Package ‘TUGLab’

July 21, 2025

Type Package

Title A Laboratory for TU Games

Version 0.0.1

Date 2025-06-05

Maintainer Álvaro de Prado Saborido <alvarodepradosaborido@gmail.com>

Description Cooperative game theory models decision-making situations in which a group of agents, called players, may achieve certain benefits by cooperating to reach an optimal outcome. It has great potential in different fields, since it offers a scenario to analyze and solve problems in which cooperation is essential to achieve a common goal. The 'TUGLab' (Transferable Utility Games Laboratory) R package contains a set of scripts that could serve as a helpful complement to the books and other materials used in courses on cooperative game theory, and also as a practical tool for researchers working in this field. The 'TUGLab' project was born in 2006 trying to highlight the geometrical aspects of the theory of cooperative games for 3 and 4 players. 'TUGlabWeb' is an online platform on which the basic functions of 'TUGLab' are implemented, and it is being used all over the world as a resource in degree, master's and doctoral programs. This package is an extension of the first versions and enables users to work with games in general (computational restrictions aside). The user can check properties of games, compute well-known games and calculate several set-valued and single-valued solutions such as the core, the Shapley value, the nucleolus or the core-center. The package also illustrates how the Shapley value flexibly adapts to various cooperative game settings, including weighted players and coalitions, a priori unions, and restricted communication structures. In keeping with the original philosophy of the first versions, special emphasis is placed on the graphical representation of the solution concepts for 3 and 4 players.

URL http://tuglabweb.uvigo.es/TUGlabWEB2/index.php,
https://mmiras.webs.uvigo.es/TUGlab/

BugReports https://github.com/esanchez-coder/TUGLab/issues

Depends R (>= 3.5)

Imports geometry, plotly, rcdd, stringr, volesti

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.3.2

Author Álvaro de Prado Saborido [aut, cre] (Departamento de Estatística e Investigación Operativa. Universidade de Vigo. Spain),
Alejandro Bernárdez Ferradás [ctb] (ORCID: <https://orcid.org/0009-0006-0960-3555>, SiDOR. Departamento de Estatística e Investigación Operativa. Universidade de Vigo. CITMAga. Spain),
Miguel Ángel Mirás Calvo [aut] (ORCID: <https://orcid.org/0000-0001-7247-1926>, RGEAF. Departamento de Matemáticas. Universidade de Vigo. Spain),
Iago Núñez Lugilde [aut] (ORCID: <https://orcid.org/0000-0003-3382-0737>, Departamento de Matemáticas. MODES. Universidade da Coruña. Spain),
Carmen Quinteiro Sandomingo [aut] (ORCID: <https://orcid.org/0000-0002-2711-1945>, Departamento de Matemáticas. Universidade de Vigo. Spain),
Estela Sánchez Rodríguez [aut] (ORCID: <https://orcid.org/0000-0002-0933-6411>, SiDOR. Departamento de Estatística e Investigación Operativa. Universidade de Vigo. CITMAga. Spain),
MCIN/AEI/10.13039/501100011033 [fnd] (Project PID2021-124030NB-C33. ERDF A way of making Europe/EU)

Repository CRAN

Date/Publication 2025-06-10 09:30:05 UTC

Contents

`additivecheck` *Additive check*

Description

This function checks if the given game is additive.

Usage

```
additivecheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is additive if $v(S) = \sum_{i \in S} v(i)$ for all $S \in 2^N$.

Value

`TRUE` if the game defined by `v` is additive, `FALSE` otherwise. If `instance=TRUE` and the game is not additive, the position (binary order position if `binary=TRUE`; lexicographic order position otherwise) of a coalition for which additivity is violated is also returned.

See Also

additivegame, superadditivecheck

Examples

```
v <- c(1, 5, 40, 100, 6, 41, 101, 45, 105, 140, 46, 106, 141, 145, 146)
additivecheck(v)
additivecheck(v, binary = TRUE, instance = TRUE)
```

`additivegame` *Additive game*

Description

Given the value of each player, this function returns the characteristic function of the associated additive game.

Usage

```
additivegame(a, binary = FALSE)
```

Arguments

`a`	A vector containing the player values.
`binary`	A logical value. By default, `binary=FALSE`.

Details

The characteristic function of the additive game given by $a \in \mathbb{R}^n$ is defined for each $S \in 2^N$ by $v(S) = \sum_{i \in S} a_i$.

Value

The characteristic function of the associated additive game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

See Also

additivecheck, strategicallyequivalentcheck, superadditivecheck

Examples

```
a <- c(1,5,10,13,58)
additivegame(a, binary = FALSE)
```

airfieldgame *Airfield game*

Description

Given an airfield problem, this function returns the associated airfield game.

Usage

```
airfieldgame(c, binary = FALSE)
```

Arguments

`c`	A vector of costs defining the airfield problem.
`binary`	A logical value. By default, `binary=FALSE`.

Details

Let $N = \{1, \ldots, n\}$ denote a set of agents, and let $c \in \mathbb{R}_+^N$ be a cost vector. Each c_i represents the cost of the service required by agent i. Segmental costs are defined as the difference between a given cost and the first immediately lower cost: $c_i - c_{i-1}$ for $i \in N \backslash \{1\}$.

Each $c \in \mathbb{R}_+^N$ defines an airfield problem, which is associated to an airfield game $v_a \in G^N$, is defined by

$$v_a(S) = \max\{c_j : j \in S\} \text{ for all } S \in 2^N.$$

Airfield games, as defined, are cost games, but they can also be expressed as savings games. Additional tools and methods for addressing airfield problems are available in the **AirportProblems** package *Bernárdez Ferradás et al. (2025).*

Value

The characteristic function of the airfield game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Bernárdez Ferradás, A., Sánchez Rodríguez, E., Mirás Calvo, M., & Quinteiro Sandomingo, C. (2025). *AirportProblems: Analysis of Cost Allocation for Airport Problems*. R package version 0.1.0. `https://CRAN.R-project.org/package=AirportProblems`

Littlechild, S.C., & Owen, G. (1973). A Simple Expression for the Shapely Value in a Special Case. *Management Science*, 23, 370-372.

See Also

claimsgame, savingsgame

Examples

```
c <- c(2000,3200,4100,5100)
airfieldgame(c,binary=TRUE)
```

`balancedcheck` *Balanced check*

Description

This function checks if the given game is balanced and computes its balanced cover.

Usage

```
balancedcheck(v, game = FALSE, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

`v`	A characteristic function, as a vector.
`game`	A logical value. By default, `game=FALSE`. If set to `TRUE`, the balanced cover of the game is also returned.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Let $v \in G^N$. A family F of non-empty coalitions of N is balanced if there exists a weight family $\delta^F = \{\delta^F_S\}_{S\in F}$ such that $\delta^F_S > 0$ for each $S \in F$ and $\sum_{S\in F} \delta^F_S e^S = e^N$, being e^S the characteristic vector of S, that is, the vector $(e^S_i)_{i\in N}$ in which $e^S_i = 1$ if $i \in S$ and $e^S_i = 0$ if $i \notin S$).

The game v is balanced if, for each balanced family F, it is true that

$$\sum_{S\in F} \delta^F_S v(S) \leq v(N).$$

The balanced cover of v is the game $\tilde{v}$ defined by $\tilde{v}(S) = v(S)$ for all $S \neq N$ and

$$\tilde{v}(N) = \max_{\delta\in P} \sum_{S\subset N} \delta_S v(S),$$

being P the set of the weight families associated with the balanced families of N.

A game is balanced if and only if it coincides with its balanced cover. By the Bondareva-Shapley Theorem, a game has a non-empty core if and only if it is balanced.

Value

TRUE if the game is balanced, FALSE otherwise. If game=TRUE, the balanced cover of the game is also returned.

References

Maschler, M., Solan, E., & Zamir, S. (2013). *Game Theory*. Cambridge University Press.

See Also

totallybalancedcheck

Examples

```
balancedcheck(c(12,10,20,20,50,70,70), game=TRUE)
balancedcheck(c(rep(0,4), rep(30,6), rep(0,4), 50))
v <- runif(2^3-1,0,10) # random three-player game
balancedcheck(v, game=TRUE)
balancedcheck(balancedcheck(v, game=TRUE)$game) # balanced cover is indeed balanced
balancedcheck(runif(2^(15)-1,min=10,max=20)) # random game
```

---

balancedfamilycheck   *Balanced family check*

Description

This function checks if the given family is balanced.

Usage

```
balancedfamilycheck(Fam, n = NULL, tol = 100 * .Machine$double.eps)
```

Arguments

Fam	A vector containing the binary order positions of a family of coalitions.
n	The number of players in the set of players from which Fam is taken. When not specified, n is assumed to be the the number of players present in Fam.
tol	A tolerance parameter, as a non-negative number. By default, tol=100*.Machine$double.eps.

Details

A family F of non-empty coalitions of a set of players N is balanced if there exists a weight family $\delta^F = \{\delta^F_S\}_{S\in F}$ such that $\delta^F_S > 0$ for each $S \in F$ and $\sum_{S\in F}\delta^F_S e^S = e^N$, being e^S the characteristic vector of S, that is, the vector $(e^S_i)_{i\in N}$ in which $e^S_i = 1$ if $i \in S$ and $e^S_i = 0$ if $i \notin S$).

A balanced family F is said to be minimal if there does not exist a balanced family F' such that $F' \subsetneq F$.

Value

This function returns three outputs: `check`, `minimal` and `delta`. If `Fam` is not a balanced family: `check=FALSE` and both `minimal` and `delta` are `NULL`. If `Fam` is a balanced family: `check=TRUE`, `minimal=TRUE` if `Fam` is minimal (`minimal=FALSE` otherwise), and `delta` returns an associated weight family.

References

Maschler, M., Solan, E., & Zamir, S. (2013). *Game Theory*. Cambridge University Press.

See Also

balancedcheck, kohlbergcriterion, totallybalancedcheck

Examples

```
balancedfamilycheck(c(3,6,13,8)) # balanced and minimal
balancedfamilycheck(c(3,5,9,4,8,14)) # balanced but not minimal
balancedfamilycheck(c(1,2,4,12,13)) # not balanced
```

`belong2corecheck` *Belong to core*

Description

This function checks if an allocation belongs to the core of a game.

