Package ‘antitrust’

February 15, 2013

Type Package

Title Antitrust Library

Version 0.9

Date 2012-10-10

Author Michael Sandfort and Charles Taragin

Maintainer Charles Taragin <charles.taragin@usdoj.gov>

Depends R (>= 2.15.0), methods, nleqslv, numDeriv

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

License Unlimited

LazyLoad yes

Collate Antitrust.R Bertrand.R logit.R linear.R loglin.R logitALM.R logitNests.R logitCap.R ces.R cesNests.R aids.R pcaids.R pcaidsNests.R cmcr.R upp.R hhi.R sim.R

Repository CRAN

Date/Publication 2012-10-23 13:44:26

NeedsCompilation no

R topics documented:

`antitrust-package` *Antitrust Library*

Description

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

Details

Package:	antitrust
Type:	Package
Version:	0.9
Date:	2012-10-10
License:	Unlimited
LazyLoad:	yes

Disclaimer

The views expressed herein are entirely those of the authors and should not be purported to reflect those of the U.S. Department of Justice. The `antitrust` package has been released into the public domain without warranty of any kind, expressed or implied. Address: Economic Analysis Group, Antitrust Division, U.S. Department of Justice, 450 5th St. NW, Washington DC 20530. E-mail: charles.taragin@usdoj.gov and michael.sandfort@usdoj.gov.

Getting Started

1. Collect data on product prices, shares, margins and diversions (optional).
2. If you have data on many/all products in the market consider calibrating a demand system and simulating a merger with either a `aids`,`logit`, `ces`, `linear`, or `loglin` demand system.
3. If you only have data on the merging parties' products, consider using `cmcr.bertrand` or `cmcr.cournot` to uncover the marginal cost reductions needed to offset a post-merger increase.

Author(s)

Michael Sandfort and Charles Taragin

Maintainer: Charles Taragin `<charles.taragin@usdoj.gov>`

---

aids — *(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.

Usage

```
aids(shares,margins,prices,diversions,
                 ownerPre,ownerPost,
                 mcDelta=rep(0, length(shares)),
                 priceStart=runif(length(shares)),
                 isMax=FALSE,
                 labels=paste("Prod",1:length(shares),sep=""),
                 ...)

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

```
              mcDelta=rep(0, length(shares)),
              priceStart=runif(length(shares)),
              isMax=FALSE,
              labels=paste("Prod",1:length(shares),sep=""),
              ...)

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

Arguments

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

`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.

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

`mktElast` A negative number equal to the industry pre-merger price elasticity. Default is -1.

`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.

`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.

`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.

`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.

`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.

`labels` A k-length vector of labels.

`...` Additional options to feed to the `nleqslv` optimizer used to solve for equilibrium prices.

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 <charles.taragin@usdoj.gov>

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 \& 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
                       #representative consumer income
                       #CV can only be calculated if prices are supplied

CV(result.aids,14.5e12)     #calculate compensating variation in dollars
                            #14.5e12 is an estimate of total US GDP


