Package ‘trade’

June 29, 2020

Type Package

Title Tools for Trade Practitioners

Version 0.5.5

Author Charles Taragin

Maintainer Charles Taragin <ctaragin@ftc.gov>

Depends antitrust (>= 0.99.11)

Imports methods, stats

Suggests bookdown,knitr

VignetteBuilder knitr

Description A collection of tools for trade practitioners, including the ability to calibrate different consumer demand systems and simulate the effects of tariffs and quotas under different competitive regimes. These tools are derived from Anderson et al. (2001) <doi:10.1016/S0047-2727(00)00085-2> and Froeb et al. (2003) <doi:10.1016/S0304-4076(02)00166-5>.

License Unlimited

Encoding UTF-8

LazyLoad yes

RoxygenNote 7.1.0

Collate 'QuotaClasses.R' 'TariffClasses.R' 'TariffCournot-methods.R' 'TariffMonComRUM-methods.R' 'summary-methods.R' 'ps-methods.R' 'bertrand_quota.R' 'bertrand_tariff.R' 'cournot_tariff.R' 'initialize-methods.R' 'monopolistic_competition_tariff.R' 'trade-deprecated.R' 'trade_shiny.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2020-06-29 21:50:02 UTC

R topics documented:

`bertrand_quota` *quota Simulation With A Bertrand Pricing Game*

Description

Simulate the effect of quotas when firms play a Bertrand pricing game and consumer demand is either Logit, CES, or AIDS

Usage

```
bertrand_quota(
  demand = c("logit"),
  prices,
  quantities,
  margins,
  owner,
  mktElast = NA_real_,
  diversions,
  quotaPre = rep(Inf, length(quantities)),
  quotaPost,
  priceOutside = ifelse(demand == "logit", 0, 1),
  priceStart,
  isMax = FALSE,
  parmStart,
  control.slopes,
  control.equ,
  labels = paste("Prod", 1:length(quantities), sep = ""),
  ...
)
```

Arguments

`demand`	A character vector indicating which demand system to use. Currently allows logit (default), ces, or aids.
`prices`	A length k vector product prices. Default is missing, in which case demand intercepts are not calibrated.

`quantities`	A length k vector of product quantities.
`margins`	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
`owner`	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.
`mktElast`	A negative number equal to the industry pre-merger price elasticity. Default is 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 revenue share is assumed.
`quotaPre`	A vector of length k where each element equals the **current** quota (expressed as a proportion of pre-merger quantities) imposed on each product. Default is Inf, which assumes no quota.
`quotaPost`	A vector of length k where each element equals the **new** quota (expressed as a proportion of pre-merger quantities) imposed on each product. Default is Inf, which assumes no quota.
`priceOutside`	price of the outside good. Equals 0 for logit and 1 for ces. Not used for aids.
`priceStart`	For aids, 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. For ces and logit, the default is prices.
`isMax`	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
`parmStart`	`aids` only. A vector of length 2 who elements equal to an initial guess for "known" element of the diagonal of the demand matrix and the market elasticity.
`control.slopes`	A list of `optim` control parameters passed to the calibration routine optimizer (typically the `calcSlopes` method).
`control.equ`	A list of `BBsolve` control parameters passed to the non-linear equation solver (typically the `calcPrices` method).
`labels`	A k-length vector of labels.
`...`	Additional options to feed to the `BBsolve` optimizer used to solve for equilibrium prices.

Details

Let k denote the number of products produced by all firms. Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, `bertrand_quota` is able to recover the slopes and intercepts of the Logit, demand system. These parameters are then used to simulate the price effects of a quota under the assumption that the firms are playing a simultaneous price setting game.

Value

`bertrand_quota` returns an instance of class `QuotaLogit`.

References

Simon P. Anderson, Andre de Palma, Brent Kreider, Tax incidence in differentiated product oligopoly, Journal of Public Economics, Volume 81, Issue 2, 2001, Pages 173-192.

