Package 'antitrust'

October 12, 2022

Type Package

Title Tools for Antitrust Practitioners

Version 0.99.26

Date 2022-08-22

Maintainer Charles Taragin <ctaragin+antitrustr@gmail.com>

Imports methods, BB, numDeriv

Suggests ggplot2, knitr, bookdown, rmarkdown, competitiontoolbox

VignetteBuilder knitr

Encoding UTF-8

Description
A collection of tools for antitrust practitioners, including the ability to calibrate different consumer demand systems and simulate the effects of mergers under different competitive regimes.

URL https://github.com/luciu5/antitrust

License CC0

LazyLoad yes

Collate 'AntitrustClasses.R' 'BertrandClasses.R'
'BertrandRUMClasses.R' 'AuctionClasses.R' 'BargainingClasses.R'
'VerticalClasses.R' 'CournotClasses.R' 'OwnershipMethods.R'
'SummaryMethods.R' 'ShowMethods.R' 'AuctionCapMethods.R'
'PricesMethods.R' 'CostMethods.R' 'MarginsMethods.R'
'ParamsMethods.R' 'PriceDeltaMethods.R' 'PSMethods.R'
'OutputMethods.R' 'PlotMethods.R' 'HypoMonMethods.R'
'HHIMethods.R' 'UPPMethods.R' 'DiversionMethods.R'
'ElastMethods.R' 'CMCRMethods.R' 'CVMethods.R'
'DiagnosticsMethods.R' 'AIDSFunctions.R' 'Antitrust_Shiny.R'
'AntitrustPackage.R' 'Auction2ndCapFunctions.R'
'Auction2ndLogitFunctions.R' 'BargainingLogitFunctions.R'
'BertrandFunctions.R' 'CESFunctions.R'
'CMCRBertrandFunctions.R' 'CMCRCournotFunctions.R'
'CournotFunctions.R' 'HHIFunctions.R' 'LinearFunctions.R'
'LogitFunctions.R' 'SimFunctions.R' 'VerticalFunctions.R'
'antitrust-deprecated.R'

RoxygenNote 7.2.0

NeedsCompilation no

Author Charles Taragin [aut, cre],
Michael Sandfort [aut],
Shlok Goyal [ctb]

Repository CRAN

Date/Publication 2022-08-24 07:00:15 UTC

R topics documented:

AIDS-Functions *(Nested) AIDS Calibration and Merger Simulation*

Description

Calibrates consumer demand using (nested) AIDS and then simulates the price effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products Bertrand game.

Below let k denote the number of products produced by all firms.

Usage

```
aids(
  shares,
  margins,
  prices,
  diversions,
  ownerPre,
  ownerPost,
  mktElast = NA_real_,
  insideSize = NA_real_,
  mcDelta = rep(0, length(shares)),
  subset = rep(TRUE, length(shares)),
  parmStart = rep(NA_real_, 2),
  priceStart = runif(length(shares)),
  isMax = FALSE,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(shares), sep = ""),
  ...
)

pcaids(
  shares,
  knownElast,
  mktElast = -1,
  prices,
  diversions,
  ownerPre,
  ownerPost,
```

```
  knownElastIndex = 1,
  insideSize = NA_real_,
  mcDelta = rep(0, length(shares)),
  subset = rep(TRUE, length(shares)),
  priceStart = runif(length(shares)),
  isMax = FALSE,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(shares), sep = ""),
  ...
)

pcaids.nests(
  shares,
  margins,
  knownElast,
  mktElast = -1,
  prices,
  ownerPre,
  ownerPost,
  nests = rep(1, length(shares)),
  knownElastIndex = 1,
  insideSize = NA_real_,
  mcDelta = rep(0, length(shares)),
  subset = rep(TRUE, length(shares)),
  priceStart = runif(length(shares)),
  isMax = FALSE,
  nestsParmStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(shares), sep = ""),
  ...
)
```

Arguments

shares	A length k vector of product revenue shares. All shares must be between 0 and 1.
margins	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
prices	A length k vector product prices. Default is missing, in which case demand intercepts are not calibrated.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing, in which case diversion according to revenue share is assumed.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product before the merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.

mktElast	A negative number equal to the industry pre-merger price elasticity. Default is NA for aids and -1 for pcaids.
insideSize	Total expenditure (revenues) on products included in the simulation.
mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
parmStart	aids only. A vector of length 2 who elements equal to an initial guess for "known" element of the diagonal of the demand matrix and the market elasticity.
priceStart	A vector of length k who elements equal to an initial guess of the proportional change in price caused by the merger. The default is to draw k random elements from a [0,1] uniform distribution.
isMax	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels.
...	Additional options to feed to the BBsolve optimizer used to solve for equilibrium prices.
knownElast	A negative number equal to the pre-merger own-price elasticity for any of the k products.
knownElastIndex	An integer equal to the position of the 'knownElast' product in the 'shares' vector. Default is 1, which assumes that the own-price elasticity of the first product is known.
nests	A length k vector identifying which nest a product belongs to. Default is that all products belong to a single nest.
nestsParmStart	A vector of starting values used to solve for price coefficient and nest parameters. If missing then the random draws with the appropriate restrictions are employed.

Details

Using product market revenue shares and all of the product product margins from at least two firms, aids is able to recover the slopes in a proportionally calibrated Almost Ideal Demand System (AIDS) without income effects. aids then uses these slopes to simulate the price effects of a merger between two firms under the assumption that all firms in the market are playing a differentiated Bertrand pricing game.

If prices are also supplied, aids is able to recover the intercepts from the AIDS demand system. Intercepts are helpful because they can be used to simulate pre- and post-merger price *levels* as

well as price *changes*. Whatsmore, the intercepts are necessary in order to calculate compensating variation.

`aids` assumes that diversion between the products in the market occurs according to revenue share. This assumption may be relaxed by setting 'diversions' equal to a k x k matrix of diversion ratios. The diagonal of this matrix must equal -1, the off-diagonal elements must be between 0 and 1, and the rows must sum to 1.

`pcaids` is almost identical to `aids`, but instead of assuming that at least two margins are known, `pcaids` assumes that the own-price elasticity of any single product, and the industry-wide own-price elasticity, are known. Demand intercepts cannot be recovered using `pcaids`.

`pcaids.nests` extends `pcaids` by allowing products to be grouped into nests. Although products within the same nest still have the independence of irrelevant alternatives (IIA) property, products in different nests do not. Note that the 'diversions' argument is absent from `pcaids.nests`.

`pcaids.nests` assumes that the share diversion between nests is symmetric (i.e for 2 nests A and B, the diversion from A to B is the same as B to A). Therefore, if there are w nests, $2 \leq w \leq k$, then the model must estimate $w(w-1)/2$ distinct nesting parameters. To accomplish this, `pcaids.nests` uses margin information to produce estimates of the nesting parameters. It is important to note that the number of supplied margins must be at least as great as the number of nesting parameters in order for PCAIDS to work.

The nesting parameters are constrained to be between 0 and 1. Therefore, one way to test the validity of the nesting structure is to check whether the nesting parameters are between 0 and 1. The value of the nesting parameters may be obtained from calling either the 'summary' or 'getNestsParms' functions.

Value

`aids` returns an instance of class `AIDS`, a child class of `Linear`. `pcaids` returns an instance of class `PCAIDS`, while `pcaids.nests` returns an instance of `PCAIDSNests`. Both are children of the `AIDS` class.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Epstein, Roy and Rubinfeld, Daniel (2004). "Merger Simulation with Brand-Level Margin Data: Extending PCAIDS with Nests." *The B.E. Journal of Economic Analysis and Policy*, **advances.4**(1), pp. 2.

Epstein, Roy and Rubinfeld, Daniel (2004). "Effects of Mergers Involving Differentiated Products."

See Also

`linear` for a demand system based on quantities rather than revenue shares.

Examples

```
## Simulate a merger between two single-product firms A and B in a
## three-firm market (A, B, C). This example assumes that the merger is between
## the firms A and B and that A's own-price elasticity is
## known.
## Source: Epstein and Rubinfeld (2004), pg 9, Table 2.


prices    <- c(2.9,3.4,2.2) ## optional for aids, unnecessary for pcaids
shares    <- c(.2,.3,.5)


## The following are used by aids but not pcaids
## only two of the margins are required to calibrate the demand parameters
margins <- c(0.33, 0.36, 0.44)

## The following are used by pcaids, but not aids
knownElast<- -3
mktElast  <- -1


## Define ownership using a vector of firm identities
ownerPre <- c("A","B","C")
ownerPost <- c("A","A","C")

## Alternatively, ownership could be defined using matrices
#ownerPre=diag(1,length(shares))
#ownerPost=ownerPre
#ownerPost[1,2] <- ownerPost[2,1] <- 1


## AIDS: the following assumes both prices and margins are known.
##       Prices are not needed to estimate price changes



result.aids <- aids(shares,margins,prices,ownerPre=ownerPre,ownerPost=ownerPost,labels=ownerPre)



print(result.aids)           # return predicted price change
summary(result.aids)         # summarize merger simulation

elast(result.aids,TRUE)      # returns premerger elasticities
elast(result.aids,FALSE)     # returns postmerger elasticities

diversion(result.aids,TRUE)  # return premerger diversion ratios
diversion(result.aids,FALSE) # return postmerger diversion ratios


cmcr(result.aids)            #calculate compensating marginal cost reduction
upp(result.aids)             #calculate Upwards Pricing Pressure Index
```

```
## Implement the Hypothetical Monopolist Test
## for products A and B using a 5\% SSNIP

HypoMonTest(result.aids,prodIndex=1:2)


CV(result.aids)        #calculate compensating variation as a percent of
#total expenditure
#CV can only be calculated if prices are supplied


## Get a detailed description of the 'AIDS' class slots
showClass("AIDS")

## Show all methods attached to the 'AIDS' Class
showMethods(classes="AIDS")

## Show which class have their own 'elast' method
showMethods("elast")

## Show the method definition for 'elast' and Class 'AIDS'
getMethod("elast","AIDS")



## PCAIDS: the following assumes that only one product's elasticity is
##       known as well as the market elasticity.



result.pcaids <- pcaids(shares,knownElast,mktElast,
                        ownerPre=ownerPre,ownerPost=ownerPost,
                        labels=ownerPre)

print(result.pcaids)           # return predicted price change
summary(result.pcaids)         # summarize merger simulation

elast(result.pcaids,TRUE)      # returns premerger elasticities
elast(result.pcaids,FALSE)     # returns postmerger elasticities

diversion(result.pcaids,TRUE)  # return premerger diversion ratios
diversion(result.pcaids,FALSE) # return postmerger diversion ratios


cmcr(result.pcaids)            #calculate compensating marginal cost reduction


## Implement the Hypothetical Monopolist Test
## for products A and B using a 5\% SSNIP

HypoMonTest(result.pcaids,prodIndex=1:2)
```

```
## Nested PCAIDS: in addition to the PCAIDS information requirements,
##                users must supply the nesting structure as well as margin infromation.

nests <- c('H','L','L') # product A assigned to nest H, products B and C assigned to nest L


result.pcaids.nests <- pcaids.nests(shares,knownElast,mktElast,margins=margins,
                                    nests=nests,ownerPre=ownerPre,
                                    ownerPost=ownerPost,labels=ownerPre)
```

`Antitrust-Class` *"Antitrust" Classes*

Description

The "Antitrust" class is a building block used to create other classes in this package. As such, it is most likely to be useful for developers who wish to code their own calibration/simulation routines.

Let k denote the number of products produced by all firms below.

Slots

`pricePre` A length k vector of simulated pre-merger prices.

`pricePost` A length k vector of simulated post-merger prices.

`ownerPre` A k x k matrix of pre-merger ownership shares.

`ownerPost` A k x k matrix of post-merger ownership shares.

`labels` A length k vector of labels.

`control.slopes` A list of `optim` control parameters passed to the calibration routine optimizer (typically the `calcSlopes` method).

`control.equ` A list of `BBsolve` control parameters passed to the non-linear equation solver (typically the `calcPrices` method).

Objects from the Class

Objects can be created by calls of the form `new("Antitrust", ...)`.

The "matrixOrList", "matrixOrVector" and "characterOrList" Classes

The "matrixOrList","matrixOrVector" and "characterOrList" classes are virtual classes used for validity checking in the 'ownerPre' and 'ownerPost' slots of "Antitrust" and the 'slopes' slot in "Bertrand".

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showClass("Antitrust")          # get a detailed description of the class
```

Auction-Classes *Class "Auction"*

Description

The "Auction2ndCap" class contains all the information needed to calibrate a 2nd price auction with capacity constraints

The "Auction2ndLogit" class contains all the information needed to calibrate a Logit demand system and perform a merger simulation analysis under the assumption that firms are setting offers in a 2nd-score auction.

The "Auction2ndLogitNests" class contains all the information needed to calibrate a Nested Logit demand system and perform a merger simulation analysis under the assumption that firms are setting offers in a 2nd-score auction.

The "Auction2ndLogitALM" class contains all the information needed to calibrate a Logit demand system with unobserved outside share and perform a merger simulation analysis under the assumption that firms are setting offers in a 2nd-score auction.

Below, let k denote the number of firms.

Slots

`capacities` A length k vector of firm capacities.

`margins` A length k vector of product margins, some of which may equal NA.

`prices` A length k vector of product prices.

`reserve` A length 1 vector equal to observed buyer's reserve price. May equal NA.

`shareInside` A length 1 vector equal to the probability that a buyer does not select the outside option. May equal NA.

`sellerCostCDF` A length 1 character vector equal to the name of the function that calculates the Cumulative Distribution (CDF) of SellerCosts.

`sellerCostCDFLowerTail` A length 1 logical vector equal to TRUE if the probabilities are P[X <= x], otherwise, P[X > x].

`sellerCostPDF` A function returning the Probability Density of Seller Costs.

`sellerCostBounds` The bounds on the seller's CDF.

`sellerCostParms` The parameters of the seller's CDF.

`buyerValuation` Buyer's self-supply cost.

`reservePre` Buyer's optimal pre-merger reservation price.

`reservePost` Buyer's optimal post-merger reservation price.

`mcDelta` A length k vector equal to the proportional change in a firm's capacity following the merger.

`parmsStart` A vector of starting values.

Objects from the Class

Auction2ndCap: Objects can be created by using the constructor function `auction2nd.cap`.

Auction2ndLogit: Objects can be created by using the constructor function `auction2nd.logit`.

Auction2ndLogitNests: Objects can be created by using the constructor function `auction2nd.logit.nests`.

Auction2ndLogitALM: Objects can be created by using the constructor function `auction2nd.logit.alm`.

Extends

Auction2ndCap: Class `Antitrust`, directly.

Auction2ndLogit: Class `Logit`, directly. Class `Bertrand`, by class `Logit`, distance 2. Class `Antitrust`, by class `Bertrand`, distance 3.

Auction2ndLogitALM: Class `Auction2ndLogit`, directly. Class `Logit`, distance 2. Class `Bertrand`, by class `Logit`, distance 3. Class `Antitrust`, by class `Bertrand`, distance 4.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showClass("Auction2ndCap")          # get a detailed description of the class
showClass("Auction2ndLogit")          # get a detailed description of the class
showClass("Auction2ndLogitALM")          # get a detailed description of the class
```

`Auction2ndCap-Functions`

(Capacity Constrained) 2nd Price Auction Model

Description

Calibrates the parameters of bidder cost distributions and then simulates the price effect of a merger between two firms under the assumption that firms are competing in a (Capacity Constrained) 2nd price auction.

Let k denote the number of firms bidding in the auction below.

Usage

```
auction2nd.cap(
  capacities,
  margins,
  prices,
  reserve = NA,
  shareInside = NA,
  sellerCostCDF = c("punif", "pexp", "pweibull", "pgumbel", "pfrechet"),
  ownerPre,
  ownerPost,
  mcDelta = rep(0, length(capacities)),
  constrain.reserve = TRUE,
  parmsStart,
  control.slopes,
  labels = as.character(ownerPre),
  ...
)
```

Arguments

capacities	A length k vector of firm capacities OR capacity shares.
margins	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
prices	A length k vector product prices. Prices may be NA.
reserve	A length 1 vector equal to the buyer's reserve price. Default is NA.
shareInside	A length 1 vector equal to the probability that the buyer does not select the outside option. Default is NA.
sellerCostCDF	A length 1 character vector indicating which probability distribution will be used to model bidder cost draws. Possible options are "punif", "pexp", "pweibull", "pgumbel", "pfrechet". Default is "punif".
ownerPre	A length k factor whose values indicate which firms are present in the market pre-merger.
ownerPost	A length k factor whose values indicate which firms are present in the market post-merger.
mcDelta	A vector of length k where each element equals the proportional change in a firm's capacity due to the merger. Default is 0, which assumes that the merger does not affect any products' capacity.
constrain.reserve	If TRUE, the buyer's post-merger optimal reserve price is assumed to equal the buyer's pre-merger optimal reserve price. If FALSE, the buyer re-calculates her optimal reserve price post-merger.
parmsStart	A vector of starting values for calibrated parameters. See below for more details.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).

labels	A k-length vector of labels. Default is "Firm", where '#' is a number between 1 and the length of 'capacities'.
...	Additional options to feed to either optim or constrOptim.

Details

auction2nd.cap examines how a merger affects equilbrium bidding behavior when a single buyer is running a 2nd price procurement auction with bidders whose marginal cost of supplying a homogenous product is private information. This version of the model assumes that bidders are differentiated by their capacities in the sense that firms with greater capacity are more likely to have lower costs than firms with smaller capacities.

