Programas informáticos orientados a juegos TU

David Mirás Calvo

Trabajo tutelado:	Programas informáticos orientados a juegos TU
Autor:	David Mirás Calvo
Curso de doctorado:	2145-03 Estatística e investigación de operacións
Bienio:	2003-2005

Directores:

Miguel Ángel Mirás Calvo
Departamento de Matemáticas

Estela Sánchez Rodríguez
Departamento de Estadística e I.O.

Universidad de Vigo

Septiembre, 2008

Índice general

Introducción

Podríamos definir a la teoría de juegos como una disciplina matemática que estudia situaciones de conflicto de intereses en las que intervienen dos o más jugadores. La finalidad de la teoría de juegos es investigar de qué modo los individuos racionales deberían comportarse cuando sus intereses se contraponen. La competencia entre líneas aéreas, las coaliciones formadas con ánimo de ejercer presiones políticas, la elección de emplazamientos industriales, las fusiones de empresas, son ejemplos de situaciones en las que sería posible aplicar esta teoría. La teoría de juegos también se utiliza, o se ha utilizado, para optimizar precios, para tomar decisiones en inversiones, para designar jurados o para distribuir los tanques en una batalla. A cada juego se le busca una solución, es decir, una descripción de lo que cada jugador debería hacer y de cuál sería, en consecuencia, el resultado de sus acciones. Las bases de la moderna teoría de juegos fueron sentadas con la publicación en 1944 del libro *The theory of games and economic behavior* (*Teoría de los juegos y del comportamiento económico*), obra del matemático John von Neumann y del economista Oskar Morgenstern.

En esta memoria nos centraremos en los juegos cooperativos con utilidad transferible, concretamente, en los llamados juegos coalicionales. Básicamente, un juego coalicional de n jugadores viene determinado por una función, denominada función característica, que asigna a cada posible coalición de jugadores la utilidad (o beneficio) que sus miembros obtendrían en el juego si cooperaran entre ellos sin contar con los que no forman parte del grupo. El objetivo fundamental de la teoría es proponer métodos de reparto (una solución) del beneficio global de la cooperación entre los jugadores. Descontando la coalición vacía, a la que siempre asignaremos el valor nulo, para describir la función característica de un juego de n jugadores necesitamos $2^n - 1$ valores. Esto significa que, para juegos de más de 4 jugadores es imprescindible la ayuda de una computadora para realizar los cálculos de las soluciones. Así, por ejemplo, Littlechild y Thompson (1977) analizaron el problema de las tasas de utilización de las pistas en el aeropuerto de Birmingham en el periodo 1968-1969 aplicables a las distintas compañías en función del tipo de aviones utilizados (esta clase de juegos han dado en llamarse "juegos del aeropuerto"), modelándolo como un juego de 11 jugadores (correspondientes a 11 tipos de aeronaves).

Consecuentemente, la necesidad de la utilización de herramientas informáticas para la resolución de juegos que deriven de situaciones reales es manifiesta. Así, Jean Derks ha desarrollado un paquete específico orientado al cálculo del nucleolo. Pero incluso para juegos de 3 y 4 jugadores hay aspectos geométricos y computacionales para los que la ayuda de un ordenador se antoja conveniente. El docente y el investigador, por ejemplo, necesitan a menudo proponer ejemplos que ilustren un concepto y, para ello, la posibilidad de "ver" lo que está ocurriendo es una ventaja nada desdeñable. Precisamente, resaltar

los aspectos geométricos de los juegos de 3 y 4 jugadores (los únicos para los que una representación gráfica es posible) es la motivación esencial del programa TUGlab, véase Mirás y Sánchez (2008), y la del paquete **tugames1** que analizaremos en la Sección 2.4.

El objetivo central de este trabajo es extender el paquete TUGlab para abarcar juegos con un número indeterminado de jugadores. Denominaremos TUGlabExtended a esta ampliación. Naturalmente, la versión ampliada de TUGlab deberá renunciar de antemano a la orientación geométrica original y centrarse en los aspectos puramente computacionales. En todo caso, el carácter genérico de esta extensión, es decir, la pretensión de que pueda ser utilizada para analizar cualquier juego con cualquier número de jugadores, impone una serie de limitaciones insoslayables. Pongamos un ejemplo. Dado un juego estrictamente convexo de 11 jugadores, para guardar la información relativa a los vectores de contribuciones marginales se requeriría una matriz numérica con 11 columnas y... ¡casi 40 millones de filas!, algo más de 3 giga-bytes de información. Ciertamente, la universalidad del programa implica su "lentitud" y su "despilfarro" de memoria frente a aplicaciones específicas para determinadas clases de juegos (como las empleadas por Littlechild y Thompson (1977) en su análisis del juego del aeropuerto). Habrá muchos juegos específicos para los que nuestro programa colapse o para los que tarde demasiado en calcular una solución concreta. No puede esperarse que nuestra herramienta sea una panacea para los juegos TU. Al contrario, nos contentaríamos con que sirviese como un elemento de contraste para un primer análisis en problemas aplicados o como un instrumento útil para analizar ejemplos complejos que puedan surgir en el desempeño docente o investigador.

En la presente memoria vamos, en primer lugar, a introducir el método utilizado en TUGlabExtended para el tratamiento de la función característica. Expondremos algunas de las definiciones y propiedades de los juegos TU y su manejo con este nuevo paquete de MATLAB. Trataremos algunos aspectos de las soluciones y principalmente trabajaremos con la solución puntual más utilizada: la regla de reparto dada por el valor de Shapley. Derivaremos una expresión matricial para su cálculo que nos permitirá utilizar toda la potencia del cálculo matricial de MATLAB. En el capítulo 2 presentamos una guía de las nuevas órdenes de MATLAB que hemos definido en TUGlabExtended y repasaremos algunas de las aplicaciones informáticas existentes orientadas a la teoría de juegos.

Capítulo 1

Programación de juegos TU en MATLAB

A lo largo del presente trabajo vamos a tratar con juegos cooperativos con utilidad transferible (juegos TU). El objetivo fundamental es desarrollar una aplicación informática dirigida al cálculo y la interpretación de los principales aspectos relacionados con estos juegos. Para ello ampliaremos el paquete TUGlab, véase Mirás y Sánchez (2008), que originalmente estaba orientado a la docencia y que pretende destacar las implicaciones geométricas de los juegos de 3 y 4 jugadores, y disponer así de un programa que nos permita trabajar con un número de jugadores en principio indeterminado. Al igual que TUGlab, la mencionada ampliación se programará en MATLAB®.[1] Veremos más adelante que ciertos problemas computacionales limitarán en la práctica el número efectivo de jugadores. En general, podemos afirmar que el programa TUGlabExtended ejecutará los cálculos en un tiempo "razonable" para juegos de hasta 10 jugadores.

Comencemos por destacar algunos aspectos generales del uso de MATLAB:

1. MATLAB es un programa que distingue en sus órdenes entre mayúsculas y minúsculas, por este motivo se debe poner mucho cuidado en la sintaxis de los comandos.

2. MATLAB no está disponible en castellano, por este motivo hemos nombrado los comandos y las funciones de TUGlabExtended directamente en inglés.

3. La nomenclatura de los comandos de TUGlabExtended es similar a la de TUGlab. En la mayoría de las órdenes simplemente hemos suprimido la palabra "game" del nombre original en TUGlab y añadido la letra "E" al final para formar el nombre en TUGlabExtended. así, la sentencia **convex** de TUGlab, que nos permite averiguar si un juego es convexo o no, tiene su equivalente **convexE** en TUGlabExtended.

4. Para trabajar con TUGlabExtended basta con tener las órdenes en una carpeta y direccionar el directorio de MATLAB hacia ella, escribiendo en la ventana de comandos las órdenes como se presentan en esta memoria.

El material íntegro que configura el paquete de TUGlabExtended estará disponible, en su versión más actual, en la página web del grupo SaGaTh (Santiago Game Theory

[1] MATLAB® es una marca registrada de The MathWorks, Inc.

Group), el grupo de teoría de juegos del Departamento de Estadística e Investigación Operativa de la Universidad de Santiago de Compostela.

`http://eio.usc.es/pub/io/xogos`.

Dado que esta es la primera versión de TUGlabExtended, la mayoría de las funciones no han sido comprobadas exhaustivamente. Agradecemos de antemano cualquier información acerca de un error de funcionamiento así como las sugerencias y comentarios que el usuario quiera enviarnos.

1.1. Codificación de la función característica

Básicamente, los juegos TU vienen determinados por una función escalar v, la función característica, que asigna a cada coalición S (un subconjunto del conjunto finito N de jugadores) un valor $v(S)$. Formalmente:

Definición 1.1 *Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica*

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

Una decisión importante a la hora de hacer programas informáticos para trabajar con estos juegos es decidir "como etiquetar" las coaliciones, esto es, como introducir la función característica v. En TUGlab, dado que sólo se trabajaba con juegos de 3 y 4 jugadores, se escogió una opción intuitiva: ordenar lexicográficamente las coaliciones. Así, para 3 jugadores, las coaliciones estarían ordenadas de la forma:

$$\{1\} \quad \{2\} \quad \{3\} \quad \{1,2\} \quad \{1,3\} \quad \{2,3\} \quad \{1,2,3\}$$

Consecuentemente, la función v vendría dada por un vector con 7 coordenadas, de modo que, por ejemplo, si la quinta coordenada de v fuese 34 entonces, como la quinta coalición en el orden es $\{1,3\}$, tendríamos $v(\{1,3\}) = 34$.

