Package: mergersim (via r-universe)

July 3, 2026

Title Merger Simulation and Calibration

Version 0.1.0

Description Analyze mergers between firms. Models of competition include differentiated Bertrand price-setting, Nash bargaining, and second score auctions, as implemented in Panhans and Taragin (2023) <doi:10.1016/j.ijindorg.2023.102986>. Calibrates demand systems including standard logit, nested logit, and generalized nested logit as implemented in Panhans and Wiemer (2026) <doi:10.1093/joclec/nhaf037>.

License CC0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports BB, numDeriv, rootSolve, stats

Suggests antitrust, knitr, rmarkdown

VignetteBuilder knitr

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

Date/Publication 2026-06-26 14:35:46 UTC

RemoteUrl https://github.com/mpanhans/mergersim

RemoteRef HEAD

RemoteSha 8f324c06ba710392614495abb451f507f1355468

Contents

bargain_calibrate 2
bargain_foc 3
bargain_foc_vert_sim 4
bargain_NP_vert_seq 5
bargain_NP_vert_seq_gnl 7
bargain_vert_seq_calibrate 9
bargain_vert_sim_calibrate 10

bargain_vert_sim_calibrate_gnl 11
bertrand_calibrate 13
bertrand_calibrate_gnl 14
bertrand_foc 16
bertrand_foc_vert 17
bertrand_foc_vert_gnl 18
bertrand_vert_calibrate 20
diversion_calc 20
match_share 21
share_calc 22
ssa_calibrate 23
ssa_calibrate_gnl 24
ssbargain_calibrate 25
ssbargain_foc 27

Index **28**

`bargain_calibrate` *Nash bargaining calibration*

Description

Nash bargaining calibration

Usage

```
bargain_calibrate(
  param,
  own,
  price,
  shares,
  cost,
  weight,
  lambda,
  includeMUI = TRUE
)
```

Arguments

`param`	Price coefficient alpha and mean values delta, parameters to calibrate
`own`	Ownership matrix
`price`	Price
`shares`	Observed market shares
`cost`	Marginal costs for each product
`weight`	Weighting matrix
`lambda`	Bargaining power of the buyer

includeMUI	logical; whether to include marginal utility of income in buyer's payoff, thereby translating dollars to utility. Default is True, interpreted as buyer maximizing utility. Setting equal to False would have interpretation that buyer maximizes profts.

Details

This function calculate the first-order conditions from a Bertrand price-setting model of competition

Value

The first-order conditions

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
c_j <- c(.05,.31,.30)
own_pre = diag(3)
p0 <- c_j*1.1
share1 <- (exp(delta + alpha*p0))/(1+sum(exp(delta + alpha*p0)))
wt_matrix <- diag(c(1,1,1,1000,1000,1000))

bargain_calibrate(param = c(alpha,delta),own = own_pre,price = p0,
shares = share1,cost = c_j, weight = wt_matrix,
lambda = 0.5)
```

---

bargain_foc *Nash bargaining first-order conditions*

Description

Nash bargaining first-order conditions

Usage

```
bargain_foc(price, own, alpha, delta, cost, lambda, includeMUI = TRUE)
```

Arguments

price	Price
own	Ownership matrix
alpha	Price coefficient
delta	Mean values
cost	Marginal costs for each product
lambda	Bargaining power of the buyer

includeMUI — logical; whether to include marginal utility of income in buyer's payoff, thereby translating dollars to utility. Default is True, interpreted as buyer maximizing utility. Setting equal to False would have interpretation that buyer maximizes profts.

Details

This function calculate the first-order conditions from a Bertrand price-setting model of competition

Value

The first-order conditions

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
c_j <- c(.05,.31,.30)
own_pre = diag(3)
p0 <- c_j*1.1

bertrand_foc(price = p0,
own = own_pre, alpha= alpha,
delta = delta, cost = c_j)
```

---

bargain_foc_vert_sim *Nash Product FOCs for use in vertical model with simultaneous timing*

Description

Nash Product FOCs for use in vertical model with simultaneous timing

Usage

```
bargain_foc_vert_sim(
  price_w,
  own_down,
  own_up,
  alpha,
  delta,
  cost_w,
  cost_r,
  lambda,
  price_r,
  sumFOC = TRUE
)
```