Examples

```
## Calibration and simulation results from a 80% quota on non-US beers "OTHER-LITE"
## and "OTHER-REG"
## Source: Epstein/Rubenfeld 2004, pg 80

prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
owner <-c("BUD","OLD STYLE","MILLER","MILLER","OTHER-LITE","OTHER-REG")
price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)
quantities   <- c(.066,.172,.253,.187,.099,.223)*100
margins <- c(.3830,.5515,.5421,.5557,.4453,.3769)
quota <- c(Inf,Inf,Inf,Inf,.8,.8)

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


result.logit <- bertrand_quota(demand = "logit",prices=price,quantities=quantities,
                       margins = margins,owner=owner, quotaPost = quota, labels=prodNames)

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

---

bertrand_tariff *Tariff Simulation With A Bertrand Pricing Game*

Description

Simulate the effect of tariffs when firms play a Bertrand pricing game and consumer demand is either Logit, CES, or AIDS

Usage

```
bertrand_tariff(
  demand = c("logit", "ces", "aids"),
  prices,
  quantities,
  margins,
  owner,
  mktElast = NA_real_,
  diversions,
```

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

Arguments

`demand`	A character vector indicating which demand system to use. Currently allows logit (default), ces, or aids.
`prices`	A length k vector product prices. Default is missing, in which case demand intercepts are not calibrated.
`quantities`	A length k vector of product quantities.
`margins`	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
`owner`	EITHER a vector of length k whose values indicate which firm produced a product before the tariff OR a k x k matrix of pre-merger ownership shares.
`mktElast`	A negative number equal to the industry pre-merger price elasticity. Default is 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 revenue share is assumed.
`tariffPre`	A vector of length k where each element equals the **current** *ad valorem* tariff (expressed as a proportion of the consumer price) imposed on each product. Default is 0, which assumes no tariff.
`tariffPost`	A vector of length k where each element equals the **new** *ad valorem* tariff (expressed as a proportion of the consumer price) imposed on each product. Default is 0, which assumes no tariff.
`priceOutside`	price of the outside good. Equals 0 for logit and 1 for ces. Not used for aids.
`priceStart`	For aids, 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. For ces and logit, the default is prices.
`isMax`	If TRUE, checks to see whether computed price equilibrium locally maximizes firm profits and returns a warning if not. Default is FALSE.
`parmStart`	`aids` only. A vector of length 2 whose elements equal to an initial guess for each "known" element of the diagonal of the demand matrix and the market elasticity.
`control.slopes`	A list of `optim` control parameters passed to the calibration routine optimizer (typically the `calcSlopes` method).

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

Details

Let k denote the number of products produced by all firms. Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, bertrand_tariff is able to recover the slopes and intercepts of either a Logit, CES, or AIDS demand system. These parameters are then used to simulate the price effects of an *ad valorem* tariff under the assumption that the firms are playing a simultaneous price setting game.

Value

bertrand_tariff returns an instance of class TariffLogit, TariffCES, or TariffAIDS, depending upon the value of the "demand" argument.

References

Simon P. Anderson, Andre de Palma, Brent Kreider, Tax incidence in differentiated product oligopoly, Journal of Public Economics, Volume 81, Issue 2, 2001, Pages 173-192.

See Also

monopolistic_competition_tariff to simulate the effects of a tariff under monopolistic competition.

Examples

```
## Calibration and simulation results from a 10% tariff on non-US beers "OTHER-LITE"
## and "OTHER-REG"
## Source: Epstein/Rubenfeld 2004, pg 80

prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
owner <-c("BUD","OLD STYLE","MILLER","MILLER","OTHER-LITE","OTHER-REG")
price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)
quantities   <- c(.066,.172,.253,.187,.099,.223)*100
margins <- c(.3830,.5515,.5421,.5557,.4453,.3769)
tariff <- c(0,0,0,0,.1,.1)

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


result.logit <- bertrand_tariff(demand = "logit",prices=price,quantities=quantities,
                                margins = margins,owner=owner,
```

```
                                  tariffPost = tariff, labels=prodNames)

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

---

cournot_tariff *Tariff Simulation With A Cournot Quantity Setting Game*

Description

Simulate the effect of tariffs when firms play a cournot quantity setting game and consumer demand is either linear or log-linear

Usage

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

Arguments

prices	A length k vector product prices.
quantities	An n x k matrix of product quantities. All quantities must either be positive, or if the product is not produced by a plant, NA

