Package ‘TUvalues’

December 16, 2025

Type Package

Title Tools for Calculating Allocations in Game Theory using Exact and Approximated Methods

Version 1.1.0

Description The main objective of cooperative Transferable-Utility games (TU-games) is to allocate a good among the agents involved. The package implements major solution concepts including the Shapley value, Banzhaf value, and egalitarian rules, alongside their extensions for structured games: the Owen value and Banzhaf-Owen value for games with a priori unions, and the Myerson value for communication games on networks. To address the inherent exponential computational complexity of exact evaluation, the package offers both exact algorithms and linear approximation methods based on sampling, enabling the analysis of large-scale games. Additionally, it supports core set-based solutions, allowing computation of the vertices and the centroid of the core.

License AGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/mariaguilleng/TUvalues

BugReports https://github.com/mariaguilleng/TUvalues/issues

Imports utils, gtools, highs

NeedsCompilation no

Author Maria D. Guillen [cre, aut] (ORCID: <https://orcid.org/0000-0002-2445-5654>), Juan Carlos Gonçalves [aut] (ORCID: <https://orcid.org/0000-0002-0867-0004>)

Maintainer Maria D. Guillen <maria.guilleng@umh.es>

Repository CRAN

Date/Publication 2025-12-16 09:30:16 UTC

Contents

`banzhaf` *Banzhaf value*

Description

Calculate the Banzhaf value

Usage

```
banzhaf(
  characteristic_func,
  n_players = 0,
  method = "exact",
  n_rep = 10000,
  replace = FALSE,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`method`	Method used to calculate the Banzhaf value. Valid methods are: `exact` for the exact calculation or `appro` for approximated polynomial calculation based on sampling.
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation
`replace`	Should sampling be with replacement?
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The Banzhaf value for each player

Examples

```
n <- 8
v <- function(coalition) {
if (length(coalition) > n/2) {
   return(1)
 } else {
   return(0)
 }
}
banzhaf(v, method = "exact", n_players = n)
banzhaf(v, method = "appro", n_rep = 2000, n_players = n, replace = TRUE)

v<-c(0,0,0,1,2,1,3)
banzhaf(v, method = "exact")
banzhaf(v, method = "appro", n_rep = 2000, replace = TRUE)
```

`banzhaf_appro` *Banzhaf Index (approximated)*

Description

Calculate the approximated Banzhaf Index based on sampling

Usage

```
banzhaf_appro(characteristic_func, n_players, n_rep, replace, echo)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	The number of players
`n_rep`	The number of iterations to perform in the approximated calculation
`replace`	Should sampling be with replacement?
`echo`	Show progress of the calculation.

Value

The Shapley value for each player

`banzhaf_exact` *Banzhaf Index (exact)*

Description

Calculate the approximated Banzhaf Index

Usage

```
banzhaf_exact(characteristic_func, n_players)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	The number of players in the game.

Value

The Banzhaf Index for each player

banzhaf_owen *Banzhaf-Owen value*

Description

Calculate the Banzhaf-Owen value

Usage

```
banzhaf_owen(
  characteristic_func,
  union,
  n_players = 0,
  method = "exact",
  n_rep = 10000,
  replace = TRUE,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`method`	Method used to calculate the Owen value. Valid methods are: `exact` for the exact calculation or `appro` for approximated polynomial calculation based on sampling proposed by Saavedra-Nieves & Fiestras-Janeiro (2021).
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation
`replace`	Should sampling be with replacement?
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The Banzhaf-Owen value for each player

References

Saavedra-Nieves, A., & Fiestras-Janeiro, M. G. (2021). Sampling methods to estimate the Banzhaf–Owen value. Annals of Operations Research, 301(1), 199-223.

Examples

```
characteristic_func <- c(0,0,0,0,30,30,40,40,50,50,60,70,80,90,100)
union <- list(c(1,3),c(2),c(4))
banzhaf_owen(characteristic_func, union)
banzhaf_owen(characteristic_func, union, method = "appro", n_rep = 4000)
```

`banzhaf_owen_appro` *Banzhaf-Owen Value*

Description

Calculate the approximated Banzhaf-Owen value using the algorithm proposed by Saavedra-Nieves & Fiestras-Janeiro (2021).