## AIDS: 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.aids,prodIndex=1:2)
```

AIDS-class *Class "AIDS"*

Description

The "AIDS" class contains all the information needed to calibrate a AIDS demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

Objects from the Class

Objects can be created by using the constructor function `aids`.

Slots

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

`mktElast`: A negative number equal to the industry pre-merger price elasticity.

`priceStart`: A length k vector who elements equal to an initial guess of the proportional change in prices caused by the merger.

`priceDelta`: A length k vector containing the simulated price effects from the merger.

Extends

Class `Linear`, directly. Class `Bertrand`, by class "Linear", distance 2.

Methods

For all of methods containing the ‘preMerger’ argument, ‘preMerger’ takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcMargins signature(object ,preMerger=TRUE)` Calculates pre-merger or post-merger equilibrium margins.

calcPriceDelta signature(object,isMax=FALSE,...) Computes the proportional change in each products' price from the merger under the assumptions that consumer demand is AIDS and firms play a differentiated product Bertrand Nash pricing game.When isMax equals TRUE, a check is run to determine if the calculated equilibrium price vector locally maximizes profits. '...' may be used to change the default values of nleqslv, the non-linear equation solver.

calcPrices signature(object, preMerger = TRUE) Compute 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. return a vector of length-k vector of NAs if user did not supply prices.

calcPriceDeltaHypoMon signature(object,prodIndex,...) Calculates the price changes that a Hypothetical Monopolist would impose on its products relative to pre-merger prices.

calcShares signature(object, preMerger = TRUE) Computes either pre-merger or post-merger equilibrium quantity shares under the assumptions that consumer demand is AIDS and firms play a differentiated product Bertrand Nash pricing game.

calcSlopes signature(object) Uncover AIDS demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

cmcr signature(object) Calculates compensated marginal cost reduction, the percentage decrease in the marginal costs of the merging parties' products needed to offset a post-merger price increase.

CV signature(object) Calculate the amount of money a representative consumer would need to be paid to be just as well off as they were before the merger. Requires a length-k vector of pre-merger prices.

diversion signature(object, preMerger= TRUE) Computes a k x k matrix of diversion ratios.

elast signature(object , preMerger = TRUE) Computes a k x k matrix of own and cross-price elasticities.

show signature(object) Displays the percentage change in prices due to the merger.

summary signature(object,revenue=TRUE,parameters=FALSE,digits=2,..) Summarizes the effect of the merger, including price and revenue changes. Setting 'revenue' equal to FALSE reports quantity rather than revenue changes. Setting 'parameters' equal to TRUE reports all demand parameters. 'digits' controls the number of significant digits reported in output.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("AIDS")
```

Antitrust-class *Class "Antitrust"*

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.

Objects from the Class

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

Slots

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

`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.

Methods

Many of the methods described below contain a ‘preMerger’ argument. 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.

`calcPriceDelta signature(object)` Calculates the proportional change in product prices from a merger.

`ownerToMatrix signature(object, preMerger = TRUE)` Converts an ownership vector (or factor) to a k x k matrix of 1s and 0s.

`ownerToVec signature(object, preMerger = TRUE)` Converts a k x k ownership matrix to a length-k vector whose values identify an owner.

`show signature(object)` Displays the percentage change in prices due to the merger.

The "matrixOrList" and "matrixOrVector" Classes

The "matrixOrList" and "matrixOrVector" 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 <charles.taragin@usdoj.gov>

Examples

```
showClass("Antitrust")
```

`Bertrand-class` *Class "Bertrand"*

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.

Objects from the Class

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

Slots

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

`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

Methods

Many of the methods described below contain a 'preMerger' and 'revenue' argument. The 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the premerger values, while FALSE invokes the method using the post-merger ownership structure. The 'revenue' argument also takes on a value of TRUE or FALSE, where TRUE invokes the method using revenues, while FALSE invokes the method using quantities

`calcMC signature(object,preMerger=TRUE)` 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.

`calcMargins signature(object, preMerger = TRUE)` Compute either pre-merger or post-merger equilibrium margins under the assumption that firms play a differentiated product Bertrand Nash pricing game.

`cmcr signature(object)` Calculates compensated marginal cost reduction, the percentage decrease in the marginal costs of the merging parties' products needed to offset a post-merger price increase.

`HypoMonTest signature(object,prodIndex,ssnip=.05,...)` `HypoMonTest` implements the Hypothetical Monopolist Test for a given 'ssnip'.

`calcPriceDeltaHypoMon signature(object,prodIndex,...)` Compute 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 Bertrand equilibrium. '...' may be used to pass arguments to the optimizer.

diversionHypoMon `signature(object,prodIndex,...)` 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'*.

hhi `signature(object, preMerger= TRUE,revenue=FALSE)` Compute either the pre-merger or post-merger Herfindahl-Hirschman Index (HHI) under the assumption that firms play a differentiated product Bertrand Nash pricing game.

diversion `signature(object, preMerger = TRUE)` Computes a k x k matrix of diversion ratios.

summary `signature(object,revenue=TRUE,shares=TRUE,parameters=FALSE,digits=2)` Summarizes the effect of the merger, including price and revenue changes. Setting 'revenue' equal to FALSE reports quantities rather than revenues. Setting 'shares' to FALSE reports quantities rather than than shares (when possible). Setting 'parameters' equal to TRUE reports all demand parameters. 'digits' controls the number of significant digits reported in output.