Usage

```
belong2corecheck(v, binary = FALSE, x, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`x`	An allocation, as a vector.
`instance`	A logical value. By default, `instance=FALSE`.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

TRUE if x belongs to the core of v, FALSE otherwise. If instance=TRUE and x does not belong to the core of v, a justification is also provided: if efficiency is violated, not efficient is returned; if efficiency is not violated, the position (binary order position if binary=TRUE; lexicographic order position otherwise) of a coalition for which rationality is violated is returned.

References

Gillies, D. (1953). *Some theorems on n-person games*. PhD thesis, Princeton, University Press Princeton, New Jersey.

Examples

```
v <- c(0, 0, 0, 2, 1, 4, 6)
a <- c(3, 1, 2) # an allocation for v
b <- c(2, 2, 2) # egalitarian solution for v
belong2corecheck(v = v, binary = TRUE, x = a, instance = TRUE)
belong2corecheck(v = v, binary = FALSE, x = b, instance = TRUE)

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
cost.x <- c(1,2,2) # core allocation of cost.v
belong2corecheck(v = -cost.v, x = -cost.x)
```

---

bin2lex   *Binary order to lexicographic order*

Description

Given a characteristic function in binary order, this function returns the characteristic function in lexicographic order.

Usage

```
bin2lex(v)
```

Arguments

v	A characteristic function, as a vector in binary order.

Details

The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$. Lexicographic order arranges coalitions in ascending order according to size, and applies lexicographic order to break ties among coalitions of the same size.

Value

The characteristic function, as a vector in lexicographic order.

See Also

codebin2lex, codelex2bin, lex2bin

Examples

```
v <- seq(1:31)
bin2lex(v)
lex2bin(bin2lex(v))==v
```

`claimsgame` *Pessimistic claims game associated with a claims problem*

Description

Given a claims problem, this function returns the associated pessimistic claims game.

Usage

```
claimsgame(E, d, binary = FALSE)
```

Arguments

`E`	An endowment, as a positive number.
`d`	A vector of claims.
`binary`	A logical value. By default, `binary=FALSE`.

Details

A claims problem is a triple (N, E, d) where $E \geq 0$ is an amount to be distributed among a set N of agents and $d \in \mathbb{R}^{|N|}$ is a vector of claims satisfying $\sum_{i=1}^{n} d_i \geq E$.

Given a claims problem (N, E, d), its associated claims game, $v_{E,d} \in G^N$, is defined by

$$v_{E,d}(S) = \max\{0,\ E - \sum_{i \in N \setminus S} d_i\} \text{ for all } S \in 2^N.$$

For further analysis and computational methods related to conflicting claims problems, see the **ClaimsProblems** package *Sánchez Rodríguez et al. (2025)*.

Value

The characteristic function of the pessimistic claims game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

O'Neill, B. (1982). A problem of rights arbitration from the Talmud. *Mathematical Social Sciences*, 2, 345–371.

Sánchez Rodríguez, E., Núñez Lugilde, I., Mirás Calvo, M., & Quinteiro Sandomingo, C. (2025). *ClaimsProblems: Analysis of Conflicting Claims*. R package version 1.0.0.

https://CRAN.R-project.org/package=ClaimsProblems

See Also

airfieldgame

Examples

```
E <- 10
d <- c(2,4,7,8)
claimsgame(E,d)
```

`coalitionweightedshapleyvalue`

Coalition-weighted Shapley value

Description

Given a game and a weight family, this function returns the coalition-weighted Shapley value.

Usage

```
coalitionweightedshapleyvalue(v, delta, binary = FALSE, game = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`delta`	A weight family. It can be introduced in two different ways: as a non-negative vector whose length is the number of coalitions (thus specifying all coalition weights) or as a non-negative vector whose length is the number of players (thus specifying the weights of single-player coalitions and implying that the rest of weights are 0). In any case, if the introduced weights do not add up to 1, the weight family is computed by normalization.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` and `delta` (assuming `delta` was introduced by specifying all coalition weights; otherwise there is no difference) are introduced in binary order instead of lexicographic order.
`game`	A logical value. By default, `game=FALSE`. If set to `TRUE`, the coalition-weighted game is also returned.

Details

A weight family is a collection of $2^{|N|} - 1$ real numbers $\delta = \{\delta_T\}_{T \in 2^N \setminus \emptyset}$ such that $\delta_T \geqslant 0$ for each $T \in 2^N \setminus \emptyset$ and $\sum_{T \in 2^N \setminus \emptyset} \delta_T = 1$. For each $v \in G^N$ and each $T \in 2^N$, the T-marginal game of v, denoted $v^T \in G^N$, is defined as

$$v^T(S) = v(S \cup (N \setminus T)) - v(N \setminus T) + v(S \cap (N \setminus T)),\ S \in 2^N.$$

For each game $v \in G^N$ and each weight family δ, the δ-weighted game $v^\delta \in G^N$ is defined as

$$v^\delta = \sum_{T \in 2^N \setminus \emptyset} \delta_T v^T.$$

Given a game $v \in G^N$, its δ-weighted Shapley value, $\Phi^\delta(v)$, is the Shapley value of the δ-weighted game:

$$\Phi^\delta(v) = Sh(v^\delta).$$

Value

The coalition-weighted Shapley value, as a vector. If `game=TRUE`, the coalition-weighted game is also returned, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Sánchez Rodríguez, E., Mirás Calvo, M. A., Quinteiro Sandomingo, C., & Núñez Lugilde, I. (2024). Coalition-weighted Shapley values. *International Journal of Game Theory*, 53, 547-577.

See Also

marginalgame, shapleyvalue, weightedshapleyvalue

Examples

```
v <- c(0,0,0,0,0,0,1,0,0,1,3,4,6,8,10)
delta <- c(0.3,0.1,0,0.6,0,0,0,0,0,0,0,0,0,0,0)
coalitionweightedshapleyvalue(v, delta, binary=TRUE)

v <- c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
delta <- c(0.25,0.25,0.25,0.25)
a <- coalitionweightedshapleyvalue(v, delta, game=TRUE)
b <- coalitionweightedshapleyvalue(a$game, delta, game=TRUE)
c <- coalitionweightedshapleyvalue(b$game, delta, game=TRUE)
plotcoresets(rbind(v, a$game, b$game, c$game), imputations=FALSE)

# Games a, b and c have the same Shapley value:
all(sapply(list(a$value, b$value, c$value, shapleyvalue(v)),
           function(x) all.equal(x, a$value) == TRUE))
```

codebin2lex *Binary order position to lexicographic order position*

Description

Given the binary order position of a coalition, this function returns the corresponding lexicographic order position.

Usage

```
codebin2lex(n, Nbin)
```

Arguments

n	Number of players, as an integer.
Nbin	A binary order position, as an integer between 1 and $2^n - 1$.

Details

The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$. Lexicographic order arranges coalitions in ascending order according to size, and applies lexicographic order to break ties among coalitions of the same size.

Value

The corresponding lexicographic order position, as an integer between 1 and $2^n - 1$.

See Also

bin2lex, codelex2bin, lex2bin

Examples

```
codebin2lex(5, 4)
```

codelex2bin *Lexicographic order position to binary order position*

Description

Given the lexicographic order position of a coalition, this function returns the corresponding binary order position.

Usage

```
codelex2bin(n, Nlex)
```

Arguments

`n`	Number of players.
`Nlex`	A lexicographic order position, as an integer between 1 and $2^{\text{n}} - 1$.

Details

Lexicographic order arranges coalitions in ascending order according to size, and applies lexicographic order to break ties among coalitions of the same size. The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$.

Value

The corresponding binary order position, as an integer between 1 and $2^{\text{n}} - 1$.

See Also

bin2lex, codebin2lex, lex2bin

Examples

```
codelex2bin(5, 4)
```

`compromiseadmissiblecheck` *Compromise-admissible check*

Description

This function checks if the given game is compromise-admissible.

Usage

```
compromiseadmissiblecheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

Let $v \in G^N$. The utopia payoff of player $i \in N$ is defined as $M_i(v) = v(N) - v(N \backslash i)$. The minimal right of player $i \in N$ is defined as $m_i(v) = \max_{S:i \in S}(v(S) - \sum_{j \in S \backslash i} M_j(v))$.

The game $v \in G^N$ is said to be compromise-admissible if its core-cover is not empty, that is, if the following conditions hold:

1) $m(v) \leq M(v)$.

2) $\sum_{i \in N} m_i(v) \leq v(N) \leq \sum_{i \in N} M_i(v)$.

Value

`TRUE` if the game is compromise-admissible, `FALSE` otherwise. If `instance=TRUE` and $\{i \in N : m_i(v) > M_i(v)\} \neq \emptyset$, one of the players in that set is also returned.

Examples

```
compromiseadmissiblecheck(c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160))
compromiseadmissiblecheck(c(1,2,2), instance=TRUE)

# What if the game is a cost game?
cost.v <- c(30, 20, 50, 40, 60, 60, 75) # compromise-admissible cost game
compromiseadmissiblecheck(-c(30, 20, 50, 40, 60, 60, 75))
```

`constantsumgame` *Constant sum game*

Description

This function computes the characteristic function of the specified constant sum game.

Usage

```
constantsumgame(halfv, vN)
```

Arguments

`halfv`	The first half (according to lexicographic or binary order) of the characteristic function (excluding the grand coalition), as a vector.
`vN`	The utility of the grand coalition.

Details

A game $v \in G^N$ is a constant sum game if, for each $S \in 2^N$, $v(S) + v(N \setminus S) = v(N)$. Thus, if v is a constant sum game and F is a family of $2^{n-1} - 1$ coalitions such that $S \cup T \neq N$ for any $S, T \in F$, the full characteristic function of v is strictly determined by the utilities of the coalitions in F and the utility of the grand coalition.

Value

The characteristic function of the constant sum game. It is to be interpreted according to the order that `halfv` is introduced in.

Examples

```
constantsumgame(c(0,0,0), 1) # the dollar game
# Building a random constant sum game:
players <- sample(3:6,1) # random number of players between three and six
halfv <- runif(2^(players-1)-1, 0, 10) # random halfv
vN <- runif(1,30,50) # random vN
constantsumgame(halfv, vN)
```

---

convexcheck *Convex check*

Description

This function checks if the given game is convex.

Usage

```
convexcheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is convex if $v(S \cap T) + v(S \cup T) \geq v(S) + v(T)$ for all $S, T \in 2^N$. Zumsteg, S. (1995) shows that v is convex if $v(S \cup i \cup j) + v(S) \geq v(S \cup i) + v(S \cup j)$ for all $S \in 2^N$ and $i, j \in N \backslash S$ such that $i \neq j$.

A game $v \in G^N$ is concave if $-v$ is convex.

Value

`TRUE` if the game is convex, `FALSE` otherwise. If `instance=TRUE` and the game is not convex, the function also returns the positions (binary order positions if `binary=TRUE`; lexicographic order positions otherwise) of a pair of coalitions violating the Zumsteg convexity characterization.

References

Zumsteg, S. (1995). *Non-cooperative aspects of cooperative game theory and related computational problems*. PhD thesis, ETH Zurich.

See Also

strategicallyequivalentcheck, superadditivecheck

Examples

```
v1 <- c(5, 2, 2, 1, 8, 8, 6, 4, 3, 3, 12, 10, 10, 6, 14)
convexcheck(v1)
v2 <- c(0, 0, 0, 2, 2, 1, 3)
convexcheck(v2, binary = FALSE, instance = TRUE)

# How to check if a game is concave:
v.conc <- c(4, 3, 3, 2, 6, 6, 5, 5, 4, 4, 7, 6, 6, 6, 7) # concave game
convexcheck(-v.conc)
```

---

corecenterhitrun — *Core-center estimation by hit-and-run*

Description

Given a game with a full-dimensional core, this function computes a hit-and-run estimation of its core center.

Usage

```
corecenterhitrun(v, k = 1000, getpoints = FALSE, binary = FALSE)
```

Arguments

`v`	A characteristic function of a game with, as a vector.
`k`	The number of points to be generated by the hit-and-run method, as an integer. By default, `k=1000`.
`getpoints`	A logical value. By default, `getpoints=FALSE`. If set to`getpoints=TRUE`, the points generated by the hit-and-run method are also returned, as a matrix in which each row is a point.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$. A game is said to be balanced if its core is not empty.

The core-center of a balanced game v, $CC(v)$, is defined as the expectation of the uniform distribution over $C(v)$, and thus can be interpreted as the centroid or center of gravity of $C(v)$. Let μ be the $(n-1)$-dimensional Lebesgue measure and let $V(C) = \mu(C(v))$ be the volume (measure) of the core. If $V(C) > 0$, then, for each $i \in N$,

$$CC_i(v) = \frac{1}{V(C)} \int_{C(v)} x_i d\mu$$

.

The hit-and-run method (Smith, 1984) is a Monte Carlo algorithm that generates uniformly distributed random points within a bounded and convex region, such as a polytope or a convex body in high-dimensional space.

The hit-and-run method is based on the following process. Step 0: choose a point inside the convex body. Step 1: choose a uniformly distributed random direction from the unit sphere in the given dimension. Step 2: determine the longest line segment that, starting from the chosen point and taking the chosen direction, remains entirely within the convex body. Step 3: choose a uniformly distributed along the line segment random point. Step 4: go to Step 1.