Usage

```
banzhaf_owen_appro(characteristic_func, union, n_players, n_rep, replace, echo)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players
`n_players`	The number of players
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`replace`	Should sampling be with replacement?
`echo`	Show progress of the calculation.

Value

The Banzhaf-Owen Index for each player

References

Saavedra-Nieves, A., & Fiestras-Janeiro, M. G. (2021). Sampling methods to estimate the Banzhaf–Owen value. Annals of Operations Research, 301(1), 199-223.

`banzhaf_owen_exact` *Banzhaf-Owen Value*

Description

Calculate the approximated Banzhaf-Owen value

Usage

```
banzhaf_owen_exact(characteristic_func, union, n_players)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players
`n_players`	The number of players in the game.

Value

The Banzhaf Index for each player

`centroid` *Centroid of the core of the game*

Description

Calculate the centroid of core of the game if it exits.

Usage

```
centroid(
  characteristic_func,
  n_players = 0,
  method = "exact",
  n_rep = 1000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`method`	Method used to calculate the core. Valid methods are: `exact` for the exact calculation or `appro` for approximated core based on Camacho et al. (2025).
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The centroid of the core if it exists.

References

Camacho, J., Gonçalves-Dosantos, J. C., & Sánchez-Soriano, J. (2025). A Linear Programming Approach to Estimate the Core in Cooperative Games. arXiv preprint arXiv:2510.01766.

Examples

```
v <- c(2,3,5,5,7,8,10)
centroid(v, method = "exact")
centroid(v, method = "appro", n_rep = 100)

n <- 3
v <- function(coalition) {
 size <- length(coalition)
 if (size <= 1) {
   return(0)
 } else if (size == 2) {
   return(10)
 } else if (size == 3) {
   return(24)
 } else {
   return(0)
 }
}
centroid(v, n, method = "exact")
centroid(v, n, method = "appro", n_rep = 200)
```

`coalitions` *coalitions*

Description

Create all the possible coalitions given the number of players

Usage

```
coalitions(n_players)
```

Arguments

`n_players`	Number of players

Value

A list containing a `data.frame` of the binary representation of the coalitions and a `vector` of the classical representation (as sets) of the coalitions

`coreVertex` *Vertices of the core of the game*

Description

Calculate the vertices of core of the game if it exits.

Usage

```
coreVertex(
  characteristic_func,
  n_players = 0,
  method = "exact",
  n_rep = 1000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`method`	Method used to calculate the core. Valid methods are: `exact` for the exact calculation or `appro` for approximated core based on Camacho et al. (2025).
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The vertices of the core if it exists.

References

Camacho, J., Gonçalves-Dosantos, J. C., & Sánchez-Soriano, J. (2025). A Linear Programming Approach to Estimate the Core in Cooperative Games. arXiv preprint arXiv:2510.01766.

Examples

```
v <- c(2,3,5,5,7,8,10)
coreVertex(v, method = "exact")
coreVertex(v, method = "appro", n_rep = 100)

n <- 3
v <- function(coalition) {
 size <- length(coalition)
 if (size <= 1) {
   return(0)
 } else if (size == 2) {
   return(10)
 } else if (size == 3) {
   return(24)
 } else {
   return(0)
 }
}
coreVertex(v, n, method = "exact")
coreVertex(v, n, method = "appro", n_rep = 200)
```

`core_appro` *Approximated core of the game*

Description

Calculate the vertices of the core of the game following Camacho et al. (2025)

Usage

```
core_appro(characteristic_func, n_players = 0, n_rep = 1000, echo)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`n_rep`	The number of iterations to perform in the algorithm.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The vertices of the estimated core

References

Camacho, J., Gonçalves-Dosantos, J. C., & Sánchez-Soriano, J. (2025). A Linear Programming Approach to Estimate the Core in Cooperative Games. arXiv preprint arXiv:2510.01766.

`core_exact` *Exact core of the game*

Description

Calculate the vertices of core of the game.

Usage

```
core_exact(characteristic_func, n_players = 0)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.

Value

The vertices of the core

`egalitarian` *Egalitarian value*

Description

Calculate the egalitarian value

Usage

```
egalitarian(characteristic_func, n_players = 0)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.

Value

The egalitarian value for each player

Examples

```
n <- 10
v <- function(coalition) {
  if (length(coalition) > n/2) {
    return(1)
  } else {
    return(0)
  }
}
egalitarian(v,n)

v <- c(1,1,2,1,2,2,2)
egalitarian(v)
```

`egalitarian_unions` *Egalitarian value with a priori unions*

Description

Calculate the egalitarian value in games with a priori unions

Usage

```
egalitarian_unions(characteristic_func, union, n_players = 0)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.