upp `signature(object)` Calculate the Upwards Pricing Pressure (upp) index.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("Bertrand")
```

---

ces *(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.

Usage

```
ces(prices,shares,margins,
                ownerPre,ownerPost,
                shareInside = 1,
                normIndex=ifelse(sum(shares)<1,NA,1),
                mcDelta=rep(0,length(prices)),
                priceStart = prices,
                isMax=FALSE,
                labels=paste("Prod",1:length(prices),sep=""),
                ...
                )
```

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

Arguments

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

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.
nests	A length k vector identifying the nest that each product belongs to.
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.
shareInside	The proportion that a typical consumer spends on all products included in the 'prices' vector. Only needed to calculate compensating variation. Default is 1, meaning that all of a consumer's income is spent on products within the market.
normIndex	An integer specifying the product index against which the mean values of all other products are normalized. Default is 1.
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.
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.
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.

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.

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 nleqslv optimizer used to solve for equilibrium prices

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

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

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,
                 shareInside=.01,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
CV(result.ces,1e9)          #calculate compensating variation in dollars
                            #1e9 is an estimate of total US beer expenditure

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

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

`CES-class` *Class "CES"*

Description

The "CES" class contains all the information needed to calibrate a CES demand system and perform a merger analysis under the assumption that firms are playing a differentiated Bertrand pricing game.

Objects from the Class

Objects can be created by using the constructor function `ces`.

Slots

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

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

Extends

Class `Logit`, directly. Class `Bertrand`, by class "Logit", distance 2.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcShares signature(object, preMerger = TRUE,revenue=FALSE)` Compute either pre-merger or post-merger equilibrium revenue shares under the assumptions that consumer demand is CES and firms play a differentiated product Bertrand Nash pricing game. 'revenue' takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

`calcSlopes signature(object)` Uncover CES demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`CV signature(object, revenueInside)` Calculates compensating variation. If 'revenueInside' is missing, then CV returns compensating variation as a percent of the representative consumer's income. If 'revenueInside' equals the total expenditure on all products inside the market, then CV returns compensating variation in levels.

`elast signature(object, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("CES")
```

CESNests-class *Class "CESNests"*

Description

The "CESNests" class contains all the information needed to calibrate a nested CES demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

Objects from the Class

Objects can be created by using the constructor function `ces.nests`.

Slots

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

`nests`: A length k vector identifying the nest that each product belongs to.

`parmsStart`: A length k vector who elements equal an initial guess of the nesting parameter values.

`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.

Extends

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

Methods

For all of methods containing the ‘preMerger’ argument, ‘preMerger’ takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcShares signature(object, preMerger = TRUE, revenue = FALSE)` Compute either pre-merger or post-merger equilibrium revenue shares under the assumptions that consumer demand is nested CES and firms play a differentiated product Bertrand Nash pricing game. ‘revenue’ takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

`calcSlopes signature(object)` Uncover nested CES demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

CV `signature(object, revenueInside)` Calculates compensating variation. If 'revenueInside' is missing, then CV returns compensating variation as a percent of the representative consumer's income. If 'revenueInside' equals the total expenditure on all products inside the market, then CV returns compensating variation in levels.

elast `signature(object, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin `<charles.taragin@usdoj.gov>`

Examples

```
showClass("CESNests")
```

---

cmcr-methods *Methods For Calculating Compensating Marginal Cost Reductions and Upwards Pricing Pressure Index (Bertrand)*

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 'ANY'
cmcr(object)
## S4 method for signature 'ANY'
upp(object)
```

Arguments

object	An instance of one of the classes listed above.

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.

Like `cmcr`, 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

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

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

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

---

cmcr.bertrand — *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.

Usage

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


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

	Let k denote the number of products produced by the merging parties.
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.
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.
labels	A length-k vector of product labels.

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 upwards pricing pressure for each of the merging's parties' products , net any efficiency claims.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

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))
```

`cmcr.cournot` *Compensating Marginal Cost Reductions and Upwards Pricing Pressure (Cournot)*

Description

Calculate the average marginal cost reduction necessary to restore pre-merger prices, or the net Upwards Pricing Pressure in a two-product merger involving firms playing a homogeneous product Cournot pricing game.