Arguments

price_w	Upstream or wholesale prices
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_w	Marginal costs for upstream firm for each product
cost_r	Marginal costs for downstream firm for each product
lambda	Bargaining power of the buyer/retailer
price_r	Retail prices starting values
sumFOC	logical; if true, returns the sum of first-order conditions, if false, returns vector of FOC's for each product

Details

This function calculate the first-order conditions from a Nash bargaining model. For use in a vertical supply chain with simultaneous timing

Value

The first-order conditions

Examples

```
bargain_foc_vert_sim(price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
own_down = paste0("R",rep(c(1,2,3),each=2)),
own_up = paste0("W",rep(c(1,2),3)),
alpha = -0.9, delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
cost_w = rep(.2, times = 6),
cost_r = rep(.1, times = 6),
lambda = 0.5,
price_r = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0))
```

---

bargain_NP_vert_seq *Nash Product for use in vertical model with sequential timing*

Description

Nash Product for use in vertical model with sequential timing

Usage

```
bargain_NP_vert_seq(
  w_start,
  product_max,
  price_w,
  own_down,
  own_up,
  alpha,
  delta,
  cost_w,
  cost_r,
  lambda,
  p_R0,
  sigma,
  showAll = FALSE,
  maxJointProfits = FALSE
)
```

Arguments

w_start	Initial wholesale price for product for which to maximize Nash Product
product_max	Index of product for which to maximize Nash Product
price_w	Upstream or wholesale prices
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_w	Marginal costs for upstream firm for each product
cost_r	Marginal costs for downstream firm for each product
lambda	Bargaining power of the buyer/retailer
p_R0	Retail prices starting values
sigma	Contract type; value between 0 and 1 where 0 is linear price, 1 is two-part tariff
showAll	logical; if true, returns gains for trade for every product
maxJointProfits	Experimental option; if true, will maximize joint profits rather than gains from trade

Details

This function calculate the Nash Product from a Nash bargaining model

Value

The first-order conditions

Examples

```
bargain_NP_vert_seq(w_start = 1.5, product_max = 1,
price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
own_down = paste0("R",rep(c(1,2,3),each=2)),
own_up = paste0("W",rep(c(1,2),3)),
alpha = -0.9, delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
cost_w = rep(.2, times = 6),
cost_r = rep(.1, times = 6),
lambda = 0.5, sigma = 0,
p_R0 = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0))
```

---

bargain_NP_vert_seq_gnl

Nash Product for use in vertical model with sequential timing

---

Description

Nash Product for use in vertical model with sequential timing

Usage

```
bargain_NP_vert_seq_gnl(
  w_start,
  product_max,
  price_w,
  own_down,
  own_up,
  alpha,
  delta,
  cost_w,
  cost_r,
  lambda,
  p_R0,
  sigma,
  nest_allocation,
  mu,
  showAll = FALSE
)
```

Arguments

w_start	Initial wholesale price for product for which to maximize Nash Product
product_max	Index of product for which to maximize Nash Product
price_w	Upstream or wholesale prices
own_down	Ownership matrix for downstream firms

own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_w	Marginal costs for upstream firm for each product
cost_r	Marginal costs for downstream firm for each product
lambda	Bargaining power of the buyer/retailer
p_R0	Retail prices starting values
sigma	Contract type; value between 0 and 1 where 0 is linear price, 1 is two-part tariff
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
showAll	logical; if true, returns gains for trade for every product

Details

This function calculate the Nash Product from a Nash bargaining model. This version allows for nested logit with overlapping nests.

