Package ‘GameTheory’

July 4, 2017

Type Package
Title Cooperative Game Theory
Version 2.7
Date 2017-07-04
Author Sebastian Cano-Berlanga
Maintainer Sebastian Cano-Berlanga <cano.berlanga@gmail.com>
Depends lpSolveAPI, combinat, gtools, ineq, kappalab
Description Implementation of a common set of punctual solutions for Cooperative Game Theory.
License GPL (>= 2)
Suggests R.rsp
VignetteBuilder R.rsp
Repository CRAN
NeedsCompilation no
Encoding latin1
Date/Publication 2017-07-04 11:35:39 UTC

R topics documented:

GameTheory-package .................................................. 2
AdjustedProportional .................................................. 4
AllRules ................................................................. 5
AlphaMin ................................................................. 6
CEA ................................................................... 7
CEL ................................................................... 8
DefineGame ............................................................ 9
LorenzRules ............................................................ 10
Nucleolus .............................................................. 10
NucleolusCapita ........................................................ 12
plot.ClaimsRules ....................................................... 13
Proportional ........................................................... 14
RandomArrival ......................................................... 14
Description

Implementation of a common set of punctual solutions for Cooperative Game Theory.

Details

Package: GameTheory
Type: Package
Version: 1.0
Date: 2015-02-04
License: GPL (>= 2)

Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References


Examples

## TRANSFERABLE UTILITY

### 3 PLAYER SHAPLEY VALUE

# Begin defining the game

```r
COALITIONS <- c(46125,17437.5,5812.5,69187.5,53812.5,30750,90000)
LEMAIRE<DefineGame(3,COALITIONS)
summary(LEMAIRE)
```

# End defining the game

```r
NAMES <- c("Investor 1","Investor 2","Investor 3")
LEMAIRESHAPLEY <- ShapleyValue(LEMAIRE,NAMES)
summary(LEMAIRESHAPLEY)
```

### 3 PLAYER NUCLEOLUS OF A GAINS GAME

```r
LEMAIRENUCLEOLUS<-Nucleolus(LEMAIRE)
summary(LEMAIRENUCLEOLUS)
```

## 4 PLAYER SHAPLEY VALUE

```r
COALITIONS <- c(26,27,55,57,81,83,82,84,110,110,110,110)
AIR<DefineGame(4,COALITIONS)
NAMES <- c("Airline 1","Airline 2","Airline 3","Airline 4")
AIRSHAPLEY<ShapleyValue(AIR,NAMES)
summary(AIRSHAPLEY)
```

### 4 PLAYER NUCLEOLUS OF A COST GAME

```r
AIRNUCLEOLUS<-Nucleolus(AIR,type="Cost")
summary(AIRNUCLEOLUS)
```

## SHAPLEY - SHUBIK POWER INDEX

### 2003 Elections

```r
SEATS<-c(46,42,23,15,9)
PARTIES<-c("CiU","PSC","ERC","PP","ICV")
E2003<-ShapleyShubik(68,SEATS,PARTIES)
summary(E2003)
```

### 2006 Elections

```r
SEATS<-c(48,37,21,14,12,3)
PARTIES<-c("CiU","PSC","ERC","PP","ICV","C's")
E2006<-ShapleyShubik(68,SEATS,PARTIES)
summary(E2006)
```
# 2012 Elections
SEATS <- c(50L, 20L, 19L, 13L, 9L, 3L)
PARTIES <- c("CiU", "PSC", "ERC", "PP", "ICV", "C's", "CUP")
E2012 <- ShapleyShubik(SEATS, PARTIES)
summary(E2012)

### CONFLICTING CLAIMS PROBLEM

### replication of Gallastegui et al. (2003), Table 7.
CLAIMS <- c(158L, 299L, 927L, 2196L, 4348L, 6256L, 13952L)
COUNTRIES <- c("Germany", "Netherlands", "Belgium", "Ireland", "UK", "Spain", "France")
INARRA <- AllRules(13500L, CLAIMS, COUNTRIES)
summary(INARRA)
plot(INARRA, 5) # Display allocations for UK
LorenzRules(INARRA) # Inequality graph

---

### AdjustedProportional

**Adjusted Proportional Rule**

**Description**

This function calculates how to distribute a given endowment by the Adjusted Proportional rule.