Value

A hit-and-run estimation of the core-center, as a vector; and, if `getpoints=TRUE`, a matrix containing the points generated by the hit-and-run method.

References

Gonzalez-Díaz, J. & Sánchez-Rodríguez, E. (2007). A natural selection from the core of a TU game: the core-center. International Journal of Game Theory, 36(1), 27-46.

Espinoza-Burgos, N. H. (2020). *Comparación de métodos exactos y aproximados para calcular el core-center del juego del aeropuerto*. TFM, Máster en Técnicas Estadísticas, `http://eio.usc.es/pub/mte/descargas/ProyectosFinMaster/Proyecto_1791.pdf`.

Smith, R. L. (1984). Efficient Monte Carlo Procedures for Generating Points Uniformly Distributed Over Bounded Regions. Operations Research, 32(6), 1296-1308.

See Also

balancedcheck, corecentervalue, coredimension, corevertices, corevertices234

Examples

```
v1 <- claimsgame(E=8,d=c(3,5,6))
corecenterhitrun(v1,k=1e5)

v2 <- c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
corecenterhitrun(v2,k=1e5)

# Plotting the corecenter and its hit-and-run estimation:
plotcoreset(v2,solutions="corecenter",allocations=corecenterhitrun(v2))

# Plotting the points generated by the hit-and-run method:
hrpoints <- corecenterhitrun(v2,k=100,getpoints=TRUE)$points
plotcoreset(v2,allocations=hrpoints)
```

```
# What if the game is not full-dimensional because of a dummy player?
v3 <- c(440,0,0,0,440,440,440,15,14,7,455,454,447,60,500)
# For coredimension to detect that, tolerance has to be appropriate:
coredimension(v3,tol=100*.Machine$double.eps) # tolerance too small
coredimension(v3) # default tolerance, 1e-12, big enough

# Now how to compute the hit-and-run estimation of the core-center?
# Knowing that player 1 is a dummy and that the core-center assigns
# dummies their individual worth...
v3.without1 <- subgame(v3,S=14) # subgame without player 1
( cc.hr <- c(v3[1],corecenterhitrun(v3.without1,k=100)) )

# Plotting the points when there is a dummy player:
points.without1 <- corecenterhitrun(v3.without1,k=100,getpoints=TRUE)$points
points.with1 <- cbind(v3[1],points.without1)
plotcoreset(v3,allocations=points.with1)

# This function does not work if the core is not full-dimensional:
v4 <- c(0,0,0,0,2,5,0,5,0,0,10,2,5,5,10)
corecenterhitrun(v4,k=1e5)
```

---

corecentervalue *Core-center*

Description

Given a game, this function computes its core center.

Usage

```
corecentervalue(v, binary = FALSE, tol = 1e-12)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, binary=FALSE. Should be set to TRUE if v is introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, tol=1e-12.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$. A game is said to be balanced if its core is not empty.

The core-center of a balanced game v, $CC(v)$, is defined as the expectation of the uniform distribution over $C(v)$, and thus can be interpreted as the centroid or center of gravity of $C(v)$. Let μ be the $(n-1)$-dimensional Lebesgue measure and let $V(C) = \mu(C(v))$ be the volume (measure) of the core. If $V(C) > 0$, then, for each $i \in N$,

$$CC_i(v) = \frac{1}{V(C)} \int_{C(v)} x_i d\mu$$

.

Value

The core-center, as a vector.

References

Gonzalez-Díaz, J. & Sánchez-Rodríguez, E. (2007). A natural selection from the core of a TU game: the core-center. International Journal of Game Theory, 36(1), 27-46.

See Also

balancedcheck, corecenterhitrun, coredimension, corevertices, corevertices234

Examples

```
v1 <- claimsgame(E=8,d=c(3,5,6))
corecentervalue(v1)
plotcoreset(v1,solutions="corecenter")

v2 <- c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
corecentervalue(v2)
plotcoreset(v2,solutions="corecenter")

# What if the game is not full-dimensional because of a dummy player?
v3 <- c(440,0,0,0,440,440,440,15,14,7,455,454,447,60,500)
dummynull(v3) # player 1 is a dummy in v3, so the core is degenerate
# For coredimension to detect that, tolerance has to be appropriate:
coredimension(v=v3,tol=100*.Machine$double.eps) # tolerance too small
coredimension(v=v3) # default tolerance, 1e-12, big enough

# Now how to compute the corecenter?
# When given a degenerate game, corecentervalue computes an approximation:
( cc.approx <- corecentervalue(v=v3) ) # approximate core-center
# However, knowing that player 1 is a dummy and that the core-center assigns
# dummies their individual worth...
v3.without1 <- subgame(v=v3,S=14) # subgame without player 1
( cc.exact <- c(v3[1],corecentervalue(v3.without1)) ) # "exact" core-center

# Plotting both results:
plotcoreset(v3,allocations=rbind(cc.approx,cc.exact),projected=TRUE)
```

coredimension *Core dimension*

Description

Given a game, this function computes the dimension of its core.

Usage

```
coredimension(v, binary = FALSE, tol = 1e-12)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

The dimension of the core of `v`, as an integer.

References

Edgeworth, F. Y. (1881). *Mathematical psychics: An essay on the application of mathematics to the moral sciences*. CK Paul.

Gillies, D. (1953). *Some theorems on n-person games*. PhD thesis, Princeton, University Press Princeton, New Jersey.

See Also

balancedcheck, corevertices, corevertices234, plotcoreset, plotcoresets

Examples

```
v1 <- c(rep(0,5),rep(1,4),0,rep(1,3),2,2)
plotcoreset(v1)
coredimension(v1)

v2 <- c(rep(0,5),rep(1,4),0,rep(1,4),2)
plotcoreset(v2)
coredimension(v2)

v3 <- marginalgame(c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10),1)
plotcoreset(v3)
coredimension(v3)

v4 <- c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
plotcoreset(v4)
coredimension(v4)
```

---

corevertices — *Core vertices*

Description

Given a game, this function computes its core vertices.

Usage

```
corevertices(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

If the core of `v` is non-empty, the core vertices are returned, as a matrix in which each row is a vertex.

Note

Function `corevertices234` can also compute the core vertices of games with less than five players, but takes a different approach.

References

Edgeworth, F. Y. (1881). *Mathematical psychics: An essay on the application of mathematics to the moral sciences*. CK Paul.

Gillies, D. (1953). *Some theorems on n-person games*. PhD thesis, Princeton, University Press Princeton, New Jersey.

See Also

balancedcheck, corevertices234, plotcoreset, plotcoresets

Examples

```
v=c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
corevertices(v)

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
-corevertices(-cost.v) # core vertices of the cost game
```

---

corevertices234 *Core vertices of games with two, three or four players*

Description

Given a game with no more than four players, this function computes its core vertices.

Usage

```
corevertices234(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S)\ \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

If the core of v is non-empty, the core vertices are returned, as a matrix in which each row is a vertex.

Note

Function `corevertices` can also compute the core vertices of games with less than five players, but takes a different approach.

See Also

balancedcheck, corevertices, plotcoreset,

Examples

```
# 2 players:
corevertices234(c(-58,4,13))

# 3 players:
corevertices234(c(1,5,10,6,11,15,16)) # additive game

# 4 players:
corevertices234(c(0,0,0,0,4,3,5,2,4,5,10,19,20,30,100)) # convex game
corevertices234(c(0,0,0,0,1,2,1,1,1,1,4,3,2,1,7)) # not convex game

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
-corevertices234(-cost.v) # core vertices of the cost game
```

`degeneratecheck` *Degenerate check*

Description

This function checks if the given game is degenerate.

Usage

```
degeneratecheck(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

A game $v \in G^N$ is degenerate if $v(N) = \sum_{i \in N} v(i)$.

Value

TRUE if the game is degenerate, FALSE otherwise.

See Also

essentialcheck

Examples

```
v <- c(1, 5, 10, 0, 0, 0, 16)
degeneratecheck(v)
w <- c(1, 5, 10, 0, 0, 0, 15)
degeneratecheck(w)
```

---

dualgame — *Dual game*

Description

Given the characteristic function of a game, this function returns the characteristic function of the dual game.

Usage

```
dualgame(v)
```

Arguments

v	A characteristic function, as a vector.

Details

The dual game of $v \in G^N$ is defined by $v^D(S) = v(N) - v(N \backslash S)$ for all $S \in 2^N$.

Value

The characteristic function of the dual game. It is to be interpreted according to the order that v is introduced in.

Examples

```
v <- c(rep(0,4),rep(5,6),rep(20,4),40)
dualgame(v)
v <- seq(1:31)
dualgame(v)
dualgame(dualgame(v)) == v
```

dummynull — *Dummy and null players*

Description

Given a game, this function identifies its dummy players and null players.

Usage

```
dummynull(v, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given a game $v \in G^N$, $i \in N$ is said to be a dummy player if $v(S) + v(\{i\}) = v(S \cup \{i\})$ for all $S \subset N \setminus \{i\}$.

A dummy player $i \in N$ is said to be a null player if $v(\{i\}) = 0$.

Value

Two different vectors are returned: one containing the dummy players and the other containing the null players.

Examples

```
v <- c(0,1,0,1,0,1,1)
dummynull(v)
# Checking if a particular player is a dummy player:
2 %in% dummynull(v)$dummy # player 2 is a dummy player in v
2 %in% dummynull(v)$null # player 2 is not a null player in v
```

essentialcheck *Essential check*

Description

This function checks if the given game is essential.

Usage

```
essentialcheck(v, binary = FALSE)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, binary=FALSE. Should be set to TRUE if v is introduced in binary order instead of lexicographic order.

Details

A game $v \in G^N$ is essential if its set of imputations is non-empty, that is, if $v(N) \geq \sum_{i \in N} v(i)$.

Value

TRUE if the game is essential, FALSE otherwise.

See Also

degeneratecheck

Examples

```
v <- c(0, 0, 0, 2, 3, 4, 1)
essentialcheck(v, binary = TRUE)
essentialcheck(v, binary = FALSE)

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # essential cost game
essentialcheck(-cost.v)
```

`excesses` *Coalition excesses*

Description

Given a game and an allocation, this function computes the excess of each coalition.

Usage

```
excesses(v, binary = FALSE, x)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`x`	An allocation, as a vector.

Details

Given a game $v \in G^N$ and an allocation x, the excess of coalition $S \in 2^N$ with respect to x is defined as $e(x,S) = v(S) - x(S)$, where $x(S) = \sum_{i \in S} x_i$.

Value

The excesses of all coalitions, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

See Also

nucleolusvalue, nucleoluspcvalue

Examples

```
excesses(v=c(0,0,3,0,3,8,6,0,6,9,15,8,16,17,20), binary=TRUE, x=c(8,7,2,3))
excesses(v=c(1,5,10,6,11,15,16), x=c(1,5,10)) <= 0 # core allocation
```

getcoalition	*Get coalition*

Description

This function returns the players that form the coalition whose binary order position coincides with the given integer.

Usage

```
getcoalition(num)
```

Arguments

num	A binary order position of a coalition, as an integer.

Details

A coalition $S \in 2^N$ can be represented by the n-digit binary number $s_1 \ldots s_n$ in which $s_i = 1$ if $i \in S$ and $s_i = 0$ otherwise. The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$.

Value

The players that form the coalition whose binary order position is the given integer, as a vector.

See Also

codebin2lex, codelex2bin, getcoalitionnumber

Examples

```
num <- 5
getcoalition(num)
n <- 4
for (i in 1:(2^n - 1)){
  cat("[", i, "]", paste(getcoalition(i)),"\n")
}
```

getcoalitionnumber *Get coalition number*

Description

This function returns the binary order position of the coalition formed by the given players.