Ahora bien, si uno pretende ir un poco más allá de 4 jugadores la elección del orden lexicográfico no parece la más adecuada. Una posibilidad, que ya hemos visto en otros programas (es la empleada por Jean Derks en su paquete para el cálculo del prenucleolo), es utilizar un sistema de codificación binario. Dado un juego con n jugadores, asociaremos a cada coalición S un número en base 2 de n cifras, obtenido siguiendo el siguiente método: la primera cifra será 0 si el jugador 1 no pertenece a la coalición S y 1 si pertenece; la segunda cifra puede ser de nuevo 1 o 0, si el jugador 2 pertenece o no a S; y así sucesivamente hasta completar los n jugadores. Por ejemplo, la coalición $S = \{1,3\}$ vendría representada por la expansión binaria 101; la coalición $S = \{3\}$ por 001, etc. Ahora, calculando el número natural correspondiente a cada expansión binaria tenemos una correspondencia biunívoca entre coaliciones y números naturales. Así, por ejemplo, en el caso de 3 jugadores,

natural	**binario**	**coalición**
1	100	$\{1\}$
2	010	$\{2\}$
3	110	$\{1,2\}$
4	001	$\{3\}$
5	101	$\{1,3\}$
6	011	$\{2,3\}$
7	111	$\{1,2,3\}$

Obviamente, la función característica v depende del método escogido para ordenar las coaliciones. Para 3 jugadores, la tercera componente de una función característica v que siga la ordenación lexicográfica se corresponde con $v(\{3\})$ mientras que si sigue la ordenación binaria se correspondería con $v(\{1,2\})$.

Destaquemos algunas de las ventajas del método binario frente al lexicográfico.

1. La correspondencia entre números naturales (coordenadas) y coaliciones es muy fácil de calcular, incluso para muchos jugadores.

2. El natural asociado a cada coalición es independiente del número de jugadores.

3. Como veremos a continuación, las operaciones conjuntistas entre coaliciones (intersección, unión, complementario) son fácilmente trasladables a operaciones numéricas.

Como ya hemos indicado, el programa TUGlab exige que la función característica esté introducida siguiendo el orden lexicográfico. Ello es debido a que TUGlab es un programa orientado a destacar los aspectos geométricos de los juegos TU y, por consiguiente, está restringido a juegos de 3 y 4 jugadores. Para estos juegos, quizás, el orden lexicográfico resulte más intuitivo. Pero para un número de jugadores superior plantea problemas.

Una primera aproximación nos llevaría a pensar en diseñar un método para traducir de un sistema al otro (en las dos direcciones). Para el caso de 3 y 4 jugadores, esta traducción es sencilla y viene dada por las funciones **bin2lex** y **lex2bin**, que actúan del siguiente modo:

Ejemplo 1.2 *Si escribimos,*

```
>> bin2lex([1 0 3 4 3 2 1])
ans=
   1     0     4     3     3     2     1
```

obtenemos reordenada siguiendo el orden binario la misma función característica que hemos introducido pero siguiendo el orden lexicográfico. Por lo tanto, en el otro sentido, si tecleamos

```
>> lex2bin([1     0     4     3     3     2     1])
ans=
     1     0     3     4     3     2     1
```

recuperamos la función característica en el formato lexicográfico original.

Las ventajas del orden binario son muchísimas, pues tenemos una biyección entre los números naturales y las coaliciones. Esta biyección viene dada por la expresión en binario de cualquier número natural, siendo independiente del número de jugadores y abriendo la posibilidad de utilizar las operaciones binarias que MATLAB posee. Un ejemplo de estos comandos sería la orden **bitget**$(\mathbf{n}, \mathbf{1} : \mathbf{m})$ que devuelve la expansión binaria del natural n hasta el m-ésimo lugar.

Ejemplo 1.3 *La respuesta obtenida al teclear*

```
>> bitget(7,1:8)
ans=
     1     1     1     0     0     0     0     0
```

se corresponde con la expresión binaria del número 7 hasta la octava cifra.

La biyección entre naturales y coaliciones queda descrita en TUGlabExtended a través de las funciones **getcoalition** y **getcoalitionnumber**. Vemos a continuación un par de ejemplos que ilustran su funcionamiento.

Ejemplo 1.4 *Queremos conocer cual es la coalición que se corresponde con el número* 9*. Para ello tecleamos,*

```
>>  [a,b]=getcoalition(9)
a =
14
b =
     1     4
```

Observamos que la coalición buscada es la formada por los jugadores 1 *y* 4*. La función nos devuelve la misma respuesta escrita en dos formatos diferentes: la primera como una cadena de caracteres, la segunda como una matriz numérica.*

Ejemplo 1.5 *Si queremos determinar cual es el número natural asociado a la coalición* $\{1, 3, 4\}$ *escribiremos*

```
>> getcoalitionnumber([1 3 4])
ans=
    13
```

Se han incluido en TUGlabExtended otras funciones para ayudar al usuario a introducir la función característica y obtener información básica acerca de ella. Así tenemos **characteristicfunction**, **characteristicdata** y **getplayers**.

Ejemplo 1.6 *Siendo precisos* **characteristicfunction** *no es propiamente una función de MATLAB sino un guión (un ejecutable). Está diseñado para ver en la pantalla la biyección completa entre los naturales y las coaliciones de modo que el usuario disponga de una plantilla a la hora de introducir los datos de la función característica. Por ejemplo, para 4 jugadores, al ejecutar* **characteristicfunction** *observaremos que el programa nos pide el dato del número de jugadores del juego. Una vez proporcionado este valor, aparecerá en la ventana de comandos de MATLAB la correspondencia entre números y coaliciones.*

```
>> characteristicfunction
Number of players: 4
n =
     4

ans =
 1-1
 2-2
 3-12
 4-3
 5-13
 6-23
 7-123
 8-4
 9-14
10-24
11-124
12-34
13-134
14-234
15-1234
```

Ejemplo 1.7 *Tecleamos en la ventana de comandos*

```
>> [n,S]=characteristicdata([1 2 1 3 4 1 2 2 1 2 1 3 4 1 5])

n =
     4
S =
    15
```

Es decir, el número de jugadores del juego introducido es 4 y el número de coaliciones 15 (siempre prescindimos de la coalición vacía).

Ejemplo 1.8 *Si escribimos*

```
>>  [P,S]=getplayers(45)
P =
     1     4     8    32
S =
     1     3     4     6
```

Obtenemos que la coalición correspondiente al número 45 está formada por los jugadores de S que, a su vez, se corresponden con las coaliciones unitarias representadas por los números dados en P.

Por lo tanto resulta indiferente hablar de números naturales o de coaliciones, pues hemos encontrado una biyección entre ellos a través de la expresión binaria. El siguiente

paso es establecer las operaciones de la unión e intersección de coaliciones. Supongamos que tenemos las coaliciones $S = \{2, 3, 4\}$ y $T = \{1, 2, 3, 5\}$. Lógicamente se trata de un juego de, al menos, 5 jugadores en el cual tiene sentido hablar de la coalición $S \cup T = \{1, 2, 3, 4, 5\}$ y de la coalición $S \cap T = \{2, 3\}$. Asociadas a las coaliciones S y T tenemos los números naturales 14 y 23, respectivamente. Las operaciones de la unión y de la intersección definen dos nuevas operaciones en el conjunto de los números naturales $\mathbb{N}$. Así, en nuestro ejemplo, podemos escribir $14 \cup 23 = 31$ y $14 \cap 23 = 6$, que son los naturales asociados a las coaliciones correspondientes. Estos resultados son los obtenidos al aplicar las operaciones del OR lógico y del AND lógico a las expresiones binarias de los números naturales. Así el $14 = 01110_2$ y el $23 = 11101_2$.

14	23	OR	AND
0	1	1	0
1	1	1	1
1	1	1	1
1	0	1	0
0	1	1	0

$11111_2 = 31$ y $01100_2 = 6$

De este modo es muy sencillo calcular las intersecciones y uniones de coaliciones, o bien comprobar contenidos, necesarias para las definiciones y propiedades posteriores de los juegos TU, utilizando las sentencias de MATLAB que efectúan las operaciones lógicas (OR y AND) con las expresiones binarias de los números naturales (operaciones BIT): **bitand** Y **bitor**. En resumen, la unión $S \cup T$ del ejemplo precedente se calcularía en MATLAB con el comando **bitor(14,23)** y la intersección $S \cap T$ con la orden **bitand(14,23)**.

Una vez elegido el método utilizado en el TUGlabExtended para el tratamiento de la función característica expondremos algunas de las definiciones y propiedades de los juegos TU y su manejo con este nuevo paquete de MATLAB. Trataremos algunos aspectos de las soluciones y trabajaremos principalmente con la solución puntual más conocida, el valor de Shapley, para la que proporcionaremos una expresión matricial para su cálculo.