`margins`	An n x k matrix of product margins. All margins must be either be between 0 and 1, or NA.
`demand`	A length k character vector equal to "linear" if a product's demand curve is assumed to be linear or "log" if a product's demand curve is assumed to be log-linear.
`cost`	A length k character vector equal to "linear" if a plant's marginal cost curve is assumed to be linear or "constant" if a plant's marginal curve is assumed to be constant. Returns an error if a multi-plant firm with constant marginal costs does not have capacity constraints.
`tariffPre`	An n x k matrix where each element equals the **current** *ad valorem* tariff (expressed as a proportion of consumer price) imposed on each product. Default is 0, which assumes no tariff.
`tariffPost`	An n x k matrix where each element equals the **new** *ad valorem* tariff (expressed as a proportion of consumer price) imposed on each product. Default is 0, which assumes no tariff.
`mcfunPre`	a length n list of functions that calculate a plant's marginal cost under the current tariff structure. If empty (the default), assumes quadratic costs.
`mcfunPost`	a length n list of functions that calculate a plant's marginal cost under the new tariff structure. If empty (the default), assumes quadratic costs.
`vcfunPre`	a length n list of functions that calculate a plant's variable cost under the current tariff structure. If empty (the default), assumes quadratic costs.
`vcfunPost`	a length n list of functions that calculate a plant's variable cost under the new tariff structure. If empty (the default), assumes quadratic costs.
`capacitiesPre`	A length n numeric vector of plant capacities under the current tariff regime. Default is Inf.
`capacitiesPost`	A length n numeric vector of plant capacities under the new tariff regime. Default is Inf.
`productsPre`	An n x k matrix that equals TRUE if under the current tariff regime, a plant produces a product. Default is TRUE if 'quantities' is not NA.
`productsPost`	An n x k matrix that equals TRUE if under the new tariff regime, a plant produces a product. Default equals 'productsPre'.
`owner`	EITHER a vector of length n whose values indicate which plants are commonly owned OR an n x n matrix of ownership shares.
`mktElast`	A length k vector of product elasticities. Default is a length k vector of NAs
`quantityStart`	A length k vector of quantities used as the initial guess in the nonlinear equation solver. Default is 'quantities'.
`control.slopes`	A list of `optim` control parameters passed to the calibration routine optimizer (typically the `calcSlopes` method).
`control.equ`	A list of `BBsolve` control parameters passed to the non-linear equation solver (typically the `calcPrices` method).
`labels`	A k-length vector of labels.
`...`	Additional options to feed to the `BBsolve` optimizer used to solve for equilibrium quantities.

Details

Let k denote the number of products and n denote the number of plants. Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, `cournot_tariff` is able to recover the slopes and intercepts of either a Linear or Log-linear demand system. These parameters are then used to simulate the price effects of a tariff under the assumption that the firms are playing a homogeneous products simultaneous quantity setting game.

Value

`cournot_tariff` returns an instance of class `Cournot` from package **antitrust**, depending upon the value of the “demand” argument.

References

Simon P. Anderson, Andre de Palma, Brent Kreider, The efficiency of indirect taxes under imperfect competition, Journal of Public Economics, Volume 81, Issue 2, 2001,Pages 231-251.

Examples

```
## Simulate the effect of a 75% ad valorem tariff in a
## 5-firm, single-product market with linear demand and quadratic costs
## Firm 1 is assumed to be foreign, and so subject to a tariff


n <- 5 #number of firms in market
cap <- rnorm(n,mean = .5, sd = .1)
int <- 10
slope <- -.25
tariffPre <- tariffPost <- rep(0, n)
tariffPost[1] <- .75

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

#prep inputs for Cournot
owner.pre <- diag(n)



result.c <- cournot_tariff(prices = price.pre.c,quantities = as.matrix(quantity.pre.c),
                    margins=as.matrix(margin.pre.c),
                    owner=owner.pre,
                    tariffPre =  as.matrix(tariffPre),
                    tariffPost = as.matrix(tariffPost))
```

```
summary(result.c, market = TRUE)         # summarize tariff (high-level)
summary(result.c, market = FALSE)         # summarize tariff (detailed)
```

`initialize-methods` *Initialize Methods*

Description

Initialize methods for the `TariffBertrand` and `TariffCournot` classes

Usage

```
## S4 method for signature 'TariffBertrand'
initialize(.Object, ...)

## S4 method for signature 'QuotaBertrand'
initialize(.Object, ...)

## S4 method for signature 'TariffCournot'
initialize(.Object, ...)
```