Usage

```
getcoalitionnumber(S)
```

Arguments

S	The players forming the coalition, as a vector.

Details

A coalition $S \in 2^N$ can be represented by the n-digit binary number $s_1 \dots s_n$ where $s_i = 1$ if $i \in S$ and 0 otherwise. The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$.

Value

The binary order position of the coalition formed by the given players.

See Also

codebin2lex, codelex2bin, getcoalition

Examples

```
N <- c(1:5)
S <- c(1, 2, 3)
getcoalitionnumber(S)
n <- length(N) # number of players
NS <- setdiff(N,S) # complementary coalition
getcoalitionnumber(S) + getcoalitionnumber(NS) == 2^n - 1
```

getpermutation *Get permutation*

Description

Given a number of players and a position, this function returns the permutation of players that occupies the given position when permutations are arranged according to the Lehmer code.

Usage

```
getpermutation(n, pos)
```

Arguments

n	Number of players, as an integer.
pos	Position according to the Lehmer code, as an integer.

Details

The Lehmer code makes use of the fact that there are $n!$ permutations of a sequence of n numbers. If a permutation σ is specified by the sequence $(\sigma_i)_{i=1}^n$, its Lehmer code is the sequence $L(\sigma) = (L(\sigma)_i)_{i=1}^n$, where $L(\sigma)_i = |\{j > i : \sigma_j < \sigma_i\}|$.

The position of permutation σ according to the Lehmer code order is

$$L_\sigma = 1 + \sum_{i=1}^{n} (n-i)! L(\sigma)_i$$

.

Value

The permutation of n players whose Lehmer code position is pos, as a vector.

See Also

getpermutationnumber

Examples

```
getpermutation(4, 5)
n <- 4
for (i in 1:factorial(n)) {
  cat("[", i, "]", paste(getpermutation(n,i)), "\n")
}
```

`getpermutationnumber` *Get permutation number*

Description

Given a permutation, this function returns its position in the Lehmer code order.

Usage

```
getpermutationnumber(permutation)
```

Arguments

`permutation`	A permutation, as a vector.

Details

The Lehmer code makes use of the fact that there are $n!$ permutations of a sequence of n numbers. If a permutation σ is specified by the sequence $(\sigma_i)_{i=1}^n$, its Lehmer code is the sequence $L(\sigma) = (L(\sigma)_i)_{i=1}^n$, where $L(\sigma)_i = |\{j > i : \sigma_j < \sigma_i\}|$.

The position of permutation σ according to the Lehmer code order is

$$L_\sigma = 1 + \sum_{i=1}^{n} (n-i)! L(\sigma)_i$$

.

Value

The position of the permutation according to the Lehmer code order, as an integer.

See Also

getpermutation

Examples

```
getpermutationnumber(c(1, 2, 5, 4, 3))
```

harsanyidividend *Harsanyi dividend*

Description

This function computes the Harsanyi dividend of the given coalition in the given game.

Usage

```
harsanyidividend(v, S, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`S`	The position of a coalition, as an integer.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` and `S` are introduced according to binary order instead of lexicographic order.

Details

The Harsanyi dividends of $v \in G^N$ are the coordinates of the game in the base of unanimity games. They are defined, for all $S \in 2^N$, by

$$c_S = \sum_{S' \subset S} (-1)^{|S|-|S'|} v(S')$$

.

Value

The Harsanyi dividend of the coalition that occupies the given position in the given order.

References

Hammer, P.J., Peled, U.N., & Sorensen, S. (1977). Pseudo-boolean function and game theory I. Core elements and Shapley value. *Cahiers du Centre d'Etudes de Recherche Opérationnelle*, 19, 156-176.

See Also

unanimitygame

Examples

```
n <- 3
v <- c(1, 5, 10, 7, 11, 15, 16) # introduced in lexicographic order

coalitionsvector<-character()
dividendsvector<-numeric()

for (i in 1:(2^n-1)){
  coalitionsvector <- c(coalitionsvector,
                        paste(getcoalition(i)[getcoalition(i) != 0],collapse = " "))
  dividendsvector <- c(dividendsvector,
                       harsanyidividend(v, codelex2bin(n,i), binary = FALSE))
}

data.frame(Coalition = coalitionsvector, Dividend = dividendsvector)
data.frame(Coalition = bin2lex(coalitionsvector), Dividend = bin2lex(dividendsvector))
```

---

kohlbergcriterion   *Kohlberg criterion for the prenucleolus*

Description

This function applies the Kohlberg criterion to check if the given efficient allocation is the prenucleolus of the given game.

Usage

```
kohlbergcriterion(v, x, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

v	A characteristic function, as a vector.
x	An efficient allocation, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given $v \in G^N$ and $x \in \mathbb{R}^n$ with $\sum_{i \in N} x_i = v(N)$, let $k(x)$ be the number of different excesses in x. According to the Kohlberg criterion for the prenucleolus, x is the prenucleolus of v if and only if, for each $j \in \{1, \ldots, k(x)\}$, $\bigcup_{t=1}^{j} F^t$ is a balanced family, being F^t the set of coalitions associated with the excess that occupies position t when excesses are arranged in decreasing order.

Value

`TRUE` if `x` is the prenucleolus of `v`, `FALSE` otherwise.

References

Kohlberg, E. (1971). On the Nucleolus of a Characteristic Function Game. *SIAM Journal on Applied Mathematics*, 20(1), 62–66.

See Also

balancedfamilycheck, excesses, prenucleolusvalue

Examples

```
v <- c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160)
x <- prenucleolusvalue(v)
kohlbergcriterion(v, x) # x is the prenucleolus of v
y <- prenucleolusvalue(v) + c(1,-1,0,0)
kohlbergcriterion(v, y) # y is not the prenucleolus of v

# If the game is 0-monotonic, its nucleolus coincides with its prenucleolus,
# and therefore must pass the Kohlberg criterion for the prenucleolus:
v4 <- c(-2,-2,-2,7,7,7,6)
zeromonotoniccheck(v4)
kohlbergcriterion(v4, nucleolusvalue(v4))
```

---

leastcore *Least core*

Description

Given a game, this function computes its least core.

Usage

```
leastcore(v, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given a game $v \in G^N$ and a number $\varepsilon \in \mathbb{R}$, the ε-core of v is defined as

$$C_\varepsilon(v) = \{x \in \mathbb{R}^n : x(N) = v(N) \text{ and } x(S) \geq v(S) - \varepsilon \; \forall S \in 2^N \setminus \{\emptyset, N\}\},$$

where $x(S) = \sum_{i \in S} x_i$. The least core of v is defined as the intersection of all non-empty ε-cores of v:

$$LC(v) = \{ \bigcap_{\varepsilon \in \mathbb{R} \,:\, C_\varepsilon(v) \neq \emptyset} C_\varepsilon(v) \}.$$

The implementation of this function is based on the algorithm presented in Derks and Kuipers (1997) and on the MATLAB package WCGT2005 by J. Derks.

Value

This function returns four outputs:

t	The excess value that defines the least core.
sat	The positions (binary order positions if binary=TRUE; lexicographic order positions otherwise) of the saturated coalitions, as a vector.
x	A least core allocation, as a vector.
vt	The game whose core is the least core of v, as a vector in binary order if binary=TRUE and in lexicographic order otherwise.

References

Derks, J. & Kuipers, J. (1997). Implementing the simplex method for computing the prenucleolus of transferable utility games.

Software by J. Derks (Copyright 2005 Universiteit Maastricht, dept. of Mathematics), available in package *MatTuGames*,

https://www.shorturl.at/i6aTF.

See Also

excesses, nucleoluspcvalue, nucleolusvalue, prenucleolusvalue

Examples

```
v <- c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160)
( vt <- leastcore(v)$vt )
# Plotting the core and the least core of v:
plotcoresets(games = rbind(v,vt), imputations = FALSE)

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # characteristic function of the cost game
-leastcore(-cost.v)$t # the excess value that defines the least core of cost.v
leastcore(-cost.v)$sat # the saturated coalitions
-leastcore(-cost.v)$x # a least core allocation
-leastcore(-cost.v)$vt # the cost game whose core is the least core of cost.v
```

lex2bin — *Lexicographic order to binary order*

Description

Given a characteristic function in lexicographic order, this function returns the characteristic function in binary order.

Usage

```
lex2bin(v)
```

Arguments

v	A characteristic function, as a vector in lexicographic order.

Details

Lexicographic order arranges coalitions in ascending order according to size, and applies lexicographic order to break ties among coalitions of the same size. The binary order position of a coalition $S \in 2^N$ is given by $\sum_{i \in S} 2^{i-1}$.

Value

The characteristic function, as a vector in binary order.

See Also

bin2lex, codebin2lex, codelex2bin

Examples

```
v <- seq(1:31)
lex2bin(v)
bin2lex(lex2bin(v))==v
```

lorenzdominancerelation — *Lorenz dominance relation*

Description

Given two awards vectors, this function returns the Lorenz dominance relation between them.

Usage

```
lorenzdominancerelation(x, y)
```

Arguments

x	A vector.
y	A vector.

Details

In order to compare two vectors $x, y \in \mathbb{R}^n$ through the Lorenz criterion, both of them must be rearranged in non-decreasing order; thus, let $\bar{x}$ and $\bar{y}$ be the vectors obtained by rearranging x and y, respectively, in non-decreasing order. It is said that x Lorenz-dominates y (or that y is Lorenz-dominated by x) if all the cumulative sums of $\bar{x}$ are not less than those of $\bar{y}$. That is, x Lorenz-dominates y if $\sum_{j=1}^{n} \bar{x}_j = \sum_{j=1}^{n} \bar{y}_j$ and, for each $k = 1, \ldots, n-1$,

$$\sum_{j=1}^{k} \bar{x}_j \geq \sum_{j=1}^{k} \bar{y}_j.$$

If x Lorenz-dominates y and y Lorenz-dominates x, then x and y are said to be Lorenz-equal.

If x does not Lorenz-dominate y and y does not Lorenz-dominate x, then x and y are not Lorenz-comparable.

Value

There are four possible outputs:

-1	if the introduced vectors are not Lorenz-comparable.
0	if the vectors are Lorenz-equal.
1	if the vectors are not Lorenz-equal and the first one Lorenz-dominates the second one.
2	if the vectors are not Lorenz-equal and the second one Lorenz-dominates the first one.

References

Lorenz, M. O. (1905). Methods of Measuring the Concentration of Wealth. *Publications of the American Statistical Association*, 9(70), 209-219.

Examples

```
lorenzdominancerelation(c(1,2,3), c(1,1,4))
lorenzdominancerelation(c(1,2,7,2), c(1,1,4,6))
```

`marginalgame` *Marginal game*

Description

Given a game and a coalition, this function returns the characteristic function of the corresponding marginal game.

Usage

```
marginalgame(v, S, binary = FALSE)
```

Arguments

v	Characteristic function, as a vector.
S	The position of a coalition, as an integer.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` and `S` are introduced according to binary order instead of lexicographic order.

Details

Given a game $v \in G^N$ and a coalition $S \in 2^N$, the S-marginal game, $v^S \in G^N$, is defined by

$$v^S(R) = v(R \cup (N \backslash S)) - v(N \backslash S) + v(R \cap (N \backslash S)) \text{ for all } R \in 2^N.$$

Value

The characteristic function of the `S`-marginal game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Sánchez Rodríguez, E., Mirás Calvo, M.A., Quinteiro Sandomingo, C., & Núñez Lugilde, I. (2024). Coalition-weighted Shapley values. International Journal of Game Theory 53, 547-577.

Examples

```
v <- c(0, 0, 0, 2, 3, 10, 20)
marginalgame(v, 5, binary = TRUE) # coalition {1,3}
n <- 3
for (i in 1:(2^n - 1)) {
  cat("[", i, "]", paste(marginalgame(lex2bin(v),codebin2lex(n,i),binary=TRUE)),"\n")
}
for (i in 1:(2^n - 1)) {
  cat("[", i, "]", paste(marginalgame(v,i)),"\n")
}
```

`marginalvector` *Marginal contributions vector*

Description

Given a game and a permutation, this function returns the corresponding marginal contributions vector.

Usage

```
marginalvector(v, binary = FALSE, permutation)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`permutation`	Position of the permutation in the Lehmer code order, as an integer.