1.2. Definiciones y comandos básicos en TUGlabExtended

En este sección presentamos las órdenes básicas de TUGlabExtended relacionándolas con los correspondientes conceptos de la teoría de juegos TU y mostrando ejemplos que ilustran su funcionamiento más elemental. Posteriormente, en el Capítulo 2 analizaremos con más detalle cada una de estas funciones.

Definición 1.9 *Sea (N,v) un juego TU. Se dice que (N, V) es superaditivo si*

$$v(S \cup T) \geq v(S) + v(T), \text{, para todo } S, T \in 2^N, S \cap T = \emptyset.$$

La función para comprobar si un juego es superaditivo se denomina **superadditiveE**. Las funciones de TUGlabExtended que sirven para comprobar si una propiedad se cumple o no seguirán siempre el mismo criterio: devolverán el valor 0 si la propiedad falla y un 1 si se cumple.

Ejemplo 1.10 *Tecleemos en la ventana de comandos*

```
>> A=[1 2 2 3 4 5 6];[respuesta,info]=superadditiveE(A)
respuesta =
     0
info =
     1     2
```

Así pues, el juego no es superaditivo y las coaliciones 1 y 2 son un caso en el que falla la definición. En efecto $v(1,2) = 2$ *que es menor que* $v(1) + v(2) = 1 + 2 = 3$.

En TUGlabExtended, dado el orden binario establecido para las componentes de la función característica, resulta muy sencillo establecer las condiciones de unión e intersección de coaliciones, pues, como ya hemos visto, equivalen a las operaciones OR y AND de las expresiones binarias asociadas. Analicemos, a modo de ejemplo del método de programación utilizado, el código completo de la función **superadditiveE**.

```
function [respuesta,info]=superadditiveE(v)
% la sintaxis de la función en MATLAB

[n,nC]=characteristicdata(v);
% Calculamos el  número de jugadores y de coaliciones.
for ii=1:nC
% Para toda coalición S (el natural ii)
    for jj=1:nC
% Para toda coalición T (el natural jj)
        if bitand(ii,jj)==0
%  Si  S y T son disjuntos.
            if v(bitor(ii,jj)) < (v(ii)+v(jj))
% Si  v(S U T) < v(S)+ v(T)
                respuesta=0;info=[ii jj];return
% No se verifica la condición para las coaliciones S y T.
            end
        end
    end
end
respuesta=1;info=[];
% En caso contrario el juego es superaditivo.
```

La técnica de programación es similar para las demás funciones de TUGlabExtended.

Definición 1.11 *Un juego (N,v) se dice que es 0-normalizado si* $v(i) = 0$*, para todo* $i \in N$*. El juego se dirá que es 0-1-normalizado si* $v(i) = 0$*, para todo* $i \in N$ *y* $v(N) = 1$.

Dado un juego (N, v) se puede obtener su 0-normalización, (N, w), haciendo para cada $S \subset N$, $w(S) = v(S) - \sum_{i \in S} v(i)$. La 0-1 normalización de un juego (N, v) es el juego dado por $u(S) = kv(S) + \sum_{i \in S} \alpha_i$ donde $k = \dfrac{1}{v(N) - \sum_{i \in N} v(i)}$ y $\alpha_i = -kv(i)$, $i \in N$. Se tiene que la 0-1 normalización existe si y sólo si $v(N) - \sum_{i \in N} v(i) \neq 0$.

En TUGlabExtended la nueva función que permite calcular la 0-normalización y la 0-1-normalización de un juego TU es **normalizedE** y un ejemplo de como se puede utilizar este comando es el siguiente:

Ejemplo 1.12 *Tecleamos*

```
>> A=[2 1 9 0 2 1 6];[v0,v1]=normalizeE(A)
v0 =
     0     0     6     0     0     0     3
v1 =
     0     0     2     0     0     0     1
```

Luego, $v0$ es el vector de la función característica de la 0-normalización y $v1$ el de la 0-1 normalización.

Dado un juego TU podemos asociarle el juego dual que es aquel que a cada coalición le hace corresponder lo que queda del valor de la gran coalición si satisfacemos a la coalición complementaria.

Definición 1.13 *El juego dual (N, w) de juego (N, v) viene dado por $w(S) = v(N) - v(N \setminus S)$, para cada coalición $S \subset N$.*

Naturalmente, la orden en TUGlabExtended que permite calcular el dual de un juego dado es **dualE**.

Ejemplo 1.14 *Tecleamos*

```
>> A=[0 0 2 0 4 4 23];
>> dualE(A)
ans =
    19    19    23    21    23    23    23
```

Es decir, w es el vector de la función característica del juego dual de A.

Otras definiciones importantes son las de monotonía y convexidad que describimos a continuación.

Definición 1.15 *Se dice que un juego (N, v) es monótono cuando se verifica que $v(S) \leq v(T)$ para todas las coaliciones $S, T \in 2^N$ tales que $S \subset T$.*

Definición 1.16 *Se dice que un juego (N, v) es convexo cuando se verifica que*

$$v(S \cup T) + v(S \cap T) \geq v(S) + v(T)$$

para todo par de coaliciones $S, T \in 2^N$.

La utilización de TUGlabExtended para la comprobación de la monotonía y convexidad de un juego TU se basa en dos nuevas funciones **monotonicE** y **convexE**. Veamos ahora un par de ejemplos para entender el sencillo funcionamiento de estas funciones.

Ejemplo 1.17 *En la ventana de comandos introducimos la función característica y la orden correspondiente*

```
>> A=[0 0 2 0 5 5 25];monotonicE(A)
ans =
     1
```

Lo que garantiza la monotonía de nuestro juego.

Ejemplo 1.18 *Si introducimos*

```
>> [respuesta,info]=convexE([0 0 0 0 0 0 0 0 5 0 6 5 5 10])
respuesta =
     0
info =
     1    10
```

vemos que el juego no es convexo y un caso en el que falla la condición de convexidad viene dado por las coaliciones correspondientes a los naturales 1 y 10. En efecto, utilizando la orden **getcoalition(10)** *el programa nos dice que ésta es la coalición formada por los jugadores* $\{2,4\}$*. Además,* $S \cup T = \{1,2,4\}$ *y* $S \cap T = \emptyset$*. Utilizando* **getcoalitionnumber([1 2 4])** *obtenemos que el número asociado es 11. Así, la condición de convexidad queda escrita como* $v(\{1,2,4\}) \geq v(\{1\}) + v(\{2,4\})$*, es decir,* $v(11) \geq v(1) + v(10)$ *que claramente es falsa, pues* $v(11) = v(1) = 0$ *y* $v(10) = 5$*.*

La condición de convexidad de un juego se puede formular de otro modo. Dada una coalición S y un jugador i, que no pertenezca a dicha coalición, el incremento $v(S \cup \{i\}) - v(S)$ mide la contribución de i a S. La convexidad implica que la contribución de un jugador a una coalición no decrece cuantos más jugadores tenga dicha coalición. Se tiene el siguiente resultado:

Proposición 1.19 *Un juego* (N, v) *es convexo si, y sólo si,*

$$v(S \cup \{i\}) - v(S) \geq v(T \cup \{i\}) - v(T)$$

para todo par de coaliciones $S, T \in 2^N$ *tales que* $T \subset S \subset N \setminus \{i\}$*.*

Definición 1.20 *Se dice que un juego* (N, v) *es estrictamente convexo cuando se verifica que*

$$v(S \cup T) + v(S \cap T) > v(S) + v(T)$$

para todo par de coaliciones $S, T \in 2^N$*, o equivalentemente,*

$$v(S \cup \{i\}) - v(S) > v(T \cup \{i\}) - v(T)$$

para todo par de coaliciones $S, T \in 2^N$ *tales que* $T \subset S \subset N \setminus \{i\}$*.*

El objetivo fundamental de los juegos TU es que se forme la gran coalición N, y que los jugadores se repartan el beneficio de la cooperación. Así, el problema radica en repartir el valor $v(N)$ del juego entre los N jugadores. Un reparto es simplemente un vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, en donde cada componente x_i representa la cantidad asignada al jugador i. Dado un reparto $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, la suma de las cantidades asignadas a una coalición $S \subset N$ se denotará por $x(S) = \sum_{i \in S} x_i$.

De entre todos los posibles repartos nos interesan aquellos que satisfagan:

- Racionalidad individual. Un reparto $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ satisface la propiedad de racionalidad individual si cada jugador recibe un pago que no es inferior a lo que puede garantizarse por sí mismo, es decir, $x_i \geq v(i)$ para todo $i \in \mathbb{N}$.
- Eficiencia. Un reparto $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ satisface la propiedad de eficiencia si distribuye el valor de la gran coalición $v(N)$ entre los jugadores, esto es, si $v(N) = x_1 + \cdots + x_n$.

Los repartos eficientes que son individualmente racionales se denominan imputaciones.

Definición 1.21 *Sea (N, v) un juego TU.*

- *Se define el conjunto de preimputaciones del juego (N, v) como el conjunto de todos los repartos eficientes.*

$$I^*(N, v) = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = v(N)\}.$$

- *El conjunto de imputaciones de un juego (N, v) está formado por los repartos eficientes que verifiquen además la propiedad de racionalidad individual.*

$$I(N, v) = \{x = (x_1, \ldots, x_n) \in I^*(N, v) : x_i \geq v(\{i\}), \text{ para todo } i \in N\}.$$

Un juego se dirá esencial si el conjunto de imputaciones es no vacío.