Arguments

`.Object`	an instance of class `TariffBertrand` or `TariffCournot`
`...`	arguments to pass to initialize

`monopolistic_competition_tariff` *Tariff Simulation With A Monopolistic Competition Pricing Game*

Description

Simulate the effect of tariffs when firms play a Monopolistic Competition game and consumer demand is either Logit or CES

Usage

```
monopolistic_competition_tariff(
  demand = c("logit", "ces"),
  prices,
  quantities,
  margins,
  mktElast = NA_real_,
  mktSize,
```

```
  tariffPre = rep(0, length(quantities)),
  tariffPost = rep(0, length(quantities)),
  priceOutside = ifelse(demand == "logit", 0, 1),
  labels = paste("Prod", 1:length(quantities), sep = "")
)
```

Arguments

`demand`	A character vector indicating which demand system to use. Currently allows “logit" or “ces" .
`prices`	A length k vector product prices. Default is missing, in which case demand intercepts are not calibrated.
`quantities`	A length k vector of product quantities.
`margins`	A length k vector of product margins. All margins must be either be between 0 and 1, or NA.
`mktElast`	A negative number equal to the industry pre-tariff price elasticity. Default is NA .
`mktSize`	A positive number equal to the industry pre-tariff market size. Market size equals total quantity sold,*including sales to the outside good.*
`tariffPre`	A vector of length k where each element equals the **current** *ad valorem* tariff (expressed as a proportion of the consumer price) imposed on each product. Default is 0, which assumes no tariff.
`tariffPost`	A vector of length k where each element equals the **new** *ad valorem* tariff (expressed as a proportion of the consumer price) imposed on each product. Default is 0, which assumes no tariff.
`priceOutside`	price of the outside good. Default 0 for logit and 1 for ces. Not used for aids.
`labels`	A k-length vector of labels.

Details

Let k denote the number of products produced by all firms. Using price, and quantity, information for all products in each market, as well as margin information for at least one products in each market, `monopolistic_competition_tariff` is able to recover the slopes and intercepts of a Logit demand system. These parameters are then used to simulate the price effects of an *ad valorem* tariff under the assumption that the firms are playing a monopolisitcally competitive pricing game

Value

`monopolistic_competition_tariff` returns an instance of class `TariffMonComLogit` , depending upon the value of the “demand” argument.

References

Simon P. Anderson, Andre de Palma, Brent Kreider, Tax incidence in differentiated product oligopoly, Journal of Public Economics, Volume 81, Issue 2, 2001, Pages 173-192. Anderson, Simon P., and André De Palma. Economic distributions and primitive distributions in monopolistic competition. Centre for Economic Policy Research, 2015.

See Also

bertrand_tariff to simulate the effects of a tariff under a Bertrand pricing game.

Examples

```
## Calibration and simulation results from a 10% tariff on non-US beers "OTHER-LITE"
## and "OTHER-REG"
## Source: Epstein/Rubenfeld 2004, pg 80

prodNames <- c("BUD","OLD STYLE","MILLER","MILLER-LITE","OTHER-LITE","OTHER-REG")
price    <- c(.0441,.0328,.0409,.0396,.0387,.0497)
quantities   <- c(.066,.172,.253,.187,.099,.223)*100
margins <- c(.3830,.5515,.5421,.5557,.4453,.3769)
tariff <- c(0,0,0,0,.1,.1)

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


result.logit <- monopolistic_competition_tariff(demand = "logit",prices=price,quantities=quantities,
                                margins = margins,
                                 tariffPost = tariff, labels=prodNames)

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

result.ces <- monopolistic_competition_tariff(demand = "ces",prices=price,quantities=quantities,
                                margins = margins,
                                 tariffPost = tariff, labels=prodNames)

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

---

ps-methods *Methods To Calculate Producer Surplus*

Description

Producer Surplus methods for the TariffBertrand and TariffCournot classes

Usage

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

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

Arguments

`object` an instance of class `TariffBertrand` or `TariffCournot`

`preMerger` when TRUE, calculates producer surplus under the existing tariff regime. When FALSE, calculates tariffs under the new tariff regime. Default is TRUE.

Value

product-level (or in the case of Cournot, plant-level) producer surplus

---

`Quota-classes` *S4 classes to model quotas*

Description

Extend classes from the **antitrust** package to accomodate quotas.

Slots

`quotaPre` For QuotaCournot, a matrix containing **current** plant-level (rows) AND product-level (columns) quotas. Default is a matrix of 0s. For all other classes, a vector containing **current** product-level quotas. Quotas are expressed as a proportion of pre-merger output. Default is a vector of Infs.

`quotaPost` a For QuotaCournot, a matrix containing **new** plant-level (rows) AND product-level (columns) quotas. Default is a matrix of Infs. For all other classes, a vector containing **new** product-level quotas. quotas are expressed as a proportion of pre-merger output. Default is a vector of Infss.

`summary-methods` *Summary Methods*

Description

Summary methods for the `TariffBertrand`, `QuotaBertrand`, and `TariffCournot` classes

Usage