**Usage**

`AdjustedProportional(E, C, Names = NULL)`

**Arguments**

- `E` : Endowment
- `C` : Claims of the agents
- `Names` : Labels of the agents

**Note**

In order to calculate the rule properly, input the claims of the agents in ascending order.

**Author(s)**

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

**References**

**AllRules**

*All conflicting claims rules simultaneously*

**Description**

This function runs simultaneously all conflicting claims rules available in the package. It also calculates the Gini Index to check inequality among them.

**Usage**

```r
AllRules(E, C, Names = NULL, pct = 0, r = 2)
```

**Arguments**

- `E`: Endowment
- `C`: Claims
- `Names`: Labels of the agents
- `pct`: Format of the results. If `pct=1`, the output is given in percentage
- `r`: Decimals of the table

**Note**

In order to calculate the rule properly, input the claims of the agents in ascending order.

**Author(s)**

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

**References**


**Examples**

```r
## replication of Gallastegui et al. (2003), Table 7.

CLAIMS <- c(158,299,927,2196,4348,6256,13952)
COUNTRIES <- c("Germany","Netherlands","Belgium","Ireland","UK","Spain","France")
INARRA <- AllRules(13500,CLAIMS,COUNTRIES)
summary(INARRA)

plot(INARRA,5) ## Display allocations for UK
LorenzRules(INARRA) ## Inequality graph
```
**AlphaMin**

**AlphaMin Rule**

**Description**

This function calculates how to distribute a given endowment by the Alphamin rule.

**Usage**

```
AlphaMin(E, C, Names = NULL)
```

**Arguments**

- `E`: Endowment
- `C`: Claims of the agents
- `Names`: Labels of the agents

**Details**

For each endowment and each claim, the $\alpha - \min$ rule ensures an equal division of the endowment among the claimants as far as the smallest claim is totally honoured; then, the remaining endowment is distributed proportionally among the revised claims.

**Note**

In order to calculate the rule properly, input the claims of the agents in ascending order.

**Author(s)**

Maria Jose Solis-Baltodano <mary2014sep@gmail.com>

**References**


**Examples**

```r
CLAIMS<-c(10,20,30,40)
AGENTS<-c("Paul","John","George","Ringo")
AlphaMin(67,CLAIMS,AGENTS)->ALPHA
summary(ALPHA)

# Assignment according to the Alpha-min Rule rule for an Endowment of 67

#    Claims Amin
```
Description

This function calculates how to distribute a given endowment by the CEA rule.

Usage

CEA(E, C, Names = NULL)

Arguments

E  Endowment
C  Claims of the agents
Names  Labels of the agents

Details

The constrained equal awards (CEA) rule (Maimonides, 12th century), proposes equal awards to all agents subject to no one receiving more than his claim.

Note

In order to calculate the rule properly, input the claims of the agents in ascending order.

Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References

CEL

Constrained Equal Losses Rule

Description

This function calculates how to distribute a given endowment by the CEL rule.

Usage

CEL(E, C, Names = NULL)

Arguments

E  Endowment
C  Claims of the agents
Names  Labels of the agents

Details

The constrained equal losses (CEL) rule (Maimonides, 12th century and Aumann, 1985), chooses the awards vector at which all agents incur equal losses, subject to no one receiving a negative amount.

Note

In order to calculate the rule properly, input the claims of the agents in ascending order.

Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References

DefineGame

Transferable Utility Game

Description

Definition of a Transferable-Utility Game

Usage

DefineGame(n, V)

Arguments

n  Number of agents
V  Coalition values in lexicographic order

Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

Examples

Lemaire<-DefineGame(3,c(46125,17437.5,5812.5,69187.5,53812.5,30750,90000))
summary(Lemaire)

# Characteristic form of the game
# Number of agents: 3
# Coalition Value(s)

# v(i)
# 1  46125.0
# 2  17437.5
# 3  5812.5
# 12  69187.5
# 13  53812.5
# 23  30750.0
# 123  90000.0
LorenzRules  

**Inequality plot among rules**

**Description**
Displays a graph with a Lorenz curve for each conflicting claims rule.

**Usage**
LorenzRules(x)

**Arguments**
x  
Output object from AllRules

**Examples**

```r
## replication of Gallastegui et al. (2003), Table 7.
CLAIMS <- c(158,299,927,2196,4348,6256,13952)
COUNTRIES <- c("Germany","Netherlands","Belgium","Ireland","UK","Spain","France")
INARRA <- AllRules(13500,CLAIMS,COUNTRIES)
summary(INARRA)

plot(INARRA,5)  ## Display allocations for UK
LorenzRules(INARRA)  ## Inequality graph
```

---

Nucleolus  

**Nucleolus solution**

**Description**
This function computes the nucleolus solution of a game with a maximum of 4 agents.

**Usage**
Nucleolus(x, type = "Gains")

**Arguments**
x  
Object of class Game

type  
Specify if the game refers to Gains or Cost
Details

The nucleolus looks for an individually rational distribution of the worth of the grand coalition in which the maximum dissatisfaction is minimized. The nucleolus selects the element in the core, if this is nonempty, that lexicographically minimizes the vector of non-increasing ordered excesses of coalitions. In order to compute this solution we consider a sequence of linear programs, which looks for an imputation that minimizes the maximum excess among all coalitions.

Value

The command returns a table with the following elements:

- \( v(S) \)  Individual value of player \( i \)
- \( x(S) \)  Nucleolus solution of the player \( i \)
- \( \varepsilon_i \)  Excess of the player \( i \)

Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References


Examples

```r
## EXAMPLE FROM LEMAIRE (1991)

# Begin defining the game

COALITIONS <- c(46125, 17437.5, 5812.5, 69187.5, 53812.5, 30750, 90000)
LEMAIRE<-DefineGame(3,COALITIONS)

# End defining the game

LEMAIRENUCLEOLUS<-Nucleolus(LEMAIRE)
summary(LEMAIRENUCLEOLUS) # Gains Game, the excess should be negative
```
## NucleolusCapita

### Per Capita Nucleolus

**Description**

This function computes the per capita nucleolus solution of a gains game with a maximum of 4 agents.

**Usage**

```r
NucleolusCapita(x, type = "Gains")
```

**Arguments**

- `x`: Object of class Game
- `type`: Specify if the game refers to Gains or Cost

**Details**

The per capita nucleolus represents a measure of dissatisfaction per capita of such a coalition. It is also an individually rational distribution of the worth of the grand coalition in which the maximum per capita dissatisfaction is minimized. Formally, is defined like the nucleolus but taking into the account the per capita excess.

**Value**

The command returns a table with the following elements:

- `v(S)`: Individual value of player `i`
- `x(S)`: Nucleolus solution of the player `i`
- `Ei`: Excess of the player `i`

**Author(s)**

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

**References**


Examples

```r
## DATA FROM LEMAIRE (1991)

# Begin defining the game

COALITIONS <- c(46125, 17437.5, 5812.5, 69187.5, 53812.5, 30750, 90000)
LEMAIRE <- DefineGame(3, COALITIONS)

# End defining the game

LEMAIRENUCLEOLUS <- NucleolusCapita(LEMAIRE)
summary(LEMAIRENUCLEOLUS)
```

---

**plot.Claimsrules**  
*Plot all conflicting claims rules*

**Description**

Plot results of every rule for a given player.