Details

Given a game $v \in G^N$ and an order π of the players in N, the marginal contributions associated with order π is defined, for all $i \in N$, as $m_i^\pi = v(Pre^\pi(i) \cap i) - v(Pre^\pi(i))$, being $Pre^\pi(i) = \{j : \pi(j) < \pi(i)\}$.

Value

The vector of marginal contributions.

See Also

getpermutation, getpermutationnumber

Examples

```
n <- 3
v <- c(1, 5, 10, 30, 60, 90, 200)
for (i in 1:factorial(n)) {
  cat("[", i, "]", paste(getpermutation(3,i)),"  ",
  paste(marginalvector(v,binary=FALSE,i)), "\n")
}
```

`minimalrightsvector` *Minimal rights vector*

Description

This function computes the minimal rights vector of a game.

Usage

```
minimalrightsvector(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

Given $v \in G^N$, the utopia payoff of player $i \in N$ is defined as $M_i(N, v) = v(N) - v(N \backslash i)$.

The minimal right of player $i \in N$ is defined as $m_i(N, v) = \max_{S: i \in S}(v(S) - \sum_{j \in S \backslash i} M_j(N, v))$.

Value

The minimal rights vector.

See Also

utopiapayoffsvector

Examples

```
v <- c(0, 0, 0, 1, 1, 1, 2)
minimalrightsvector(v)
convexcheck(v)
minimalrightsvector(v) == c(v[1],v[2],v[3])
w <- c(0,0,0,4,7,6,10)
convexcheck(w)
minimalrightsvector(w) == c(w[1],w[2],w[3])
```

`monotoniccheck` *Monotonic check*

Description

This function checks if the given game is monotonic.

Usage

```
monotoniccheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is monotonic if $v(S) \leq v(T)$ for all $S, T \in 2^N$ such that $S \subset T$.

Value

`TRUE` if the game is monotonic, `FALSE` otherwise. If `instance=TRUE` and the game is not monotonic, the function also returns the positions (binary order positions if `binary=TRUE`; lexicographic order positions otherwise) of a pair of coalitions violating monotonicity.

See Also

additivecheck, superadditivecheck, zeromonotoniccheck

Examples

```
v <- c(0, 0, 1, 5, 1, 1, 2)
monotoniccheck(v, binary=FALSE, instance=TRUE)
```

`museumpassgame` *Museum pass game*

Description

This function returns the characteristic function of the described museum pass game.

Usage

```
museumpassgame(V, p = rep(1, dim(V)[2]), binary = FALSE)
```

Arguments

`V`	A matrix of zeros and ones where each row represents a museum and each column represents a visitor. If museum i is visited by visitor j, $V_{ij} = 1$; otherwise, $V_{ij} = 0$.
`p`	A vector containing the price that each visitor pays for their pass. By default, it is a vector of ones.
`binary`	A logical value. By default, `binary=FALSE`.

Details

Let N be a non-empty and finite set of museums and let U be a non-empty and finite set of visitors. The museum matrix, $V \in \{0,1\}^{N\times U}$, specifies which museums are visited by which visitors: $V_{ij} = 1$ if and only if museum $i \in N$ is visited by visitor $j \in U$. The vector $p \in \mathbb{R}^|U|_+$ represents, for each visitor j, the price they pay for their museum pass (all passes are equal, in the sense that they grant access to the same set of museums, but the price may not be the same for all visitors).

The total revenue is to be divided among the museums. Given a museum pass situation (N, U, V, p), the museum pass game is defined by

$$v(S) = \sum_{j\in U : N^j \subset S} p_j \text{ for each coalition } S \in 2^N,$$

where $N^j = \{i \in N : V_{ij} = 1\}$ is the set of museums visited by $j \in U$.

Value

The characteristic function of the museum pass game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Ginsburgh, V. & Zang, I. (2003). The museum pass game and its value. *Games and economic behavior*, 43(2), 322-325.

Examples

```
V <- rbind(c(1,0,1,1,0), c(0,1,1,1,0), c(1,1,0,0,1), c(1,0,1,0,1))
museumpassgame(V, p=c(1,1,4,5,8))
```

myersonvalue *Myerson value*

Description

Given a game and a communications network, this function computes the Myerson value.

Usage

```
myersonvalue(v, binary = FALSE, communications, game = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`communications`	An undirected communications network, as a list of vectors (their order being irrelevant), each containing two different players (their order being irrelevant). If two players are able to communicate with each other, a bidimensional vector containing them should be present in the list; otherwise, the vector should be absent. When `communications` is not specified, it is assumed that all players can communicate with each other.
`game`	A logical value. By default, `game=FALSE`. If set to `TRUE`, the game with restricted communication is also returned.

Details

Let $v \in G^N$. Assuming that communication between players is necessary for their cooperation, the game with restricted communication, v^A, is defined by $v^A(S) = v(S)$ if the players of S can communicate and $v^A(S) = 0$ otherwise, for each $S \in 2^N$.

The Myerson value is the Shapley value of the game v^A.

Value

The corresponding Myerson value, as a vector.

References

Myerson, R. B. (1977). Graphs and cooperation in games. *Mathematics of Operations Research*, 2(3), 225-229.

See Also

shapleyvalue

Examples

```
v <- c(0,0,0,0,30,30,40,40,50,50,60,70,80,90,100)
communications <- list(c(1,2), c(1,3), c(1,4))
myersonvalue(v, binary=FALSE, communications)
```

`normalizedgame` *Normalized game*

Description

Given a game, this function returns the characteristic function of its 0-1-normalization, its 0-(-1) normalization or its 0-0 normalization, as appropriate.

Usage

```
normalizedgame(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

A game $v \in G^N$ is: 0-1 normalized if $v(i) = 0$ for all $i \in N$ and $v(N) = 1$; 0-0 normalized if $v(i) = 0$ for all $i \in N$ and $v(N) = 0$; and 0-(-1) normalized if $v(i) = 0$ for all $i \in N$ and $v(N) = -1$.

If $v(N) > \sum_{i \in N} v(i)$, the 0-1 normalized game of v, $v_{0,1} \in G^N$, is defined by

$$v_{0,1}(S) = \frac{v(S) - \sum_{i \in S} v(i)}{v(N) - \sum_{i \in N} v(i)}$$

for all $S \in 2^N$.

If $v(N) < \sum_{i \in N} v(i)$, the 0-(-1) normalized game of v, $v_{0,-1} \in G^N$, is defined by

$$v_{0,-1}(S) = -\frac{v(S) - \sum_{i \in S} v(i)}{v(N) - \sum_{i \in N} v(i)}$$

for all $S \in 2^N$.

If $v(N) = \sum_{i \in N} v(i)$, the 0-0 normalized game of v, $v_{0,0} \in G^N$, is defined by

$$v_{0,0}(S) = v(S) - \sum_{i \in S} v(i)$$

for all $S \in 2^N$.

Value

The characteristic function of the 0-1-normalized game, the 0-(-1) normalized game or the 0-0 normalized game; as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

See Also

strategicallyequivalentcheck, zeronormalizedcheck, zeronormalizedgame

Examples

```
v <- c(1, 5, 11, 6, 11, 15, 16)
normalizedgame(v, binary = TRUE)
w <- c(4, 3, 8, 16, 17, 18, 15)
normalizedgame(w)
z <- c(2,3,5,10,12,14,5)
normalizedgame(z)
```

`nucleoluspcvalue` *Per capita nucleolus*

Description

Given a game, this function computes its per capita nucleolus.

Usage

```
nucleoluspcvalue(v, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given a game $v \in G^N$ and an allocation $x \in I(v)$, the per capita excess of each coalition $S \in 2^N$ with respect to x is defined as

$$e^p(v, x, S) = \frac{v(S) - \sum_{i \in S} x_i}{|S|}.$$

The per capita excesses of all non-empty coalitions, sorted in non-increasing order, are stored in the per capita excesses vector, $\theta^p(x)$. For any game $v \in G^N$ with a non-empty set of imputations, the per capita nucleolus is defined as the only imputation $pcn(v) \in I(v)$ that satisfies $\theta^p(pcn(v))_i \leqslant \theta^p(y)_i$ for each $i \in \{1, \ldots, 2^N - 1\}$ and for all $y \in I(v)$. This function is programmed following the algorithm of Potters, J.A., et al. (1996).

Value

The per capita nucleolus of the game, as a vector.

References

Grotte, J. (1970). *Computation of and Observations on the Nucleolus, the Normalized Nucleolus and the Central Games*. Master's thesis), Cornell University, Ithaca.

Potters, J. A., Reijnierse, J. H., & Ansing, M. (1996). Computing the nucleolus by solving a prolonged simplex algorithm. *Mathematics of Operations Research*, 21(3), 757-768.

See Also

excesses, leastcore, nucleolusvalue, prenucleolusvalue

Examples

```
nucleoluspcvalue(c(1,5,10,6,11,15,16))
nucleoluspcvalue(c(0,0,0,30,30,80,100))

# Computing the per capita nucleolus of a random essential game:
n <- 10 # number of players in the game
v <- c(rep(0,n),runif(2^n-(n+1),min=10,max=20)) # random essential game
nucleoluspcvalue(v)

# What if the game is a cost game?
cost.v <- airfieldgame(c(1,5,10,15)) # cost game
-nucleoluspcvalue(-cost.v) # per capita nucleolus of the cost game
```

---

nucleolusvalue *Nucleolus*

Description

Given a game, this function computes its nucleolus.

Usage

```
nucleolusvalue(v, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given a game $v \in G^N$ and an allocation x, the excess of coalition $S \in 2^N$ with respect to x is defined as $e(v, x, S) = v(S) - x(S)$, where $x(S) = \sum_{i \in S} x_i$. By sorting the excesses of all coalitions in non-increasing order, a $2^{|N|}$-tuple of complaints, denoted by $\theta(x)$, is obtained. Thus, $\theta_i(x) \geqslant \theta_j(x)$ for all $i, j \in \{1, 2, \ldots, 2^n - 1\}$ with $i < j$.

The nucleolus can be computed through the following process. First, consider only the imputations that would minimize the first complaint, that is, find the set $I_1 = \{x \in I(v) : \theta_1(x) \leqslant \theta_1(y)$ for all $y \in I(v)\}$. Then, among those imputations, consider only those that would minimize the second complaint, that is, find the set $I_2 = \{x \in I_1 : \theta_2(x) \leqslant \theta_2(y)$ for all $y \in I_1\}$. Repeat the same operation with successive complaints. Eventually, a set $I_{2^{|N|}}$ is reached. This is the nucleolus.

If v is essential, the nucleolus exists and comprises a single imputation: the only imputation $\eta \in I(v)$ that satisfies $e(\eta) \leq e(x)$ (lexicographically) for all $x \in I(v)$.

If the core of v is not empty, the nucleolus belongs to it.

This function is programmed following the algorithm of Potters, J.A., et al. (1996).

Value

The nucleolus of the game, as a vector.

References

Potters, J. A., Reijnierse, J. H., & Ansing, M. (1996). Computing the nucleolus by solving a prolonged simplex algorithm. *Mathematics of Operations Research*, 21(3), 757-768.

Schmeidler, D. (1969). The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 17(6), 1163-1170.

