4 method for signature 'market_fit,missing,missing'
shortages(fit)

## S4 method for signature 'market_fit,missing,missing'
normalized_shortages(fit)

## S4 method for signature 'market_fit,missing,missing'
relative_shortages(fit)

## S4 method for signature 'market_fit,missing,missing'
shortage_probabilities(fit)

## S4 method for signature 'market_fit,missing,missing'
shortage_indicators(fit)

## S4 method for signature 'market_fit,missing,missing'
shortage_standard_deviation(fit)
```

Arguments

`fit`	A fitted model object.
`model`	A market model object.
`parameters`	A vector of parameters at which the shortages are evaluated.

Details

shortages: Returns the predicted shortages at a given point.

normalized_shortages: Returns the shortages normalized by the variance of the difference of the shocks at a given point.

relative_shortages: Returns the shortages normalized by the supplied quantity at a given point.

shortage_probabilities: Returns the shortage probabilities, i.e. the probabilities of an observation coming from an excess demand state, at the given point.

shortage_indicators: Returns a vector of indicators (Boolean values) for each observation. An element of the vector is TRUE for observations at which the estimated shortages are non-negative, i.e. the market at in an excess demand state. The remaining elements are FALSE. The evaluation of the shortages is performed using the passed parameter vector.

shortage_standard_deviation: Returns the standard deviation of excess demand.

Value

A vector with the (estimated) shortages.

Functions

- `shortages()`: Shortages.
- `normalized_shortages()`: Normalized shortages.
- `relative_shortages()`: Relative shortages.
- `shortage_probabilities()`: Shortage probabilities.
- `shortage_indicators()`: Shortage indicators.
- `shortage_standard_deviation()`: Shortage standard deviation.

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+5))
)

# get estimated normalized shortages
head(normalized_shortages(fit))

# get estimated relative shortages
head(relative_shortages(fit))

# get the estimated shortage probabilities
head(shortage_probabilities(fit))

# get the estimated shortage indicators
head(shortage_indicators(fit))

# get the estimated shortages
head(shortages(fit))

# get the estimated shortage standard deviation
shortage_standard_deviation(fit)
```

`show,market_model-method`

Prints a short description of the model

Description

Sends basic information about the model to standard output.

Usage

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

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

Arguments

`object`	A model object.

Value

No return value, called for side effects (print basic model information).

Examples

```
fit <- equilibrium_model(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  estimation_options = list(method = "2SLS")
)

# print model information
show(fit@model)

# print fit information
show(fit)
```

`single_call_estimation` *Single call estimation*

Description

Single call estimation

Usage

```
diseq_basic(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
```

```
)

## S4 method for signature 'formula'
diseq_basic(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

diseq_deterministic_adjustment(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

## S4 method for signature 'formula'
diseq_deterministic_adjustment(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

diseq_directional(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

## S4 method for signature 'formula'
diseq_directional(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

diseq_stochastic_adjustment(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

## S4 method for signature 'formula'
diseq_stochastic_adjustment(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

equilibrium_model(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)

## S4 method for signature 'formula'
equilibrium_model(
  specification,
  data,
  correlated_shocks = TRUE,
  verbose = 0,
  estimation_options = list()
)
```

Arguments

`specification`	The model's formula.
`data`	The data to be used with the model.
`correlated_shocks`	Should the model's system entail correlated shocks? By default the argument is set to `TRUE`.
`verbose`	The verbosity with which operations on the model print messages. By default the value is set to `0`, which prints only errors.
`estimation_options`	A list with options to be used in the estimation call. See `estimate` for the available options.

Details

The functions of this section combine model initialization and estimation into a single call. They also provide a less verbose interface to the functionality of the package. The functions expect a formula following the specification described in formula, a dataset, and optionally further initialization and estimation options (see model initialization and model estimation respectively).

Estimation options are expected to be given in the argument `estimation_options` in a form of a `list`. The list names should correspond to variables of the `estimate` function. As a result, optimization options, which are customized using the `control` argument of `estimate` can be passed as an element of `estimation_options`.

Each of these functions parses the given formula, initializes the model specified by the function's name, fits the model to the given data using the estimation options and returns fitted model.

Value

The fitted model.