Usage

```
cmcr.cournot(shares,mktElast)

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

Arguments

`shares`	A length-2 vector containing merging party quantity shares.
`mktElast`	A length-1 containing the industry elasticity.
`prices`	A length-2 vector of product prices.
`margins`	A length-2 vector of product margins.
`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.
`labels`	A length-2 vector of product labels.

Details

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

Value

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.

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.

See Also

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

Examples

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


cmcr.cournot(shares,industryElast)


 ## Calculate the necessary percentage cost reductions 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))
```

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.

Methods

`signature(object = c(Logit,LogitNests))` All the information needed to compute CV is already available within the Logit and Nested Logit classes.

`signature(object = c(CES, CESNests), revenueInside)` The CV method for the "CES" and nested "CES" classes has an additional parameter, 'revenueInside', which must be set equal to the total amount that consumers have spent on products inside the market in order for CV to be calculated.

`signature(object = AIDS , totalRevenue)` The CV method for "AIDS" has an additional parameter, 'totalRevenue', which should aggregate income (e.g. GDP). If supplied computes CV in terms of dollars. If missing, CV is calculated as a percentage change in aggregate in income. must be set equal to the vector of pre-merger prices for all products in the market in order for CV to be calculated.

`signature(object = c(Linear,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.

`defineMarketTools-methods`
Methods For Implementing The Hypothetical Monopolist Test

Description

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

Usage

```
## S4 method for signature 'Bertrand'
HypoMonTest(object,prodIndex,ssnip=.05,...)
## S4 method for signature 'ANY'
calcPricesHypoMon(object,prodIndex)
## S4 method for signature 'ANY'
calcPriceDeltaHypoMon(object,prodIndex)
## S4 method for signature 'Bertrand'
diversionHypoMon(object,prodIndex,...)
## S4 method for signature 'AIDS'
diversionHypoMon(object)
```

Arguments

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

Details

`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. http://www.justice.gov/atr/public/guidelines/hmg-2010.html (accessed July 29, 2011).

`diversion-methods` *Methods For Calculating Diversion*

Description

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

Usage

```
## S4 method for signature 'ANY'
diversion(object,preMerger=TRUE,revenue=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.
`revenue`	If TRUE, calculates revenue diversion. If FALSE, calculates quantity diversion. Default is TRUE for 'Bertrand' and FALSE for 'AIDS'.

Value

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

Methods

diversion `signature(object=Bertrand,preMerger=TRUE,revenue=FALSE)` 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.

diversion `signature(object=AIDS,preMerger=TRUE,revenue=TRUE)` 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 $\frac{\beta_{ji}}{\beta_{ij}}$, 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.

`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 'ANY'
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` *Herfindahl-Hirschman Index*

Description

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

Usage

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

Arguments

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

A length-k vector of product quantity shares.

`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 <charles.taragin@usdoj.gov>

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

other-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)
```

`linear` *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.

Usage

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


loglinear(prices,quantities,margins,
               diversions,
               ownerPre,ownerPost,
```

```
                mcDelta=rep(0,length(prices)),
                priceStart=prices,
                labels=paste("Prod",1:length(prices),sep=""),
                ...
        )
```

Arguments

	Let k denote the number of products produced by all firms.
`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 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.
`priceStart`	A length k vector of prices used as the initial guess in the nonlinear equation solver. Default is 'prices'.
`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.

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-Log demand system, and then uses these demand parameters to simulate the price effects of a merger two firms under the assumption that the firms are playing a differentiated Bertrand pricing game.

'diversion' must equal a square matrix whose elements are be between -1 and 1. If 'diversion' is missing, then diversion according to quantity share is assumed. If a square matrix is supplied, the off-diagonal elements [i,j] of 'diversion' 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). Off-diagonal elements are restricted to be positive (products are assumed to be substitutes). Diagonal elements must equal -1.

'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 `nleqslv` to find equilibrium prices. `...` may be used to change the default options for `nleqslv`.

Value

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

Author(s)

Charles Taragin `<charles.taragin@usdoj.gov>`

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). http://ideas.repec.org/a/ags/jlaare/31118.html.

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)


slopes <- matrix(
c(-2.3,  0.18,0.28,
   0.11, -2.4,0.1,
```

```
   0.13,   .16,-2.7),ncol=n)


margin <- -1/diag(slopes)


#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)
```

`Linear-class` *Class "Linear"*

Description

The "Linear" class contains all the information needed to calibrate a Linear demand system and perform a merger analysis under the assumption that firms are playing a differentiated Bertrand products pricing game.