Definición 1.22 *Un juego (N, v) es esencial si*

$$\sum_{i=1}^{n} v(i) \leq v(N)$$

y es degenerado si

$$\sum_{i=1}^{n} v(i) = v(N).$$

La función de TUGlabExtended que sirve para comprobar si un juego es esencial o degenerado es **essentialE**.

Ejemplo 1.23 *El juego definido por el vector v es esencial no degenerado*

```
>> v=[0 0 3 1 0 3 4 1 1 1 3 4 5 1 6];
>> [E,D]=essentialE(v)
E=
      1
D=
      0
```

Si un juego es esencial degenerado, el conjunto de imputaciones es unitario, mientras que si es esencial no degenerado el conjunto de imputaciones es un simplex cuyos vértices podemos determinar con la orden **imputationverticesE**.

Ejemplo 1.24 *E conjunto de imputaciones del siguiente juego es un triángulo cuyos vértices son los vectores de las filas de V.*

```
>> V=imputationverticesE([0 0.5 3 1 1 1 2])
V =
    0.5000    0.5000    1.0000
         0    0.5000    1.0000
         0    0.5000    0.5000
```

Pasamos ahora a analizar y a tratar con TUGlabExtended algunas de las soluciones tipo conjunto y puntuales para la resolución de los juegos TU. Las soluciones tipo conjunto establecen criterios para eliminar del conjunto de imputaciones vectores de pagos mientras que las soluciones puntuales establecen criterios o propiedades básicas que deben satisfacer las soluciones y que garantizan existencia y unicidad de solución en la clase general de juegos o en subclases determinadas. Formalmente la definición de solución de juego cooperativo es la siguiente:

Definición 1.25 *Una solución definida en algún subdominio $\Omega \subset \mathbb{R}^n$, es una correspondencia Ψ, que asocia a cada juego (N, v), un subconjunto*

$$\Psi(N,v) \subset I^*(N,v) = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = v(N)\}$$

.

$$\Psi \colon \Omega \subset G^n \to \mathbb{R}^n$$

En el caso de que la solución sea siempre unitaria hablamos de solución puntual o regla de reparto. En otro caso, se trata de una solución tipo conjunto.

Definición 1.26 *El núcleo de un juego (N, V) es el conjunto:*

$$C(N,v) = \{x \in I(N,v) : \sum_{i \in S} x_i \geq v(S) \text{ para todo } S \subset N, S \neq \emptyset\}.$$

En el TUGlabExtended la función que nos dice si un reparto está o no está en el núcleo es **belong2coreE**.

Ejemplo 1.27 *Para saber si un reparto está o no en el núcleo debemos introducir en la ventana de comandos la función característica y el reparto:*

```
>>  v=[0 0 3 0 2 2 8];x=[1 2 8];[R,info]=belong2coreE(v,x)
R =
     0
info =
     7
```

Lo cual quiere decir que el reparto x no está en el núcleo pues no se cumple la correspondiente propiedad del núcleo para la coalición 7 que, en este caso, resulta ser la condición de eficiencia. ciertamente, $v(N) = 8$ y $\sum_{i \in N} x_i = 11$.

Tomemos ahora un reparto eficiente y comprobemos si pertenece al núcleo.

```
>>  y=[1 2 5];[R,info]=belong2coreE(v,y)
R =
     1
info =
     []
```

Luego, el reparto y está en el núcleo.

Definición 1.28 *Sea $\sigma \colon \mathbb{N} \to \mathbb{N}$ una permutación de N. Se define el vector de contribuciones marginales asociado al orden σ, $m^\sigma(N, V) = (m_i^\sigma(v))_{i \in N} \in \mathbb{R}^n$, como*

$$m_i^\sigma(v) = v(Pre_\sigma(i) \cup \{i\}) - v(Pre_\sigma(i))$$

donde $Pre_\sigma(i) = \{j \in N : \sigma(j) < \sigma(i)\}$, es decir el conjunto de jugadores que preceden al jugador i en el orden σ.

A la hora de programar en MATLAB funciones que calculen las contribuciones marginales, hay que tener en cuenta el cálculo y la ordenación de las permutaciones de los n jugadores. El programa dispone de una sentencia **Perms(1:n)**, o bien **Perms(v)** siendo v un vector de n componentes. En la programación de TUGlabExtended utilizaremos una nueva función **permutationE**, que es una variante de la equivalente programada por Jean Derks y que difiere de **Perms** en el orden en que se generan las permutaciones. Nuestra ordenación se obtiene aplicando el siguiente criterio: Primero se ordenan las permutaciones que tienen el 1 en la posición 1; luego las que tienen el 1 en la posición 2, etc. Dentro de cada bloque de 1's, primero van las que tienen el 2 en la posición 2, luego el 2 en la posición 3, etc. y así sucesivamente. Ilustremos el procedimiento con un ejemplo.

Ejemplo 1.29 *Generemos, en primer lugar, todas las permutaciones de 4 elementos en el orden descrito arriba. Para ello tecleamos:*

```
>> A=[];for ii=1:factorial(4);A=[A;permutationE(4,ii)];end;A
A =
     1     2     3     4
     1     2     4     3
     1     3     2     4
     1     4     2     3
     1     3     4     2
```

```
     1     4     3     2
     2     1     3     4
     2     1     4     3
     3     1     2     4
     4     1     2     3
     3     1     4     2
     4     1     3     2
     2     3     1     4
     2     4     1     3
     3     2     1     4
     4     2     1     3
     3     4     1     2
     4     3     1     2
     2     3     4     1
     2     4     3     1
     3     2     4     1
     4     2     3     1
     3     4     2     1
     4     3     2     1
```

Sin necesidad de generar todas las permutaciones, para encontrar la que ocupa el lugar 6 de entre todas las posibles para 4 jugadores, basta con escribir

```
>> permutationE(4,6)
ans =
     1     4     3     2
```

Al trabajar en MATLAB con permutaciones existe una limitación de capacidad muy clara. El problema radica en que para $n = 11$ hay $11! = 39{,}916{,}800$ permutaciones, es decir del orden de 40 millones. Ello implica una cantidad de memoria enorme para almacenar la correspondiente matriz de orden $11 \times 11!$. Si pedimos la ayuda de la sentencia **Perms** en MATLAB, leemos:

```
>> help perms

 PERMS  All possible permutations.
    PERMS(1:N), or PERMS(V) where V is a vector of length N, creates a
    matrix with N! rows and N columns containing all possible
    permutations of the N elements.

    This function is only practical for situations where N is less
    than about 10 (for N=11, the output takes over 3 giga-bytes).
```

De lo que se deduce que todos los cálculos que exijan manejar todas las $n!$ permutaciones van a funcionar bien hasta un máximo de 10 jugadores. Con esta limitación en mente presentamos dos nuevas funciones relacionadas con las contribuciones marginales: **marginalvectorE** y **allmarginalsE**.

Ejemplo 1.30 *Introduzcamos la función característica y la orden correspondiente:*

```
>> A=[ 0 0 7 0 6 6 12];allmarginalsE(A)
ans =
     0     7     5
     0     6     6
     7     0     5
     6     6     0
     6     0     6
     6     6     0
```

Obtenemos todos los vectores de contribuciones marginales asociados a las permutaciones de 3 jugadores. Si tecleamos

```
>> marginalvectorE(A,3)
ans =
     7     0     5
```

la respuesta dada es el vector de contribuciones marginales asociado a la tercera permutación.

Definimos a continuación los juegos de unanimidad, véase Shapley (1953b), que constituyen una base del espacio vectorial de los juegos de n jugadores, G^n, de modo que cualquier otro juego se puede escribir de forma única como combinación lineal de los juegos de unanimidad. Las coordenadas de un juego en esta base se denominan los dividendos de Harsanyi del juego.

Definición 1.31 *Se define el juego de unanimidad de la coalición $S \subset N, S \neq \emptyset$, como aquel juego (N, u_S) cuya función característica viene dada por:*

$$u_S(T) = \begin{cases} 1 & si\ S \in T \\ 0 & en\ otro\ caso \end{cases} .$$

Proposición 1.32 *Sea $(N, v) \in G^n$, se verifica que*

$$v = \sum_{\emptyset \neq T \in N} c_T u_T$$

siendo $c_T = \sum_{T' \in T} (-1)^{t-t'} v(T')$. Estos valores se conocen como los dividendos de Harsanyi del juego.

Los dividendos de Harsanyi se obtienen resolviendo el sistema de ecuaciones

$$v(S) = \sum_{T \in S} c_T, \text{ para } S \in N.$$

Para ello basta tomar $c_\emptyset = 0$ y calcular, de manera recursiva en el cardinal de coaliciones, los valores

$$c_T = v(T) - \sum_{T' \in T, T' \neq T} c'_T.$$

En TUGlabExtended se dispone de dos funciones válidas para calcular los juegos de unanimidad y los dividendos de Harsanyi: **unanimityE** y **harsanyidividendsE**.