```
## S4 method for signature 'TariffBertrand'
summary(
  object,
  revenue = FALSE,
  levels = FALSE,
  parameters = FALSE,
  market = FALSE,
  insideOnly = TRUE,
```

```
  digits = 2
)

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

## S4 method for signature 'TariffCournot'
summary(
  object,
  market = FALSE,
  revenue = FALSE,
  levels = FALSE,
  parameters = FALSE,
  digits = 2
)
```

Arguments

`object`	an instance of class `TariffBertrand`, `QuotaBertrand`, or `TariffCournot`
`revenue`	When TRUE, returns revenues, when FALSE returns quantitities. Default is FALSE.
`levels`	When TRUE returns changes in levels rather than percents and quantities rather than shares, when FALSE, returns changes as a parcent and shares rather than quantities. Default is FALSE.
`parameters`	When TRUE, displays demand and cost parameters. Default is FALSE.
`market`	When TRUE, displays aggregate information about the effect of a tariff. When FALSE displays product-specific (or in the case of Cournot, plant-specific) effects. Default is FALSE
`insideOnly`	When TRUE, rescales shares on inside goods to sum to 1. Default is FALSE.
`digits`	Number of digits to report. Default is 2.

Value

Prints either market or product/plant-level summary and invisibly returns a data frame containing the same information.

`Tariff-classes` *S4 classes to model tariffs*

Description

Extend classes from the **antitrust** package to accomodate tariffs.

Slots

`tariffPre` For TariffCournot, a matrix containing **current** plant-level (rows) AND product-level (columns) tariffs. Default is a matrix of 0s. For all other classes, a vector containg **current** product-level tariffs. *ad valorem* taxes are expressed as a proportion of the consumer price. Default is a vector of 0s.

`tariffPost` a For TariffCournot, a matrix containing **new** plant-level (rows) AND product-level (columns) tariffs. Default is a matrix of 0s. For all other classes, a vector containing **new** product-level tariffs. *ad valorem* taxes are expressed as a proportion of the consumer price. Default is a vector of 0s.

`TariffCournot-methods` *Additional methods for TariffCournot Class*

Description

Producer Surplus methods for the `TariffBertrand` and `TariffCournot` classes

Usage

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

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

Arguments

`object`	an instance of class `TariffCournot`
`preMerger`	when TRUE, computes result under the existing tariff regime. When FALSE, calculates tariffs under the new tariff regime. Default is TRUE.
`market`	when TRUE, computes market-wide results. When FALSE, calculates plant-specific results.

Value

`calcSlopes` return a TariffCournot object containing estimated slopes. `CalcQuantities` returns a matrix of equilbrium quantities under either the current or new tariff.

TariffMonComRUM-methods *Additional methods for TariffMonComLogit, TariffMonComCES Classes*

Description

Producer Surplus methods for the `TariffMonComLogit` and `TariffMonComCES` classes

Usage

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

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

## S4 method for signature 'TariffMonComLogit'
calcMargins(object, preMerger = TRUE)

## S4 method for signature 'TariffMonComCES'
calcMargins(object, preMerger = TRUE)

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

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

Arguments

`object`	an instance of class `TariffMonComLogit` or class `TariffMonComCES`
`preMerger`	when TRUE, computes result under the existing tariff regime. When FALSE, calculates tariffs under the new tariff regime. Default is TRUE.

Value

`calcSlopes` return a `TariffMonComLogit` or `TariffMonComCES` object containing estimated slopes. `CalcQuantities` returns a matrix of equilbrium quantities under either the current or new tariff.

Index