Objects from the Class

Objects can be created by using the constructor function `linear`.

Slots

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

`intercepts`: A length k vector of demand intercepts.

`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 between 0 and 1.

`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.

`symmetry`: If TRUE, requires the matrix of demand slope coefficients to be consistent with utility maximization theory.

Extends

Class `Bertrand`, directly.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcPrices signature(object, preMerger = TRUE,...)` Compute 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. '...' may be used to change the default values of `constrOptim`, the non-linear equation solver used to enforce non-negative equilibrium quantities.

`calcPriceDeltaHypoMon signature(object,prodIndex,...)` Calculates the price changes that a Hypothetical Monopolist would impose on its products relative to pre-merger prices.

`calcQuantities signature(object, preMerger = TRUE)` Compute either pre-merger or post-merger equilibrium quantities under the assumptions that consumer demand is Linear and firms play a differentiated product Bertrand Nash pricing game.

`calcShares signature(object, preMerger = TRUE, revenue = FALSE)` Compute either pre-merger or post-merger equilibrium quantity shares under the assumptions that consumer demand is Linear and firms play a differentiated product Bertrand Nash pricing game.

`calcSlopes signature(object)` Uncover slopes and intercept from a Linear demand system. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`CV signature(object = "Linear")` Calculate the amount of money a representative consumer would need to be paid to be just as well off as they were before the merger.

`elast signature(object, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin `<charles.taragin@usdoj.gov>`

Examples

```
showClass("Linear")
```

---

logit *(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.

Usage

```
logit(prices,shares,margins,
                ownerPre,ownerPost,
                normIndex=ifelse(sum(shares)<1,NA,1),
                mcDelta=rep(0,length(prices)),
                priceStart = prices,
                isMax=FALSE,
                labels=paste("Prod",1:length(prices),sep=""),
                ...
                )
logit.alm(prices,shares,margins,
                ownerPre,ownerPost,
                mcDelta=rep(0,length(prices)),
                priceStart = prices,
                isMax=FALSE,
                parmsStart,
                labels=paste("Prod",1:length(prices),sep=""),
                ...
                )

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

```
logit.cap(prices,shares,margins,
                ownerPre,ownerPost,
                capacities,
                mktSize,
                normIndex=ifelse(sum(shares)<1,NA,1),
                mcDelta=rep(0,length(prices)),
                priceStart = prices,
                isMax=FALSE,
                labels=paste("Prod",1:length(prices),sep=""),
                ...
                )
```

Arguments

	Let k denote the number of products produced by all firms playing the Bertrand pricing game.
`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.
`nests`	A length k vector identifying the nest that each product belongs to.
`capacities`	A length k vector of product capacities. Capacities must be at least as great as shares * mktSize.
`mktSize`	An integer equal to the number of potential customers. If an outside option is present, should include individuals who chose that option.
`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.
`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 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.
`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.
`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.

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.

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 nleqslv 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.

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

`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.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 <charles.taragin@usdoj.gov>

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, \& 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)

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

result.logit <- logit(price,shares,margins,ownerPre=ownerPre,ownerPost=ownerPost,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

## Construct a matrix containing all candidate product markets satisfying a 5% SSNIP

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

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


#
# Logit With capacity Constraints
#

mktSize <- 1000
cap     <- c(66,200,300,200,99,300) # BUD and OTHER-LITE are capacity constrained
result.cap <- logit.cap(price,shares,margins,capacities=cap,mktSize=mktSize,ownerPre=ownerPre,ownerPost=ownerP
print(result.cap)
```

Logit-class *Class "Logit"*

Description

The "Logit" class contains all the information needed to calibrate a Logit demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

Objects from the Class

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

Slots

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

`prices`: A length k vector of product prices.

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

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

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

`priceStart`: A length k vector of starting values used to solve for equilibrium price.

`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.