See Also

excesses, leastcore, nucleoluspcvalue, prenucleolusvalue

Examples

```
v1 <- c(0,0,3,0,3,8,6,0,6,9,15,8,16,17,20)
nucleolusvalue(v1,binary=TRUE)

v2 <- c(0,0,0.7,0,0.4925,0.68,0.83,0,0.56,0.74,0.64,0.46,0.55,0.57,0.61,0,
0.35,0.56,0.72,0.8125,0.69,0.48,0.95,0.88,0.71,0.91,0.44,0.89,0.37,0.63,1)
nucleolusvalue(v2,binary=TRUE)

# Computing the nucleolus of a random essential game:
n <- 10 # number of players in the game
v3 <- c(rep(0,n),runif(2^(n)-(n+1),min=10,max=20)) # random essential game
nucleolusvalue(v3)

# If the game is 0-monotonic, its nucleolus coincides with its prenucleolus,
# and therefore must pass the Kohlberg criterion for the prenucleolus:
v4 <- c(-2,-2,-2,7,7,7,6)
zeromonotoniccheck(v4)
```

```
kohlbergcriterion(v4,nucleolusvalue(v4))

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
-nucleolusvalue(-cost.v) # nucleolus of the cost game
```

`owenvalue` *Owen value*

Description

Given a game and a partition of the set of players, this function computes the Owen value.

Usage

```
owenvalue(v, binary = FALSE, partition = NULL, game = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`partition`	A partition of the set of players, as a list of vectors. When not specified, it is taken to be the partition whose only element is the set of all players.
`game`	A logical value. By default, `game=FALSE`. If set to `TRUE`, the associated quotient game is also returned.

Details

Let $v \in G^N$ and let $C = \{C_1, \ldots, C_m\}$ be a partition of the set of players. For each $T \in 2^N \setminus \emptyset$, let $R'_T = \{j : C_j \cap T \neq \emptyset\}$ and $R_j^T = C_j \cap T$ for each $j \in \{1, \ldots, m\}$. Being c_T the Harsanyi dividend of coalition $T \in 2^N$, the Owen value of each player $i \in N$ is defined as

$$O_i(v, C) = \sum_{T \in 2^N : j \in R'_T, i \in R_j^T} \frac{c_T}{|R'_T||R_j^T|}.$$

Value

The corresponding Owen value, as a vector; and, if `game=TRUE`, the associated quotient game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Owen, G. (1977). Values of Games with a Priori Unions. In R. Henn and O. Moeschlin (Eds.), *Mathematical Economics and Game Theory* (pp. 76-88), Springer.

See Also

shapleyvalue, harsanyidividend

Examples

```
v <- c(0,0,0,0,30,30,40,40,50,50,60,70,80,90,100) # in lexicographic order
owenvalue(v, partition=list(c(1,3),c(2),c(4)))
owenvalue(v)
round(owenvalue(v),10) == round(shapleyvalue(v),10)
w <- c(0,0,0,0,0,10,10,20,10,20,10,20,10,20,10,20,40,20,40,20,40,
       20,40,20,20,80,60,80,80,60,100) # in lexicographic order
owenvalue(w, partition=list(c(1,2,3),c(4,5)))
```

`perfectcoregame` *Perfect core game*

Description

This function returns the perfect core game with a given number of players.

Usage

```
perfectcoregame(n, binary = FALSE)
```

Arguments

`n`	A number of players, as an integer.
`binary`	A logical value. By default, `binary=FALSE`.

Details

The perfect core game of n players is defined by

$$v_P(S) = s - \sqrt{\frac{s(n-s)}{n-1}} \text{ for all } S \in 2^N,$$

where $s = |S|$.

Value

The characteristic function of the perfect core game with `n` players, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Shapley, L. S. (1971). Cores of convex games. *International Journal of Game Theory*, 1(3), 11-26.

Examples

```
perfectcoregame(6)
```

`plotcoreset` *Plot core set*

Description

Given a game with two, three or four players, this function plots its core set and set of imputations.

Usage

```
plotcoreset(
  v,
  binary = FALSE,
  imputations = TRUE,
  projected = FALSE,
  solutions = NULL,
  allocations = NULL,
  color = "blue"
)
```

Arguments

`v`	A characteristic function, as a vector. The game represented by `v` is assumed to be a profit game (i.e., a game in which a greater allocation is a more desirable allocation), not a cost game (i.e., a game in which a smaller allocation is a more desirable allocation).
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`imputations`	A logical value. By default, `imputations=TRUE`. When set to `imputations=FALSE`, the set of imputations is not drawn.
`projected`	A logical value. By default, `projected=FALSE`. When set to `projected=TRUE`, for games with three or four players the function draws a projection of the core set (and a projection of the set of imputations, as long as `imputations=TRUE`) instead of a full-dimensional representation.
`solutions`	Optional. A character vector containing a solution or a series of solutions to be added to the plot. Valid solutions: `"corecenter"`, `"nucleolus"`, `"nucleoluspc"`, `"shapleyvalue"`, `"tauvalue"`.
`allocations`	Optional. A matrix containing an allocation or a series of allocations to be added to the plot. The matrix should have as many columns as players in `v` and as many rows as allocations are introduced, so that each row contains an allocation.
`color`	The color in which the core set is to be drawn. By default, `color="blue"`.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S) \, \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

A core set plot with the specified features.

See Also

plotcoresets

Examples

```
v1 <- claimsgame(E=8,d=c(3,5,6))
plotcoreset(v1,solutions=c("nucleolus","shapleyvalue"))

v2 <- c(0,0,0,0,0,0,0,0,1,4,1,3,6,8,10)
plotcoreset(v2,solutions=c("corecenter","nucleoluspc"))
```

---

plotcoresets — *Plot multiple core sets*

Description

Given multiple games with two, three or four players, this function draws in a single plot their projected core sets and sets of imputations.

Usage

```
plotcoresets(games, binary = FALSE, imputations = TRUE)
```

Arguments

`games`	A matrix in which each row is a characteristic function.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if the characteristic functions in `games` are introduced in binary order instead of lexicographic order.
`imputations`	A logical value. By default, `imputations=TRUE`. When set to `imputations=FALSE`, the sets of imputations are not drawn.

Details

The core of a game $v \in G^N$ is the set of all its stable imputations:

$$C(v) = \{x \in \mathbb{R}^n : x(N) = v(N), x(S) \geq v(S) \; \forall S \in 2^N\},$$

where $x(S) = \sum_{i \in S} x_i$.

Value

A plot of the given core sets.

See Also

plotcoreset

Examples

```
# Plotting the core and the least core of a game:
v <- c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160)
vt <- leastcore(v)$vt
plotcoresets(games = rbind(v,vt), imputations = FALSE)
```

`prenucleolusvalue` *Prenucleolus*

Description

Given a game, this function computes its prenucleolus.

Usage

```
prenucleolusvalue(v, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Given a game $v \in G^N$ and an allocation x, the excess of coalition $S \in 2^N$ with respect to x is defined as $e(v,x,S) = v(S) - x(S)$, where $x(S) = \sum_{i \in S} x_i$. Let $\theta(x)$ be a vector of excesses at x arranged in non-increasing order. It is said that a vector α is lexicographically greater than another vector β if $\alpha \neq \beta$ and the first non-zero coordinate of vector $\alpha - \beta$ is positive.

The prenucleolus is the set of the efficient allocations that produce a lexicographically minimal vector of excesses. It is always non-empty and it actually comprises a single allocation, which in zero-monotonic games coincides with the nucleolus.

The implementation of this function is based on the algorithm presented in Derks and Kuipers (1997) and on the MATLAB package WCGT2005 by J. Derks.

Value

The prenucleolus of the game, as a vector.

References

Derks, J. & Kuipers, J. (1997). Implementing the simplex method for computing the prenucleolus of transferable utility games.

Schmeider, D. (1969). The Nucleolus of a Characteristic Function Game. *SIAM Journal on Applied Mathematics*, 17(6), 1163–1170.

Software by J. Derks (Copyright 2005 Universiteit Maastricht, dept. of Mathematics), available in package *MatTuGames*,

https://www.shorturl.at/i6aTF.

See Also

excesses, kohlbergcriterion, leastcore, nucleoluspcvalue, nucleolusvalue

Examples

```
prenucleolusvalue(c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160))
v <- runif(2^6-1, min = 10, max = 20) # random 6-player game
prenucleolusvalue(v)

# The prenucleolus of v must pass the Kohlberg criterion.
# In some cases, though, the tolerance might have to be adjusted
# to avoid numerical error:
kohlbergcriterion(v,prenucleolusvalue(v))
kohlbergcriterion(v,prenucleolusvalue(v),tol=10^(-6))

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
-prenucleolusvalue(-cost.v) # prenucleolus of the cost game
```

---

savingsgame *Savings game*

Description

Given a cost game, this function returns the associated savings game.

Usage

```
savingsgame(c, binary = FALSE)
```

Arguments

c	The characteristic function of a cost game, as a vector.
binary	A logical value. By default, binary=FALSE. Should be set to TRUE if c is introduced in binary order instead of lexicographic order.

Details

Let $c \in G^N$ be a cost game. Its associated savings game, $v_c \in G^N$, is defined by

$$v_c(S) = \sum_{i \in S} c(i) - c(S) \text{ for each } S \in 2^N.$$

Thus, for each coalition S, $v_c(S)$ can be interpreted as the collective reduction of cost resulting from the cooperation of the members of S, with respect to the scenario of non-cooperation.

Value

The characteristic function of the savings game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

See Also

airfieldgame, zeronormalizedgame

Examples

```
savingsgame(c(360,60,288,390,468,318,468))
v.random <- rnorm(2^5-1,58,13)
savingsgame(v.random) == -zeronormalizedgame(v.random)
```

---

sequencinggame *Sequencing game*

Description

Given a sequencing situation with an initial order, this function returns the characteristic function of the associated sequencing game.

Usage

```
sequencinggame(p, alpha, pi0, binary = FALSE)
```

Arguments

`p`	A vector containing the processing time of each job.
`alpha`	A vector containing the cost per unit of time that each job generates while unfinished.
`pi0`	An initial order, as a vector.
`binary`	A logical value. By default, `binary=FALSE`.

Details

Given a coalition $S \in 2^N$, $\Pi(S)$ is the set of orders of S, that is, the set of all bijective functions from S to $\{1, \ldots, s\}$. A generic order of S is denoted by $\pi_S \in \Pi(S)$. Given $i \in S$ and $\pi_S \in \Pi(S)$, let $Pre^{\pi}(i) = \{j \in S : \pi_S(j) < \pi_S(i)\}$ be the set of predecessors of i according to order π_S.

A sequencing situation is a triple (N, p, α) and, possibly, some (information on the) initial order, where $N = \{1, \ldots, n\}$ is a finite set of agents, each one having one job that has to be processed on a machine. To simplify, agent i's job is identified with i. The processing times of the jobs are given by $p = (p_i)_{i \in N}$ with $p_i > 0$ for all $i \in N$. For each agent $i \in N$ there is a cost function $c_i : (0, \infty) \to \mathbb{R}$, so that $c_i(t)$ represents the cost incurred when job i is completed exactly at time t. Assuming that c_i is linear for all $i \in N$, there exist $\alpha_i, \beta_i \geq 0$ such that $c_i(t) = \beta_i + \alpha_i t$ for all $i \in N$, where β_i is a fixed service cost and $\alpha_i t$ is the completion cost.

For any $\pi \in \Pi(N)$, $C(S, \pi)$ is the aggregate completion cost of coalition S in the order π, formally defined as

$$C(S, \pi) = \sum_{i \in S} \alpha_i \Big(p_i + \sum_{j \in Pre^{\pi}(i)} p_j \Big).$$

A sequencing situation with initial order is a quadruple (N, p, α, π_0) where $\pi_0 \in \Pi(N)$ is the initial order of the jobs.