**Usage**

```r
## S3 method for class 'Claimsrules'
plot(x, y, ...)
```

**Arguments**

- `x` Object of class Claimsrules
- `y` Agent
- `...` Other graphical parameters

**Author(s)**

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

**Examples**

```r
## replication of Gallastegui et al. (2003), Table 7.

CLAIMS <- c(158, 299, 927, 2196, 4348, 6256, 13952)
COUNTRIES <- c("Germany", "Netherlands", "Belgium", "Ireland", "UK", "Spain", "France")
INARRA <- AllRules(13500, CLAIMS, COUNTRIES)
summary(INARRA)

plot(INARRA, 5) ## Display allocations for UK
LorenzRules(INARRA) ## Inequality graph
```
Proportional Rule

Description
This function calculates how to distribute a given endowment by the Proportional rule.

Usage
Proportional(E, C, Names = NULL)

Arguments
- E: Endowment
- C: Claims of the agents
- Names: Labels of the agents

Note
In order to calculate the rule properly, input the claims of the agents in ascending order.

Author(s)
Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

Random Arrival Rule

Description
This function calculates how to distribute a given endowment by the Random Arrival rule.

Usage
RandomArrival(E, C, Names = NULL)

Arguments
- E: Endowment
- C: Claims of the agents
- Names: Labels of the agents
Details
The random arrival rule (O’Neill, 1982) works in the following fashion: suppose that each claim is fully honored until the endowment runs out following the order of the claimants arrival. In order to remove the unfairness of the first-come first-serve scheme associated with any particular order of arrival, the rule proposes to take the average of the awards vectors calculated in this way when all orders are equally probable.

Note
In order to calculate the rule properly, input the claims of the agents in ascending order.

Author(s)
Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References

Description
This function computes Shapley - Shubik Power Index of a coalition.

Usage
ShapleyShubik(quota, y, Names = NULL)

Arguments
quota Minimum amount of votes to pass a vote
y Seats of every party
Names Labels of the parties

Details
The Shapley and Shubik index works as follows. There is a group of individuals all willing to vote on a proposal. They vote in order and as soon as a majority has voted for the proposal, it is declared passed and the member who voted last is given credit for having passed it. Let us consider that the members are voting randomly. Then we compute the frequency with which an individual is the one that gets the credit for passing the proposal. That measures the number of times that the action of that individual joining the coalition of their predecessors makes it a winning coalition. Note that if this index reaches the value of 0, then it means that this player is a dummy. When the index reaches the value of 1, the player is a dictator.
Author(s)
Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References

Examples

```r
## SHAPLEY - SHUBIK POWER INDEX APPLIED TO THE CATALAN PARLIAMENT

# 2012 Elections
SEATS<-c(50,20,21,19,13,9,3)
PARTIES<-c("CiU", "PSC", "ERC", "PP", "ICV", "C's", "CUP")
E2012<-ShapleyShubik(68, SEATS, PARTIES)
summary(E2012)

# Results for 2012 elections

# | CiU | PSC | ERC | PP | ICV | C's | CUP |
# ---|-----|-----|-----|----|-----|-----|-----|
# Votes | 50.000 | 20.000 | 21.000 | 19.000 | 13.0000 | 9.0000 | 3.0000 |
# Shares (R) | 0.370 | 0.148 | 0.156 | 0.141 | 0.0963 | 0.0667 | 0.0222 |
# Shapley-Shubik | 0.533 | 0.133 | 0.133 | 0.133 | 0.0333 | 0.0333 | 0.0000 |
```

---

**ShapleyValue**  
**Shapley Value Solution**

**Description**
Calculates the Shapley value for a N-agent cooperative game.

**Usage**

```r
ShapleyValue(x, Names = NULL)
```

**Arguments**

- **x**: object of class Game
- **Names**: Labels of the agents

**Details**
Please check ShapleyShubik for an extension to voting power index.
Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References


Examples

# Begin defining the game

COALITIONS <- c(46125,17437.5,5812.5,69187.5,53812.5,30750,90000)
LEMAIRE <- DefineGame(3, COALITIONS)

# End defining the game

NAMES <- c("Investor 1","Investor 2","Investor 3")
LEMAIRESHAPLEY <- ShapleyValue(LEMAIRE, NAMES)
summary(LEMAIRESHAPLEY)

---

**summary.ClaimsRule**  
*Summary Method for ClaimsRule Objects*

Description

*summary* method for class "ClaimsRule".