Ejemplo 1.33 *Para un juego de 3 jugadores*

```
>> unanimityE(3,5)
ans =
     0
     0
     0
     0
     1
     0
     1
```

devuelve el juego de unanimidad de 3 jugadores correspondiente a la coalición número 5, es decir, la formada por los jugadores 1 y 3. Para el cálculo de la matriz completa basta con escribir

```
>> unanimityE(3)
ans =
     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     1     1     1     0     0     0     0
     0     0     0     1     0     0     0
     1     0     0     1     1     0     0
     0     1     0     1     0     1     0
     1     1     1     1     1     1     1
```

Ejemplo 1.34 *Los dividendos de Harsanyi del juego de 5 jugadores con función característica A se calculan con la orden*

```
>> A=[0 0 1 0 1 2 0 1 3 5 2 1 1 2 3 6 5 2 1 4 2 1 0 1 2 4 5 1  2   3 10];
>> harsanyidividendE(A)
ans =
  Columns 1 through 12
     0     0     1     0     1     2    -4     1     2     4    -6     0
  Columns 13 through 24
    -3    -5    10     6    -1    -4    -1    -2    -2    -1     5    -6
  Columns 25 through 31
     0     3     6     2     4     3    -5
```

Recordamos ahora los conceptos de pago de utopía (contribución marginal del jugador a la coalición total, representa lo máximo que el jugador podría pedir), mínimo derecho de un jugador (es el máximo de entre las coaliciones a las que pertenece, de las cantidades que resultan de calcular para cada jugador lo que le queda a cada coalición a la que pertenece si el resto de jugadores se llevasen su pago de utopía), la solución tipo conjunto core-cover (asegura a cada jugador su mínimo derecho y no más de su pago de utopía) y juego de compromiso admisible (aquel en el que el core-cover es no vacío).

Definición 1.35 *Dado un juego $(N, v) \in G^n$ y un jugador $i \in N$:*

1. *Se define el pago de utopía del jugador i como el valor*

$$M_i(N, v) = v(N) - v(N \setminus \{i\}).$$

2. *Se define el mínimo derecho del jugador i como el valor*

$$m_i(N, v) = \max_{S: i \in S} \{v(S) - \sum_{j \in S \setminus \{i\}} M_j(N, v)\}.$$

Definición 1.36 *Sea $(N, v) \in G^n$. Se define el core-cover como el conjunto*

$$CC(N, v) = \{x \in I(N, v) : m(N.v) \leq x \leq M(N, v)\}.$$

Definición 1.37 *Un juego (N, v) se dice que es de compromiso admisible, o también que es cuasi-equilibrado. si verifica que*

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

2. $\sum_{i=1}^{n} m_i(N, v) \leq v(N) \leq \sum_{i=1}^{n} M_i(N, v)$.

Proposición 1.38 *Sea $(N, v) \in G^n$. El core-cover es no vacío si, y sólo si, el juego (N, v) es de compromiso admisible*

En TUGlabExtended estos conceptos se tratan en la función **utopiapayoffsE**. A continuación se presenta un ejemplo de utilización de esta función.

Ejemplo 1.39 *Introducimos la siguiente función característica de un juego de 5 jugadores:*

```
>> A=[0 0 1 0 1 1 2 0 1 1 2 0 2 2 3 0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 5];
>> [M,m,R,info]=utopiapayoffsE(A)
M =
     2     2     2     2     2
m =
     0     0     0     0     0
R =
     1
info =
     []
```

Aquí, M es el vector de los pagos de utopía y m el vector de mínimos derechos, resultando que el juego es de compromiso admisible.

Si ahora utilizamos otra función característica:

```
>> A=[2 0 1 0 4 1 2 0 2 1 2 3 2 2 3 0 3 5 2 1 5 2 3 1 2 2 4 2 3 3 5];
>> [M,m,R,info]=utopiapayoffE(A)
M =
     2     2     1     2     2
m =
     3     3     2     2     3
R =
     0
info =
     1     2     3     5
```

observamos que el juego no es de compromiso admisible, ya que para los jugadores 1, 2, 3 y 5 se verifica que $m_i(N,v) > M_i(N,v)$.

Para los juegos de compromiso admisible se define una solución tipo puntual llamada el τ-valor, que elige un reparto del core-cover de modo que cada jugador reciba su mínimo derecho más una cantidad añadida que depende de su pago de utopía.

Definición 1.40 *Sea* $(N,v) \in G^n$ *un juego TU de compromiso admisible, con* $m(N,v)$ *el vector de mínimos derechos y* $M(N,v)$ *el vector de pagos de utopía de los jugadores. Se define el* τ*-valor como el reparto que satisface*

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

con $\alpha \in \mathbb{R}$ *tal que* $\sum_{i \in N} \tau_i(N,v) = v(N)$.

En TUGlabExtended la función que calcula el τ-valor de un juego de compromiso admisible es **tauvalueE**.

Ejemplo 1.41 *Para el siguiente juego TU ordenamos el cálculo del* τ*-valor*

```
>> A=[0 0 1 0 1 1 2 0 1 1 2 0 2 2 3 0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 5];
>> tauvalueE(A)
ans =
     1     1     1     1     1
```

Para el juego

```
>> A=[2 0 1 0 4 1 2 0 2 1 2 3 2 2 3 0 3 5 2 1 5 2 3 1 2 2 4 2 3 3 5];
>> tauvalueE(A)
The core-cover is empty
ans =
     []
```

no es posible el cálculo del τ*-valor puesto que el juego dado no es de compromiso admisible.*

1.3. Representación matricial del valor de Shapley

El valor de Shapley, sin lugar a duda la regla de reparto más conocida, se calcula promediando los vectores de contribuciones marginales asociados a todos los posibles órdenes de los jugadores. Dicho de otro modo, el valor de Shapley es la media de los $n!$ vectores de contribuciones marginales, aunque alguno esté repetido varias veces (multiplicidad).

Definición 1.42 *El valor de Shapley $Sh(N,v)$ de un juego $(N,v) \in G^n$ es el reparto,*

$$Sh_i(N,v) = \frac{1}{n!} \sum_{S \subset N : i \in S} (s-1)!(n-s)!(v(S) - v(S \backslash \{i\})), \text{ para cada } i \in N.$$

o, equivalentemente,

$$\begin{aligned} Sh_i(N,v) &= \frac{1}{n!} \sum_{S \subset N \backslash \{i\}} s!(n-s-1)!(v(S \cup \{i\}) - v(S)) \\ &= \frac{1}{n!} \sum_{\sigma \in \Pi(N)} v(Pre_\sigma(i) \cup \{i\}) - v(Pre_\sigma(i)) \\ &= \frac{1}{n!} \sum_{\sigma \in \Pi(N)} m^\sigma(N,v). \end{aligned}$$

Habitualmente el valor de Shapley se interpreta suponiendo que los jugadores llegan aleatoriamente a un lugar, y al llegar cada jugador recibe como utilidad la aportación marginal a los jugadores que ya estaban presentes. Así, el valor de Shapley se puede ver como el vector de utilidades esperadas de los jugadores bajo este procedimiento.

Alternativamente, el valor de Shapley puede verse como una aplicación $Sh\colon \mathbb{R}^{2^n-1} \to \mathbb{R}$ que, obviamente, es lineal. Luego podemos calcular la matriz asociada al valor de Shapley respecto a las bases canónicas. Para ello, basta observar que, para cada $i \in N$,

$$\begin{aligned} Sh_i(N,v) &= \frac{1}{n!} \sum_{S \subset N : i \in S} (s-1)!(n-s)!(v(S) - v(S \backslash \{i\})) \\ &= \sum_{S \subset N : i \in S} \frac{(s-1)!(n-s)!}{n!} v(S) - \sum_{S \subset N : i \notin S} \frac{s!(n-s-1)!}{n!} v(S) \\ &= \sum_{\emptyset \neq S \subset N} a(i,S) v(S) \end{aligned}$$

donde

$$a(i,S) = \begin{cases} \frac{(s-1)!(n-s)!}{n!} & \text{si } i \in S \\ -\frac{s!(n-s-1)!}{n!} & \text{si } i \notin S \end{cases} \tag{1.1}$$

La matriz $A(n) = \big(a(i,S)\big)$ de orden $n \times (2^n - 1)$ es la matriz asociada al valor de Shapley respecto a las bases canónicas. Es claro que $A(n)$ sólo depende del número de jugadores y no de la función característica v.

Proposición 1.43 *Sean (N,v) un juego TU con n jugadores y $A(n)$ la matriz asociada al valor de Shapley. Se tiene que $Sh(N,v) = A(n)v$.*

Así, para cualquier juego con 3 jugadores la matriz asociada al valor de Shapley es

$$A(3) = \begin{pmatrix} \frac{1}{3} & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{3} & \frac{1}{3} \\ -\frac{1}{6} & \frac{1}{3} & \frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} & \frac{1}{6} & \frac{1}{3} \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} & \frac{1}{3} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} \end{pmatrix}$$

Ahora, dada la función característica

$$v(1) = v(2) = v(3) = 0, v(\{1,2\}) = 6, v(\{1,3\}) = v(\{2,3\}) = 5, v(\{1,2,3\}) = 10$$

escribimos v como un vector con sus coordenadas siguiendo el criterio del orden binario, $v = (0,0,6,0,5,5,10)$ y

$$Sh(N,v) = \begin{pmatrix} \frac{1}{3} & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{3} & \frac{1}{3} \\ -\frac{1}{6} & \frac{1}{3} & \frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} & \frac{1}{6} & \frac{1}{3} \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} & \frac{1}{3} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 6 \\ 0 \\ 5 \\ 5 \\ 10 \end{pmatrix} = \begin{pmatrix} \frac{7}{2} \\ \frac{7}{2} \\ 3 \end{pmatrix}$$

Las nuevas funciones en TUGlabExtended que calculan la matriz de Shapley y el valor de Shapley a través de dicha matriz son **shapleymatrix** y **shapleyE**.