Using firm prices, shares, and margins, as well as information on the auction reserve price as well as the proportion of buyers who choose not to purchase from any bidder, auction2nd.cap calibrates the parameters of the common distribution from which bidder's costs are drawn (and, if not supplied, the implied reserve price) and then uses these calibrated parameters to calibrate the value to the buyer of selecting the outside option. Once these parameters have been calibrated, auction2nd.cap computes the buyer's optimal pre-merger reservation price, and if 'constrain.reserve' is FALSE, computes the buyer's optimal post-merger reservation price (setting 'constrain.reserve' to TRUE sets the buyer's post-merger optimal reserve equal to the buyer's pre-merger optimal reserve). The pre- and post-merger expected price, conditional on a particular bidder winning, are then calculated.

Currently, the common distribution from which costs may be drawn is restricted to be either: Uniform ("punif"), Exponential ("pexp"), Weibull ("pweibull"), Gumbel ("pgumbel"), or Frechet ("pfrechet"). Note that the Exponential is a single parameter distribution, the Uniform and Weibull are two parameter distributions, and the Gumbel and Frechet are 3 parameter distributions. Accordingly, sufficient price, margin, reserve, and outside share information must be supplied in order to calibrate the parameters of the specified distribution. auction2nd.cap returns an error if insufficient information is supplied.

Value

auction2nd.cap returns an instance of class Auction2ndCap.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>, with code contributed by Michael Sandfort and Nathan Miller

References

Keith Waehrer and Perry, Martin (2003). "The Effects of Mergers in Open Auction Markets", *Rand Journal of Economics*, **34(2)**, pp. 287-304.

Examples

```
##Suppose there are 3 firms (A,B,C) participating in a procurement auction with
## an unknown reservation price and that firm A acquires firm B.

caps <- c(0.65,0.30,0.05)           # total capacity normalized to 1 in this example
inShare    <- .67                   # probability that buyer does not select
```

```
# any bidder
prices     <- c(3.89, 3.79, 3.74)   # average price charged by each firm
margins    <- c(.228, .209, 0.197)  # average margin earned by each firm
ownerPre   <- ownerPost   <-c("A","B","C")
ownerPost[ownerPost=="B"] <- "A"

##assume costs are uniformly distributed with unknown bounds
result.unif = auction2nd.cap(
  capacities=caps,
  margins=margins,prices=prices,reserve=NA,
  shareInside=inShare,
  sellerCostCDF="punif",
  ownerPre=ownerPre,ownerPost=ownerPost,
  labels=ownerPre
)

print(result.unif)
summary(result.unif)

## Get a detailed description of the 'Auction2ndCap' class slots
showClass("Auction2ndCap")

## Show all methods attached to the 'Auction2ndCap' Class
showMethods(classes="Auction2ndCap")
```

Auction2ndLogit-Functions

2nd Score Procurement Auction Model with (Nested) Logit Demand

---

Description

Calibrates consumer demand using (Nested) Logit and then simulates the price effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products 2nd score auction game.

Let k denote the number of products produced by all firms playing the auction game below.

Usage

```
auction2nd.logit(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
 normIndex = ifelse(isTRUE(all.equal(sum(shares), 1, check.names = FALSE)), 1, NA),
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  insideSize = NA_real_,
```

```
  mcDeltaOutside = 0,
  control.slopes,
  labels = paste("Prod", 1:length(prices), sep = "")
)

auction2nd.logit.nests(
  prices,
  shares,
  margins,
  nests,
  diversions,
  ownerPre,
  ownerPost,
 normIndex = ifelse(isTRUE(all.equal(sum(shares), 1, check.names = FALSE)), 1, NA),
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  insideSize = NA_real_,
  mcDeltaOutside = 0,
  parmsStart,
  constraint = TRUE,
  control.slopes,
  labels = paste("Prod", 1:length(prices), sep = "")
)

auction2nd.logit.alm(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  mktElast = NA_real_,
  insideSize = NA_real_,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  mcDeltaOutside = 0,
  parmsStart,
  control.slopes,
  labels = paste("Prod", 1:length(prices), sep = "")
)
```

Arguments

prices	A length k vector of product prices.
shares	A length k vector of product (quantity) shares. Values must be between 0 and 1.
margins	A length k vector of product margins (in levels, not percents), some of which may equal NA.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.

ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
normIndex	An integer equalling the index (position) of the inside product whose mean valuation will be normalized to 1. Default is 1, unless 'shares' sum to less than 1, in which case the default is NA and an outside good is assumed to exist.
mcDelta	A vector of length k where each element equals the (level) change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
insideSize	An integer equal to total pre-merger units sold. If shares sum to one, this also equals the size of the market.
mcDeltaOutside	A length 1 vector indicating the change in the marginal cost of the outside good. Default is 0.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
nests	A length k factor of product nests.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing, in which case diversion according to share is assumed.
parmsStart	For auction2nd.logit.alm, a length 2 vector of starting values used to solve for price coefficient and the share of the outside good. The first element should always be the price coefficient and the second should be the outside good. For auction2nd.logit.nests, a length n+1 vector of starting values used to solve for price coefficient and the nesting parameters. The first element should always be the price coefficient and the remaining elements should be the nesting parameters.
constraint	if TRUE, then the nesting parameters for all non-singleton nests are assumed equal. If FALSE, then each non-singleton nest is permitted to have its own value. Default is TRUE.
mktElast	a negative value indicating market elasticity. Default is NA.

Details

Using product prices, quantity shares and all of the product margins from at least one firm, auction2nd.logit is able to recover the price coefficient and product mean valuations in a Logit demand model. auction2nd.logit then uses these calibrated parameters to simulate a merger between two firms, under the assumption that firms are participating in a 2nd score procurement auction.

auction2nd.logit.nests is identical to auction2nd.logit except that it assumes that products can be grouped into nests. Additional margin information is needed to identify th nesting parameters.

auction2nd.logit.alm is identical to auction2nd.logit except that it assumes that an outside product exists and uses additional margin information to estimate the share of the outside good.

Value

auction2nd.logit returns an instance of Auction2ndLogit, a child class of Logit. auction2nd.logit.nests returns an instance of Auction2ndLogitNests. auction2nd.logit returns an instance of Auction2ndLogitALM.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Miller, Nathan (2014). "Modeling the effects of mergers in procurement" *International Journal of Industrial Organization* , **37**, pp. 201-208.

See Also

logit,logit.nests for simulating mergers under a Nash-Bertrand pricing game with Logit demand

Examples

```
## Calibration and simulation results from a merger between firms 2 and 3
## of a 4-firm market
## Source: Miller 2014 backup materials http://www.nathanhmiller.org/research

share = c(0.29,0.40,0.28,0.03)

price = c(35.53,  154, 84.08, 53.16)*1e3
cost  =  c(NA, 101, NA, NA)*1e3

ownerPre <- ownerPost <- diag(length(share))

#Suppose products 2 and 3 merge
ownerPost[2,3] <- ownerPost[3,2] <- 1

margin = price - cost

result.2nd <- auction2nd.logit(price,share,margin,
                           ownerPre=ownerPre,ownerPost=ownerPost,normIndex=2)


print(result.2nd)
summary(result.2nd,revenue=FALSE)

##re-run without any price information except Firm 2

price <- rep(NA_real_, length(price))

result.noprice <- auction2nd.logit(price,share,margin,
                                   ownerPre=ownerPre,ownerPost=ownerPost,normIndex=2)

print(result.noprice)
```

```
summary(result.noprice,revenue=FALSE)

##changing the units of prices and margins can yield dramatically different results

price = c(35.53,  154, 84.08, 53.16)
cost  =  c(NA, 101, NA, NA)
margin <- price - cost

result.units <- auction2nd.logit(price,share,margin,
                                 ownerPre=ownerPre,ownerPost=ownerPost,normIndex=2)

print(result.units)
summary(result.units,revenue=FALSE)

## Get a detailed description of the 'Auction2ndLogit' class slots
showClass("Auction2ndLogit")

## Show all methods attached to the 'Auction2ndLogit' Class
showMethods(classes="Auction2ndLogit")
```

AuctionCap-Methods *Auction Cap Methods*

Description

calcBuyerExpectedCost computes the expected amount that the buyer will pay to the auction winner.

calcBuyerValuation computes the value to the buyer of the outside option.

calcExpectedLowestCost computes the expected lowest cost of the winning bid.

calcExpectedPrice computes the expected price paid by the buyer.

calcOptimalReserve computes the bidder's optimal reserve price.

calcSellerCostParms calibrates the parameters of the Seller Cost CDF, as well as the reserve price, if not supplied.

cdfG calculates the probability that a cost draw less than or equal to 'c' is realized for each firm. If 'c' is not supplied, the buyer reserve and total capacity is used.

Usage

```
## S4 method for signature 'Auction2ndCap'
calcSellerCostParms(object, ...)

## S4 method for signature 'Auction2ndCap'
calcBuyerValuation(object)

## S4 method for signature 'Auction2ndCap'
calcOptimalReserve(object, preMerger = TRUE, lower, upper)
```

```
## S4 method for signature 'Auction2ndCap'
calcBuyerExpectedCost(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
cdfG(object, c, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
calcExpectedPrice(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
calcExpectedLowestCost(object, preMerger = TRUE)
```

Arguments

object	An instance of the respective class (see description for the classes)
...	Additional arguments to pass to calcSellerCostParms
preMerger	If TRUE, the pre-merger ownership structure is used. If FALSE, the post-merger ownership structure is used. Default is TRUE.
lower	The minimum for the bidder's reserve price.
upper	The maximum for the bidder's reserve price.
c	cdfG calculates the probability that a cost draw less than or equal to 'c' is realized for each firm. If 'c' is not supplied, the buyer reserve and total capacity is used.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showMethods(classes="Auction2ndCap") # show all methods defined for the class
```

Bargaining-Classes *"Bargaining" Classes*

Description

Each class contains all the information needed to calibrate a specific type of demand system and perform a merger simulation analysis under the assumption that firms are playing a differentiated products Nash Bargaining game.

The "Bargaining" class is a building block used to create other classes in this package. As such, it is most likely to be useful for developers who wish to code their own calibration/simulation routines.

The "BargainingLogit" class has the information for a Nash Bargaining game with Logit demand.

Let k denote the number of products produced by all firms below.

Slots

`bargpowerPre` A length k vector of pre-merger bargaining power parameters.

`bargpowerPre` A length k vector of post-merger bargaining power parameters.

`prices` A length k vector of of observes prices.

`margins` A length k vector of of observes margins.

Objects from the Class

Objects can be created by calls of the form `new("Bargaining", ...)`.

BargainingLogit-Functions

Nash Bargaining Model with Logit Demand

Description

Calibrates consumer demand using Logit and then simulates the price effect of a merger between two firms under the assumption that firms and customers in the market are playing a differentiated products Nash Bargaining game.

Let k denote the number of products produced by all firms playing the Nash Bargaining game below.

Usage

```
bargaining.logit(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  bargpowerPre = rep(0.5, length(prices)),
  bargpowerPost = bargpowerPre,
 normIndex = ifelse(isTRUE(all.equal(sum(shares), 1, check.names = FALSE)), 1, NA),
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceStart = prices,
  insideSize = NA_real_,
  priceOutside = 0,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = "")
)

bargaining2nd.logit(
  prices,
  shares,
```

```
  margins,
  ownerPre,
  ownerPost,
  bargpowerPre = rep(0.5, length(prices)),
  bargpowerPost = bargpowerPre,
 normIndex = ifelse(isTRUE(all.equal(sum(shares), 1, check.names = FALSE)), 1, NA),
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  insideSize = NA_real_,
  mcDeltaOutside = 0,
  control.slopes,
  labels = paste("Prod", 1:length(prices), sep = "")
)
```

Arguments

prices	A length k vector of product prices.
shares	A length k vector of product (quantity) shares. Values must be between 0 and 1.
margins	A length k vector of product margins (in levels, not percents), some of which may equal NA.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
bargpowerPre	A length k vector of pre-merger bargaining power parameters. Values must be between 0 (sellers have the power) and 1 (buyers the power). NA values are allowed, though must be calibrated from additional margin and share data. Default is 0.5.
bargpowerPost	A length k vector of post-merger bargaining power parameters. Values must be between 0 (sellers have the power) and 1 (buyers the power). NA values are allowed, though must be calibrated from additional margin and share data. Default is 'bargpowerPre'.
normIndex	An integer equalling the index (position) of the inside product whose mean valuation will be normalized to 1. Default is 1, unless 'shares' sum to less than 1, in which case the default is NA and an outside good is assumed to exist.
mcDelta	A vector of length k where each element equals the (level) change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
priceStart	A vector of length k who elements equal to an initial guess of equilibrium prices. default is 'prices'.
insideSize	An integer equal to total pre-merger units sold. If shares sum to one, this also equals the size of the market.

priceOutside	A positive real number equal to the price of the outside good. Default equals 0 for Logit demand.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
mcDeltaOutside	A length 1 vector indicating the change in the marginal cost of the outside good. Default is 0.

Details

Using product prices, quantity shares and all of the product margins from at least one firm, auction2nd.logit is able to recover the price coefficient and product mean valuations in a Logit demand model. auction2nd.logit then uses these calibrated parameters to simulate a merger between two firms, under the assumption that firms are participating in a A Nash Bargaining Game (bargaining.logit) or splitting the full surplus (bargaining2nd.logit).

Value

bargaining.logit returns an instance of BargainingLogit, a child class of Logit. bargaining2nd.logit returns an instance of Bargaining2ndLogit, a child class of Auction2ndLogit.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Miller, Nathan (2014). "Modeling the effects of mergers in procurement" *International Journal of Industrial Organization* , **37**, pp. 201-208.

See Also

logit for simulating mergers under a Nash-Bertrand pricing game with Logit demand, and ,auction2nd.logit for simulating mergers under a 2nd score auction with Logit demand.

Examples

```
## Calibration and simulation results from a merger between firms 2 and 3
## of a 4-firm market
## Source: Miller 2014 backup materials http://www.nathanhmiller.org/research

share = c(0.29,0.40,0.28,0.03)
bargpower <- rep(0.6,4) # buyer has advantage
price = c(35.53,  154, 84.08, 53.16)
cost  =  c(NA, 101, NA, NA)
```

```
ownerPre <- ownerPost <- diag(length(share))

#Suppose products 2 and 3 merge
ownerPost[2,3] <- ownerPost[3,2] <- 1

margin = (price - cost)/price

result.barg <- bargaining.logit(price,share,margin,bargpowerPre=bargpower,
                           ownerPre=ownerPre,ownerPost=ownerPost,normIndex=2)


print(result.barg)
summary(result.barg,revenue=FALSE)


## Get a detailed description of the 'BargainingLogit' class slots
showClass("BargainingLogit")

## Show all methods attached to the 'BargainingLogit' Class
showMethods(classes="BargainingLogit")
```

Bertrand-Functions *Bertrand Calibration and Merger Simulation With Logit, CES and AIDS Demand*

Description

Calibrates consumer demand using either a Logit, CES, or AIDS demand system and then simulates the prices effect of a merger between two firms under the assumption that all firms in the market are playing a Nash-Bertrand price setting game.

Let k denote the number of products produced by all firms below.

Usage

```
bertrand.alm(
  demand = c("logit", "ces", "aids"),
  prices,
  quantities,
  margins,
  ownerPre,
  ownerPost,
  mktElast = NA_real_,
 insideSize = ifelse(demand == "logit", sum(quantities, na.rm = TRUE), sum(prices *
   quantities, na.rm = TRUE)),
  diversions,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
```