A coalition $S \in 2^N$ is said to be connected in order π if, for all $i, j \in S$ and $k \in N$, $\pi(i) < \pi(k) < \pi(j)$ implies $k \in S$. We say that a coalition S' is a component of S if $S' \subset S$, S' is connected, and for every $i \in S \backslash S'$, $S' \cup i$ is not connected. The components of S form a partition of S that is denoted by S/π_0. Curiel et al. (1989) define the gain of swapping i and j as $g_{ij} = \max\{0, \alpha_j p_i - \alpha_i p_j\}$.

The sequencing game (N, v_{π_0}) is defined, for all $S \in 2^N$, by

$$v_{\pi_0}(S) = \sum_{S' \in S/\pi_0} \left(\sum_{i,j \in S' : \pi_0(i) < \pi_0(j)} g_{ij} \right).$$

Value

The characteristic function of the sequencing game (interpreted as a savings game), as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Curiel, I., Pederzoli, G., & Tijs, S. (1989). Sequencing games. *European Journal of Operational Research*, 40(3), 344-351.

See Also

tailgame

Examples

```
p <- c(1,2,3,4)
alpha <- c(4,5,1,2)
pi0 <- c(2,3,1,4)
sequencinggame(p, alpha, pi0)
```

shapleyvalue *Shapley value*

Description

Given a game, this function computes its Shapley value.

Usage

```
shapleyvalue(v, binary = FALSE)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, binary=FALSE. Should be set to TRUE if v is introduced in binary order instead of lexicographic order.

Details

Given $v \in G^N$, the Shapley value of each player $i \in N$ can be defined as

$$Sh_i(v) = \sum_{S \subset N \setminus \{i\}} \frac{s!(n-s-1)!}{n!}(v(S \cup \{i\}) - v(S)).$$

It is also possible to compute it as

$$Sh_i(v) = \sum_{\emptyset \neq S \subset N} M_{i,S} v(S),$$

where $M_{i,S} = \frac{(s-1)!(n-s)!}{n!}$ if $i \in S$ and $M_{i,S} = -\frac{s!(n-s-1)!}{n!}$ if $i \notin S$.

Value

The Shapley value of the game, as a vector.

References

Le Creurer, I. J., Mirás Calvo, M. A., Núñez Lugilde, I., Quinteiro Sandomingo, C., & Sánchez Rodríguez, E. (2024). On the computation of the Shapley value and the random arrival rule. Available at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4293746.

Shapley, L. S. (1953). A value for n-person games. *Contribution to the Theory of Games*, 2.

See Also

marginalvector

Examples

```
shapleyvalue(c(0,0,3,0,3,8,6,0,6,9,15,8,16,17,20), binary=TRUE)
shapleyvalue(claimsgame(E=69.420,d=runif(10,5,10)))
```

`solidarityvalue` *Solidarity value*

Description

Given a game, this function computes its solidarity value.

Usage

```
solidarityvalue(v, binary = FALSE, amc = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`amc`	A logical value. By default, `amc=FALSE`. If set to `TRUE`, the average marginal contributions are also returned.

Details

Given $v \in G^N$, the average marginal contribution to coalition $S \in 2^N$ is defined as

$$AMC(S) = \frac{1}{|S|} \sum_{k \in S} (v(S) - v(S \backslash \{k\})).$$

The solidarity value of each player $i \in N$ can be defined as

$$\phi_i(v) = \sum_{S:i \in S} \frac{(n-|S|)!(|S|-1)!}{|N|!} AMC(S).$$

Value

The solidarity value of the game, as a vector. If `amc=TRUE`, a vector (in binary order if `binary=TRUE` and in lexicographic order otherwise) containing the average marginal contribution to each coalition is also returned.

References

Nowak, A. S. & Radzik, T. (1994). A solidarity value for n-person transferable utility games. *International Journal of Game Theory*, 23, 43-48.

See Also

shapleyvalue

Examples

```
solidarityvalue(c(0,0,0,1,1,1,2), binary=TRUE)
solidarityvalue(bin2lex(c(0,0,1,2,5,5,7)))
solidarityvalue(bin2lex(c(0,0,2,7,9,10,12,9,11,12,14,19,21,22,24)), amc=TRUE)
```

`solvels` *Solve linear system*

Description

This function classifies and solves the given linear system.

Usage

```
solvels(A, tol = 100 * .Machine$double.eps)
```

Arguments

`A`	The augmented matrix of a linear system.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Value

This function returns two outputs: `solution` and `flag`. If the introduced linear system is inconsistent: `flag=-1` and `solution=Inf`. If it is consistent and has infinitely many solutions: `flag=0` and `solution` returns one of the solutions, as a vector. If it is consistent and has a unique solution: `flag=1` and `solution` returns the unique solution, as a vector.

Examples

```
# Consistent and determinate system:
solvels(matrix(c(1,1,1,6,2,-1,1,3,-1,-1,1,0), byrow=TRUE, nrow = 3, ncol = 4))
# Consistent and indeterminate system:
solvels(matrix(c(1,1,-3,0,2,-1,-3,3,4,1,-9,3), byrow=TRUE, nrow = 3, ncol = 4))
# Inconsistent system:
solvels(matrix(c(-2,1,1,1,1,-2,1,1,1,1,-2,1), byrow=TRUE, nrow = 3, ncol = 4))
```

`strategicallyequivalentcheck`
Strategically equivalent check

Description

This function checks if two games are strategically equivalent.

Usage

```
strategicallyequivalentcheck(v, w, binary = FALSE, parameters = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`w`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` and `w` are introduced in binary order instead of lexicographic order.
`parameters`	A logical value. By default, `parameters=FALSE`.

Details

Games $v \in G^N$ and $w \in G^N$ are strategically equivalent if there exist $k > 0$ and an additive game $a \in G^N$ such that $v(S) = kw(S) + a(S)$ for all $S \in 2^N$.

Value

`TRUE` if `v` and `w` are strategically equivalent, `FALSE` otherwise. If `parameters=TRUE`, whenever `v` and `w` are strategically equivalent, the function also returns `k` (a positive integer) and `a` (the characteristic function of an additive game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise) such that $\texttt{v} = \texttt{kw} + \texttt{a}$.

See Also

additivegame, normalizedgame, zeronormalizedgame

Examples

```
w <- c(1000, 0, 0, 2000, 3000, 2000, 4000)
v <- 4.5 * w + additivegame(c(4, 6, 1), binary = TRUE)
strategicallyequivalentcheck(v, w, binary = TRUE, parameters = TRUE)
```

`subgame` *Subgame of a coalition*

Description

Given a game and a coalition, this function returns the characteristic function of the subgame of the given coalition.

Usage

```
subgame(v, S, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`S`	The position of a coalition, as an integer.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` and `S` are introduced according to binary order instead of lexicographic order.

Details

Given $v \in G^N$, the subgame of coalition $S \in 2^N$ is defined by $v_S(T) = v(T)$ for all $T \in 2^S$.

Value

The characteristic function of the subgame of the given coalition, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

Examples

```
v <- c(0, 0, 0, 0, 2, 3, 4, 5, 6, 9, 1, 15, 20, 30, 100)
S <- 13
subgame(v, S)
n <- 4
for (i in 1 : (2^n-1)) {
  cat("[", i, "]", paste(subgame(v,i)), "\n")
}
subgame(v,15)==v
```

superadditivecheck *Superadditive check*

Description

This function checks if the given game is superadditive.

Usage

```
superadditivecheck(v, binary = FALSE, instance = FALSE)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
instance	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is superadditive if $v(S \cup T) \geq v(S) + v(T)$ for all $S, T \in 2^N$ with $S \cap T = \emptyset$.

A game $v \in G^N$ is subadditive if $-v$ is superadditive.

Value

`TRUE` if the game is superadditive, `FALSE` otherwise. If `instance=TRUE` and the game is not superadditive, the function also returns the positions (binary order positions if `binary=TRUE`; lexicographic order positions otherwise) of a pair of coalitions violating superadditivity.

See Also

additivecheck, convexcheck, monotoniccheck, strategicallyequivalentcheck

Examples

```
v <- c(2, 2, 4, 2, 4, 5, 6)
superadditivecheck(v, binary = TRUE, instance = TRUE)

# How to check if a game is subadditive:
v.sub <- c(40, 30, 50, 60, 70, 65, 90) # subadditive game
superadditivecheck(-v.sub)
```

symmetrycheck *Symmetry check*

Description

Given a game and two players, this function checks if those are symmetric players.

Usage

```
symmetrycheck(v, i, j, binary = FALSE, tol = 100 * .Machine$double.eps)
```

Arguments

v	A characteristic function, as a vector.
i	The position of an individual coalition, as an integer.
j	The position of another individual coalition, as an integer.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v`, `i` and `j` are introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

Let $v \in G^N$. Players $i, j \in N$ are symmetric in v if, for each $S \subset N$ with $i, j \in S$, $v(S \setminus \{i\}) = v(S \setminus \{j\})$.

Value

`TRUE` if `i` and `j` are symmetric in `v`, `FALSE` otherwise.

Examples

```
symmetrycheck(c(13,13,0,0,58,58,0),1,2) # players 1 and 2 are symmetric
```

tailgame *Tail game*

Description

Given a sequencing situation without an initial order, this function returns the characteristic function of the associated tail game.

Usage

```
tailgame(p, alpha, binary = FALSE)
```

Arguments

p	A vector containing the processing time of each job.
alpha	A vector containing the cost per unit of time that each job generates while unfinished.
binary	A logical value. By default, `binary=FALSE`.

Details

Given $S \in 2^N$, $\Pi(S)$ is the set of orders of S, that is, the set of all bijective functions from S to $\{1, \ldots, s\}$. A generic order of S is denoted by $\pi_S \in \Pi(S)$. Given $i \in S$ and $\pi_S \in \Pi(S)$, let $Pre^{\pi}(i) = \{j \in S : \pi_S(j) < \pi_S(i)\}$ be the set of predecessors of i according to order π_S.

A sequencing situation is a triple (N, p, α) and, possibly, some (information on the) initial order, where $N = \{1, \ldots, n\}$ is a finite set of agents, each one having one job that has to be processed on a machine. To simplify, agent i's job is identified with i. The processing times of the jobs are given by $p = (p_i)_{i \in N}$ with $p_i > 0$ for all $i \in N$. For each agent $i \in N$ there is a cost function $c_i : (0, \infty) \to \mathbb{R}$, so that $c_i(t)$ represents the cost incurred when job i is completed exactly at time t. Assuming that c_i is linear for all $i \in N$, there exist $\alpha_i, \beta_i \geq 0$ such that $c_i(t) = \beta_i + \alpha_i t$ for all $i \in N$, where β_i is a fixed service cost and $\alpha_i t$ is the completion cost.

For any $\pi \in \Pi(N)$, $C(S, \pi)$ is the aggregate completion cost of coalition S in the order π, formally defined as

$$C(S, \pi) = \sum_{i \in S} \alpha_i \Big(p_i + \sum_{j \in Pre^{\pi}(i)} p_j \Big).$$

A sequencing situation without initial order is a triple (N, p, α) in which there is no information about an initial order.

An order that minimizes the aggregate completion cost of coalition N is called an optimal order and denoted by $\hat{\pi}$. Defining the urgency index of each $i \in N$ as $u_i = \frac{\alpha_i}{p_i}$, an optimal order can be obtained by arranging jobs in such a way that the corresponding arrangement of their urgency indices is non-increasing. Given a sequencing situation (N, p, α), $\Omega(N, p, \alpha)$ denotes the set of those optimal orders that also satisfy the following condition: if two jobs share the same urgency index but not the same processing, the one with shortest processing time goes first.

The characteristic function of the tail game associated to a sequencing situation (N, p, α) is defined, for each $S \in 2^N$, by

$$c_{tail}(S) = C(S, (\pi_{N \setminus S}, \hat{\pi}_S)),$$

where $\pi_{N \setminus S} \in \Pi(N \setminus S)$ and $\hat{\pi}_S \in \Omega(S, p_S, \alpha_S)$.

Having no information about an initial order, coalitions assume they will be processed at the tail of some "artificial" initial order.

Value

The characteristic function of the tail game (interpreted as a cost game), as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

References

Klijn, F. & Sánchez, E. (2006). Sequencing games without initial order. *Mathematical Methods of Operations Research*, 63, 53-62.

See Also

sequencinggame

Examples

```
p <- c(1,2,3,4)
alpha <- c(4,5,1,2)
tailgame(p,alpha)
```

---

tauvalue *τ-value*

Description

Given a game, this function computes its τ-value.

Usage

```
tauvalue(v, binary = FALSE)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

The τ-value of $v \in G^N$ is given by

$$\tau(v) = m(v) + \alpha(M(v) - m(v)),$$

where $M(v)$ is the vector of utopia payoffs, $m(v)$ is the vector of minimal rights, and α is the value for which $\sum_{i \in N} \tau_i(v) = v(N)$.