Functions

- `diseq_basic()`: Basic disequilibrium model.
- `diseq_deterministic_adjustment()`: Disequilibrium model with deterministic price adjustments.
- `diseq_directional()`: Directional disequilibrium model.
- `diseq_stochastic_adjustment()`: Disequilibrium model with stochastic price adjustments.
- `equilibrium_model()`: Equilibrium model

Examples

```
# An example of estimating the equilibrium model
eq <- equilibrium_model(
  HS | RM | ID | TREND ~ RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
    RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(), estimation_options = list(control = list(maxit = 5000))
)

# An example of estimating the deterministic adjustment model
da <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~ RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
    RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  verbose = 2,
  estimation_options = list(control = list(maxit = 5000))
)

# An example of estimating the directional model
dr <- diseq_directional(
  HS | RM | ID | TREND ~ TREND + W + CSHS + L1RM + L2RM |
    RM + TREND + W + MA6DSF + MA3DHF + MONTH,
  fair_houses(), estimation_options = list(
    method = "Nelder-Mead", control = list(maxit = 5000)
  )
)
```

```
# An example of estimating the basic model
start <- coef(eq)
start <- start[names(start) != "RHO"]
bs <- diseq_basic(
  HS | RM | ID | TREND ~ RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
    RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(), verbose = 2, correlated_shocks = FALSE,
  estimation_options = list(
    start = start,
    control = list(maxit = 5000)
  )
)

# An example of estimating the stochastic adjustment model
sa <- diseq_stochastic_adjustment(
  HS | RM | ID | TREND ~ RM + TREND + W + CSHS + MONTH |
    RM + TREND + W + L1RM + L2RM + MA6DSF + MA3DHF + MONTH |
    TREND + L2RM + L3RM,
  fair_houses() |> dplyr::mutate(L3RM = dplyr::lag(RM, 3)),
  correlated_shocks = FALSE,
  estimation_options = list(
    control = list(maxit = 5000), standard_errors = c("W")
  )
)
```

`summary` *Model and fit summaries*

Description

Methods that summarize models and their estimates.

`market_model`: Prints basic information about the passed model object. In addition to the output of the `show` method, `summary` prints

- the number of observations,
- the number of observations in each equation for models with sample separation, and
- various categories of variables.

`market_fit`: Prints basic information about the passed model fit. In addition to the output of the model's `summary` method, the function prints basic estimation results. For a maximum likelihood estimation, the function prints

- the used optimization method,
- the maximum number of allowed iterations,
- the relative convergence tolerance (see `optim`),
- the convergence status,
- the initializing parameter values,

- the estimated coefficients, their standard errors, Z values, and P values, and
- $-2 \log L$ evaluated at the maximum.

For a linear estimation of the equilibrium system, the function prints

- the used method,
- the summary of the first stage regression,
- the summary of the demand (second stage) regression, and
- the summary of the supply (second stage) regression.

Usage

```
## S4 method for signature 'market_model'
summary(object)

## S4 method for signature 'market_fit'
summary(object)
```

Arguments

`object`	An object to be summarized.

Value

No return value, called for for side effects (print summary).

Methods (by class)

- `summary(market_model)`: Summarizes the model.
- `summary(market_fit)`: Summarizes the model's fit.

Examples

```
model <- simulate_model(
  "diseq_stochastic_adjustment", list(
    # observed entities, observed time points
    nobs = 500, tobs = 3,
    # demand coefficients
    alpha_d = -0.1, beta_d0 = 9.8, beta_d = c(0.3, -0.2), eta_d = c(0.6, -0.1),
    # supply coefficients
    alpha_s = 0.1, beta_s0 = 5.1, beta_s = c(0.9), eta_s = c(-0.5, 0.2),
    # price equation coefficients
    gamma = 1.2, beta_p0 = 3.1, beta_p = c(0.8)
  ),
  seed = 556
)

# print model summary
summary(model)
```

```
# estimate
fit <- estimate(model)

# print estimation summary
summary(fit)
```

`vcov,market_fit-method`
Variance-covariance matrix for a fitted market model

Description

Returns the variance-covariance matrix of the estimated coefficients for the fitted model. Specializes the `vcov` function for fitted market models.

Usage

```
## S4 method for signature 'market_fit'
vcov(object)
```

Arguments

`object`	A fitted model object.

Value

A matrix of covariances for the estimated model coefficients.

Examples

```
# estimate a model using the houses dataset
fit <- diseq_deterministic_adjustment(
  HS | RM | ID | TREND ~
    RM + TREND + W + CSHS + L1RM + L2RM + MONTH |
      RM + TREND + W + L1RM + MA6DSF + MA3DHF + MONTH,
  fair_houses(),
  correlated_shocks = FALSE,
  estimation_options = list(control = list(maxit = 1e+6))
)

# access the variance-covariance matrix
head(vcov(fit))
```

Index