```
  priceOutside = ifelse(demand == "logit", 0, 1),
  priceStart = prices,
  isMax = FALSE,
  parmStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)
```

Arguments

demand	A character vector indicating which demand system to use. Currently allows logit (default), ces, or aids.
prices	A length k vector product prices. Default is missing, in which case demand intercepts are not calibrated.
quantities	A length k vector of product quantities.
margins	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product before the merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
mktElast	A negative number equal to the industry pre-merger price elasticity. Default is NA.
insideSize	Size of all units included in the market. For logit, this defaults to total quantity, while for aids and ces this defaults to total revenues.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing, in which case diversion according to revenue share is assumed.
mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded. Default is a length k vector of TRUE.
priceOutside	A postive real number equal to the price of the outside good. Default either equals 1 for Logit demand or 0 for CES demand.
priceStart	A vector of length k who elements equal to an initial guess of the proportional change in price caused by the merger. For aids, the default is to draw k random elements from a [0,1] uniform distribution. For ces and logit, the default is prices.
isMax	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
parmStart	aids only. A vector of length 2 who elements equal to an initial guess for "known" element of the diagonal of the demand matrix and the market elasticity.

control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels.
...	Additional options to feed to the BBsolve optimizer used to solve for equilibrium prices.

Details

The main purpose of this function is to provide a more convenient front-end for the aids, logit.alm and ces functions.

Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, bertrand.alm is able to recover the slopes and intercepts of either a Logit, CES, or AIDS demand system. These parameters are then used to simulate the price effects of a merger between two firms under the assumption that the firms are playing a simultaneous price setting game.

'ownerPre' and 'ownerPost' values will typically be equal to either 0 (element [i,j] is not commonly owned) or 1 (element [i,j] is commonly owned), though these matrices may take on any value between 0 and 1 to account for partial ownership.

Value

bertrand.alm returns an instance of class LogitALM, CESALM, or AIDS, depending upon the value of the "demand" argument.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

BertrandOther-Classes *"Bertrand" Classes*

Description

The "Bertrand" class is a building block used to create other classes in this package. As such, it is most likely to be useful for developers who wish to code their own merger calibration/simulation routines.

Each class below contains all the information needed to calibrate a specific type of demand system and perform a merger simulation analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

The "Linear" class has the information for a Linear demand system.

The "LogLin" class has the information for a Log-Linear demand system.

The "AIDS" class has the information for a AIDS demand system.

The "PCAIDS" class has the information for a PCAIDS demand system

The "PCAIDSNests" class has the information for a nested PCAIDS demand system

Below, let k denote the number of products produced by all firms.

Slots

`shares` A length k vector containing observed output. Depending upon the model, output will be measured in units sold, quantity shares, or revenue shares.

`mcDelta` A length k vector where each element equals the proportional change in a product's marginal costs due to the merger.

`slopes` A k x (k+1) matrix of linear demand intercepts and slope coefficients

`subset` A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.

`intercepts` A length k vector of demand intercepts. (Linear only)

`prices` A length k vector product prices. (Linear only)

`quantities` A length k vector of product quantities. (Linear only)

`margins` A length k vector of product margins. All margins must be between 0 and 1. (Linear only)

`diversion` A k x k matrix of diversion ratios with diagonal elements equal to -1.

`priceStart` A length k vector of prices used as the initial guess in the nonlinear equation solver. (Linear and AIDS only)

`symmetry` If TRUE, requires the matrix of demand slope coefficients to be consistent with utility maximization theory. Default is false. (Linear and LogLin only)

`insideSize` A positive number equal to total pre-merger revenues for all products included in the simulation. (AIDS only)

`mktElast` A negative number equal to the industry pre-merger price elasticity. (AIDS only)

`parmStart` A length 2 vector who elements equal to an initial of a single diagonal element of the matrix of slope coefficients, as well as the market elasticity. (AIDS only)

`priceDelta` A length k vector containing the simulated price effects from the merger. (AIDS only)

`knownElast` A negative number equal to the pre-merger own-price elasticity for any of the k products. (PCAIDS only)

`knownElastIndex` An integer equal to the position of the 'knownElast' product in the 'shares' vector. (PCAIDS only)

`nests` A length k vector identifying which nest a product belongs to. (Nested PCAIDS only)

`nestsParms` A length k vector containing nesting parameters. (Nested PCAIDS only)

Objects from the Class

For Bertrand, objects can be created by calls of the form `new("Bertrand", ...)`.

For Linear, objects can be created by using the constructor function `linear`.

For LogLin, objects can be created by using the constructor function `loglin`.

For AIDS, objects can be created by using the constructor function `aids`.

For PCAIDS, objects can be created by using the constructor `pcaids`.

For nested PCAIDS, objects can be created by using the constructor `pcaids.nests`.

Extends

Bertrand: Class Antitrust, directly.

Linear: Class Bertrand, directly. Class Antitrust, by class Bertrand, distance 2.

LogLin: Class Linear, directly. Class Bertrand, by class Linear, distance 2. Class Antitrust, by class Bertrand, distance 3.

AIDS: Class Linear, directly. Class Bertrand, by class "Linear", distance 2.

PCAIDS: Class AIDS, directly. Class Linear, by class AIDS, distance 2. Class Bertrand, by class Linear, distance 3. Class Antitrust, by class Bertrand, distance 4.

Nested PCAIDS: Class PCAIDS, directly. Class AIDS, by class PCAIDS, distance 2. Class Linear, by class AIDS, distance 3. Class Bertrand, by class Linear, distance 4. Class Antitrust, by class Bertrand, distance 5.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showClass("Bertrand")          # get a detailed description of the class
showClass("Linear")          # get a detailed description of the class
showClass("LogLin")          # get a detailed description of the class
showClass("AIDS")          # get a detailed description of the class
showClass("PCAIDS")          # get a detailed description of the class
showClass("PCAIDSNests")          # get a detailed description of the class
```

BertrandRUM-Classes *"Bertrand RUM" Classes*

Description

Each class contains all the information needed to calibrate a specific type of demand system and perform a merger simulation analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

The "Logit" class has the information for a Logit demand system.

The "LogitCap" class has the information for a Logit demand system and assumes that firms are playing a differentiated products Bertrand pricing game with capacity constraints. "LogitCapALM" extends "LogitCap" to allow for an unobserved outside share.

The "LogitNests" class has the information for a nested Logit

The "LogitNestsALM" class has the information for a nested Logit demand system under the assumption that the share of the outside product is not known. Once the model parameters have been calibrated, methods exist that perform a merger simulation analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

The "LogitALM" class has the information for a Logit demand system assuming that firms are playing a differentiated products Bertrand pricing game with unknown market elasticity.

The "CES" class has the information for a CES demand system

The "CESALM" class has the information for a CES demand system and assumes that firms are playing a differentiated products Bertrand pricing game with unknown market elasticity.

The "CESNests" class has the information for a nested CES demand system.

Let k denote the number of products produced by all firms below.

Slots

`prices` A length k vector of product prices.

`margins` A length k vector of product margins, some of which may equal NA.

`normIndex` An integer specifying the product index against which the mean values of all other products are normalized.

`shareInside` The share of customers that purchase any of the products included in the 'prices' vector.

`priceOutside` The price of the outside good. Default is 0.

`slopes` A list containing the coefficient on price ('alpha') and the vector of mean valuations ('meanval').

`mktElast` A length 1 vector of market elasticities.

`priceStart` A length-k vector of starting prices for the non-linear solver.

`insideSize` A positive number equal to total pre-merger quantities (revenues for CES) for all products included in the simulation.

`mktSize` A positive number equal to total quantities (revenues for CES) pre-merger for all products in the simulations as well as the outside good.

`capacitiesPre` A length k vector whose elements equal pre-merger product capacities. (LogitCap and LogitCapALM only)

`capacitiesPost` A length k vector whose elements equal post-merger product capacities. (LogitCap and LogitCapALM only)

`nests` A length k vector identifying the nest that each product belongs to. (LogitNests and CESNests Only)

`parmsStart` A length k vector who elements equal an initial guess of the nesting parameter values. (LogitNests and CESNests Only)

`constraint` A length 1 logical vector that equals TRUE if all nesting parameters are constrained to equal the same value and FALSE otherwise. Default is TRUE. (LogitNests and CESNests Only)

`parmsStart` A length 2 vector whose first element equals an initial guess of the price coefficient and whose second element equals an initial guess of the outside share. The price coefficient's initial value must be negative and the outside share's initial value must be between 0 and 1. (LogitALM and CESALM only)

`slopes` A list containing the coefficient on the numeraire ('alpha'), the coefficient on price ('gamma'), and the vector of mean valuations ('meanval') (CES only)

`priceOutside` The price of the outside good. Default is 1. (CES only)

Objects from the Class

For Logit, objects can be created by using the constructor function `logit`.

For LogitALM, objects can be created by using the constructor function `logit.alm`.

For LogitCap and LogitCapALM, objects can be created by using the constructor function `logit.cap` and `logit.cap.alm`.

For LogitNests, objects can be created by using the constructor function `logit.nests`.

For LogitNestsALM, objects can be created by using the constructor function `logit.nests.alm`.

For CES, objects can be created by using the constructor function `ces`.

For CESALM, objects can be created by using the constructor function `ces.alm`.

For CESNests, objects can be created by using the constructor function `ces.nests`.

Extends

Logit: Class `Bertrand`, directly. Class `Antitrust`, by class `Bertrand`, distance 2.

LogitCap: Class `Logit`, directly. Class `Bertrand`, by class `Logit`, distance 2. Class `Antitrust`, by class `Bertrand`, distance 3.

#'LogitCapALM: Class `LogitCap`, directly. Class `Logit`, by class `LogitCap`, distance 2. Class `Bertrand`, by class `Logit`, distance 3. Class `Antitrust`, by class `Bertrand`, distance 4.

LogitNests: Class `Logit`, directly. Class `Bertrand`, by class `Logit`, distance 2.

LogitNestsALM: Class `LogitNests`, directly. Class `Logit`, by class `LogitNests`, distance 2. Class `Bertrand`, by class `Logit`, distance 3. Class `Antitrust`, by class `Bertrand`, distance 4.

LogitALM: Class `Logit`, directly. Class `Bertrand`, by class `Logit`, distance 2. Class `Antitrust`, by class `Bertrand`, distance 3.

CES: Class `Logit`, directly. Class `Bertrand`, by class `Logit`, distance 2. Class `Antitrust`, by class `Bertrand`, distance 3.

CESALM: Class `CES`, directly. Class `Logit`, by class `CES`, distance 2. Class `Bertrand`, by class `Logit`, distance 3. Class `Antitrust`, by class `Bertrand`, distance 4.

CESNests: Class `CES`, directly. Class `Logit`, by class `CES`, distance 2. Class `Bertrand`, by class `Logit`, distance 3. Class `Antitrust`, by class `Bertrand`, distance 4.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showClass("Logit")           # get a detailed description of the class
showClass("LogitCap")           # get a detailed description of the class
showClass("LogitNests")           # get a detailed description of the class
showClass("LogitNestsALM")           # get a detailed description of the class
showClass("LogitALM")           # get a detailed description of the class
showClass("CES")          # get a detailed description of the class
showClass("CESALM")           # get a detailed description of the class
showClass("CESNests")           # get a detailed description of the class
```

CES-Functions *(Nested) Constant Elasticity of Substitution Demand Calibration and Merger Simulation*

Description

Calibrates consumer demand using (Nested) Constant Elasticity of Substitution (CES) and then simulates the price effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products Bertrand pricing game.

Let k denote the number of products produced by all firms playing the Bertrand pricing game below.

Usage

```
ces(
  prices,
  shares,
  margins,
  diversions,
  ownerPre,
  ownerPost,
  normIndex = ifelse(sum(shares) < 1, NA, 1),
  mktElast = NA_real_,
  insideSize = NA_real_,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 1,
  priceStart = prices,
  isMax = FALSE,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

ces.alm(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  mktElast = NA_real_,
  insideSize = NA_real_,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 1,
  priceStart = prices,
```

```
  isMax = FALSE,
  parmsStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

ces.nests(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  nests = rep(1, length(shares)),
  normIndex = ifelse(sum(shares) < 1, NA, 1),
  insideSize = NA_real_,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 1,
  priceStart = prices,
  isMax = FALSE,
  constraint = TRUE,
  parmsStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)
```

Arguments

prices	A length k vector of product prices.
shares	A length k vector of product revenue shares.
margins	A length k vector of product margins, some of which may equal NA.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
normIndex	An integer specifying the product index against which the mean values of all other products are normalized. Default is 1.
mktElast	a negative value indicating market elasticity. Default is NA.
insideSize	total revenues included in the market.

mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
priceOutside	A length 1 vector indicating the price of the outside good. Default is 1.
priceStart	A length k vector of starting values used to solve for equilibrium price. Default is the 'prices' vector.
isMax	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
...	Additional options to feed to the BBsolve optimizer used to solve for equilibrium prices.
parmsStart	A vector of starting values used to solve for price coefficient and nest parameters. The first element should always be the price coefficient and the remaining elements should be nesting parameters. Theory requires the nesting parameters to be greater than the price coefficient. If missing then the random draws with the appropriate restrictions are employed.
nests	A length k vector identifying the nest that each product belongs to.
constraint	if TRUE, then the nesting parameters for all non-singleton nests are assumed equal. If FALSE, then each non-singleton nest is permitted to have its own value. Default is TRUE.

Details

Using product prices, revenue shares and all of the product margins from at least one firm, ces is able to recover the price coefficient and product mean valuations in a Constant Elasticity of Substitution demand model. ces then uses these calibrated parameters to simulate the price effects of a merger between two firms under the assumption that that all firms in the market are playing a differentiated products Bertrand pricing game.

ces.alm is identical to ces except that it assumes that an outside product exists and uses additional margin information to estimate the share of the outside good.

ces.nests is identical to ces except that it includes the 'nests' argument which may be used to assign products to different nests. Nests are useful because they allow for richer substitution patterns between products. Products within the same nest are assumed to be closer substitutes than products in different nests. The degree of substitutability between products located in different nests is controlled by the value of the nesting parameter sigma. The nesting parameters for singleton nests (nests containing only one product) are not identified and normalized to 1. The vector of sigmas is calibrated from the prices, revenue shares, and margins supplied by the user.

By default, all non-singleton nests are assumed to have a common value for sigma. This constraint may be relaxed by setting 'constraint' to FALSE. In this case, at least one product margin must be supplied from a product within each nest.

In both `ces` and `ces.nests`, if revenue shares sum to 1, then one product's mean value is not identified and must be normalized to 1. 'normIndex' may be used to specify the index (position) of the product whose mean value is to be normalized. If the sum of revenue shares is less than 1, both of these functions assume that the exists a k+1st product in the market whose price and mean value are both normalized to 1.

Value

`ces` returns an instance of class `CES`. `ces.alm` returns an instance of class `CESALM`. `ces.nests` returns an instance of `CESNests`, a child class of `CES`.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Anderson, Simon, Palma, Andre, and Francois Thisse (1992). *Discrete Choice Theory of Product Differentiation*. The MIT Press, Cambridge, Mass.

Epstein, Roy and Rubinfeld, Daniel (2004). "Effects of Mergers Involving Differentiated Products."

Sheu G (2011). "Price, Quality, and Variety: Measuring the Gains From Trade in Differentiated Products." U.S Department of Justice.

See Also

`logit`

Examples

```
## Calibration and simulation results from a merger between Budweiser and
## Old Style. Assume that typical consumer spends 1% of income on beer,
## and that total beer expenditure in US is 1e9
## Source: Epstein/Rubenfeld 2004, pg 80

prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
ownerPre <-c("BUD","OLD STYLE","MILLER","MILLER","OTHER-LITE","OTHER-REG")
ownerPost <-c("BUD","BUD","MILLER","MILLER","OTHER-LITE","OTHER-REG")
nests <- c("R","R","R","L","L","R")

price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)
shares   <- c(.071,.137,.251,.179,.093,.269)
margins  <- c(.3830,.5515,.5421,.5557,.4453,.3769)

names(price) <-
  names(shares) <-
  names(margins) <-
  prodNames
```

```
result.ces <-ces(price,shares,margins,ownerPre=ownerPre,ownerPost=ownerPost,
                 labels=prodNames)

print(result.ces)           # return predicted price change
summary(result.ces)         # summarize merger simulation

elast(result.ces,TRUE)      # returns premerger elasticities
elast(result.ces,FALSE)     # returns postmerger elasticities

diversion(result.ces,TRUE)  # return premerger diversion ratios
diversion(result.ces,FALSE) # return postmerger diversion ratios

cmcr(result.ces)            #calculate compensating marginal cost reduction
upp(result.ces)             #calculate Upwards Pricing Pressure Index

CV(result.ces)              #calculate compensating variation as a percent of
#representative consumer income

## Implement the Hypothetical Monopolist Test
## for BUD and OLD STYLE using a 5\% SSNIP

HypoMonTest(result.ces,prodIndex=1:2)


## Get a detailed description of the 'CES' class slots
showClass("CES")

## Show all methods attached to the 'CES' Class
showMethods(classes="CES")

## Show which class have their own 'elast' method
showMethods("elast")

## Show the method definition for 'elast' and Class 'CES'
getMethod("elast","CES")
```

CMCR-Methods *Methods For Calculating Compensating Marginal Cost Reductions*

Description

Calculate the marginal cost reductions necessary to restore premerger prices in a merger, or the Upwards Pricing Pressure Index for the products of merging firms playing a differentiated products Bertrand pricing game.

Usage

```
## S4 method for signature 'Bertrand'
```

```
cmcr(object, market = FALSE, levels = FALSE, rel = c("cost", "price"))

## S4 method for signature 'Cournot'
cmcr(object, market = TRUE, levels = FALSE, rel = c("cost", "price"))

## S4 method for signature 'AIDS'
cmcr(object, market = FALSE, rel = c("cost", "price"))

## S4 method for signature 'Auction2ndLogit'
cmcr(object, market = FALSE, levels = FALSE, rel = c("cost", "price"), ...)
```

Arguments

object	An instance of one of the classes listed above.
market	If TRUE, calculates (post-merger) share-weighted average of metric. Default is FALSE.
levels	If TRUE calculates CMCR in levels rather than as a percentage of pre-merger costs. Default is FALSE.
rel	A length 1 character vector indicating whether CMCR should be calculated relative to pre-merger cost ("cost") or pre-merger price ("price"), Default is "cost". Ignored when levels is TRUE.
...	Additional arguments to pass to cmcr.

Details

cmcr uses the results from the merger simulation and calibration methods associates with a particular class to compute the compensating marginal cost reduction (CMCR) for each of the merging parties' products.

Value

cmcr returns a vector of length k equal to CMCR for the merging parties' products and 0 for all other products.

See Also

cmcr.bertrand is a function that calculates CMCR without the need to first calibrate a demand system and simulate a merger.

CMCRBertrand-Functions

Compensating Marginal Cost Reductions and Upwards Pricing Pressure (Bertrand)

Description

Calculate the marginal cost reductions necessary to restore premerger prices (CMCR), or the net Upwards Pricing Pressure (UPP) in a merger involving firms playing a differentiated products Bertrand pricing game.

Let k denote the number of products produced by the merging parties below.

Usage