Value

The first-order conditions

Examples

```
own_down <- paste0("R",rep(c(1,2,3),each=2))
own_up <- paste0("W",rep(c(1,2),3))

B1 <- 1 * matrix( c(own_down == "R1",
                    own_down == "R2",
                    own_down == "R3",
                    own_up == "W1",
                    own_up == "W2"),
                  ncol = 5, nrow = 6)
a1 <- B1 * 0.5
mu1 <- rep(1.0,5)

bargain_NP_vert_seq_gnl(w_start = 1.5, product_max = 1,
                        price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
                        own_down = own_down,
                        own_up = own_up,
                        alpha = -0.9, delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
                        cost_w = rep(.2, times = 6),
                        cost_r = rep(.1, times = 6),
                        lambda = 0.5, sigma = 0,
                        p_R0 = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0),
                        nest_allocation = a1, mu = mu1)
```

`bargain_vert_seq_calibrate` *Calibrate costs for Nash bargain in vertical model with sequential timing*

Description

Calibrate costs for Nash bargain in vertical model with sequential timing

Usage

```
bargain_vert_seq_calibrate(
  c_w_val,
  price_w,
  own_down,
  own_up,
  alpha,
  delta,
  lambda,
  cost_r,
  price_r,
  sigma,
  setTol = 0.01,
  setMaxIter = 500,
  showAll = FALSE,
  symmetricCosts = FALSE
)
```

Arguments

`c_w_val`	Wholesale/upstream costs
`price_w`	Upstream or wholesale prices
`own_down`	Ownership matrix for downstream firms
`own_up`	Ownership matrix for upstream firms
`alpha`	Price coefficient
`delta`	Mean values
`lambda`	Bargaining power of the buyer/retailer
`cost_r`	Marginal costs for downstream firm for each product
`price_r`	Retail prices
`sigma`	Contract type; value between 0 and 1 where 0 is linear price, 1 is two-part tariff
`setTol`	tolerance for convergence
`setMaxIter`	Maximum iterations for convergence loop
`showAll`	logical; if true, returns gains for trade for every product
`symmetricCosts`	Constrain costs to be symmetric

Details

This function can be used to calibrate the costs in a Nash bargain which is the upstream market of a vertical supply chain. Assumes logit demand.

Value

The value of objective function

Examples

```
bargain_vert_seq_calibrate(c_w_val = rep(.2, times = 6),
price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
own_down = paste0("R",rep(c(1,2,3),each=2)),
own_up = paste0("W",rep(c(1,2),3)),
alpha = -0.9, delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
lambda = 0.5, cost_r = rep(.1, times = 6),
price_r = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0),
sigma = 0, setTol = 1.5)
```

---

bargain_vert_sim_calibrate

Calibrate lambda for Nash bargain in vertical model with simultaneous timing

---

Description

Calibrate lambda for Nash bargain in vertical model with simultaneous timing

Usage

```
bargain_vert_sim_calibrate(
  lambda,
  price_w,
  own_down,
  own_up,
  alpha,
  delta,
  cost_w,
  cost_r,
  price_r
)
```

Arguments

lambda	Bargaining power of the buyer/retailer
price_w	Upstream or wholesale prices
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_w	Marginal costs for upstream firm for each product
cost_r	Marginal costs for downstream firm for each product
price_r	Retail prices

Details

This function can be used to calibrate the bargaining parameter in a Nash bargain which is the upstream market of a vertical supply chain. Assumes logit demand.

Value

The value of objective function

Examples

```
bargain_vert_sim_calibrate(lambda = 0.4,
price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
own_down = paste0("R",rep(c(1,2,3),each=2)),
own_up = paste0("W",rep(c(1,2),3)),
alpha = -0.9,
delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
cost_w = rep(.2, times = 6),
cost_r = rep(.1, times = 6),
price_r = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0))
```

---

bargain_vert_sim_calibrate_gnl

Calibrate upstream Nash bargain in vertical model with simultaneous timing

---

Description

Calibrate upstream Nash bargain in vertical model with simultaneous timing

Usage

```
bargain_vert_sim_calibrate_gnl(
  param,
  lambda = NA,
  cost_w = NA,
  own_down,
  own_up,
  alpha,
  delta,
  cost_r,
  price_r,
  price_w,
  nest_allocation = NA,
  mu = NA,
  sumFOC = FALSE
)
```

Arguments

param	vector of parameters to calibrate, will be either bargaining parameter or demand parameters, depending on what other information is supplied
lambda	Bargaining power of the buyer/retailer
cost_w	Marginal costs for upstream firm for each product
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_r	Marginal costs for downstream firm for each product
price_r	Retail prices
price_w	Upstream or wholesale prices
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
sumFOC	logical; whether to return the sum of squares of the first-order conditions. Defaults to FALSE, in which case it returns each product first-order condition as a vector.