Ejemplo 1.44 *Para un juego de 3 jugadores la matriz de Shapley asociada se calcularía:*

```
>> shapleymatrix(3)
ans =
    0.3333   -0.1667    0.1667   -0.1667    0.1667   -0.3333    0.3333
   -0.1667    0.3333    0.1667   -0.1667   -0.3333    0.1667    0.3333
   -0.1667   -0.1667   -0.3333    0.3333    0.1667    0.1667    0.3333
```

Ejemplo 1.45 *El valor de Shapley del juego dado por la función característica v se calcularía del siguiente modo:*

```
>> v=[0 0 1 0 2 0 1 3 0 2 5 5 0 1 2 3 1 1 4 1 4 1 4 0 2 2 6 9 8 5 14];
>>shapleyE(v)
ans =
    2.2833    1.3667    2.3667    3.7833    4.2000
```

La formulación matricial del valor de Shapley, que hasta donde sabemos no es "estándar" en la literatura, resulta especialmente adecuada para trabajar en MATLAB (no olvidemos que el nombre MATLAB procede de MATrix LABoratory). Ello nos permite, por ejemplo, sacar partido de la capacidad de cálculo simbólico de MATLAB para estudiar juegos paramétricos.

Ejemplo 1.46 *Utilizamos el cálculo simbólico de MATLAB para calcular el valor de Shapley de la familia de juegos paramétricos definida por la función característica v,*

```
>> syms a
>> syms v
>> v=[a a+1 3 a  3*a 3 4 a+2 3*a 3 3 4+a 3 4  10];
>> shapleyE(v)
ans =
[   5/12*a+3/2, -7/12*a+19/6,   1/12*a+5/2,  1/12*a+17/6]
```

La respuesta es el valor de Shapley de v en función del parámetro utilizado.

Aplicando los principios básicos del álgebra lineal podemos utilizar la matriz $A(v)$ para obtener información relativa al valor de Shapley. Pongamos algunos ejemplos sencillos.

Dado un vector $x \in \mathbb{R}^n$ es fácil calcular el subespacio vectorial (de dimensión 2^n-1-n) formado por todos los juegos cuyo valor de Shapley sea igual a x.

También podemos encontrar la matriz asociada al valor de Shapley respecto a la base de los juegos de unanimidad. Denotemos por C la base canónica en $\mathbb{R}^{2^n-1}$ y por $\mathcal{U}$ la de los juegos de unanimidad. El siguiente diagrama es, entonces, conmutativo:

$$\begin{array}{ccc} (\mathbb{R}^{2^n-1}, C) & \xrightarrow{Sh} & (\mathbb{R}^n, C) \\ id \uparrow & \nearrow_{Sh} & \\ (\mathbb{R}^{2^n-1}, \mathcal{U}) & & \end{array}$$

Por lo tanto, la matriz asociada al valor de Shapley respecto a la base $\mathcal{U}$ puede calcularse mediante

$$A_{\mathcal{U}} = A(v)M_{\mathcal{U}}.$$

donde $M_{\mathcal{U}}$ es la matriz cambio de base de $\mathcal{U}$ a la canónica. En el Ejemplo 1.33 es fácil ver que esta matriz es es triangular superior (la matriz que genera TUGlabExtended es justamente la traspuesta). Estamos pues en condiciones de escribir la formulación matricial del valor de Shapley de un juego (N, v) en función de sus dividendos de Harsanyi, $c(v)$, ya que éstos son las coordenadas del juego en la base $\mathcal{U}$,

$$Sh(N, v) = A_{\mathcal{U}}c(v).$$

Naturalmente, una vez conocida la matriz $A(v)$ que genera TUGlabExtended, los cálculos implicados en los ejemplos descritos son inmediatos usando MATLAB.

Por otra parte, dado que la función Sh que asigna el valor de Shapley a cada juego de n jugadores, es lineal, pensada como una función de $\mathbb{R}^{2^n-1}$ en $\mathbb{R}^n$, también es continua y diferenciable en $\mathbb{R}^{2^n-1}$. La continuidad es una propiedad deseable de una regla de reparto y, de hecho, la mayor parte de las soluciones puntuales la poseen: el valor de Shapley, el *tau*-valor, el nucleolo, el core-center, etc. En contraste, la diferenciabilidad de una regla de reparto no es una propiedad que se analice con frecuencia en la literatura de os juegos TU. En Mirás y Sánchez (2008) se presenta un ejemplo que muestra que el nucleolo no es, en general, una regla diferenciable. Por contra, como hemos visto, el valor de Shapley sí es diferenciable y , además, por ser lineal, su diferencial es precisamente la matriz de Shapley que hemos obtenido. Así pues, los coeficientes $a(i, S)$ pueden interpretarse como derivadas parciales.

Proposición 1.47 *La aplicación* $Sh : \mathbb{R}^{2^n-1} \longrightarrow \mathbb{R}^n$ *que asigna a cada juego de n jugadores su valor de Shapley es diferenciable en* $\mathbb{R}^{2^n-1}$. *Además,* $DSh(v) = A(v)$. *Por lo tanto, para cada* $i \in N$ *y* $S \subset N$, *se tiene que*

$$a(i,S) = \frac{\partial Sh_i}{\partial x_S}(v)$$

donde x_S *es la coordenada correspondiente al valor* $v(S)$ *en* v.

En definitiva, $a(i,S)$ mide la tasa de variación que experimenta la coordenada i-ésima del valor de Shapley con respecto a variaciones del valor de $v(S)$. Luego, de la expresión (1,1) deducimos directamente que si $i \in S$ entonces un incremento del valor de $v(S)$ producirá un incremento de la asignación del valor de Shapley al jugador i mientras que si $i \notin S$ un incremento en $v(S)$ producirá una disminución de Sh_i.

Capítulo 2

TUGlabExtended y otras aplicaciones informáticas

Dedicaremos este capítulo a describir con detalle las funciones definidas en el paquete TUGlabExtended y a analizar otras aplicaciones informáticas orientadas a la teoría de juegos a las que puede accederse a través de internet.

2.1. El módulo TUGlabExtended

Como ya hemos indicado, TUGlabExtended es una ampliación del paquete TUGlab (Transferable Utility Games laboratory), véase Mirás y Sánchez (2008). Se trata de una colección de funciones de MATLAB diseñadas para servir de apoyo y complemento tanto a los instructores y alumnos en los cursos de teoría de juegos como al investigador en esta materia. TUGlab es válido para juegos de 3 o 4 jugadores, mientras que las órdenes de TUGlabExtended admiten, en principio, juegos con un número arbitrario de jugadores, y sólo las limitaciones de almacenamiento y velocidad de procesamiento de las máquinas restringen su rango efectivo.

Las funciones de TUGlabExtended pueden ejecutarse con cualquier distribución de MATLAB superior a la versión 6 y en cualquier plataforma en la que se encuentre disponible el programa (Unix, PC o Macintosh). Esta primera versión de TUGlabExtended consta de 24 funciones que clasificamos en dos tipos:

1. Funciones principales(16 funciones) que se refieren a conceptos directos de la teoría de juegos.

2. Funciones auxiliares (8 funciones) necesarias para el cálculo pero que no se refieren directamente a conceptos de la teoría de juegos.

Todas las funciones, sean principales o auxiliares, se ejecutan directamente desde la ventana de comandos de MATLAB. El nombre de estas funciones se distingue de sus "predecesoras" de TUGlab en que se ha suprimido la palabra "game" y se ha añadido en su lugar la letra mayúscula "E". De este modo evitamos cualquier confusión posible a la hora de escoger entre las funciones de TUGlab y las de TUGlabExtended.

El material íntegro que configura el paquete de TUGlabExtended estará disponible, en su versión más actual, en la página web

`http://eio.usc.es/pub/io/xogos`.

Esta página pertenece al grupo SaGaTh (Santiago Game Theory Group), el grupo de teoría de juegos del Departamento de Estadística e Investigación Operativa de la Universidad de Santiago de Compostela.