```
cmcr.bertrand(
  prices,
  margins,
  diversions,
  ownerPre,
  ownerPost = matrix(1, ncol = length(prices), nrow = length(prices)),
  rel = c("cost", "price"),
  labels = names(prices)
)

upp.bertrand(
  prices,
  margins,
  diversions,
  ownerPre,
  ownerPost = matrix(1, ncol = length(prices), nrow = length(prices)),
  mcDelta = rep(0, length(prices)),
  labels = paste("Prod", 1:length(prices), sep = "")
)
```

Arguments

prices	A length-k vector of product prices.
margins	A length-k vector of product margins.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1.
ownerPre	EITHER a vector of length k whose values indicate which of the merging parties produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	A k x k matrix of post-merger ownership shares. Default is a k x k matrix of 1s.
rel	A length 1 character vector indicating whether CMCR should be calculated relative to pre-merger cost ("cost") or pre-merger price ("price"), Default is "cost".
labels	A length-k vector of product labels.
mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.

Details

All 'prices' elements must be positive, all 'margins' elements must be between 0 and 1, and all 'diversions' elements must be between 0 and 1 in absolute value. In addition, off-diagonal elements (i,j) of 'diversions' must equal an estimate of the diversion ratio from product i to product j (i.e. the estimated fraction of i's sales that go to j due to a small increase in i's price). Also, 'diversions' elements are positive if i and j are substitutes and negative if i and j are complements.

'ownerPre' will typically be a vector whose values equal 1 if a product is produced by firm 1 and 0 otherwise, though other values including firm name are acceptable. Optionally, 'ownerPre' may be set equal to a matrix of the merging firms pre-merger ownership shares. These ownership shares must be between 0 and 1.

'ownerPost' is an optional argument that should only be specified if one party to the acquisition is assuming partial control of the other party's assets. 'ownerPost' elements must be between 0 and 1.

Value

`cmcr.bertrand` returns a length-k vector whose values equal the percentage change in each products' marginal costs that the merged firms must achieve in order to offset a price increase.

`upp.bertrand` returns a length-k vector whose values equal the generalized pricing pressure (GePP) for each of the merging's parties' products, net any efficiency claims. GePP is a generalization of Upwards Pricing Pressure (UPP) that accomodates multi-product firms.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Farrell, Joseph and Shapiro, Carl (2010). "Antitrust Evaluation of Horizontal Mergers: An Economic Alternative to Market Definition." *The B.E. Journal of Theoretical Economics*, **10**(1), pp. 1-39.

Jaffe, Sonia and Weyl Eric (2012). "The First-Order Approach to Merger Analysis." *SSRN eLibrary*

Werden, Gregory (1996). "A Robust Test for Consumer Welfare Enhancing Mergers Among Sellers of Differentiated Products." *The Journal of Industrial Economics*, **44**(4), pp. 409-413.

See Also

`cmcr.cournot` for a homogeneous products Cournot version of CMCR, and `cmcr-methods` for calculating CMCR and UPP after calibrating demand system parameters and simulating a merger.

Examples

```
## Let k_1 = 1 and and k_2 = 2 ##

p1 = 50;      margin1 = .3
p2 = c(45,70); margin2 = c(.4,.6)
isOne=c(1,0,0)
diversions = matrix(c(-1,.5,.01,.6,-1,.1,.02,.2,-1),ncol=3)
```

```
cmcr.bertrand(c(p1,p2), c(margin1,margin2), diversions, isOne)
upp.bertrand(c(p1,p2), c(margin1,margin2), diversions, isOne)


## Calculate the necessary percentage cost reductions for various margins and
## diversion ratios in a two-product merger where both products have
## equal prices and diversions (see Werden 1996, pg. 412, Table 1)


margins = seq(.4,.7,.1)
diversions = seq(.05,.25,.05)
prices = rep(1,2) #assuming prices are equal, we can set product prices to 1
isOne = c(1,0)
result = matrix(ncol=length(margins),nrow=length(diversions),dimnames=list(diversions,margins))

for(m in 1:length(margins)){
  for(d in 1:length(diversions)){

    dMatrix = -diag(2)
    dMatrix[2,1] <- dMatrix[1,2] <- diversions[d]

    firmMargins = rep(margins[m],2)

    result[d,m] = cmcr.bertrand(prices, firmMargins, dMatrix, isOne)[1]

  }}

print(round(result,1))
```

CMCRCournot-Functions *Compensating Marginal Cost Reductions and Upwards Pricing Pressure (Cournot)*

Description

Calculate the marginal cost reductions necessary to restore premerger prices (CMCR), or the net Upwards Pricing Pressure (UPP) in a merger involving firms playing a homogeneous product Cournot pricing game.

Usage

```
cmcr.cournot(
  shares,
  mktElast,
  party = FALSE,
  rel = c("cost", "price"),
  labels = names(shares)
)
```

```
cmcr.cournot2(
  margins,
  rel = c("cost", "price"),
  party = FALSE,
  labels = names(margins)
)

upp.cournot(
  prices,
  margins,
  ownerPre,
  ownerPost = matrix(1, ncol = length(prices), nrow = length(prices)),
  mcDelta = rep(0, length(prices)),
  labels = names(margins)
)
```

Arguments

shares	A length-2 vector containing merging party quantity shares.
mktElast	A length-1 containing the industry elasticity.
party	If TRUE calculate a length-2 vector of individial party CMCRs. If FALSE calculate share-weighted CMCR relative to share-weighted pre-merger marginal costs. Default is FALSE
rel	A length 1 character vector indicating whether CMCR should be calculated relative to pre-merger cost ("cost") or pre-merger price ("price"), Default is "cost".
labels	A length-2 vector of product labels.
margins	A length-2 vector of product margins.
prices	A length-2 vector of product prices.
ownerPre	EITHER a vector of length 2 whose values indicate which of the merging parties produced a product pre-merger OR a 2 x 2 matrix of pre-merger ownership shares.
ownerPost	A 2 x 2 matrix of post-merger ownership shares. Default is a 2 x 2 matrix of 1s.
mcDelta	A vector of length 2 where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.

Details

The 'shares' (or 'margins') vector must have 2 elements, and all 'shares' and 'margins' elements must be between 0 and 1. The 'mktElast' vector must have 1 non-negative element.

Value

when 'party' is FALSE (default), `cmcr.cournot`, `cmcr.cournot2` return a vector with 1 element whose value equals the percentage change in the products' average marginal costs that the merged

firms must achieve in order to offset a price increase. When 'party' is TRUE, `cmcr.cournot`, `cmcr.cournot2` return a vector with 2 element whose value equals the percentage change in *each* parties' marginal costs necessary to offset a price increase. When 'rel' equals "cost" (default) results are in terms of per-merger marginal costs. Otherwise, results are in terms of pre-merger price.

Author(s)

Charles Taragin

References

Froeb, Luke and Werden, Gregory (1998). "A robust test for consumer welfare enhancing mergers among sellers of a homogeneous product." *Economics Letters*, **58**(3), pp. 367 - 369.

Werden, Gregory and Froeb, Luke (2008). "Unilateral Competitive Effects of Horizontal Mergers", in Paolo Buccirossi (ed), Handbook of Antitrust Economics (MIT Press).

See Also

`cmcr.bertrand` for a differentiated products Bertrand version of this measure.

Examples

```
shares=c(.05,.65)
industryElast = 1.9
margins=shares/industryElast

##  calculate average CMCR as a percentage of pre-merger costs
cmcr.cournot(shares,industryElast, rel="cost")

##  calculate average CMCR as a percentage of pre-merger price
cmcr.cournot(shares,industryElast, rel="price")

##  calculate average CMCR using margins as a percentage of pre-merger costs
cmcr.cournot2(margins, party=TRUE,rel="cost")

##  calculate the average CMCR for various shares and
##  industry elasticities in a two-product merger where both firm
##  products have identical share (see Froeb and
##  Werden, 1998, pg. 369, Table 1)



deltaHHI = c(100, 500, 1000, 2500, 5000) #start with change in HHI
shares = sqrt(deltaHHI/(2*100^2)) #recover shares from change in HHI
industryElast = 1:3

result = matrix(nrow=length(deltaHHI),ncol=length(industryElast),
                dimnames=list(deltaHHI,industryElast))

for(s in 1:length(shares)){
  for(e in 1:length(industryElast)){
```

```
    result[s,e] = cmcr.cournot(rep(shares[s],2),industryElast[e])[1]

  }}

print(round(result,1))
```

Cost-Methods *Methods for Calculating marginal and Variable Costs*

Description

For Auction2ndCap, calcMC calculates (constant) marginal cost for each product. For those classes that do not require prices, returns a length-k vector of NAs when prices are not supplied.

For Bertrand, calcMC computes either pre- or post-merger marginal costs. Marginal costs are assumed to be constant. Post-merger marginal costs are equal to pre-merger marginal costs multiplied by 1+'mcDelta', a length-k vector of marginal cost changes. 'mcDelta' will typically be between 0 and 1.

For Auction2ndLogit, calcMC computes constant marginal costs impied by the model.

For Cournot, calcMC calculates marginal cost for each product.

calcdMC computes the derivative of either pre- or post-merger marginal costs. The derivative of Marginal costs is assumed to be constant. Post-merger marginal costs are equal to pre-merger marginal costs multiplied by 1+'mcDelta', a length-k vector of marginal cost changes. 'mcDelta' will typically be between 0 and 1.

calcVC computes either pre- or post-merger variable costs. Variable costs are assumed to be quadratic by default. Post-merger variable costs are equal to pre-merger variable costs multiplied by 1+'mcDelta', a length-k vector of marginal cost changes. 'mcDelta' will typically be between 0 and 1.

Usage

```
## S4 method for signature 'Bertrand'
calcMC(object, preMerger = TRUE)

## S4 method for signature 'VertBargBertLogit'
calcMC(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
calcMC(object, t, preMerger = TRUE, exAnte = TRUE)

## S4 method for signature 'Cournot'
calcMC(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndLogit'
calcMC(object, preMerger = TRUE, exAnte = FALSE)
```

```
## S4 method for signature 'Stackelberg'
calcdMC(object, preMerger = TRUE)

## S4 method for signature 'Cournot'
calcVC(object, preMerger = TRUE)
```

Arguments

object	An instance of the respective class (see description for the classes)
preMerger	If TRUE, the pre-merger ownership structure is used. If FALSE, the post-merger ownership structure is used. Default is TRUE.
t	The capacity profile of each supplier. Default is 'preMerger' capacities.
exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE.

Cournot-classes *"Cournot" Classes*

Description

The "Cournot" and "Stackelberg" classes are building blocks used to create other classes in this package. As such, they are most likely to be useful for developers who wish to code their own merger calibration/simulation routines.

Note below that k is the number of products and n is the number of plants.

Slots

`intercepts` A length k vector containing the calibrated demand intercept.

`mcfunPre` A length n list whose elements equal a function that calculates a plant's pre-merger marginal cost.

`mcfunPost` A length n list whose elements equal a function that calculates a plant's post-merger marginal cost.

`vcfunPre` A length n list whose elements equal a function that calculates a plant's pre-merger variable cost.

`vcfunPost` A length n list whose elements equal a function that calculates a plant's post-merger variable cost.

`prices` A length k vector of product prices.

`quantities` An n x k matrix of plant quantities produced for each product.

`margins` An n x k matrix of plant product margins.

`quantityPre` An n x k matrix of predicted pre-merger quantities.

`quantityPost` An n x k matrix of predicted post-merger quantities.

`quantityStart` A length n x k vector of starting quantities for the non-linear solver.

`productsPre` An n x k logical matrix qhose elements are TRUE if a plant produces a product pre-merger and FALSE otherwise.

`productsPost` An n x k logical matrix qhose elements are TRUE if a plant produces a product post-merger and FALSE otherwise.

`capacitiesPre` A length-n logical vector whose elements equal to pre-merger plant capacities. Infinite values are allowed.

`capacitiesPost` A length-n logical vector whose elements equal to post-merger plant capacities. Infinite values are allowed.

`demand` A length k character vector specifying whether product demand is linear ("linear") or log-linear ("log").

`cost` A length k character vector equal to "linear" if a plant's marginal cost curve is assumed to be linear or "constant" if a plant's marginal curve is assumed to be constant. Returns an error if a multi-plant firm with constant marginal costs does not have capacity constraints.

`mktElast` A length k vector of market elasticities.

`dmcfunPre` A length n list whose elements equal a function that calculates the derivative of a plant's pre-merger marginal cost with respect to that plant's output. (Stackelberg only)

`dmcfunPost` A length n list whose elements equal a function that calculates the derivative of a plant's post-merger marginal cost with respect to that plant's output. (Stackelberg only)

`isLeaderPre` An n x k logical matrix qhose elements are TRUE if a plant produces a product pre-merger and FALSE otherwise. (Stackelberg only)

`isLeaderPost` An n x k logical matrix qhose elements are TRUE if a plant produces a product post-merger and FALSE otherwise. #'@slot dmcfunPre A length n list whose elements equal a function that calculates the derivative of a plant's pre-merger marginal cost with respect to that plant's output. (Stackelberg only)

Objects from the Class

For Cournot, objects can be created by calls of the form `new("Cournot", ...)`.

For Stackelberg, objects can be created by calls of the form `new("Stackelberg", ...)`.

Extends

Cournot: Class `Bertrand`, directly. Class `Antitrust`, by class `Bertrand`, distance 2.

Stackelberg: Class `Cournot`, directly. Class `Bertrand`, by class `Cournot`, distance 2. Class `Antitrust`, by class `Bertrand`, distance 3.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showClass("Cournot")          # get a detailed description of the class
showClass("Stackelberg")          # get a detailed description of the class
```

Cournot-Functions *Multi-product Cournot/Stackelberg Calibration and Merger Simulation With Linear or Log-Linear Demand*

Description

Calibrates consumer demand for multiple products using either a linear or log-linear demand system and then simulates the prices effect of a merger between two multi-plant firms under the assumption that all firms in the market are playing either a Cournot or Stackelberg quantity setting game.

Let k denote the number of products and n denote the number of plants below.

Usage

```
cournot(
  prices,
  quantities,
  margins = matrix(NA_real_, nrow(quantities), ncol(quantities)),
  demand = rep("linear", length(prices)),
  cost = rep("linear", nrow(quantities)),
  mcfunPre = list(),
  mcfunPost = mcfunPre,
  vcfunPre = list(),
  vcfunPost = vcfunPre,
  capacitiesPre = rep(Inf, nrow(quantities)),
  capacitiesPost = capacitiesPre,
  productsPre = !is.na(quantities),
  productsPost = productsPre,
  ownerPre,
  ownerPost,
  mktElast = rep(NA_real_, length(prices)),
  mcDelta = rep(0, nrow(quantities)),
  quantityStart = as.vector(quantities),
  control.slopes,
  control.equ,
  labels,
  ...
)

stackelberg(
  prices,
  quantities,
  margins,
  demand = rep("linear", length(prices)),
  cost = rep("linear", nrow(quantities)),
  isLeaderPre = matrix(FALSE, ncol = ncol(quantities), nrow = nrow(quantities)),
  isLeaderPost = isLeaderPre,
```

```
  mcfunPre = list(),
  mcfunPost = mcfunPre,
  vcfunPre = list(),
  vcfunPost = vcfunPre,
  dmcfunPre = list(),
  dmcfunPost = dmcfunPre,
  capacitiesPre = rep(Inf, nrow(quantities)),
  capacitiesPost = capacitiesPre,
  productsPre = !is.na(quantities),
  productsPost = productsPre,
  ownerPre,
  ownerPost,
  mcDelta = rep(0, nrow(quantities)),
  quantityStart = as.vector(quantities),
  control.slopes,
  control.equ,
  labels,
  ...
)
```

Arguments

prices	A length k vector product prices.
quantities	An n x k matrix of product quantities. All quantities must either be positive, or if the product is not produced by a plant, NA.
margins	An n x k matrix of product margins. All margins must be either be between 0 and 1, or NA.
demand	A length k character vector equal to "linear" if a product's demand curve is assumed to be linear or "log" if a product's demand curve is assumed to be log-linear.
cost	A length n character vector equal to "linear" if a plant's marginal cost curve is assumed to be linear or "constant" if a plant's marginal curve is assumed to be constant. Returns an error if a multi-plant firm with constant marginal costs does not have capacity constraints.
mcfunPre	a length n list of functions that calculate a plant's pre-merger marginal cost. If empty (the default), assumes quadratic costs.
mcfunPost	a length n list of functions that calculate a plant's post-merger marginal cost. If empty (the default), equals 'mcfunPre'
vcfunPre	a length n list of functions that calculate a plant's pre-merger variable cost. If empty (the default), assumes quadratic variable costs.
vcfunPost	a length n list of functions that calculate a plant's post-merger variable cost. If empty (the default), equals 'vcfunPre'
capacitiesPre	A length n numeric vector of pre-merger plant capacities. Default is Inf.
capacitiesPost	A length n numeric vector of post-merger plant capacities. Default 'capacitiesPre'.

productsPre	An n x k matrix that equals TRUE if pre-merger, a plant produces a product. Default is TRUE if 'quantities' is not NA.
productsPost	An n x k matrix that equals TRUE if post-merger, a plant produces a product. Default equals 'productsPre'.
ownerPre	EITHER a vector of length n whose values indicate which plants are commonly owned pre-merger OR an n x n matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length n whose values indicate which plants will be commonly owned after the merger OR an n x n matrix of post-merger ownership shares.
mktElast	A length k vector of product elasticities. Default is a length k vector of NAs
mcDelta	A length n vector where each element equals the proportional change in a firm's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
quantityStart	A length k vector of quantities used as the initial guess in the nonlinear equation solver. Default is 'quantities'.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcQuantities method).
labels	A list with 2 elements. The first element is a vector of firm names, while the second element is a vector of products names. Default is 'O1:On', and 'P1:Pk'.
...	Additional options to feed to the solver. See below.
isLeaderPre	An n x k logical matrix equal to TRUE if a firm is a "leader" pre-merger for a particular product and FALSE otherwise. Default is FALSE, which is equivalent to cournot.
isLeaderPost	An n x k logical matrix equal to TRUE if a firm is a "leader" pre-merger for a particular product and FALSE otherwise. Default is FALSE, which is equivalent to cournot.
dmcfunPre	a length n list of functions that calculate the derivative of a plant's pre-merger marginal cost. If empty (the default), assumes quadratic variable costs.
dmcfunPost	a length n list of functions that calculate the derivative of a plant's post-merger marginal cost. If empty (the default), equals 'mcfunPre'

Details

Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, cournot is able to recover the slopes and intercepts of either a Linear or Log-linear demand system as well as the cost parameters (see below for further details). These parameters are then used to simulate the price effects of a merger between two firms under the assumption that the firms are playing a homogeneous products simultaneous quantity setting game.

stackelberg, is similar to cournot, except that for a given product, firms are either "leaders" or "followers". leaders gain a first mover advantage over followers, which allows the leaders to

anticipate how changes to their output will effect the follower's output decisions. Firms can be the leader for some products but the follower in others.