Details

This function can be used to calibrate the bargaining parameter in a Nash bargain which is the upstream market of a vertical supply chain. Assumes logit demand.

Value

The value of objective function

Examples

```
bargain_vert_sim_calibrate_gnl(param = 0.4, lambda=NA,
cost_w = rep(.2, times = 6),
own_down = paste0("R",rep(c(1,2,3),each=2)),
own_up = paste0("W",rep(c(1,2),3)),
alpha = -0.9,
delta = c(0.2, 0.3, 0.9, 1.0, 0.8, 0.9),
cost_r = rep(.1, times = 6),
price_r = c(2.9, 2.9, 3.0, 3.0, 3.0, 3.0),
price_w = c(1.6, 1.6, 1.6, 1.6, 1.6, 1.6),
nest_allocation=NA, mu=NA, sumFOC = TRUE)
```

---

bertrand_calibrate *Bertrand model calibration*

Description

Bertrand model calibration

Usage

```
bertrand_calibrate(
  param,
  own,
  price,
  shares,
  cost,
  weight,
  returnOutcomes = FALSE
)
```

Arguments

param	Price coefficient alpha parameter to calibrate
own	Ownership matrix
price	Price
shares	Observed market shares
cost	Marginal costs for each product
weight	Weighting matrix
returnOutcomes	logical; should equilibrium objects be returned (mean value parameter, prices, shares, costs) as a list.

Details

This function calculate the first-order conditions from a Bertrand price-setting model of competition. This function is only for standard logit demand. For nested logit or generalized nested logit, see bertrand_calibrate_gnl().

Value

Distance between observed values and model predicted values for prices and shares

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
c_j <- c(.05,.31,.30)

own_pre = diag(3)

p0 <- c_j*1.1

share1 <- (exp(delta + alpha*p0))/(1+sum(exp(delta + alpha*p0)))
x00 <- c(-1)
wt_matrix <- diag(c(1,1,1,1000,1000,1000))

bertrand_calibrate( param = x00,
                    own = own_pre, price = p0,
                    shares = share1, cost  = c_j,
                    weight = wt_matrix)
```

---

bertrand_calibrate_gnl

Calibrate Bertrand model with GNL demand

Description

Calibrate Bertrand model with GNL demand

Usage

```
bertrand_calibrate_gnl(
  param,
  own,
  price,
  shares,
  cost,
  weight = c(1, 1, 1, 1),
  nest_allocation,
  div_matrix,
  mu_constraint_matrix = NA,
  div_calc_marginal = TRUE,
  optimizer = "BBoptim",
  optimizer_c = "optim",
  returnOutcomes = FALSE,
  maxitval = 1500,
```

```
  maxitval_c = 1500,
  fast_version = FALSE
)
```

Arguments

param	Vector of demand parameters (alpha,mu)
own	Ownership matrix
price	Observed prices
shares	Observed market shares
cost	Marginal costs for each product
weight	Vector of length four with weights given to prices, shares, diversions, and costs, respectively. Default is c(1,1,1,1).
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
div_matrix	A matrix of observed diversions from product in row j to product in column k.
mu_constraint_matrix	is a (K-by-K') matrix indicating which nesting parameters are constrained to be equal to each other, where K is the number of nests and K' is the number of freely varying nesting parameters. mu_full = mu_constraint_matrix %*% mu_prime. Where mu_full is a vector of length K of the nesting parameter value for each nest, and mu_prime is a vector of length K' of parameters to be calculated. It must be the case that K is greater than K'.
div_calc_marginal	is a logical if function should match to marginal diversions (if TRUE) or second choice diversions (if FALSE). Default to TRUE.
optimizer	Which optimization routine should be used to find equilibrium prices, either BBoptim or multiroot
optimizer_c	Which optimization routine should be used to calibrate costs, either BBoptim or optim
returnOutcomes	logical; should equilibrium objects be returned (mean value parameter, prices, shares, costs) as a list.
maxitval	Max iterations during price optimization from foc
maxitval_c	Max iterations during cost optimization from foc
fast_version	logical; if True, uses only first-order conditions for products that have non-missing costs. If False, uses the model to predict costs for all products, and then uses all first-order conditions, which takes longer to calibrate. Default is False.

Details

This function calibrates a Bertrand model with generalized nested logit (GNL) demand

Value

Difference between model predicted and observed values of prices, shares, and diversions.

Examples