Value

The egalitarian value for each player

Examples

```
n <- 10
v <- function(coalition) {
  if (length(coalition) > n/2) {
    return(1)
  } else {
    return(0)
  }
}
union <- list(1:4,5:n)
egalitarian_unions(v,union,n)

v <- c(1,1,2,1,2,2,2)
union <- list(c(1,2),c(3))
egalitarian_unions(v, union)
```

`equal_surplus_division` *Equal Surplus Division value*

Description

Calculate the equal surplus division value

Usage

```
equal_surplus_division(characteristic_func, n_players = 0)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.

Value

The equal surplus division value for each player

Examples

```
n <- 10
v <- function(coalition) {
  if (length(coalition) > n/2) {
    return(1)
  } else {
    return(0)
  }
}
equal_surplus_division(v,n)

v <- c(1,1,2,1,2,2,2)
equal_surplus_division(v)
```

equal_surplus_division_unions

Equal Surplus Division value with a priori unions

Description

Calculate the equal surplus division value in games with a priori unions

Usage

```
equal_surplus_division_unions(
  characteristic_func,
  union,
  n_players = 0,
  type = 1
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`type`	Number indicating the type of equal surplus division value to compute following Alonso-Meijide et al. (2020). Values 1, 2 and 3 are implemented.

Value

The equal surplus division value for each player

References

Alonso-Meijide, J. M., Costa, J., García-Jurado, I., & Gonçalves-Dosantos, J. C. (2020). On egalitarian values for cooperative games with a priori unions. Top, 28(3), 672-688.

Examples

```
n <- 3
v <- function(coalition) {
 size <- length(coalition)
 if (size <= 1) {
   return(0)
 } else if (size == 2) {
   return(10)
 } else if (size == 3) {
   return(24)
 } else {
   return(0)
 }
}
union <- list(1:4,5:n)
equal_surplus_division_unions(v,union,n,type = 1)

v <- c(1,1,2,1,2,2,2)
union <- list(c(1,2),c(3))
equal_surplus_division_unions(v, union, type = 2)
```

myerson *Myerson value*

Description

Calculate the Myerson value in a communication game.

Usage

```
myerson(
  characteristic_func,
  graph_edges,
  n_players = 0,
  method = "exact",
  n_rep = 10000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players. It can be provided as a vector or as a function.
`graph_edges`	Edges of the communication graph of the game. It must be a `list` of pairs indicating the connected players.

n_players	Only used if characteristic_func is a function. The number of players in the game.
method	Method used to calculate the Myerson value. Valid methods are: exact for the exact calculation or appro for approximated polynomial calculation based on sampling proposed.
n_rep	Only used if method is appro. The number of iterations to perform in the approximated calculation.
echo	Only used if method is appro. Show progress of the approximated calculation.

Value

The Myerson value for each player.

Examples

```
characteristic_func <- c(
1, 2, 0, 3,
3, 1, 4, 2, 5, 3,
3, 6, 4, 5,
15
)
graph_edges <- list(c(1, 2), c(2, 4))
myerson(characteristic_func, graph_edges, method = "exact")
myerson(characteristic_func, graph_edges, method = "appro", n_rep = 1000)

v <- function(S) {
  if (length(S) == 2) {
    return(1)
   }
   return(0)
}
n <- 3
graph_edges <- list(c(1, 2))
myerson(v, graph_edges, n_players = n, method = "exact")
myerson(v, graph_edges, n_players = n, method = "appro", n_rep = 2000)
```

---

myerson_unions *Myerson value with a priori unions*

Description

Calculate the Myerson value in a communication game with a priori unions.

Usage

```
myerson_unions(
  characteristic_func,
  n_players = 0,
  unions,
  graph_edges,
  method = "exact",
  n_rep = 10000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players. It can be provided as a vector or as a function.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`unions`	List of vectors indicating the a priori unions between the players.
`graph_edges`	Edges of the communication graph of the game. It must be a `list` of pairs indicating the connected players.
`method`	Method used to calculate the Myerson value. Valid methods are: `exact` for the exact calculation or `appro` for approximated polynomial calculation based on sampling proposed.
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The Myerson value for each player.

Examples

```
v <- c(
  0, 0, 0, 0,
  1, 1, 1, 0, 0, 0,
  1, 1, 1, 1,
  1
)
graph_edges <- list(c(2,3),c(3,1),c(1,4))
unions <- list(c(2,3),c(1),c(4))
myerson_unions(v, unions = unions, graph_edges = graph_edges, method = "exact")
myerson_unions(v, unions = unions, graph_edges = graph_edges, method = "appro", n_rep = 2000)
```