'mcfunPre' and 'mcfunPost' are length n lists whose elements are 'R' functions that return a firm's marginal cost. The first argument of each function should be total firm quantities. By default, each firm is assumed to have quadratic costs with a firm-specific parameter calibrated from a firm's margin. 'vcfunPre' and 'vcfunPost' are similarly defined. 'dmcfunPre' and 'dmcfunPost' are the changes in marginal cost and are only required for `stackelberg`.

'ownerPre' and 'ownerPost' values will typically be equal to either 0 (element [i,j] is not commonly owned) or 1 (element [i,j] is commonly owned), though these matrices may take on any value between 0 and 1 to account for partial ownership.

Under linear demand and linear marginal costs, an analytic solution to the Cournot quantity game exists. However, this solution can at times produce negative equilibrium quantities. To accommodate this issue, `cournot` uses `BBsolve` to find equilibrium quantities subject to a non-negativity constraint. . . . may be used to change the default options for `BBsolve`.

Value

`cournot` returns an instance of class `Cournot`. `stackelberg` returns an instance of class `Stackelberg`.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
## Simulate a Cournot merger between two single-plant firms
## producing a single product in a
## 5-firm market with linear demand and quadratic costs



n <- 5 #number of firms in market pre-merger
cap <- rnorm(n,mean = .5, sd = .1)
int <- 10
slope <- -.25

B.pre.c = matrix(slope,nrow=n,ncol=n)
diag(B.pre.c) = 2* diag(B.pre.c) - 1/cap
quantity.pre.c = rowSums(solve(B.pre.c) * -int)
price.pre.c = int + slope * sum(quantity.pre.c)
mc.pre.c = quantity.pre.c/cap
vc.pre.c = quantity.pre.c^2/(2*cap)
margin.pre.c = 1 - mc.pre.c/price.pre.c
ps.pre.c = price.pre.c*quantity.pre.c - vc.pre.c

mktQuant.pre.c = sum(quantity.pre.c)

## suppose firm 1 acquires firm 2
## This model has a closed form solution
B.post.c = B.pre.c
```

```
B.post.c[1,2] = 2*B.post.c[1,2]
B.post.c[2,1] = 2*B.post.c[2,1]

quantity.post.c = rowSums(solve(B.post.c) * -int)
price.post.c = int + slope * sum(quantity.post.c)
mc.post.c = quantity.post.c/cap
vc.post.c = quantity.post.c^2/(2*cap)
margin.post.c = 1 - mc.post.c/price.post.c
ps.post.c = price.post.c*quantity.post.c - vc.post.c

mktQuant.post.c = sum(quantity.post.c, na.rm=TRUE)

#check if merger is profitable for merging parties
isprofitable.c = ps.post.c - ps.pre.c
isprofitable.c= sum(isprofitable.c[1:2]) > 0


#prep inputs for Cournot
owner.pre <- diag(n)
owner.post <- owner.pre
owner.post[1,2] <- owner.post[2,1] <- 1



result.c <- cournot(prices = price.pre.c,quantities = as.matrix(quantity.pre.c),
                    margins=as.matrix(margin.pre.c),
                    ownerPre=owner.pre,ownerPost=owner.post)

print(result.c)           # return predicted price change
summary(result.c)         # summarize merger simulation

## check if 'cournot' yields the same result as closed-form solution
#print(all.equal(sum(result.c@quantityPre) , mktQuant.pre.c))
#print(all.equal(sum(result.c@quantityPost) , mktQuant.post.c))




## Simulate a Stackelberg merger between two single-plant firms
## producing a single product in a
## 5-firm market with linear demand and quadratic costs.
## Allow both merging parties to be followers pre-merger,
## but assume that they become leaders post-merger.
## Finally, assume that pre-merger, there is a single leader who ## remains a leader post-merger
## Note: This example uses setup from the above Cournot example

isLeader.pre = matrix(rep(FALSE,n), ncol=1)
isLeader.pre[n,] = TRUE
isLeader.post = isLeader.pre
isLeader.post[1:2,] = TRUE

passthru.pre = matrix(-slope^2/(2*slope - 1/cap))
passthru.post = passthru.pre
```

```
passthru.pre[isLeader.pre] = 0
passthru.post[isLeader.post] = 0

B.pre.s = matrix(slope,nrow=n,ncol=n)
diag(B.pre.s) = 2* diag(B.pre.s) - 1/cap
diag(B.pre.s)[n] = diag(B.pre.s)[n] + sum(passthru.pre)

quantity.pre.s = rowSums(solve(B.pre.s) * ( -int))
price.pre.s = int + slope * sum(quantity.pre.s)
mc.pre.s = quantity.pre.s/cap
vc.pre.s = quantity.pre.s^2/(2*cap)
margin.pre.s = 1 - mc.pre.s/price.pre.s
ps.pre.s = price.pre.s*quantity.pre.s - vc.pre.s

mktQuant.pre.s = sum(quantity.pre.s)

## suppose firm 1 acquires firm 2
## This model has a closed form solution
B.post.s = matrix(slope,nrow=n,ncol=n)
diag(B.post.s) = 2* diag(B.post.s) - 1/cap
B.post.s[1,2] = 2*B.post.s[1,2]
B.post.s[1,1:2] = B.post.s[1,1:2]  + sum(passthru.post)
B.post.s[2,1] = 2*B.post.s[2,1]
B.post.s[2,1:2] = B.post.s[2,1:2]  + sum(passthru.post)
diag(B.post.s)[n] = diag(B.post.s)[n]  +  sum(passthru.post)

quantity.post.s = rowSums(solve(B.post.s) * as.vector( -int ) )
price.post.s = int + slope * sum(quantity.post.s)
mc.post.s = quantity.post.s/cap
vc.post.s = quantity.post.s^2/(2*cap)
margin.post.s = 1 - mc.post.s/price.post.s
ps.post.s = price.post.s*quantity.post.s - vc.post.s

mktQuant.post.s = sum(quantity.post.s, na.rm=TRUE)

#check if merger is profitable for merging parties
isprofitable.s = ps.post.s - ps.pre.s
isprofitable.s = sum(isprofitable.s[1:2]) > 0


#prep inputs for Stackelberg
owner.pre <- diag(n)
owner.post <- owner.pre
owner.post[1,2] <- owner.post[2,1] <- 1



result.s <- stackelberg(prices = price.pre.s,quantities = as.matrix(quantity.pre.s),
                        margins=as.matrix(margin.pre.s),ownerPre=owner.pre,
                        ownerPost=owner.post,
                        isLeaderPre = isLeader.pre, isLeaderPost = isLeader.post)

print(result.s)           # return predicted price change
```

```
summary(result.s)         # summarize merger simulation

## check if 'stackelberg' yields the same result as closed-form solution
#print(all.equal(sum(result.s@quantityPre) , mktQuant.pre.s))
#print(all.equal(sum(result.s@quantityPost) , mktQuant.post.s))
```

CV-Methods *Methods For Calculating Compensating Variation (CV)*

Description

Calculate the amount of money a consumer would need to be paid to be just as well off as they were before the merger.

All the information needed to compute CV is already available within the Logit, Nested Logit CES and Nested CES classes. In CES and Nested CES, CV cannot be calculated if the outside share cannot be inferred.

For AIDS, if the 'insideSize' slot to the "AIDS" class equals NA, CV is calculated as a percentage of total expenditure (revenues) on products included in the simulation. Otherwise CV is calculated in terms of dollars. Pre-merger prices for all products in the market must be supplied in order for CV to be calculated.

For Linear and LogLin, although no additional information is needed to calculate CV for either the "Linear" or "LogLin" classes, The CV method will fail if the appropriate restrictions on the demand parameters have not been imposed.

Usage

```
## S4 method for signature 'Cournot'
CV(object)

## S4 method for signature 'Linear'
CV(object)

## S4 method for signature 'Logit'
CV(object)

## S4 method for signature 'LogLin'
CV(object)

## S4 method for signature 'AIDS'
CV(object)

## S4 method for signature 'LogitNests'
CV(object)

## S4 method for signature 'Auction2ndLogit'
```

```
CV(object)

## S4 method for signature 'VertBargBertLogit'
CV(object)

## S4 method for signature 'VertBarg2ndLogit'
CV(object)

## S4 method for signature 'CES'
CV(object)

## S4 method for signature 'CESNests'
CV(object)
```

Arguments

object	An instance of one of the classes listed above.

`defineMarketTools-methods`

Methods For Implementing The Hypothetical Monopolist Test

Description

An Implementation of the Hypothetical Monopolist Test described in the 2010 Horizontal Merger Guidelines.

`HypoMonTest` implements the Hypothetical Monopolist Test for a given 'ssnip'.

`calcPricesHypoMon` computes prices for a subset of firms under the control of a hypothetical monopolist under the specified demand function or auction.

`diversionHypoMon` calculates the matrix of revenue diversions between all products included in the merger simulation, *irrespective of whether or not they are also included in 'prodIndex'*.

`calcPriceDeltaHypoMon` computes the proportional difference in product prices between the prices of products in 'prodIndex' (i.e. prices set by the Hypothetical Monopolist) and prices set in the pre-merger equilibrium. '...' may be used to pass arguments to the optimizer.

Usage

```
## S4 method for signature 'Bertrand'
HypoMonTest(object, prodIndex, ssnip = 0.05, ...)

## S4 method for signature 'Cournot'
HypoMonTest(object, plantIndex, prodIndex, ssnip = 0.05, ...)

## S4 method for signature 'Cournot'
calcPricesHypoMon(object, plantIndex, prodIndex)
```

```
## S4 method for signature 'Linear'
calcPricesHypoMon(object, prodIndex)

## S4 method for signature 'Logit'
calcPricesHypoMon(object, prodIndex)

## S4 method for signature 'LogLin'
calcPricesHypoMon(object, prodIndex)

## S4 method for signature 'AIDS'
calcPricesHypoMon(object, prodIndex, ...)

## S4 method for signature 'LogitCap'
calcPricesHypoMon(object, prodIndex, ...)

## S4 method for signature 'Auction2ndLogit'
calcPricesHypoMon(object, prodIndex)

## S4 method for signature 'Bertrand'
diversionHypoMon(object, prodIndex, ...)

## S4 method for signature 'AIDS'
diversionHypoMon(object)

## S4 method for signature 'Bertrand'
calcPriceDeltaHypoMon(object, prodIndex, ...)

## S4 method for signature 'Cournot'
calcPriceDeltaHypoMon(object, prodIndex, plantIndex, ...)

## S4 method for signature 'AIDS'
calcPriceDeltaHypoMon(object, prodIndex, ...)
```

Arguments

`object`	An instance of one of the classes listed above.
`prodIndex`	A vector of product indices that are to be placed under the control of the Hypothetical Monopolist.
`ssnip`	A number between 0 and 1 that equals the threshold for a "Small but Significant and Non-transitory Increase in Price" (SSNIP). Default is .05, or 5%.
`...`	Pass options to the optimizer used to solve for equilibrium prices.
`plantIndex`	A vector of plant indices that are to be placed under the control of the Hypothetical Monopolist (Cournot).

Details

Let k denote the number of products produced by all firms playing the Bertrand pricing game above.

HypoMonTest is an implementation of the Hypothetical Monopolist Test on the products indexed by 'prodIndex' for a 'ssnip'. The Hypothetical Monopolist Test determines whether a profit-maximizing Hypothetical Monopolist who controls the products indexed by 'prodIndex' would increase the price of at least one of the merging parties' products in 'prodIndex' by a small, significant, and non-transitory amount (i.e. impose a SSNIP).

calcPriceDeltaHypoMon calculates the price changes relative to (predicted) pre-merger prices that a Hypothetical Monopolist would impose on the products indexed by 'prodIndex', holding the prices of products not controlled by the Hypothetical Monopolist fixed at pre-merger levels. With the exception of 'AIDS', the calcPriceDeltaHypoMon for all the classes listed above calls calcPricesHypoMon to compute price levels. calcPriceDeltaHypoMon is in turn called by HypoMonTest.

diversionHypoMon calculates the matrix of revenue diversions between all products included in the merger simulation, *irrespective* of whether or not they are also included in 'prodIndex'. This matrix is useful for diagnosing whether or not a product not included in 'prodIndex' may have a higher revenue diversion either to or from a product included in 'prodIndex'. Note that the 'AIDS' diversionHypoMon method does not contain the 'prodIndex' argument, as AIDS revenue diversions are only a function of demand parameters.

Value

HypoMonTest returns TRUE if a profit-maximizing Hypothetical Monopolist who controls the products indexed by 'prodIndex' would increase the price of at least one of the merging parties' products in 'prodIndex' by a 'ssnip', and FALSE otherwise. HypoMonTest returns an error if 'prodIndex' does not contain at least one of the merging parties products.

calcPriceDeltaHypoMon returns a vector of proportional price changes for all products placed under the control of the Hypothetical Monopolist (i.e. all products indexed by 'prodIndex').

calcPricesHypoMon is identical, but for price levels.

diversionHypoMon returns a k x k matrix of diversions, where element i,j is the diversion from product i to product j.

References

U.S. Department of Justice and Federal Trade Commission, *Horizontal Merger Guidelines*. Washington DC: U.S. Department of Justice, 2010. https://www.justice.gov/atr/horizontal-merger-guidelines-08192 (accessed May 5, 2021).

Diagnostics-Methods *Methods for Calculating Diagnostics*

Description

Computes the percentage difference between predicted and observed pre-merger prices, shares, margins and market elasticities (if supplied) . 'labels' is used to specify row labels.

Usage

```
## S4 method for signature 'Bertrand'
calcDiagnostics(object, labels = object@labels)

## S4 method for signature 'VertBargBertLogit'
calcDiagnostics(object, labels = object@down@labels)

## S4 method for signature 'Cournot'
calcDiagnostics(object)
```

Arguments

object	An instance of one of the classes listed above.
labels	A length-k vector of product labels. Default is object@labels.

Diversion-Methods *Methods For Calculating Diversion*

Description

Calculate the diversion matrix between any two products in the market.

Usage

```
## S4 method for signature 'Bertrand'
diversion(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'AIDS'
diversion(object, preMerger = TRUE, revenue = TRUE)

## S4 method for signature 'VertBargBertLogit'
diversion(object, preMerger = TRUE, revenue = TRUE)
```

Arguments

object	An instance of one of the classes listed above.
preMerger	If TRUE, calculates pre-merger price elasticities. If FALSE, calculates post-merger price elasticities. Default is TRUE.
revenue	If TRUE, calculates revenue diversion. If FALSE, calculates quantity diversion. Default is TRUE for 'Bertrand' and FALSE for 'AIDS'.

Details

For Bertrand, when 'revenue' is FALSE (the default), this method uses the results from the merger calibration and simulation to compute the *quantity* diversion matrix between any two products in the market. Element i,j of this matrix is the quantity diversion from product i to product j, or the proportion of product i's sales that leave (go to) i for (from) j due to a increase (decrease) in i's price. Mathematically, quantity diversion is $\frac{-\epsilon_{ji} share_j}{\epsilon_{ii} share_i}$, where ϵ_{ij} is the cross-price elasticity from i to j.

When 'revenue' is TRUE, this method computes the revenue diversion matrix between any two products in the market. Element i,j of this matrix is the revenue diversion from product i to product j, or the proportion of product i's revenues that leave (go to) i for (from) j due to a increase (decrease) in i's price. Mathematically, revenue diversion is $-\frac{\epsilon_{ji}(\epsilon_{jj}-1)r_j}{\epsilon_{jj}(\epsilon_{ii}-1)r_j}$ where r_i is the revenue share of product i.

When 'preMerger' is TRUE, diversions are calculated at pre-merger equilibrium prices, and when 'preMerger' is FALSE, they are calculated at post-merger equilibrium prices.

For AIDS, when 'revenue' is TRUE (the default), this method computes the *revenue* diversion matrix between any two products in the market. For AIDS, the revenue diversion from i to j is $\frac{\beta_{ji}}{\beta_i j}$, where β_{ij} is the percentage change in product i's revenue due to a change in j's price.

When 'revenue' is FALSE, this `callNextMethod` is invoked. Will yield a matrix of NAs if the user did not supply prices.

When 'preMerger' is TRUE, diversions are calculated at pre-merger equilibrium prices, and when 'preMerger' is FALSE, they are calculated at post-merger equilibrium prices.

Value

returns a k x k matrix of diversion ratios, where the i,jth element is the diversion from i to j.

`Elast-Methods` *Methods For Calculating Own and Cross-Price Elasticities*

Description

Calculate the own and cross-price elasticity between any two products in the market.

Usage