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

Extends

Class `Bertrand`, directly.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcPrices signature(object = Logit, preMerger = TRUE,isMax=FALSE,...)` Compute 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. When isMax equals TRUE, a check is run to determine if the calculated equilibrium price vector locally maximizes profits. '...' may be used to change the default values of `nleqslv`, the non-linear equation solver.

`calcPriceDeltaHypoMon signature(object = Logit,prodIndex,...)` Calculates the price changes that a Hypothetical Monopolist would impose on its products relative to pre-merger prices.

`calcShares signature(object = Logit, preMerger = TRUE,revenue = FALSE)` Compute 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. 'revenue' takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

`calcSlopes signature(object = Logit)` Uncover Logit demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`CV signature(object = Logit)` Calculate the amount of money a representative consumer would need to be paid to be just as well off as they were before the merger.

`elast signature(object = Logit, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("Logit")
```

LogitALM-class *Class "LogitALM"*

Description

The "LogitALM" class contains all the information needed to calibrate a Logit demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game with capacity constraints.

Objects from the Class

Objects can be created by using the constructor function `logit.cap`.

Slots

`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 .

Extends

Class `Logit`, directly. Class `Bertrand`, by class "Logit", distance 2.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcSlopes signature(object)` Uncover Logit ALM demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game with capacity constraints.

Author(s)

Charles Taragin `<charles.taragin@usdoj.gov>`

Examples

```
showClass("LogitALM")
```

LogitCap-class *Class "LogitCap"*

Description

The "LogitCap" class contains all the information needed to calibrate a Logit demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game with capacity constraints.

Objects from the Class

Objects can be created by using the constructor function `logit.cap`.

Slots

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

`mktSize`: A vector of length 1 equal to the number of consumers in the market. This count should include the number of consumers who purchase the outside option (if specified).

`capacities`: A length k vector whose elements equal product capacities.

Extends

Class `Logit`, directly. Class `Bertrand`, by class "Logit", distance 2.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcPrices signature(object, preMerger = TRUE)` Compute 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.

`calcQuantities signature(object, preMerger  = TRUE)` Compute either pre-merger or post-merger equilibrium quantities under the assumptions that consumer demand is Linear and firms play a differentiated product Bertrand Nash pricing game.

`calcMargins signature(object, preMerger = TRUE)` Computes equilibrium product margins assuming that firms are playing a Nash-Bertrand pricing game with capacity constraints. Note that margins for capacity constrained firms are not identified from the firm's first-order conditions, and so must be supplied by the user.

`calcSlopes signature(object)` Uncover Logit demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game with capacity constraints.

Author(s)

Charles Taragin `<charles.taragin@usdoj.gov>`

Examples

```
showClass("LogitCap")
```

`LogitNests-class` *Class "LogitNests"*

Description

The "LogitNests" class contains all the information needed to calibrate a nested Logit demand system and perform a merger analysis under the assumption that firms are playing a differentiated products Bertrand pricing game.

Objects from the Class

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

Slots

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

`nests`: A length k vector identifying the nest that each product belongs to.

`parmsStart`: A length k vector who elements equal an initial guess of the nesting parameter values.

`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.

Extends

Class `Logit`, directly. Class `Bertrand`, by class "Logit", distance 2.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcShares signature(object, preMerger = TRUE,revenue = FALSE)` Compute 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. 'revenue' takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

`calcSlopes signature(object)` Uncover nested Logit demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`CV signature(object)` Calculate the amount of money a representative consumer would need to be paid to be just as well off as they were before the merger.

`elast signature(object, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("LogitNests")
```

`LogLin-class` *Class "LogLin"*

Description

The "LogLin" class contains all the information needed to calibrate a Log-Linear demand system and perform a merger analysis under the assumption that firms are playing a differentiated Bertrand products pricing game.

Objects from the Class

Objects can be created by using the constructor function `loglin`.

Slots

`symmetry`: If TRUE, requires the matrix of demand slope coefficients to be consistent with utility maximization theory Default is FALSE

Extends

Class `Linear`, directly. Class `Bertrand`, by class "Linear", distance 2.

Methods

For all of methods containing the 'preMerger' argument, 'preMerger' takes on a value of TRUE or FALSE, where TRUE invokes the method using the pre-merger ownership structure, while FALSE invokes the method using the post-merger ownership structure.