Las funciones principales de TUGlabExtended son:

1. **allmarginalsE** Encuentra la matriz formada por los vectores de contribuciones marginales de un juego TU.
2. **belong2coreE** Verifica si un reparto está en el núcleo o no.
3. **convexE** Verifica si un juego TU es convexo.
4. **dualE** Encuentra el juego dual de uno dado.
5. **essentialE** Comprueba si un juego es esencial y degenerado.
6. **harsanyidividendsE** Calcula los dividendos de Harsanyi de un juego TU.
7. **imputationverticesE** Calcula los vértices del conjunto de imputaciones.
8. **marginalvectorE** Calcula el vector de contribuciones marginales asociado a una permutación dada.
9. **monotonicE** Verifica si un juego TU es monótono.
10. **normalizedE** Encuentra la 0-normalización y la 0-1-normalización, si existe de un juego TU.
11. **shapleyE** Calcula el valor de Shapley de un juego TU.
12. **shapleymatrix** Calcula la matriz de Shapley de un juego TU de un número de jugadores dado.
13. **superadditiveE** Verifica si un juego TU es superaditivo.
14. **tauvalueE** Calcula el τ-valor de un juego TU de compromiso admisible.
15. **unanimityE** Encuentra el juego de unanimidad correspondiente a una coalición dada, o bien la matriz de lo juegos de unanimidad.
16. **utopiapayoffsE** Calcula los pagos de utopía, el vector de mínimos derechos y comprueba si un juego es de compromiso admisible.

La función característica de un juego TU de n jugadores debe ser introducida como un vector de $2^n - 1$ componentes $A = [v(1), v(2), \cdots, v(2^n - 1)]$, de modo que la componente $v(i)$ representa el valor que la función característica otorga a la coalición correspondiente al natural i en la biyección que hemos establecido entre números y coaliciones basada en la expansión binaria.

Las 8 funciones auxiliares son:

bin2lex.m	characteristicfunction.m
characteristicdata.m	getcoalition.m
getcoalitionnumber.m	getplayers.m
lex2bin.m	permutationE.m

2.2. Las funciones principales de TUGlabExtended

Vamos a describir a continuación las 16 funciones principales definidas en TUGlabExtended. Para cada una de ellas incluimos la siguiente información:

1. La sintaxis.
2. Las variables de entrada (INPUT).
3. Las variables de salida(OUTPUT).
4. Algunos conceptos de la teoría de juegos que ayuden a comprender el funcionamiento de las funciones.
5. Un ejemplo.
6. Una lista de comandos de TUGlabExtended relacionados.

ALLMARGINALS

Sintaxis: W=allmarginalsE(A)

ALLMARGINALS Calcula todos los vectores de contribuciones marginales de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes, siendo n el número de jugadores.
OUTPUT

W=allmarginalsE(A) Devuelve una matriz W con $n!$ filas y n columnas, cuyas filas están formadas por los vectores de contribuciones marginales asociados al juego A.
CONCEPTOS

Sea $\sigma : N \longrightarrow N$ una permutación de N. Se define el vector de contribuciones marginales asociado al orden σ, $m^\sigma(N, V) = (m_i^\sigma(v))_{i \in N} \in \mathbb{R}^n$, como

$$m_i^\sigma(v) = v(Pre_\sigma(i) \cup \{i\}) - v(Pre_\sigma(i))$$

donde $Pre_\sigma(i) = \{j \in N : \sigma(j) < \sigma(i)\}$, es decir el conjunto de jugadores que preceden al jugador i en el orden σ.

EJEMPLO

```
>> A=[0 0 2 0 2 2 4];

>>W= allmarginalsE(A)

W =

     0     2     2
     0     2     2
     2     0     2
     2     2     0
     2     0     2
     2     2     0
```

Ver también MARGINALVECTORE, PERMUTATIONE.

BELONG2COREE

Sintaxis: [R,info]=belong2coreE(A,x)

BELONG2COREE Determina si un reparto pertenece o no al núcleo de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores, y un reparto x que debe ser un vector de n componentes.
OUTPUT

R=belong2coreE(A,x). Si el reparto dado por el vector x pertenece al núcleo entonces R=1, en otro caso R=0.

[R,info]=belong2coreE(A,x). Si R=0, devuelve un ejemplo donde falla la definición.
CONCEPTOS

El núcleo de un juego (N, V) es el conjunto:

$$C(N,v) = \{x \in I(N,v) : \sum_{i \in S} x_i \geq v(S) \text{ para todo } S \subset N, S \neq \emptyset\}.$$

EJEMPLO

```
>> A=[0 0 1 0 1 1 2 5 0 1 1 2 1 2 2 8 0 21 1 5 1 5 5 8 2 3 3 5 3 5  20];
>> x=[ 4 4 4 4 4 ];
>> [R,info]=belong2coreE(A,x)

R =

     0


info =

     8
```

Ver también ESSENTIALE, IMPUTATIONVERTICESE

CONVEXE

Sintaxis: [respuesta,info]=convexE(A)

CONVEXE Comprueba sin un juego es convexo.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

convexE(A) Si el juego TU es convexo la respuesta es 1, en otro caso la respuesta es 0.

[respuesta,info]=convexE(A) Si el juego no es convexo, la variable info da un ejemplo de coaliciones que no verifican la definición.
CONCEPTOS

Un juego es convexo si $v(S \cup T) + v(S \cap T) \geq v(S) + v(T)$, para todas las coaliciones $S, T \subset N$.

EJEMPLO

```
>> A=[0 0 0 0 3 4 4];[convexo,info]=convexE(A)

convexo =

     0


info =

     5     6
```

Ver también SUPERADDITIVEE, MONOTONICE.

DUALE

Sintaxis: v=dualE(A)

DUALE Obtiene el juego dual de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

v=dualE(A) Si A representa la función característica de un juego TU, v es la función característica de su dual.
CONCEPTOS

El juego dual w de v es definido como $w(S) = v(N) - v(N \setminus S)$, para toda coalición S de el conjunto de jugadores N.

EJEMPLO

```
>> A=[0 0 2 0 4 4 23];
>> v=dualE(A)

v =

    19    19    23    21    23    23    23
```

Ver también NORMALIZEE.

ESSENTIALE

Sintaxis: [E,D]=essentialE(A)

ESSENTIALE Comprueba si un juego es esencial o degenerado.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

E=essentialE(A) Si el juego es esencial entonces E=1, en caso contrario E=0.

[E,D]=essentialE(A) Si el juego es degenerado entonces D=1, en caso contrario D=0.
CONCEPTOS

Un juego (N, v) es esencial si

$$\sum_{i=1}^{n} v(i) \leq v(N)$$

y es degenerado si

$$\sum_{i=1}^{n} v(i) = v(N).$$

EJEMPLO

```
>> [a,b]=essentialE([0 0.5 0 1 1 1 0.5])

a =

     0


b =

     0

>> [a,b]=essentialE([0 0.5 3 0 1 1 0.5])

a =

     1


b =

     1

>> [a,b]=essentialE([0 0.5 0 1 1 1 2])
```

```
a =

     1

b =

     0
```

Ver también IMPUTATIONVERTICESE.

HARSANYIDIVIDENDSE

Sintaxis: HD=harsanyidividendsE(A)

HARSANYIDIVIDENDSE Calcula los dividendos de Harsanyi de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

HD=harsanyidividendsE(A) Da un vector de 2^n-1 componentes que son los dividendos de Harsanyi del juego A.
CONCEPTOS

Los dividendos de Harsanyi son las coordenadas de un juego A en la base formada por los juegos de unanimidad.

EJEMPLO

```
>> A=[0 0 2 2 3 3 5 0 8 8 9 9 5 5 3 0 8 8 9 5 5 1 10 8 9 9 6 3 10 10 20];
>> harsanyidividendE(A)

ans =

  Columns 1 through 12

     0     0     2     2     1     1    -1     0     8     8    -9     7

  Columns 13 through 24

   -13   -13    10     0     8     8    -9     3    -9   -13    17     8

  Columns 25 through 31

   -15   -15    12   -17    27    31   -19
```

Ver también UNANIMITYE.

IMPUTATIONVERTICESE

Sintaxis: V=imputationverticesE(A)

IMPUTATIONVERTICESE Calcula los vértices del conjunto de imputaciones de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

V=imputationverticesE(A) Devuelve una matriz cuyas filas son los vértices del conjunto de imputaciones.
CONCEPTOS

El conjunto de imputaciones de un juego (N, v) está formado por los repartos eficientes que verifiquen además la propiedad de racionalidad individual.

$$I(N,v) = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = v(N), x_i \geq v(\{i\}), \text{ para todo } i \in N\}.$$

Un juego es esencial si y sólo si el conjunto de imputaciones no es vacío. El conjunto de imputaciones es unitario si el juego es degenerado. Si el juego es esencial no degenerado entonces l conjunto de imputaciones es un simplex en $\mathbb{R}^n$.

EJEMPLO

```
>> A=[0 0 3 1 0 3 4 1 1 1 3 4 5 1 6];
>> V=imputationvertices(A)

V =

     0     0     3     3
     0     0     5     1
     2     0     3     1
     0     2     3     1
```

Ver también ESSENTIALE.