```
## S4 method for signature 'Cournot'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Linear'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Logit'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'LogLin'
```

```
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'AIDS'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'LogitNests'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'CES'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'CESNests'
elast(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'VertBargBertLogit'
elast(object, preMerger = TRUE, market = FALSE)
```

Arguments

object	An instance of one of the classes listed above.
preMerger	If TRUE, calculates pre-merger price elasticities. If FALSE, calculates post-merger price elasticities. Default is TRUE.
market	If TRUE, calculates the market (aggregate) elasticity. If FALSE, calculates matrix of own- and cross-price elasticities. Default is FALSE.

Details

When 'market' is FALSE, this method computes the matrix of own and cross-price elasticities. Element i,j of this matrix is the percentage change in the demand for good i from a small change in the price of good j. When 'market' is TRUE, this method computes the market (aggregate) elasticities using share-weighted prices.

When 'preMerger' is TRUE, elasticities are calculated at pre-merger equilibrium prices and shares, and when 'preMerger' is FALSE, they are calculated at post-merger equilibrium prices and shares.

Value

returns a k x k matrix of own- and cross-price elasticities, where k is the number of products in the market.

HHI-Functions *Herfindahl-Hirschman Index*

Description

Calculate the Herfindahl-Hirschman Index with arbitrary ownership and control.

Let k denote the number of products produced by the merging parties below.

Usage

```
HHI(shares, owner = diag(length(shares)), control)
```

Arguments

shares	A length-k vector of product quantity shares.
owner	EITHER a vector of length k whose values indicate which of the merging parties produced a product OR a k x k matrix of ownership shares. Default is a diagonal matrix, which assumes that each product is owned by a separate firm.
control	EITHER a vector of length k whose values indicate which of the merging parties have the ability to make pricing or output decisions OR a k x k matrix of control shares. Default is a k x k matrix equal to 1 if 'owner' > 0 and 0 otherwise.

Details

All 'shares' must be between 0 and 1. When 'owner' is a matrix, the i,jth element of 'owner' should equal the percentage of product j's profits earned by the owner of product i. When 'owner' is a vector, HHI generates a k x k matrix of whose i,jth element equals 1 if products i and j are commonly owned and 0 otherwise. 'control' works in a fashion similar to 'owner'.

Value

HHI returns a number between 0 and 10,000

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Salop, Steven and O'Brien, Daniel (2000) "Competitive Effects of Partial Ownership: Financial Interest and Corporate Control" 67 Antitrust L.J. 559, pp. 559-614.

See Also

HHI-Methods for computing HHI following merger simulation.

Examples

```
## Consider a market with 5 products labeled 1-5. 1,2 are produced
## by Firm A, 2,3 are produced by Firm B, 3 is produced by Firm C.
## The pre-merger product market shares are

shares = c(.15,.2,.25,.35,.05)
owner  = c("A","A","B","B","C")
nprod  = length(shares)

HHI(shares,owner)

## Suppose that Firm A acquires a 75\% ownership stake in product 3, and
```

```
## Firm B get a 10\% ownership stake in product 1. Assume that neither
## firm cedes control of the product to the other.

owner <- diag(nprod)

owner[1,2] <- owner[2,1] <- owner[3,4] <- owner[4,3] <- 1
control <- owner
owner[1,1] <- owner[2,1] <- .9
owner[3,1] <- owner[4,1] <- .1
owner[1,3] <- owner[2,3] <- .75
owner[3,3] <- owner[4,3] <- .25

HHI(shares,owner,control)

## Suppose now that in addition to the ownership stakes described
## earlier, B receives 30\% of the control of product 1
control[1,1] <- control[2,1] <- .7
control[3,1] <- control[4,1] <- .3

HHI(shares,owner,control)
```

HHI-Methods *Methods For Calculating the Herfindahl-Hirschman Index*

Description

Computes the Herfindahl-Hirschman Index (HHI) using simulated market shares and either pre- or post-merger ownership information. Outside shares are excluded from the calculation.

Usage

```
## S4 method for signature 'Bertrand'
hhi(object, preMerger = TRUE, revenue = FALSE, insideonly = TRUE)

## S4 method for signature 'Cournot'
hhi(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'VertBargBertLogit'
hhi(object, preMerger = TRUE, revenue = FALSE, insideonly = TRUE)
```

Arguments

object	An instance of one of the classes listed above.
preMerger	If TRUE, returns pre-merger outcome. If FALSE, returns post-merger outcome. Default is TRUE.
revenue	If TRUE, returns revenues. If FALSE, returns quantities. Default is TRUE.
insideonly	If TRUE, excludes the share of the outside good from the calculation. Default is TRUE.

`Linear-Functions` *Linear and Log-Linear Demand Calibration and Merger Simulation*

Description

Calibrates consumer demand using either a linear or log-linear demand system and then simulates the prices effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products Bertrand game.

Let k denote the number of products produced by all firms.

Usage

```
linear(
  prices,
  quantities,
  margins,
  diversions,
  symmetry = TRUE,
  ownerPre,
  ownerPost,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceStart = prices,
  control.slopes,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

loglinear(
  prices,
  quantities,
  margins,
  diversions,
  ownerPre,
  ownerPost,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceStart = prices,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)
```

Arguments

`prices`	A length k vector product prices.

quantities	A length k vector of product quantities.
margins	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing, in which case diversion according to quantity share is assumed.
symmetry	If TRUE, requires the matrix of demand slope coefficients to be symmetric and homogeneous of degree 0 in prices, both of which suffice to make demand consistent with utility maximization theory. Default is TRUE.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
mcDelta	A length k vector where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
priceStart	A length k vector of prices used as the initial guess in the nonlinear equation solver. Default is 'prices'.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
...	Additional options to feed to the solver. See below.
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).

Details

Using price, quantity, and diversion information for all products in a market, as well as margin information for (at least) all the products of any firm, linear is able to recover the slopes and intercepts in a Linear demand system and then uses these demand parameters to simulate the price effects of a merger between two firms under the assumption that the firms are playing a differentiated Bertrand pricing game.

loglinear uses the same information as linear to uncover the slopes and intercepts in a Log-Linear demand system, and then uses these demand parameters to simulate the price effects of a merger of two firms under the assumption that the firms are playing a differentiated Bertrand pricing game.

'diversions' must be a square matrix whose off-diagonal elements [i,j] estimate the diversion ratio from product i to product j (i.e. the estimated fraction of i's sales that go to j due to a small increase in i's price). Off-diagonal elements are restricted to be non-negative (products are assumed to be substitutes), diagonal elements must equal -1, and rows must sum to 0 (negative if you wish to include an outside good) . If 'diversions' is missing, then diversion according to quantity share is assumed.

'ownerPre' and 'ownerPost' values will typically be equal to either 0 (element [i,j] is not commonly owned) or 1 (element [i,j] is commonly owned), though these matrices may take on any value between 0 and 1 to account for partial ownership.

Under linear demand, an analytic solution to the Bertrand pricing game exists. However, this solution can at times produce negative equilibrium quantities. To accommodate this issue, `linear` uses `constrOptim` to find equilibrium prices with non-negative quantities. `...` may be used to change the default options for constrOptim.

`loglinear` uses the non-linear equation solver `BBsolve` to find equilibrium prices. `...` may be used to change the default options for `BBsolve`.

Value

`linear` returns an instance of class `Linear`. `loglinear` returns an instance of `LogLin`, a child class of `Linear`.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

von Haefen, Roger (2002). "A Complete Characterization Of The Linear, Log-Linear, And Semi-Log Incomplete Demand System Models." *Journal of Agricultural and Resource Economics*, **27**(02). doi: 10.22004/ag.econ.31118.

See Also

`aids` for a demand system based on revenue shares rather than quantities.

Examples

```
## Simulate a merger between two single-product firms in a
## three-firm market with linear demand with diversions
## that are proportional to shares.
## This example assumes that the merger is between
## the first two firms



n <- 3 #number of firms in market
price    <- c(2.9,3.4,2.2)
quantity <- c(650,998,1801)
margin <- c(.435,.417,.370)


#simulate merger between firms 1 and 2
owner.pre <- diag(n)
owner.post <- owner.pre
owner.post[1,2] <- owner.post[2,1] <- 1
```

```
result.linear <- linear(price,quantity,margin,ownerPre=owner.pre,ownerPost=owner.post)

print(result.linear)           # return predicted price change
summary(result.linear)         # summarize merger simulation

elast(result.linear,TRUE)      # returns premerger elasticities
elast(result.linear,FALSE)     # returns postmerger elasticities

diversion(result.linear,TRUE)  # returns premerger diversion ratios
diversion(result.linear,FALSE) # returns postmeger diversion ratios

cmcr(result.linear)            # returns the compensating marginal cost reduction

CV(result.linear)              # returns representative agent compensating variation


## Implement the Hypothetical Monopolist Test
## for products 1 and 2 using a 5\% SSNIP

#HypoMonTest(result.linear,prodIndex=1:2)


## Get a detailed description of the 'Linear' class slots
showClass("Linear")

## Show all methods attached to the 'Linear' Class
showMethods(classes="Linear")

## Show which class have their own 'elast' method
showMethods("elast")

## Show the method definition for 'elast' and Class 'Linear'
getMethod("elast","Linear")
```

`Logit-Functions` *(Nested) Logit Demand Calibration and Merger Simulation)*

Description

Calibrates consumer demand using (Nested) Logit and then simulates the price effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products Bertrand pricing game.

Let k denote the number of products produced by all firms playing the Bertrand pricing game below.

Usage

```
logit(
```

```
    prices,
    shares,
    margins,
    diversions,
    ownerPre,
    ownerPost,
   normIndex = ifelse(isTRUE(all.equal(sum(shares), 1, check.names = FALSE)), 1, NA),
    mcDelta = rep(0, length(prices)),
    subset = rep(TRUE, length(prices)),
    insideSize = NA_real_,
    priceOutside = 0,
    priceStart = prices,
    isMax = FALSE,
    control.slopes,
    control.equ,
    labels = paste("Prod", 1:length(prices), sep = ""),
    ...
  )

  logit.nests(
    prices,
    shares,
    margins,
    diversions,
    ownerPre,
    ownerPost,
    nests = rep(1, length(shares)),
    normIndex = ifelse(sum(shares) < 1, NA, 1),
    mcDelta = rep(0, length(prices)),
    subset = rep(TRUE, length(prices)),
    priceOutside = 0,
    priceStart = prices,
    isMax = FALSE,
    constraint = TRUE,
    parmsStart,
    control.slopes,
    control.equ,
    labels = paste("Prod", 1:length(prices), sep = ""),
    ...
  )

  logit.nests.alm(
    prices,
    shares,
    margins,
    ownerPre,
    ownerPost,
    nests = rep(1, length(shares)),
```

```
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 0,
  priceStart = prices,
  isMax = FALSE,
  constraint = TRUE,
  parmsStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

logit.cap(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  capacitiesPre = rep(Inf, length(prices)),
  capacitiesPost = capacitiesPre,
  insideSize,
  normIndex = ifelse(sum(shares) < 1, NA, 1),
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 0,
  priceStart = prices,
  isMax = FALSE,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

logit.alm(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  mktElast = NA_real_,
  insideSize = NA_real_,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 0,
  priceStart = prices,
  isMax = FALSE,
  parmsStart,
```

```
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)

logit.cap.alm(
  prices,
  shares,
  margins,
  ownerPre,
  ownerPost,
  capacitiesPre = rep(Inf, length(prices)),
  capacitiesPost = capacitiesPre,
  mktElast = NA_real_,
  insideSize,
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  priceOutside = 0,
  priceStart = prices,
  isMax = FALSE,
  parmsStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)
```

Arguments

prices	A length k vector of product prices.
shares	A length k vector of product (quantity) shares. Values must be between 0 and 1.
margins	A length k vector of product margins, some of which may equal NA.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
normIndex	An integer equalling the index (position) of the inside product whose mean valuation will be normalized to 1. Default is 1, unless 'shares' sum to less than 1, in which case the default is NA and an outside good is assumed to exist.
mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.

subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
insideSize	An integer equal to total pre-merger units sold. If shares sum to one, this also equals the size of the market.
priceOutside	A length 1 vector indicating the price of the outside good. Default is 0.
priceStart	A length k vector of starting values used to solve for equilibrium price. Default is the 'prices' vector.
isMax	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
...	Additional options to feed to the BBsolve optimizer used to solve for equilibrium prices.
nests	A length k vector identifying the nest that each product belongs to.
constraint	if TRUE, then the nesting parameters for all non-singleton nests are assumed equal. If FALSE, then each non-singleton nest is permitted to have its own value. Default is TRUE.
parmsStart	For logit.cap.alm, a length-2 vector of starting values used to solve for the price coefficient and outside share (in that order). For logit.nets, rhe first element should always be the price coefficient and the remaining elements should be the nesting parameters. Theory requires the nesting parameters to be greater than the price coefficient. If missing then the random draws with the appropriate restrictions are employed.
capacitiesPre	A length k vector of pre-merger product capacities. Capacities must be at least as great as shares * insideSize.
capacitiesPost	A length k vector of post-merger product capacities.
mktElast	a negative value indicating market elasticity. Default is NA.

Details

Using product prices, quantity shares and all of the product margins from at least one firm, logit is able to recover the price coefficient and product mean valuations in a Logit demand model. logit then uses these calibrated parameters to simulate a merger between two firms.

logit.alm is identical to logit except that it assumes that an outside product exists and uses additional margin information to estimate the share of the outside good. If market elasticity is known, it may be supplied using the 'mktElast' argument.

logit.nests is identical to logit except that it includes the 'nests' argument which may be used to assign products to different nests. Nests are useful because they allow for richer substitution patterns between products. Products within the same nest are assumed to be closer substitutes than

products in different nests. The degree of substitutability between products located in different nests is controlled by the value of the nesting parameter sigma. The nesting parameters for singleton nests (nests containing only one product) are not identified and normalized to 1. The vector of sigmas is calibrated from the prices, revenue shares, and margins supplied by the user.

By default, all non-singleton nests are assumed to have a common value for sigma. This constraint may be relaxed by setting 'constraint' to FALSE. In this case, at least one product margin must be supplied from a product within each nest.

`logit.nests.alm` is identical to `logit.nests` except that it assumes that an outside product exists and uses additional margin information to estimate the share of the outside good.

`logit.cap` is identical to `logit` except that firms are playing the Bertrand pricing game under exogenously supplied capacity constraints. Unlike `logit`, `logit.cap` requires users to specify capacity constraints via 'capacities' and the number of potential customers in a market via 'mktSize'. 'mktSize' is needed to transform 'shares' into quantities that must be directly compared to 'capacities'.

In `logit`, `logit.nests` and `logit.cap`, if quantity shares sum to 1, then one product's mean value is not identified and must be normalized to 0. 'normIndex' may be used to specify the index (position) of the product whose mean value is to be normalized. If the sum of revenue shares is less than 1, both of these functions assume that the exists a k+1st product in the market whose price and mean value are both normalized to 0.

Value

`logit` returns an instance of class `Logit`. `logit.alm` returns an instance of `LogitALM`, a child class of `Logit`.. `logit.nests` returns an instance of `LogitNests`, a child class of `Logit`. `logit.cap` returns an instance of `LogitCap`, a child class of `Logit`.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Anderson, Simon, Palma, Andre, and Francois Thisse (1992). *Discrete Choice Theory of Product Differentiation*. The MIT Press, Cambridge, Mass.

Epstein, Roy and Rubinfeld, Daniel (2004). "Effects of Mergers Involving Differentiated Products."

Werden, Gregory and Froeb, Luke (1994). "The Effects of Mergers in Differentiated Products Industries: Structural Merger Policy and the Logit Model", *Journal of Law, Economics, and Organization*, **10**, pp. 407-426.

Froeb, Luke, Tschantz, Steven and Phillip Crooke (2003). "Bertrand Competition and Capacity Constraints: Mergers Among Parking Lots", *Journal of Econometrics*, **113**, pp. 49-67.

Froeb, Luke and Werden, Greg (1996). "Computational Economics and Finance: Modeling and Analysis with Mathematica, Volume 2." In Varian H (ed.), chapter Simulating Mergers among Noncooperative Oligopolists, pp. 177-95. Springer-Verlag, New York.

See Also

`ces`

Examples