Value

The τ-value of the game, as a vector.

References

Tijs, S. H. (1981). Bounds for the core of a game and the τ-value. In O. Moeschlin and D. Pallaschke (Eds.), *Game theory and mathematical economics* (pp. 123-132).

See Also

minimalrightsvector, utopiapayoffsvector.

Examples

```
tauvalue(c(0,0,0,0,10,40,30,60,10,20,90,90,90,130,160))

# What if the game is a cost game?
cost.v <- c(2,2,2,3,4,4,5) # cost game
-tauvalue(-cost.v) # tau-value of the cost game
```

---

totallybalancedcheck *Totally balanced check*

Description

This function checks if the given game is totally balanced and computes its totally balanced cover.

Usage

```
totallybalancedcheck(
  v,
  game = FALSE,
  binary = FALSE,
  tol = 100 * .Machine$double.eps
)
```

Arguments

v	A characteristic function, as a vector.
game	A logical value. By default, `game=FALSE`. If set to `TRUE`, the totally balanced cover of the game is also returned.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
tol	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Details

A game $v \in G^N$ is totally balanced if all of its subgames are balanced (the subgame of each coalition $S \in 2^N$ with respect to v is defined by $v_S(T) = v(T)$ for all $T \in 2^S$).

Value

`TRUE` if the game is totally balanced, `FALSE` otherwise. If `game=TRUE`, the totally balanced cover of the game is also returned.

References

Maschler, M., Solan, E., & Zamir, S. (2013). *Game Theory*. Cambridge University Press.

See Also

balancedcheck, subgame

Examples

```
totallybalancedcheck(c(0,0,0,0,1/2,0,0,1/2,0,1/2,1/2,1/2,1/2,1/2,1))
totallybalancedcheck(c(0,0,0,0,1,1,0,1,0,0,1,1,1,1,2),game=TRUE)
```

---

triangularup *Square upper triangulation*

Description

This function computes a square upper triangular version of the given matrix.

Usage

```
triangularup(V, tol = 100 * .Machine$double.eps)
```

Arguments

`V`	A matrix.
`tol`	A tolerance parameter, as a non-negative number. By default, `tol=100*.Machine$double.eps`.

Value

A square upper triangular version of the given matrix.

This function returns two outputs: `SUT`, the square upper triangular matrix. `pivot`, a vector indicating pivot rows.

Examples

```
set.seed(58)
triangularup(matrix(sample(1:10, 16, replace = TRUE), nrow = 4, ncol = 4))
triangularup(matrix(c(7,8,5,5,3,5,4,1,3,10,4,4,6,7,8,8),byrow=TRUE, nrow = 4, ncol = 4))
triangularup(matrix(c(1,2,1,1,-2,0,1,1),byrow=TRUE, nrow = 2, ncol = 4))
triangularup(matrix(c(1,2,1,-2,0,1,3,-1,1,-2,3,3),byrow=TRUE, nrow = 4, ncol = 3))
```

`unanimitygame` *Unanimity game*

Description

This function returns the characteristic function of the unanimity game of a coalition.

Usage

```
unanimitygame(n, S, binary = FALSE)
```

Arguments

`n`	Number of players, as an integer.
`S`	The position of a coalition, as an integer.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `S` is introduced according to binary order instead of lexicographic order.

Details

The characteristic function of the unanimity game of a coalition $S \in 2^N$ is defined, for each $R \in 2^N$, as $u_S(R) = 1$ if $S \subset R$ and $u_S(R) = 0$ otherwise.

Value

The characteristic function of the unanimity game of coalition `S`, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

Examples

```
unanimitygame(n=4,S=7)
```

`utopiapayoffsvector` *Utopia payoffs vector*

Description

This function computes the utopia payoffs vector of a game.

Usage

```
utopiapayoffsvector(v, binary = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

Given $v \in G^N$, the utopia payoff of player $i \in N$ is defined as $M_i(N, v) = v(N) - v(N \backslash i)$.

Value

The utopia payoffs vector.

See Also

minimalrightsvector

Examples

```
v <- c(0, 10, 200, 1, 4, 7, 7)
utopiapayoffsvector(v, binary = FALSE)
```

`weightedmajoritygame` *Weighted majority game*

Description

This function returns the characteristic function of the described weighted majority game.

Usage

```
weightedmajoritygame(q, w, binary = FALSE)
```

Arguments

`q`	A quota, as a number between 0 and the sum of player weights.
`w`	The player weights, as a vector of non-negative numbers.
`binary`	A logical value. By default, `binary=FALSE`.

Details

Given a situation in which a number of agents have to vote for or against a certain measure, let $N = \{1, \ldots, n\}$ be the set of voters, w be a non-negative vector of voter weights (the weight of each voter is the number of votes or the proportion of total votes they hold), and $q \in [0, \sum_{i \in N} w_i]$ be the quota (the minimum number of votes or the minimum proportion of total votes needed to pass the measure). The corresponding weighted majority game, v, is defined by

$$v(S) = 1 \text{ if } \sum_{i \in S} w_i \geqslant q \text{ and } v(S) = 0 \text{ otherwise, for each } S \in 2^N.$$

Value

The characteristic function of the weighted majority game associated with the described situation, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

Examples

```
q <- 39
w <- c(rep(7,5),rep(1,10))
weightedmajoritygame(q,w)
```

`weightedshapleyvalue` *Positively weighted Shapley value*

Description

Given a game, positive player weights and an ordered partition of the set of players, this function returns the corresponding weighted Shapley value.

Usage

```
weightedshapleyvalue(v, binary = FALSE, weights, partition = NULL)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`weights`	The player weights, as a vector of positive numbers.
`partition`	An ordered partition of the set of players, as a list of vectors. When not specified, it is taken to be the partition whose only element is the set of all players.

Details

A weight system ω is a pair $\omega = (\lambda, \mathcal{S})$ where $\lambda = (\lambda_i)_{i \in N}$ is a positive weight vector ($\lambda_i > 0$ for each $i \in N$) and $\mathcal{S} = (S_1, \ldots, S_m)$ is an ordered partition of N. The weighted Shapley value with weight system $\omega = (\lambda, \mathcal{S})$ is the linear map Sh^ω that assigns to each unanimity game u_T, with $T \in 2^N \setminus \emptyset$, the allocations $Sh_i^\omega(u_T) = \frac{\lambda_i}{\lambda(T \cap S_k)}$ if $i \in T \cap S_k$ and $Sh_i^\omega = 0$ if $i \notin T \cap S_k$, where $k = \max\{i \in N : S_i \cap T \neq \emptyset\}$. Then, for each $v \in G^N$ and being c_T the Harsanyi dividend of coalition $T \in 2^N$,

$$Sh^\omega(v) = \sum_{T \in 2^N \setminus \emptyset} c_T Sh^\omega(u_T).$$

Value

The positively weighted Shapley value of the game, as a vector.

References

Shapley, L. S. (1953). *Additive and non-additive set functions*. PhD thesis, Department of Mathematics, Princeton University.

See Also

coalitionweightedshapleyvalue, harsanyidividend, shapleyvalue

Examples

```
v <- c(0,0,0,0,0,0,1,0,0,1,3,4,6,8,10)
weightedshapleyvalue(v,binary=TRUE,weights=c(0.5,0.2,0.2,0.1))
w <- c(0,0,0,0,30,30,40,40,50,50,60,70,80,90,100)
weightedshapleyvalue(w,weights=c(1,2,3,4),partition=list(c(1,2),c(3,4)))
```

`zeromonotoniccheck` *0-monotonic check*

Description

This function checks if the given game is 0-monotonic.

Usage

```
zeromonotoniccheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is 0-monotonic if $v_0(S) \leq v_0(T)$ for all $S, T \in 2^N$ such that $S \subset T$, being $v_0 \in G^N$ the 0-normalization of v.

Value

`TRUE` if the game is 0-monotonic, `FALSE` otherwise. If `instance=TRUE` and the game is not 0-monotonic, the function also returns the positions (binary order positions if `binary=TRUE`; lexicographic order positions otherwise) of a pair of coalitions violating 0-monotonicity.

See Also

monotoniccheck, zeronormalizedgame, zeronormalizedcheck

Examples

```
v <- c(0, 0, 0, 1, 1, 1, 2)
zeromonotoniccheck(v, binary = TRUE)
monotoniccheck(v, binary = TRUE)

w <- c(-2,-2,-2,7,7,7,6)
zeromonotoniccheck(w)
monotoniccheck(w)

z <- c(1, 1, 1, 2, 2, 2, 2)
zeromonotoniccheck(z)
monotoniccheck(z)
```

---

zeronormalizedcheck *0-normalized check*

Description

This function checks if the given game is 0-normalized.

Usage

```
zeronormalizedcheck(v, binary = FALSE, instance = FALSE)
```

Arguments

`v`	A characteristic function, as a vector.
`binary`	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.
`instance`	A logical value. By default, `instance=FALSE`.

Details

A game $v \in G^N$ is 0-normalized if $v(i) = 0$ for all $i \in N$.

Value

`TRUE` if the game is 0-normalized, `FALSE` otherwise. If `instance=TRUE` and the game is not 0-normalized, the function also returns a player for whose value is not zero.

See Also

normalizedgame, strategicallyequivalentcheck, zeromonotoniccheck, zeronormalizedgame

Examples

```
v <- c(rep(0, 4), 1, rep(30, 20), rep(3, 5), 50) # v(5)=1
zeronormalizedcheck(v, binary = FALSE, instance = TRUE)
```

zeronormalizedgame *0-normalized game*

Description

Given a game, this function returns the characteristic function of its 0-normalization.

Usage

```
zeronormalizedgame(v, binary = FALSE)
```

Arguments

v	A characteristic function, as a vector.
binary	A logical value. By default, `binary=FALSE`. Should be set to `TRUE` if `v` is introduced in binary order instead of lexicographic order.

Details

The 0-normalization of a given $v \in G^N$ is defined by $v_0(S) = v(S) - \sum_{i \in S} v(i)$ for each $S \in 2^N$.

Value

The characteristic function of the 0-normalized game, as a vector in binary order if `binary=TRUE` and in lexicographic order otherwise.

See Also

normalizedgame, savingsgame, strategicallyequivalentcheck, zeromonotoniccheck, zeronormalizedcheck

Examples

```
zeronormalizedgame(c(0,3,7,15,17,27,30))
zeronormalizedgame(c(1,5,10,6,11,15,16))
v.random <- rnorm(2^5-1,58,13)
zeronormalizedgame(v.random) == -savingsgame(v.random)
```

Index