Usage

```r
## S3 method for class 'ClaimsRule'
summary(object, ...)
```

Arguments

- `object` an object of class "ClaimsRule"
- `...` Other parameters passed down to `print()` and `summary()`
**summary.ClaimsRules**  
*Summary methods for a ClaimsRules Object*

**Description**

Summary methods for a ClaimsRules Object

**Usage**

```r
## S3 method for class 'ClaimsRules'
summary(object, ...)
```

**Arguments**

- `object`: A ClaimsRules object
- `...`: Other parameters passed down to `print()` and `summary()`

---

**summary.Game**  
*Summary methods for a Game Object*

**Description**

Summary methods for a Game Object

**Usage**

```r
## S3 method for class 'Game'
summary(object, ...)
```

**Arguments**

- `object`: A Game object
- `...`: Other parameters passed down to `print()` and `summary()`
Summary methods for a Nucleolus Object

Description

Summary methods for a Nucleolus Object

Usage

```r
## S3 method for class 'Nucleolus'
summary(object, ...)
```

Arguments

- `object` A Nucleolus object
- `...` Other parameters passed down to `print()` and `summary()`

Summary methods for a ShapleyShubik Object

Description

Summary methods for a ShapleyShubik Object

Usage

```r
## S3 method for class 'ShapleyShubik'
summary(object, ...)
```

Arguments

- `object` A ShapleyShubik object
- `...` Other parameters passed down to `print()` and `summary()`
#### summary.ShapleyValue  
*Summary methods for a ShapleyValue Object*

**Description**

Prints the summary of the Shapley values solution for a given game.

**Usage**

```r
## S3 method for class 'ShapleyValue'
summary(object, ...)
```

**Arguments**

- `object` A ShapleyValue object
- `...` Other parameters passed down to `print()` and `summary()`

---

#### Talmud  
*Talmud Rule*

**Description**

This function calculates how to distribute a given endowment by the Talmud rule.

**Usage**

```r
talmud(E, C, Names = NULL)
```

**Arguments**

- `E` Endowment
- `C` Claims of the agents
- `Names` Labels of the agents

**Details**

The Talmud rule (Aumann 1985) proposes to apply the constrained equal awards rule, if the endowment is not enough to satisfy the half-sum of the claims. Otherwise, each agent receives the half of her claim and the constrained equal losses rule is applied to distribute the remaining endowment.

**Note**

In order to calculate the rule properly, input the claims of the agents in ascending order.
Author(s)

Sebastian Cano-Berlanga <cano.berlanga@gmail.com>

References

Aumann, R.J. and Maschler, M., (1985) Game Theoretic Analysis of a bankruptcy from the Talmud. 
Index

AdjustedProportional, 4
AllRules, 5
AlphaMin, 6

CEA, 7
CEL, 8

DefineGame, 9

GameTheory (GameTheory-package), 2
GameTheory-package, 2

LorenzRules, 10

Nucleolus, 10
NucleolusCapita, 12

plotClaimsRules, 13
Proportional, 14

RandomArrival, 14

ShapleyShubik, 15
ShapleyValue, 16
summary, 17
summary.ClaimsRule, 17
summary.ClaimsRules, 18
summary.Game, 18
summary.Nucleolus, 19
summary.ShapleyShubik, 19
summary.ShapleyValue, 20

Talmud, 20