```
## Calibration and simulation results from a merger between Budweiser and
## Old Style.
## Source: Epstein/Rubenfeld 2004, pg 80


prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
ownerPre <-c("BUD","OLD STYLE","MILLER","MILLER","OTHER-LITE","OTHER-REG")
ownerPost <-c("BUD","BUD","MILLER","MILLER","OTHER-LITE","OTHER-REG")
nests <- c("Reg","Reg","Reg","Light","Light","Reg")

price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)
shares   <- c(.066,.172,.253,.187,.099,.223)
margins <- c(.3830,.5515,.5421,.5557,.4453,.3769)

insideSize <- 1000

names(price) <-
  names(shares) <-
  names(margins) <-
  prodNames

result.logit <- logit(price,shares,margins,
                      ownerPre=ownerPre,ownerPost=ownerPost,
                      insideSize = insideSize,
                      labels=prodNames)



print(result.logit)           # return predicted price change
summary(result.logit)         # summarize merger simulation

elast(result.logit,TRUE)      # returns premerger elasticities
elast(result.logit,FALSE)     # returns postmerger elasticities

diversion(result.logit,TRUE)  # return premerger diversion ratios
diversion(result.logit,FALSE) # return postmerger diversion ratios


cmcr(result.logit)            #calculate compensating marginal cost reduction
upp(result.logit)            #calculate Upwards Pricing Pressure Index

CV(result.logit)              #calculate representative agent compensating variation


## Implement the Hypothetical Monopolist Test
## for BUD and OLD STYLE using a 5\% SSNIP

HypoMonTest(result.logit,prodIndex=1:2)
```

```
## Get a detailed description of the 'Logit' class slots
showClass("Logit")

## Show all methods attached to the 'Logit' Class
showMethods(classes="Logit")

## Show which classes have their own 'elast' method
showMethods("elast")

## Show the method definition for 'elast' and Class 'Logit'
getMethod("elast","Logit")



#
# Logit With capacity Constraints
#


cap     <- c(66,200,300,200,99,300) # BUD and OTHER-LITE are capacity constrained
result.cap <- logit.cap(price,shares,margins,capacitiesPre=cap,
                        insideSize=insideSize,ownerPre=ownerPre,
                        ownerPost=ownerPost,labels=prodNames)
print(result.cap)
```

Margins-Methods *Methods for Calculating Diagnostics*

Description

Computes equilibrium product margins assuming that firms are playing a Nash-Bertrand, Cournot, 2nd Score Auction, or Bargaining game. For "LogitCap", assumes firms are playing a Nash-Bertrand or Cournot game with capacity constraints.

Usage

```
## S4 method for signature 'Bertrand'
calcMargins(object, preMerger = TRUE, level = FALSE)

## S4 method for signature 'Bargaining2ndLogit'
calcMargins(object, preMerger = TRUE, exAnte = FALSE, level = TRUE)

## S4 method for signature 'BargainingLogit'
calcMargins(object, preMerger = TRUE, level = FALSE)

## S4 method for signature 'VertBargBertLogit'
calcMargins(object, preMerger = TRUE, level = FALSE)
```

```
## S4 method for signature 'Auction2ndCap'
calcMargins(object, preMerger = TRUE, exAnte = TRUE, level = FALSE)

## S4 method for signature 'Cournot'
calcMargins(object, preMerger = TRUE, level = FALSE)

## S4 method for signature 'AIDS'
calcMargins(object, preMerger = TRUE, level = FALSE)

## S4 method for signature 'LogitCap'
calcMargins(object, preMerger = TRUE, level = FALSE)

## S4 method for signature 'Auction2ndLogit'
calcMargins(object, preMerger = TRUE, exAnte = FALSE, level = TRUE)

## S4 method for signature 'Auction2ndLogitNests'
calcMargins(object, preMerger = TRUE, exAnte = FALSE, level = FALSE)
```

Arguments

`object`	An instance of one of the classes listed above.
`preMerger`	If TRUE, returns pre-merger outcome. If FALSE, returns post-merger outcome. Default is TRUE.
`level`	IF TRUE, return margins in dollars. If FALSE, returns margins in proportions. Default for most classes is FALSE.
`exAnte`	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE.

`Output-Methods` *Output Methods*

Description

This section contains three types of methods: calcShares, calcQuantities, and calcRevenues. calcShares computes equilibrium product shares assuming that firms are playing a Nash-Bertrand or Cournot game. 'revenue' takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

calcQuantities computes equilibrium product quantities assuming that firms are playing a Nash-Bertrand, 2nd Score Auction, or Cournot game. Setting 'market' to TRUE returns total market quantity.

calcRevenues computes equilibrium product revenues assuming that firms are playing a Nash-Bertrand, 2nd Score Auction, or Cournot game. Setting 'market' to TRUE returns total market revenue.

Usage

```
## S4 method for signature 'Cournot'
calcQuantities(object, preMerger = TRUE, market = FALSE, ...)

## S4 method for signature 'Stackelberg'
calcQuantities(object, preMerger = TRUE, market = FALSE, ...)

## S4 method for signature 'Linear'
calcQuantities(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Logit'
calcQuantities(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'LogLin'
calcQuantities(object, preMerger = TRUE, market = FALSE, ...)

## S4 method for signature 'AIDS'
calcQuantities(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'CES'
calcQuantities(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Bertrand'
calcRevenues(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Cournot'
calcRevenues(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'VertBargBertLogit'
calcRevenues(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'AIDS'
calcQuantities(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'CES'
calcRevenues(object, preMerger = TRUE, market = FALSE)

## S4 method for signature 'Auction2ndCap'
calcShares(object, preMerger = TRUE, exAnte = TRUE)

## S4 method for signature 'Cournot'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'Linear'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'VertBargBertLogit'
calcQuantities(object, preMerger = TRUE, market = FALSE)
```

```
## S4 method for signature 'VertBargBertLogit'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'VertBarg2ndLogit'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'Logit'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'AIDS'
calcShares(object, preMerger = TRUE, revenue = TRUE)

## S4 method for signature 'LogitNests'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'Auction2ndLogit'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'Auction2ndLogitNests'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'CES'
calcShares(object, preMerger = TRUE, revenue = FALSE)

## S4 method for signature 'CESNests'
calcShares(object, preMerger = TRUE, revenue = FALSE)
```

Arguments

object	An instance of one of the classes listed above.
preMerger	If TRUE, returns pre-merger outcome. If FALSE, returns post-merger outcome. Default is TRUE.
market	If TRUE, reports market-level summary. Otherwise reports product/plant level summary. Default is FALSE.
...	Additional arguments to pass to calcQuantities.
exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE.
revenue	If TRUE, returns revenues. If FALSE, returns quantities. Default is TRUE.

Ownership-methods *Methods for Manipulating Ownership Matrices*

Description

ownerToMatrix converts an ownership vector (or factor) to a k x k matrix of 1s and 0s.

ownerToVec converts a k x k ownership matrix to a length-k vector whose values identify an owner.

Usage

```
## S4 method for signature 'Antitrust'
ownerToMatrix(object, preMerger = TRUE)

## S4 method for signature 'VertBargBertLogit'
ownerToMatrix(object, preMerger = TRUE)

## S4 method for signature 'Antitrust'
ownerToVec(object, preMerger = TRUE)
```

Arguments

object	An instance of the Antitrust class.
preMerger	The 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger values, while FALSE invokes the method using the post-merger ownership structure.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Examples

```
showMethods(classes="Antitrust") # show all methods defined for the class
```

Params-Methods *Methods for Calculating Demand Parameters*

Description

The calcSlopes methods calculate demand parameters assuming that firms are playing a differentiated product Nash-Bertrand pricing game or (as in the case of the Cournot and Stackelberg classes), a Cournot game.

getNestsParms returns a matrix containing the calibrated nesting parameters.

getParms returns a list of model-specific demand parameters. 'digits' specifies the number of significant digit to return (default 10).

Usage

```
## S4 method for signature 'Cournot'
calcSlopes(object)

## S4 method for signature 'Stackelberg'
calcSlopes(object)

## S4 method for signature 'Linear'
calcSlopes(object)

## S4 method for signature 'Logit'
calcSlopes(object)

## S4 method for signature 'LogLin'
calcSlopes(object)

## S4 method for signature 'AIDS'
calcSlopes(object)

## S4 method for signature 'PCAIDS'
calcSlopes(object)

## S4 method for signature 'PCAIDSNests'
calcSlopes(object)

## S4 method for signature 'LogitCap'
calcSlopes(object)

## S4 method for signature 'LogitNests'
calcSlopes(object)

## S4 method for signature 'Auction2ndLogitNests'
calcSlopes(object)

## S4 method for signature 'LogitCapALM'
calcSlopes(object)

## S4 method for signature 'LogitNestsALM'
calcSlopes(object)

## S4 method for signature 'Auction2ndLogit'
calcSlopes(object)

## S4 method for signature 'Auction2ndLogitALM'
calcSlopes(object)

## S4 method for signature 'LogitALM'
calcSlopes(object)
```

```
## S4 method for signature 'CES'
calcSlopes(object)

## S4 method for signature 'CESALM'
calcSlopes(object)

## S4 method for signature 'CESNests'
calcSlopes(object)

## S4 method for signature 'BargainingLogit'
calcSlopes(object)

## S4 method for signature 'Bargaining2ndLogit'
calcSlopes(object)

## S4 method for signature 'VertBargBertLogit'
calcSlopes(object)

## S4 method for signature 'Bertrand'
getParms(object, digits = 10)

## S4 method for signature 'VertBargBertLogit'
getParms(object, digits = 10)

## S4 method for signature 'PCAIDSNests'
getNestsParms(object)
```

Arguments

object	An instance of the respective class (see description for the classes)
digits	Number of significant digits to report. Default is 2.

`Plot-Methods` *Methods For Calculating Upwards Pricing Pressure Index (Bertrand)*

Description

Use `ggplot` to plot pre- and post-merger demand, marginal cost and equilibria. 'scale' controls the amount above marginal cost and below equilbrium price that is plotted.

Usage

```
## S4 method for signature 'Bertrand'
plot(x, scale = 0.1)
```

Arguments

x	Used only in plot method. Should always be set equal to object.
scale	The proportion below marginal cost and above equilbrium price that should be plotted. Default is .1.

PriceDelta-Methods *Methods For Calculating Price Delta*

Description

For Antitrust, the method computes equilibrium price changes due to a merger assuming that firms are playing a Nash-Bertrand or Cournot game. This is a wrapper method for computing the difference between pre- and post-merger equilbrium prices.

For AIDS, the method computes equilibrium price changes due to a merger assuming that firms are playing a Nash-Bertrand or Cournot game and LA-AIDS. This method calls a non-linear equations solver to find a sequence of price changes that satisfy the Bertrand FOCs.

Usage

```
## S4 method for signature 'Antitrust'
calcPriceDelta(
  object,
  levels = FALSE,
  market = FALSE,
  index = c("paasche", "laspeyres"),
  ...
)

## S4 method for signature 'Cournot'
calcPriceDelta(object, levels = FALSE, market = TRUE, ...)

## S4 method for signature 'VertBargBertLogit'
calcPriceDelta(object, levels = FALSE, market = FALSE, ...)

## S4 method for signature 'AIDS'
calcPriceDelta(
  object,
  isMax = FALSE,
  levels = FALSE,
  subset,
  market = FALSE,
  index = c("paasche", "laspeyres"),
  ...
)

## S4 method for signature 'Auction2ndLogit'
```

```
calcPriceDelta(
  object,
  levels = TRUE,
  market = FALSE,
  exAnte = ifelse(market, TRUE, FALSE),
  ...
)
```

Arguments

object	An instance of one of the classes listed above.
levels	If TRUE, report results in levels. If FALSE, report results in percents. Default is FALSE.
market	If TRUE, calculates (post-merger) share-weighted average of metric. Default is FALSE.
index	If "paasche",calculates market-wide price changes using post-merger predicted shares. If "laspeyres", calculates price index using pre-merger shares. Default is "paasche".
...	Additional values that may be used to change the default values of the non-linear equation solver.
isMax	If TRUE, uses numerical derivatives to determine if equilibrium price vector is a local maximum. Default is FALSE.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE, unless 'market' is TRUE.

`Prices-Methods` *"Calculating Prices" Methods*

Description

For Auction2ndCap, the calcPrices method computes the expected price that the buyer pays, conditional on the buyer purchasing from a particular firm.

For Logit, the calcPrices method computes either pre-merger or post-merger equilibrium prices under the assumptions that consumer demand is Logit and firms play a differentiated product Bertrand Nash pricing game.

For LogitCap, the calcPrices method computes either pre-merger or post-merger equilibrium shares under the assumptions that consumer demand is Logit and firms play a differentiated product Bertrand Nash pricing game with capacity constraints.

For Logit, the calcPrices method computes either pre-merger or post-merger equilibrium prices under the assumptions that consumer demand is Logit and firms play a differentiated product Bertrand Nash pricing game.

For LogLin, the calcPrices method computes either pre-merger or post-merger equilibrium prices under the assumptions that consumer demand is Log-Linear and firms play a differentiated product Bertrand Nash pricing game.

For AIDS, the calcPrices method computes either pre-merger or post-merger equilibrium prices under the assumptions that consumer demand is AIDS and firms play a differentiated product Bertrand Nash pricing game. It returns a length-k vector of NAs if the user did not supply prices.

Usage

```
## S4 method for signature 'Cournot'
calcPrices(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
calcPrices(object, preMerger = TRUE, exAnte = TRUE)

## S4 method for signature 'Logit'
calcPrices(object, preMerger = TRUE, isMax = FALSE, subset, ...)

## S4 method for signature 'Auction2ndLogit'
calcPrices(object, preMerger = TRUE, exAnte = FALSE)

## S4 method for signature 'LogitCap'
calcPrices(object, preMerger = TRUE, isMax = FALSE, subset, ...)

## S4 method for signature 'Linear'
calcPrices(object, preMerger = TRUE, subset, ...)

## S4 method for signature 'LogLin'
calcPrices(object, preMerger = TRUE, subset, ...)

## S4 method for signature 'AIDS'
calcPrices(object, preMerger = TRUE, ...)

## S4 method for signature 'BargainingLogit'
calcPrices(object, preMerger = TRUE, isMax = FALSE, subset, ...)

## S4 method for signature 'VertBargBertLogit'
calcPrices(object, preMerger = TRUE, ...)

## S4 method for signature 'VertBarg2ndLogit'
calcPrices(object, preMerger = TRUE, ...)
```

Arguments

object	An instance of the respective class (see description for the classes)
preMerger	If TRUE, the pre-merger ownership structure is used. If FALSE, the post-merger ownership structure is used. Default is TRUE.

exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE.
isMax	If TRUE, a check is run to determine if the calculated equilibrium price vector locally maximizes profits. Default is FALSE.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded. Default is a length k vector of TRUE.
...	For Logit, additional values that may be used to change the default values of BBsolve, the non-linear equation solver. For others, additional values that may be used to change the default values of constrOptim, the non-linear equation solver used to enforce non-negative equilibrium quantities.

PS-methods *Producer Surplus Methods*

Description

In the following methods, calcProducerSurplus computes the expected profits of each supplier with the game depending on the class. The available classes are: Bertrand, Cournot, and Auction2ndCap.

calcProducerSurplusGrimTrigger is a method that may be used to explore how a merger affects firms' incentives to collude.

Usage

```
## S4 method for signature 'Bertrand'
calcProducerSurplus(object, preMerger = TRUE)

## S4 method for signature 'VertBargBertLogit'
calcProducerSurplus(object, preMerger = TRUE)

## S4 method for signature 'Auction2ndCap'
calcProducerSurplus(object, preMerger = TRUE, exAnte = TRUE)

## S4 method for signature 'Cournot'
calcProducerSurplus(object, preMerger = TRUE)

## S4 method for signature 'Bertrand'
calcProducerSurplusGrimTrigger(
  object,
  coalition,
  discount,
  preMerger = TRUE,
```

```
  isCollusion = FALSE,
  ...
)
```

Arguments

object	An instance of one of the classes listed above.
preMerger	If TRUE, returns pre-merger outcome. If FALSE, returns post-merger outcome. Default is TRUE.
exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is TRUE.
coalition	A length c vector of integers indicating the index of the products participating in the coalition.
discount	A length k vector of values between 0 and 1 that represent the product-specific discount rate for all products produced by firms particiapting in the coalition. NAs are allowed.
isCollusion	TRUE recalculates demand and cost parameters under the assumption that the coalition specified in 'coalition' is operating pre-merger. FALSE (the default) uses demand and cost parameters calculated from the 'ownerPre' matrix.
...	Additional argument to pass to calcPrices (for `calcProducerSurplusGrimTrigger`)

Details

calcProducerSurplusGrimTrigger calculates 'preMerger' product producer surplus (as well as other statistics – see below), under the assumption that firms are playing an N-player iterated Prisoner's Dilemma where in each period a coalition of firms decides whether to *cooperate* with one another by setting the joint surplus maximizing price on some 'coalition' of their products, or *defect* from the coalition by setting all of their products' prices to optimally undercut the prices of the coalition's products. Moreover, firms are assumed to play Grim Trigger strategies where each firm cooperates in the current period so long as *every* firm in the coalition cooperated last period and defects otherwise. product level 'discount' rates are then employed to determine whether a firm's discounted surplus from remaining in the coalition are greater than its surplus from optimally undercutting the coalition prices' for one period plus its discounted surplus when all firms set Nash-Bertrand prices in all subsequent periods.

Value

`calcProducerSurplusGrimTrigger` returns a data frame with rows equal to the number of products produced by any firm participating in the coalition and the following 5 columns

- Discount:The user-supplied discount rate
- Coord:Single period producer surplus from coordinating
- Defect:Single period producer surplus from defecting
- Punish:Single period producer surplus from punishing using Bertrand price

- IC:TRUE if the discounted producer surplus from coordinating across all firm products are greater than the surplus from defecting across all firm products for one period and receiving discounted Bertrand surplus for all subsequent periods under Grim Trigger.

`Show-Methods` *Show Method*

Description

Displays the percentage change in prices due to the merger.

Usage

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

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

Arguments

`object`	An instance of the Antitrust class.

`Sim-Functions` *Merger Simulation With User-Supplied Demand Parameters*

Description

Simulates the price effects of a merger between two firms with user-supplied demand parameters under the assumption that all firms in the market are playing either a differentiated products Bertrand pricing game, 2nd price (score) auction, or bargaining game.

Let k denote the number of products produced by all firms below.

Usage