`calcPrices` `signature(object, preMerger = TRUE)` Compute 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.

`calcPriceDeltaHypoMon` `signature(object,prodIndex,...)` Calculates the price changes that a Hypothetical Monopolist would impose on its products relative to pre-merger prices.

`calcQuantities` `signature(object, preMerger = TRUE)` Compute either pre-merger or post-merger equilibrium quantities under the assumptions that consumer demand is Log-Linear and firms play a differentiated product Bertrand Nash pricing game.

`calcSlopes` `signature(object)` Uncover slopes and intercept from a Log-Linear demand system. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`elast` `signature(object, preMerger = TRUE)` Computes a k x k matrix of own and cross-price elasticities.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("LogLin")
```

---

other-methods *Other Useful Methods*

Description

Below is a description of methods that users may find helpful.

Usage

```
## S4 method for signature 'ANY'
calcShares(object,preMerger=TRUE,revenue=FALSE)
## S4 method for signature 'ANY'
calcQuantities(object,preMerger=TRUE)
## S4 method for signature 'ANY'
calcPrices(object,preMerger=TRUE,...)
## S4 method for signature 'Antitrust'
calcPriceDelta(object)
## S4 method for signature 'AIDS'
calcPriceDelta(object,isMax=FALSE,...)
## S4 method for signature 'ANY'
calcMargins(object,preMerger=TRUE)
## S4 method for signature 'Bertrand'
calcMC(object,preMerger=TRUE)
## S4 method for signature 'ANY'
calcSlopes(object,preMerger=TRUE)
## S4 method for signature 'PCAIDSNests'
getNestsParms(object)
## S4 method for signature 'Bertrand'
hhi(object,preMerger=TRUE,revenue=FALSE)
## S4 method for signature 'Antitrust'
ownerToMatrix(object,preMerger=TRUE)
## S4 method for signature 'Antitrust'
ownerToVec(object,preMerger=TRUE)
## S4 method for signature 'Bertrand'
summary(object,revenue=TRUE,shares=TRUE,parameters=FALSE,digits=2,...)
```

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.
`isMax`	If TRUE, uses numerical derivatives to determine if equilibrium price vector is a local maximum. Default is FALSE.
`revenue`	If TRUE, returns revenues. If FALSE, returns quantities. Default is TRUE
`shares`	If TRUE, returns shares. If FALSE, returns levels. Default is TRUE
`parameters`	If TRUE, reports demand and cost parameters. Default is FALSE
`digits`	The number of significant digits to round printed results. Default is 2
`...`	Arguments to be passed to non-linear solver, OR for `summary` to `CV`.

Methods

calcShares `signature(object= c(Linear,AIDS,Logit,LogitNests,CES,CESNests),preMerger=TRUE, revenue=FA`
Computes equilibrium product shares assuming that firms are playing a Nash-Bertrand pricing game. 'revenue' takes on a value of TRUE or FALSE, where TRUE calculates revenue shares, while FALSE calculates quantity shares.

calcQuantities `signature(object=c(Linear,LogLin,LogitCap),preMerger=TRUE)`
Computes equilibrium product quantities assuming that firms are playing a Nash-Bertrand pricing game.

calcPrices `signature(object=c(Linear,LogLin,AIDS,Logit,LogitNests,LogitCap,CES,CESNests),preMerger=TR`
Computes equilibrium product price levels assuming that firms are playing a Nash-Bertrand pricing game. '...' may be used to feed additional options to the optimizer responsible for computing equilibrium prices. Typically, `nleqslv` is used, but see the appropriate document for further details.

calcPriceDelta `signature(object=Bertrand)` Computes equilibrium price changes due to a merger assuming that firms are playing a Nash-Bertrand pricing game. This is a wrapper method for computing the difference between pre- and post-merger equilbrium prices

calcPriceDelta `signature(object=AIDS,isMax=FALSE,...)` Computes equilibrium price changes due to a merger assuming that firms are playing a Nash-Bertrand pricing game. This method calls a non-linear equations solver to find a sequence of price changes that satisfy the Bertrand FOCs.

calcMargins `signature(object=c(Bertrand,LogitCap),preMerger=TRUE)` Computes equilibrium product margins assuming that firms are playing a Nash-Bertrand pricing game. For "LogitCap", assumes firms are playing a Nash-Bertrand pricing game with capacity constraints.

calcMC `signature(object=Bertrand,preMerger=TRUE)` 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.

calcSlopes `signature(object=c(Linear,LogLin,AIDS,PCAIDSNests,Logit,LogitNests,LogitCap,CES,CESNests)`
Computes demand parameters assuming that firms are playing a Nash-Bertrand pricing game.

getNestsParms `signature(object=PCAIDSNests)`
: Returns a vector a matrix of calibrated nesting parameters.

hhi `signature(object=Bertrand,preMerger=TRUE,revenue=FALSE)`
: Computes the Herfindahl-Hirschman Index (HHI) using simulated market shares and either pre- or post-merger ownership information.

ownerToMatrix `signature(object=Antitrust,preMerger=TRUE)`
: converts a length-k ownership vector into a k x k ownership matrix where element i,j equals 1 if products i and j are commonly owned, and 0 otherwise.

ownerToVec `signature(object=Antitrust,preMerger=TRUE)`
: converts a k x k ownership matrix into a length-k ownership vector

show `signature(object)` Displays the percentage change in prices due to the merger.

summary `signature(object,revenue=TRUE,shares=TRUE,parameters=FALSE,digits=2,...)`
: Summarizes the effect of the merger, including price and revenue changes. Setting 'revenue' equal to FALSE reports quantities rather than revenues. Setting 'shares' to FALSE reports quantities rather than than shares (when possible). Setting 'parameters' equal to TRUE reports all demand parameters. 'digits' controls the number of significant digits reported in output. '...' allows other arguments to be passed to a `CV` method.

upp `signature(object)` Calculate the Upwards Pricing Pressure (upp) index.

`PCAIDS-class` *Class "PCAIDS"*

Description

The "PCAIDS" class contains all the information needed to calibrate a PCAIDS demand system and perform a merger analysis under the assumption that firms are playing a differentiated Bertrand products pricing game.

Objects from the Class

Objects can be created by using the constructor `pcaids`.

Slots

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

`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.

Extends

Class `AIDS`, directly. Class `Linear`, by class "AIDS", distance 2. Class `Bertrand`, by class "Linear", distance 3.

Methods

`calcSlopes signature(object)` Uncover nested CES demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("PCAIDS")
```