```
nest1 <- matrix( c(1, 0, 0, 0, 1, 1), ncol = 2, nrow = 3)
divmat <- matrix( c(0, .4, .4, .4, 0, .4, .4, .4, 0), ncol = 3, nrow = 3)

bertrand_calibrate_gnl(param = c(-0.9, 1, 1),
                       own = diag(3),
                       price = c(.05, .34, .33),
                       shares = c( 0.31, 0.27, 0.25),
                       cost = c(.05,.31,.30),
                       weight = c(1,1,1,1),
                       nest_allocation = nest1, div_matrix = divmat)
```

---

bertrand_foc *Bertrand first-order conditions*

Description

Bertrand first-order conditions

Usage

```
bertrand_foc(
  price,
  own,
  alpha,
  delta,
  cost,
  nest_allocation = NA,
  mu = NA,
  sumFOC = FALSE
)

bertrand_foc_c(
  cost,
  price,
  own,
  alpha,
  delta,
  nest_allocation = NA,
  mu = NA,
  sumFOC = FALSE
)
```

Arguments

price	Price
own	Ownership matrix
alpha	Price coefficient
delta	Mean values
cost	Marginal costs for each product
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
sumFOC	logical; whether to return the sum of squares of the first-order conditions. Defaults to FALSE, in which case it returns each product first-order condition as a vector.

Details

This function calculate the first-order conditions from a Bertrand price-setting model of competition

Value

The first-order conditions

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
c_j <- c(.05,.31,.30)

own_pre = diag(3)

p0 <- c_j*1.1

bertrand_foc(price = p0,
             own = own_pre, alpha= alpha,
             delta = delta, cost = c_j)
```

---

bertrand_foc_vert *Downstream Bertrand first-order conditions*

Description

Downstream Bertrand first-order conditions

Usage

```
bertrand_foc_vert(
  price_r,
  own_down,
  own_up,
  alpha,
  delta,
  cost_r,
  price_w,
  cost_w,
  sumFOC = FALSE
)
```

Arguments

price_r	Downstream or retail prices
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_r	Marginal costs for downstream firm for each product
price_w	Upstream or wholesale prices, treated as costs by downstream firms
cost_w	Marginal costs for upstream firm for each product
sumFOC	logical; whether to return the sum of squares of the first-order conditions. Defaults to FALSE, in which case it returns each product first-order condition as a vector.

Details

This function calculates the first-order conditions from a Bertrand price-setting model of competition as the downstream market of a vertical supply chain. Assumes logit demand.

Value

The first-order conditions

---

bertrand_foc_vert_gnl *Downstream Bertrand first-order conditions*

Description

Downstream Bertrand first-order conditions

Usage

```
bertrand_foc_vert_gnl(
  price_r,
  own_down,
  own_up,
  alpha,
  delta,
  cost_r,
  price_w,
  cost_w,
  nest_allocation = NA,
  mu = NA,
  sumFOC = FALSE
)
```

Arguments

price_r	Downstream or retail prices
own_down	Ownership matrix for downstream firms
own_up	Ownership matrix for upstream firms
alpha	Price coefficient
delta	Mean values
cost_r	Marginal costs for downstream firm for each product
price_w	Upstream or wholesale prices, treated as costs by downstream firms
cost_w	Marginal costs for upstream firm for each product
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
sumFOC	logical; whether to return the sum of squares of the first-order conditions. Defaults to FALSE, in which case it returns each product first-order condition as a vector.

Details

This function calculates the first-order conditions from a Bertrand price-setting model of competition as the downstream market of a vertical supply chain. This version allows for generalized nested logit demand.