```
sim(
  prices,
  supply = c("bertrand", "auction", "bargaining", "bargaining2nd"),
 demand = c("Linear", "AIDS", "LogLin", "Logit", "CES", "LogitNests", "CESNests",
    "LogitCap"),
  demand.param,
  ownerPre,
  ownerPost,
  nests,
  capacities,
```

```
  mcDelta = rep(0, length(prices)),
  subset = rep(TRUE, length(prices)),
  insideSize = 1,
  priceOutside,
  priceStart,
  bargpowerPre = rep(0.5, length(prices)),
  bargpowerPost = bargpowerPre,
  labels = paste("Prod", 1:length(prices), sep = ""),
  ...
)
```

Arguments

prices	A length k vector of product prices.
supply	A character string indicating how firms compete with one another. Valid values are "bertrand" (Nash Bertrand), "auction2nd" (2nd score auction), "bargaining", or "bargaining2nd".
demand	A character string indicating the type of demand system to be used in the merger simulation. Supported demand systems are linear ('Linear'), log-linear('LogLin'), logit ('Logit'), nested logit ('LogitNests'), ces ('CES'), nested CES ('CESNests') and capacity constrained Logit ('LogitCap').
demand.param	See Below.
ownerPre	EITHER a vector of length k whose values indicate which firm produced a product pre-merger OR a k x k matrix of pre-merger ownership shares.
ownerPost	EITHER a vector of length k whose values indicate which firm produced a product after the merger OR a k x k matrix of post-merger ownership shares.
nests	A length k vector identifying the nest that each product belongs to. Must be supplied when 'demand' equals 'CESNests' and 'LogitNests'.
capacities	A length k vector of product capacities. Must be supplied when 'demand' equals 'LogitCap'.
mcDelta	A vector of length k where each element equals the proportional change in a product's marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
insideSize	A length 1 vector equal to total units sold if 'demand' equals "logit", or total revenues if 'demand' equals "ces".
priceOutside	A length 1 vector indicating the price of the outside good. This option only applies to the 'Logit' class and its child classes Default for 'Logit','LogitNests', and 'LogitCap' is 0, and for 'CES' and 'CesNests' is 1.
priceStart	A length k vector of starting values used to solve for equilibrium price. Default is the 'prices' vector for all values of demand except for 'AIDS', which is set equal to a vector of 0s.

bargpowerPre	A length k vector of pre-merger bargaining power parameters. Values must be between 0 (sellers have the power) and 1 (buyers the power). Ignored if 'supply' not equal to "bargaining" or bargaining2nd.
bargpowerPost	A length k vector of post-merger bargaining power parameters. Values must be between 0 (sellers have the power) and 1 (buyers the power). Default is 'bargpowerPre'. Ignored if 'supply' not equal to "bargaining".
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
...	Additional options to feed to the optimizer used to solve for equilibrium prices.

Details

Using user-supplied demand parameters, sim simulates the effects of a merger in a market where firms are playing a differentiated products pricing game.

If 'demand' equals 'Linear', 'LogLin', or 'AIDS', then 'demand.param' must be a list containing 'slopes', a k x k matrix of slope coefficients, and 'intercepts', a length-k vector of intercepts. Additionally, if 'demand' equals 'AIDS', 'demand.param' must contain 'mktElast', an estimate of aggregate market elasticity. For 'Linear' demand models, sim returns an error if any intercepts are negative, and for both 'Linear', 'LogLin', and 'AIDS' models, sim returns an error if not all diagonal elements of the slopes matrix are negative.

If 'demand' equals 'Logit' or 'LogitNests', then 'demand.param' must equal a list containing

- alphaThe price coefficient.
- meanvalA length-k vector of mean valuations 'meanval'. If none of the values of 'meanval' are zero, an outside good is assumed to exist.

If demand equals 'CES' or 'CESNests', then 'demand.param' must equal a list containing

- gamma The price coefficient,
- alphaThe coefficient on the numeraire good. May instead be calibrated using 'shareInside',
- meanvalA length-k vector of mean valuations 'meanval'. If none of the values of 'meanval' are zero, an outside good is assumed to exist,
- shareInside The budget share of all products in the market. Default is 1, meaning that all consumer wealth is spent on products in the market. May instead be specified using 'alpha'.

Value

sim returns an instance of the class specified by the 'demand' argument.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

See Also

The S4 class documentation for: Linear, AIDS, LogLin, Logit, LogitNests, CES, CESNests

Examples

```
## Calibration and simulation results from a merger between Budweiser and
## Old Style. Note that the in the following model there is no outside
## good; BUD's mean value has been normalized to zero.

## Source: Epstein/Rubenfeld 2004, pg 80


prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
ownerPre <-c("BUD","OLD STYLE","MILLER","MILLER","OTHER-LITE","OTHER-REG")
ownerPost <-c("BUD","BUD","MILLER","MILLER","OTHER-LITE","OTHER-REG")
nests <- c("Reg","Reg","Reg","Light","Light","Reg")

price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)

demand.param=list(alpha=-48.0457,
                  meanval=c(0,0.4149233,1.1899885,0.8252482,0.1460183,1.4865730)
)

sim.logit <- sim(price,supply="bertrand",demand="Logit",demand.param,
                 ownerPre=ownerPre,ownerPost=ownerPost)



print(sim.logit)           # return predicted price change
summary(sim.logit)         # summarize merger simulation

elast(sim.logit,TRUE)      # returns premerger elasticities
elast(sim.logit,FALSE)     # returns postmerger elasticities

diversion(sim.logit,TRUE)  # return premerger diversion ratios
diversion(sim.logit,FALSE) # return postmerger diversion ratios


cmcr(sim.logit)            #calculate compensating marginal cost reduction
upp(sim.logit)             #calculate Upwards Pricing Pressure Index

CV(sim.logit)              #calculate representative agent compensating variation
```

`summary-methods` *Summary Methods*

Description

Summary methods for the `Bertrand`, `Auction2ndCap`, `Cournot`, and `Auction2ndLogit` classes. Summarizes the effect of the merger, including price and revenue changes.

Usage

```
## S4 method for signature 'Bertrand'
summary(
  object,
  revenue = TRUE,
  shares = TRUE,
  levels = FALSE,
  parameters = FALSE,
  market = FALSE,
  insideOnly = TRUE,
  digits = 2,
  ...
)

## S4 method for signature 'VertBargBertLogit'
summary(
  object,
  revenue = TRUE,
  levels = FALSE,
  parameters = FALSE,
  market = FALSE,
  insideOnly = TRUE,
  digits = 2,
  ...
)

## S4 method for signature 'Auction2ndCap'
summary(object, exAnte = FALSE, parameters = FALSE, market = TRUE, digits = 2)

## S4 method for signature 'Cournot'
summary(
  object,
  market = FALSE,
  revenue = FALSE,
  shares = FALSE,
  levels = FALSE,
  parameters = FALSE,
  digits = 2,
  ...
)

## S4 method for signature 'Auction2ndLogit'
summary(object, levels = TRUE, revenue = FALSE, ...)
```

Arguments

object	an instance of class Bertrand, Auction2ndCap, Cournot, or Auction2ndLogit
revenue	When TRUE, returns revenues, when FALSE returns quantitities. Default is

	TRUE.
shares	When TRUE, returns shares, when FALSE returns quantities (when possible). Default is TRUE.
levels	When TRUE, returns changes in levels rather than percents and quantities rather than shares, when FALSE, returns changes as a percent and shares rather than quantities. Default is FALSE.
parameters	When TRUE, displays all demand parameters. Default is FALSE.
market	When TRUE, displays aggregate information about the effect of a tariff. When FALSE displays product-specific (or in the case of Cournot, plant-specific) effects. Default is FALSE.
insideOnly	When TRUE, rescales shares on inside goods to sum to 1. Default is FALSE.
digits	Number of significant digits to report. Default is 2.
...	Allows other objects to be passed to a CV method.
exAnte	If 'exAnte' equals TRUE then the *ex ante* expected result for each firm is produced, while FALSE produces the expected result conditional on each firm winning the auction. Default is FALSE.

SupplyChain-Functions *Supply Chain Merger Simulation*

Description

Calibrates consumer demand using (Nested) Logit and then simulates the price effect of a merger between two firms under the assumption that all firms in the market are playing a differentiated products Bertrand pricing game.

Let k denote the number of products produced by all firms playing the Bertrand pricing game below.

Usage

```
vertical.barg(
  supplyDown = c("bertrand", "2nd"),
  sharesDown,
  pricesDown,
  marginsDown,
  ownerPreDown,
  ownerPostDown,
  nests = rep(NA, length(pricesDown)),
  diversions = diversions,
  mcDeltaDown = rep(0, length(pricesDown)),
  pricesUp,
  marginsUp,
  ownerPreUp,
  ownerPostUp,
  mcDeltaUp = rep(0, length(pricesUp)),
```

```
 normIndex = ifelse(isTRUE(all.equal(sum(sharesDown), 1, check.names = FALSE)), 1, NA),
  subset = rep(TRUE, length(pricesDown)),
  insideSize = NA_real_,
  priceOutside = 0,
  priceStartDown = pricesDown,
  priceStartUp = pricesUp,
  isMax = FALSE,
  constrain = c("global", "pair", "wholesaler", "retailer"),
  control.slopes,
  control.equ,
  labels = paste0("Prod", 1:length(pricesUp)),
  ...
)
```

Arguments

supplyDown	A length 1 character vector that specifies whether the downstream game that firms are playing is a Nash-Bertrand Pricing game ("bertrand") or a 2nd score auction ("2nd"). Default is "bertrand".
sharesDown	A length k vector of product (quantity) shares. Values must be between 0 and 1.
pricesDown	A length k vector of downstream product prices.
marginsDown	A length k vector of downstream product margins in levels (e.g. dollars), some of which may equal NA.
ownerPreDown	A vector of length k whose values indicate which downstream firm produced a product pre-merger.
ownerPostDown	A vector of length k whose values indicate which downstream firm produced a product post-merger.
nests	A length k factor of product nests.
diversions	A k x k matrix of diversion ratios with diagonal elements equal to -1. Default is missing, in which case diversion according to share is assumed.
mcDeltaDown	A vector of length k where each element equals the proportional change in a downstream firm's product-level marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.
pricesUp	A length k vector of upstream product prices.
marginsUp	A length k vector of upstream product margins in levels (e.g. dollars), some of which may equal NA.
ownerPreUp	A vector of length k whose values indicate which upstream firm produced a product pre-merger.
ownerPostUp	A vector of length k whose values indicate which upstream firm produced a product after the merger.
mcDeltaUp	A vector of length k where each element equals the proportional change in a upstream firm's product-level marginal costs due to the merger. Default is 0, which assumes that the merger does not affect any products' marginal cost.

normIndex	An integer equalling the index (position) of the inside product whose mean valuation will be normalized to 1. Default is 1, unless 'shares' sum to less than 1, in which case the default is NA and an outside good is assumed to exist.
subset	A vector of length k where each element equals TRUE if the product indexed by that element should be included in the post-merger simulation and FALSE if it should be excluded.Default is a length k vector of TRUE.
insideSize	An integer equal to total pre-merger units sold. If shares sum to one, this also equals the size of the market.
priceOutside	A length 1 vector indicating the price of the outside good. Default is 0.
priceStartDown	A length k vector of starting values used to solve for downstream equilibrium prices. Default is the 'pricesDown' vector.
priceStartUp	A length k vector of starting values used to solve for upstream equilibrium price. Default is the 'pricesUp' vector.
isMax	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
constrain	Specify calibration strategy for estimating bargaining power parameter. "global" (default) assumes bargaining parameter is the same across all participants,"pair" assumes that all wholesaler/retailer pairs have a distinct parameter,"wholesaler" assumes that each wholesaler's parameter is identical across negotiations, "retailer" assumes that each retailer's parameter is identical across negotiations.
control.slopes	A list of optim control parameters passed to the calibration routine optimizer (typically the calcSlopes method).
control.equ	A list of BBsolve control parameters passed to the non-linear equation solver (typically the calcPrices method).
labels	A k-length vector of labels. Default is "Prod#", where '#' is a number between 1 and the length of 'prices'.
...	Additional options to feed to the BBsolve optimizer used to solve for equilibrium prices.

Details

Using product prices, quantity shares and all of the product margins from at least one firm, logit is able to recover the price coefficient and product mean valuations in a Logit demand model. logit then uses these calibrated parameters to simulate a merger between two firms.

Value

When 'supplyDown' equals "bertand", vertical.barg returns an instance of class VertBargBertLogit. When 'supplyDown' equals "2nd", vertical.barg returns an instance of class VertBarg2ndLogit

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

References

Sheu, G. and Taragin, C. (2021), Simulating mergers in a vertical supply chain with bargaining. The RAND Journal of Economics, 52: 596-632.doi: 10.1111/17562171.12385.

Examples

```
## Vertical supply with 2 upstream firms,
## 2 downstream firms, each offering
## a single product.

shareDown <- c( 0.1293482, 0.1422541, 0.4631014, 0.2152962)
marginDown <- c( 0.2067533, 0.2572215, 0.3082511, 0.3539681)
priceDown <- c( 63.08158, 50.70465, 95.82960, 83.45267)
ownerPreDown <- paste0("D",rep(c(1,2),each=2))
marginUp <- c(0.5810900, 0.5331135, 0.5810900, 0.5331135)
priceUp <- c( 40.11427, 27.73734, 40.11427, 27.73734)
ownerPreUp <- paste0("U",rep(c(1,2),2))
priceOutSide <- 10


## Simulate an upstream horizontal merger
ownerPostDown <- ownerPreDown
ownerPostUp <- rep("U1",length(ownerPreUp))


simres_up <- vertical.barg(sharesDown =shareDown,
pricesDown = priceDown,
marginsDown = marginDown,
ownerPreDown = ownerPreDown,
ownerPostDown = ownerPreDown,
pricesUp = priceUp,
marginsUp = marginUp,
ownerPreUp = ownerPreUp,
ownerPostUp = ownerPostUp,
priceOutside = priceOutSide)


print(simres_up)
summary(simres_up)

## Simulate a downstream horizontal merger
ownerPostUp <- ownerPreUp
ownerPostDown <- ownerPreDown
ownerPostDown <- rep("D1",length(ownerPreDown))

simres_down <- vertical.barg(sharesDown =shareDown,
pricesDown = priceDown,
marginsDown = marginDown,
ownerPreDown = ownerPreDown,
ownerPostDown = ownerPostDown,
pricesUp = priceUp,
marginsUp = marginUp,
```

```
ownerPreUp = ownerPreUp,
ownerPostUp = ownerPreUp,
priceOutside = priceOutSide)


print(simres_down)
summary(simres_down)


## Simulate a vertical merger
ownerPostUp <- ownerPreUp
ownerPostDown <- ownerPreDown
ownerPostDown[ownerPostDown == "D1"] <- "U1"

simres_vert <- vertical.barg(sharesDown =shareDown,
pricesDown = priceDown,
marginsDown = marginDown,
ownerPreDown = ownerPreDown,
ownerPostDown = ownerPostDown,
pricesUp = priceUp,
marginsUp = marginUp,
ownerPreUp = ownerPreUp,
ownerPostUp = ownerPreUp,
priceOutside = priceOutSide)


print(simres_vert)
summary(simres_vert)
```

UPP-Methods *Methods For Calculating Upwards Pricing Pressure Index (Bertrand)*

Description

Calculate the Upwards Pricing Pressure Index for the products of merging firms playing a differentiated products Bertrand pricing game.

Usage

```
## S4 method for signature 'Bertrand'
upp(object)

## S4 method for signature 'AIDS'
upp(object)

## S4 method for signature 'Auction2ndLogit'
upp(object)
```

Arguments

object	An instance of one of the classes listed above.

Details

upp uses the results from the merger simulation and calibration to compute the upwards pricing pressure of the merger on each merging parties' products.

Value

upp returns a vector of length k equal to the net UPP for the merging parties' products and 0 for all other products.

See Also

upp.bertrand calculates net UPP without the need to first calibrate a demand system and simulate a merger.

Vertical-Classes *"Vertical" Classes*

Description

The "Vertical" classes are building blocks used to create other classes in this package. As such, it is most likely to be useful for developers who wish to code their own calibration/simulation routines.

The "VertBargBertLogit" class has the information for a Vertical Supply Chain with Logit demand and a downstream Nash-Bertrand Pricing game.

The "VertBarg2ndLogit" class has the information fora Vertical Supply Chain with Logit demand and a downstream 2nd Score Auction.

Slots

up	an instance of "Bargaining" class.
down	For "VertBargBertLogit", an instance of "Logit" class.For "VertBarg2ndLogit", an instance of "Auction2ndLogit" class.
constrain	A length 1 character vector equal to "global", "pair", "wholesaler", or "retailer.

Author(s)

Charles Taragin <ctaragin+antitrustr@gmail.com>

Index