PCAIDSNests-class *Class "PCAIDSNests"*

Description

The "PCAIDSNests" class contains all the information needed to calibrate a nested PCAIDS demand system and perform a merger analysis under the assumption that firms are playing a differentiated Bertrand products pricing game.

Objects from the Class

Objects can be created by using the constructor `pcaids.nests`.

Slots

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

`nests`: A length k vector identifying which nest a product belongs to.

`nestsParms`: A length k vector containing nesting parameters.

Extends

Class `PCAIDS`, directly. Class `AIDS`, by class "PCAIDS", distance 2. Class `Linear`, by class "AIDS", distance 3. Class `Bertrand`, by class "Linear", distance 4.

Methods

`calcSlopes signature(object)` Uncover nested CES demand parameters. Assumes that firms are currently at equilibrium in a differentiated product Bertrand Nash pricing game.

`getNestsParms signature(object)` Returns a matrix containing the calibrated nesting parameters.

Author(s)

Charles Taragin <charles.taragin@usdoj.gov>

Examples

```
showClass("PCAIDSNests")
```

---

sim — *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 a differentiated products Bertrand pricing game.

Usage

```
sim(prices, demand=c("Linear","AIDS","LogLin","Logit","CES","LogitNests","CESNests","LogitCap"),
                      demand.param,
                      ownerPre,ownerPost,nests, capacities,
                      mcDelta=rep(0,length(prices)),
                      priceStart=prices,
                      labels=paste("Prod",1:length(prices),sep=""),...
             )
```

Arguments

	Let k denote the number of products produced by all firms.
`prices`	A length k vector of product prices.
`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.

`priceStart` A length k vector of starting values used to solve for equilibrium price. Default is the 'prices' vector.

`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.parm' 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.parm' 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.parm' 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.parm' 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 <`charles.taragin@usdoj.gov`>

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

sim.logit <- sim(price,demand="Logit",demand.parm,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
```

Index