Value

The first-order conditions

`bertrand_vert_calibrate` *Calibrate Bertrand model demand parameters*

Description

Calibrate Bertrand model demand parameters

Usage

```
bertrand_vert_calibrate(param, own_down, price_r, shares, cost, price_w)
```

Arguments

`param`	Price coefficient, alpha
`own_down`	Ownership matrix for downstream firms
`price_r`	Downstream prices
`shares`	Market shares
`cost`	Marginal costs of downstream firm
`price_w`	Upstream or wholesale prices, treated as costs by downstream firms

Details

This function can be used to calibrate the demand parameters of a Bertrand model of competition that is the downstream market of a vertical supply chain. Wholesale prices are treated as costs by the downstream firms. Assumes logit demand.

Value

Objective function value

`diversion_calc` *Market share calculations*

Description

Calculates choice probabilities based on logit or generalized nested logit demand parameters

Usage

```
diversion_calc(
  price,
  alpha,
  delta,
  nest_allocation = NA,
  mu = NA,
  marginal = FALSE,
  outsideOption = TRUE
)
```

Arguments

price	Price
alpha	Price coefficient
delta	Mean values
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
marginal	logical; if True, diversions are calculated as diversions from a small price increase. If False, diversions are calculated as second choice diversions.
outsideOption	logical; whether to include an outside option in choice set. Default is TRUE.

Details

This function calculates choice probabilities based on demand parameters for a logit or generalized nested logit demand system

Value

Returns vector of choice probabilities for each good

Examples

```
diversion_calc(price=c(2.1,2.4,2.1),alpha=-0.9,delta=c(.81,.93,.82))
```

---

match_share *Market share calculations*

Description

Calculates choice probabilities based on logit or generalized nested logit demand parameters

Usage

```
match_share(price, delta, alpha, nest_allocation = NA, mu = NA, shares_obs)
```

Arguments

price	Price
delta	Mean values
alpha	Price coefficient
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
shares_obs	Observed market shares

Details

This function calculates the difference between model predicted choice probabilities and observed market shares.

Value

Returns vector of difference between predicted shares and observed shares

Examples

```
match_share(price=c(2.1,2.4,2.1), delta=c(.81,.93,.82), alpha=-0.9,
shares_obs = c(.2, .2, .2))
```

---

share_calc *Market share calculations*

Description

Calculates choice probabilities based on logit or generalized nested logit demand parameters

Usage

```
share_calc(
  price,
  delta,
  alpha,
  nest_allocation = NA,
  mu = NA,
  returnLogsum = FALSE,
  outsideOption = TRUE
)
```

Arguments

price	Price
delta	Mean values
alpha	Price coefficient
nest_allocation	For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.
mu	Nesting parameters for each nest
returnLogsum	logical; whether to return the denominator of the choice probabilities (also known as the log-sum term). Defaults to FALSE, in which case the function returns a vector with each product's choice probability.
outsideOption	logical; whether to include an outside option in choice set. Default is TRUE.

Details

This function calculates choice probabilities based on demand parameters for a logit or generalized nested logit demand system

Value

Returns vector of choice probabilities for each good

Examples

```
share_calc(price=c(2.1,2.4,2.1),delta=c(.81,.93,.82),alpha=-0.9)
```

---

ssa_calibrate *Second score auction calibration*

Description

Second score auction calibration

Usage

```
ssa_calibrate(param, own, price, share, cost, weight)
```

Arguments

param	The price coefficient alpha to be calibrated
own	Ownership matrix
price	Price
share	Share
cost	Marginal costs for each product
weight	Weighting matrix

Details

This function calculate the first-order conditions from a second score auction model of competition

Value

The first-order conditions

Examples

```
own_pre = diag(3)
p0 <- c(.05, .34, .33)
share1 <- c( 0.31, 0.27, 0.25)
c_j <- c(.05,.31,.30)
wt_matrix <- diag(c(1,1,1))

ssa_calibrate(param = -1,own = own_pre,price=p0,share=share1,cost=c_j,
              weight = wt_matrix)
```

---

ssa_calibrate_gnl *Calibrate second score auction model with GNL demand*

Description

Calibrate second score auction model with GNL demand

Usage

```
ssa_calibrate_gnl(
  param,
  own,
  price,
  shares,
  cost,
  weight,
  nest_allocation,
  mu_constraint_matrix = NA
)
```

Arguments

param	Vector of demand parameters (alpha,mu)
own	Ownership matrix
price	Observed prices
shares	Observed market shares
cost	Marginal costs for each product

`weight` Weighting matrix of dimensions J-by-J

`nest_allocation`
For generalized nested logit demand, a J-by-K matrix where each element (j,k) designates the membership of good j in nest k. Rows should sum to 1.