`owen` *Owen value*

Description

Calculate the Owen value

Usage

```
owen(
  characteristic_func,
  union,
  n_players = 0,
  method = "exact",
  n_rep = 10000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`union`	List of vectors indicating the a priori unions between the players.
`n_players`	The number of players in the game.
`method`	Method used to calculate the Owen value. Valid methods are: `exact` for the exact calculation or `appro` for approximated polynomial calculation based on sampling proposed by Saavedra-Nieves et al. (2018).
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The Owen value for each player.

References

Saavedra-Nieves, A., García-Jurado, I., & Fiestras-Janeiro, M. G. (2018). Estimation of the Owen value based on sampling. In The mathematics of the uncertain: A tribute to Pedro Gil (pp. 347-356). Cham: Springer International Publishing.

Examples

```
n <- 10
v <- function(coalition) {
  if (length(coalition) > n/2) {
    return(1)
  } else {
    return(0)
  }
}
u <- lapply(1:(n/2), function(i) c(2*i - 1, 2*i))
owen(v, union = u, method = "appro", n_rep = 4000, n_players = n)

characteristic_func <- c(1,1,2,1,2,2,2)
union <- list(c(1,2),c(3))
owen(characteristic_func, union)
owen(characteristic_func, union, method = "appro", n_rep = 4000)
```

---

owen_appro *Owen value (approximation)*

Description

Calculate the approximated Owen value based on sampling using the algorithm proposed by Saavedra-Nieves et al. (2018).

Usage

```
owen_appro(characteristic_func, union, n_players, n_rep, echo)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players
`n_players`	The number of players
`n_rep`	The number of iterations to perform in the approximated calculation
`echo`	Show progress of the calculation.

Value

The Owen value for each player

References

Saavedra-Nieves, A., García-Jurado, I., & Fiestras-Janeiro, M. G. (2018). Estimation of the Owen value based on sampling. In The mathematics of the uncertain: A tribute to Pedro Gil (pp. 347-356). Cham: Springer International Publishing.

`owen_exact` *Owen value (exact)*

Description

Calculate the exact Owen

Usage

```
owen_exact(characteristic_func, union, n_players = NULL)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`union`	List of vectors indicating the a priori unions between the players
`n_players`	The number of players

Value

The Owen value for each player

`predecessor` *Predecessor*

Description

Given a permutation 0 of players and a player i, calculate the set of predecessors of the player i in the order 0

Usage

```
predecessor(permutation, player, include_player = FALSE)
```

Arguments

`permutation`	A permutation of the players
`player`	Number of the player i
`include_player`	Whether the player i is included as predecessor of itself or not

Value

The set of predecessors of the player i in the order 0 `shapley` *Shapley value*

Description

Calculate the Shapley value

Usage

```
shapley(
  characteristic_func,
  n_players = 0,
  method = "exact",
  n_rep = 10000,
  echo = TRUE
)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players.
`n_players`	Only used if `characteristic_func` is a `function`. The number of players in the game.
`method`	Method used to alculate the Shapley value. Valid methods are: `exact` for the exact calculation or `appro` for approximated polynomial calculation based on sampling proposed by Castro et al. (2009).
`n_rep`	Only used if `method` is `appro`. The number of iterations to perform in the approximated calculation.
`echo`	Only used if `method` is `appro`. Show progress of the approximated calculation.

Value

The Shapley value for each player.

References

Castro, J., Gómez, D., & Tejada, J. (2009). Polynomial calculation of the Shapley value based on sampling. Computers & operations research, 36(5), 1726-1730.

Examples

```
n <- 10
v <- function(coalition) {
if (length(coalition) > n/2) {
   return(1)
 } else {
   return(0)
```

```
  }
 }
 shapley(v, method = "appro", n_rep = 4000, n_players = n)

 n <- 3
 v <- c(1,1,2,1,2,2,2)
 shapley(v, method = "exact")
 shapley(v, method = "appro", n_rep = 4000)
```

shapley_appro *Shapley value (approximation)*

Description

Calculate the approximated Shapley value based on sampling using the algorithm proposed by Castro et al. (2009).

Usage

```
shapley_appro(characteristic_func, n_players, n_rep, echo)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	The number of players
`n_rep`	The number of iterations to perform in the approximated calculation
`echo`	Show progress of the calculation.

Value

The Shapley value for each player

References

Castro, J., Gómez, D., & Tejada, J. (2009). Polynomial calculation of the Shapley value based on sampling. Computers & operations research, 36(5), 1726-1730.

`shapley_exact` *Shapley value (exact)*

Description

Calculate the exact Shapley value

Usage

```
shapley_exact(characteristic_func, n_players)
```

Arguments

`characteristic_func`	The valued function defined on the subsets of the number of players
`n_players`	The number of players

Value

The Shapley value for each player

Index