`mu_constraint_matrix`
is a (K-by-K') matrix indicating which nesting parameters are constrained to be equal to each other, where K is the number of nests and K' is the number of freely varying nesting parameters. mu_full = mu_constraint_matrix %*% mu_prime. Where mu_full is a vector of length K of the nesting parameter value for each nest, and mu_prime is a vector of length K' of parameters to be calculated. It must be the case that K is greater than K'.

Details

This function calibrates a second score auction model with generalized nested logit (GNL) demand

Value

Difference between model predicted and observed values of prices, shares, and diversions.

Examples

```
nest1 <- matrix( c(1, 0, 0, 0, 1, 1), ncol = 2, nrow = 3)

ssa_calibrate_gnl(param = c(-0.9, 1, 1),
own = diag(3),
price = c(.05, .34, .33),
shares = c( 0.31, 0.27, 0.25),
cost = c(.05,.31,.30),
weight = diag(c(1,1,1)),
nest_allocation = nest1,
mu_constraint_matrix = NA)
```

---

ssbargain_calibrate *Second score bargaining first-order conditions*

Description

Second score bargaining first-order conditions

Usage

```
ssbargain_calibrate(
  param,
  own,
  price,
  shares,
```

```
  cost,
  weight,
  lambda,
  includeMUI = TRUE
)
```

Arguments

param	Price coefficient alpha and mean values delta, parameters to calibrate
own	Ownership matrix
price	Price
shares	Observed market shares
cost	Marginal costs for each product
weight	Weighting matrix
lambda	Bargaining power of the buyer
includeMUI	logical; whether to include marginal utility of income in buyer's payoff, thereby translating dollars to utility. Default is True, interpreted as buyer maximizing utility. Setting equal to False would have interpretation that buyer maximizes profits.

Details

This function calculate the first-order conditions from a bargaining model that nests the second score auction.

Value

The first-order conditions

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
own_pre = diag(3)
p0 <- c(.05, .34, .33)
c_j <- c(.05,.31,.30)
wt_matrix <- diag(c(1,1,1,1000,1000,1000))
share1 <- c( 0.31, 0.27, 0.25)

ssbargain_calibrate(param = c(alpha,delta),own = own_pre,price = p0,
                    shares = share1, cost = c_j, weight = wt_matrix,
                    lambda = 0.5)
```

ssbargain_foc *Second score bargaining first-order conditions*

Description

Second score bargaining first-order conditions

Usage

```
ssbargain_foc(price, own, alpha, delta, cost, lambda, includeMUI = TRUE)
```

Arguments

price	Price
own	Ownership matrix
alpha	Price coefficient
delta	Mean values
cost	Marginal costs for each product
lambda	Bargaining power of the buyer
includeMUI	logical; TO BE ADDED

Details

This function calculate the first-order conditions from a bargaining model that nests the second score auction.

Value

The first-order conditions

Examples

```
alpha  <- -0.9
delta <- c(.81,.93,.82)
own_pre = diag(3)
p0 <- c(.05, .34, .33)
c_j <- c(.05,.31,.30)

ssbargain_foc(price = p0,own = own_pre,alpha=alpha,delta=delta,cost=c_j,
              lambda = 0.5)
```

Index

bargain_calibrate, 2
bargain_foc, 3
bargain_foc_vert_sim, 4
bargain_NP_vert_seq, 5
bargain_NP_vert_seq_gnl, 7
bargain_vert_seq_calibrate, 9
bargain_vert_sim_calibrate, 10
bargain_vert_sim_calibrate_gnl, 11
bertrand_calibrate, 13
bertrand_calibrate_gnl, 14
bertrand_foc, 16
bertrand_foc_c *(*bertrand_foc*)*, 16
bertrand_foc_vert, 17
bertrand_foc_vert_gnl, 18
bertrand_vert_calibrate, 20

diversion_calc, 20

match_share, 21

share_calc, 22
ssa_calibrate, 23
ssa_calibrate_gnl, 24
ssbargain_calibrate, 25
ssbargain_foc, 27