MARGINALVECTORE

Sintaxis: w=marginalvectorE(A,k)

MARGINALVECTORE Calcula el vector de contribuciones marginales de un juego A correspondiente a la permutación k.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores. Un número natural k que representa la posición que ocupa la permutación entre las $n!$ posibles cuando están ordenadas según el criterio descrito en 1.29.
OUTPUT

w=marginalvectorE(A,k) Da un vector de n componentes que es el de contribuciones marginales de un juego A correspondiente a la permutación k.
CONCEPTOS

Sea $\sigma : N \longrightarrow N$ una permutación de N. Se define el vector de contribuciones marginales asociado al orden σ, $m^\sigma(N, V) = (m_i^\sigma(v))_{i \in N} \in \mathbb{R}^n$, como

$$m_i^\sigma(v) = v(Pre_\sigma(i) \cup \{i\}) - v(Pre_\sigma(i))$$

donde $Pre_\sigma(i) = \{j \in N : \sigma(j) < \sigma(i)\}$, es decir el conjunto de jugadores que preceden al jugador i en el orden σ.
EJEMPLO

```
>> A=[0 0 2 0 2 2 4]; w=marginalvectorE(A,3)

w =

     2     0     2
```

Ver también ALLMARGINALSE, PERMUTATIONE.

MONOTONICE

Sintaxis: [respuesta,info]=monotonicE(A)

MONOTONICE Comprueba si un juego TU es monótono.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

respuesta=monotonicE(A) Si un juego TU es monótono, respuesta es 1, en otro caso respuesta es 0.

[respuesta,info]=monotonicE(A) Si el juego no es monótono la variable info proporciona un ejemplo de coaliciones que no verifican la condición de monotonía para juegos TU.
CONCEPTOS

Un juego es monótono si $v(S) \leq v(T)$ para toda coalición S, T tal que $S \subset T$.

EJEMPLO

```
>> A=[4 0 3 0 1 3 5 4 0 4 1 4 6 2 30]; [respuesta,info]=monotonicE(A)

respuesta =

     0


info =

     1     3
```

Ver también CONVEXE, SUPERADDITIVEE

NORMALIZE

Sintaxis: [v0,v1]=normalizedE(A)

NORMALIZEE Obtiene la 0-normalización y la 0-1-normalización, si existe, de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

v0=normalizedE(A) v0 es un vector de $2^n - 1$ componentes que da la 0-normalización del juego dado por el vector A.

[v0,v1]=normalizedE(A) v1 es un vector de $2^n - 1$ componentes que da la 0-1 normalización del juego dado por el vector A, si existe.
CONCEPTOS

La 0-normalización de un juego TU (N, v) es el juego dado por $v_0(S) = v(S) - \sum_{i \in S} v(i)$ para $S \subset N$.

La 0-1 normalización de un juego TU (N, v) es el juego dado por $u(S) = kv(S) + \sum_{i \in S} \alpha_i$ donde $k = \frac{1}{v(N) - \sum_{i \in N} v(i)}$ y $\alpha_i = -kv(i)$, $i \in N$. Además, la 0-1 normalización existe si y sólo si $v(N) - \sum_{i \in N} v(i) \neq 0$.

EJEMPLO

```
>> A=[2 1 6 0 2 1 8];
>> [v0,v1]=normalizeE(A)

v0 =

     0     0     3     0     0     0     5


v1 =

         0         0    1.0000         0         0         0    1.6667
```

Ver también DUALGAMEE.

SHAPLEYE

Sintaxis: Sh=shapley(A)

SHAPLEYE Obtiene el valor de Shapley de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

Sh=shapleyE(A) Calcula el valor de Shapley de un juego A a través de la matriz de Shapley.
CONCEPTOS

Sea $\sigma\colon N \to \{1, \ldots, n\}$ una permutación de los jugadores y Π_N el conjunto de permutaciones de N. El vector marginal con respecto a la permutación $\sigma \in \Pi_N$ es

$$m_i^\sigma = v(\{j \in N : \sigma(j) \leq \sigma(i)\}) - v(\{j \in N : \sigma(j) < \sigma(i)\}).$$

El valor de Shapley $\phi(v)$ es la regla de reparto que asigna a cada jugador i su contribución marginal esperada, asumiendo que cada un de las $n!$ permutaciones ocurre igual:

$$\phi_i(v) = \frac{1}{n!} \sum_{\sigma \in \Pi_N} m_i^\sigma,\ i \in N.$$

Alternativamente el valor de Shapley puede verse como una aplicación $Sh\colon \mathbb{R}^{2^n-1} \to \mathbb{R}$ que, obviamente, es lineal. Luego podemos calcular la matriz asociada al valor de Shapley respecto a las bases canónicas. Para ello, basta observar que, para cada $i \in N$,

$$\begin{aligned} Sh_i(N, v) &= \frac{1}{n!} \sum_{S \subset N : i \in S} (s-1)!(n-s)!(v(S) - v(S \backslash \{i\})) \\ &= \sum_{S \subset N : i \in S} \frac{(s-1)!(n-s)!}{n!} v(S) - \sum_{S \subset N : not \in S} \frac{s!(n-s-1)!}{n!} v(S) \\ &= \sum_{\emptyset \neq S \subset N} a(i, S) v(S) \end{aligned}$$

donde

$$a(i, S) = \begin{cases} \frac{(s-1)!(n-s)!}{n!} & \text{si } i \in S \\ -\frac{s!(n-s-1)!}{n!} & \text{si } i \notin S \end{cases}$$

La matriz $A(n) = \big(a(i, S)\big)$ de orden $n \times (2^n - 1)$ es la matriz asociada al valor de Shapley respecto a las bases canónicas. De este modo el valor de Shapley se obtiene

$$Sh(N, v) = A(n)v$$

.

EJEMPLO

```
>> A=[0 0 5 0 5 5 8 0 6 6 8 6 8 8 10 0 5 5 5 5 8 8 10 6 8 8 12 8 5 12 20];
>> sh=shapleyE(A)

sh =

    3.3000    5.0500    3.5500    4.3000    3.8000
```

Ver también SHAPLEYMATRIX, MARGINALVECTORE, ALLMARGINALSE, TAUVALUEE.

SHAPLEYMATRIX

Sintaxis: Sh=shapleymatrix(n)

SHAPLEYMATRIX Obtiene la matriz de Shapley de un juego TU de n jugadores.
INPUT

n el número de jugadores.

OUTPUT

A=shapleymatrix(n) Calcula la matriz de $2^n - 1$ columnas y n filas, que es la matriz de Shapley de un juego en el que intervienen n jugadores.

CONCEPTOS

Sea $\sigma\colon N \to \{1, \ldots, n\}$ una permutación de los jugadores y Π_N el conjunto de permutaciones de N. El vector marginal con respecto a la permutación $\sigma \in \Pi_N$ es

$$m_i^\sigma = v(\{j \in N : \sigma(j) \leq \sigma(i)\}) - v(\{j \in N : \sigma(j) < \sigma(i)\}).$$

El valor de Shapley $\phi(v)$ es la regla de reparto que asigna a cada jugador i su contribución marginal esperada, asumiendo que cada un de las $n!$ permutaciones ocurre igual:

$$\phi_i(v) = \frac{1}{n!} \sum_{\sigma \in \Pi_N} m_i^\sigma, \ i \in N.$$

Alternativamente el valor de Shapley puede verse como una aplicación $Sh\colon \mathbb{R}^{2^n-1} \to \mathbb{R}$ que, obviamente, es lineal. Luego podemos calcular la matriz asociada al valor de Shapley respecto a las bases canónicas. Para ello, basta observar que, para cada $i \in N$,

$$\begin{aligned} Sh_i(N,v) &= \frac{1}{n!} \sum_{S \subset N : i \in S} (s-1)!(n-s)!(v(S) - v(S \backslash \{i\})) \\ &= \sum_{S \subset N : i \in S} \frac{(s-1)!(n-s)!}{n!} v(S) - \sum_{S \subset N : not \in S} \frac{s!(n-s-1)!}{n!} v(S) \\ &= \sum_{\emptyset \neq S \subset N} a(i,S) v(S) \end{aligned}$$

donde

$$a(i,S) = \begin{cases} \frac{(s-1)!(n-s)!}{n!} & \text{si } i \in S \\ -\frac{s!(n-s-1)!}{n!} & \text{si } i \notin S \end{cases}$$

La matriz $A(n) = \big(a(i,S)\big)$ de orden $n \times (2^n - 1)$ es la matriz asociada al valor de Shapley respecto a las bases canónicas. De este modo el valor de Shapley se obtiene

$$Sh(N,v) = A(n)v$$

.

EJEMPLO

```
>> A=shapleymatrix(4)

A =
```

```
Columns 1 through 7

  0.2500   -0.0833    0.0833   -0.0833    0.0833   -0.0833    0.0833
 -0.0833    0.2500    0.0833   -0.0833   -0.0833    0.0833    0.0833
 -0.0833   -0.0833   -0.0833    0.2500    0.0833    0.0833    0.0833
 -0.0833   -0.0833   -0.0833   -0.0833   -0.0833   -0.0833   -0.2500

Columns 8 through 14

 -0.0833    0.0833   -0.0833    0.0833   -0.0833    0.0833   -0.2500
 -0.0833   -0.0833    0.0833    0.0833   -0.0833   -0.2500    0.0833
 -0.0833   -0.0833   -0.0833   -0.2500    0.0833    0.0833    0.0833
  0.2500    0.0833    0.0833    0.0833    0.0833    0.0833    0.0833

Column 15

  0.2500
  0.2500
  0.2500
  0.2500
```

Ver también SHAPLEYE.

SUPERADDITIVEE

Sintaxis: [respuesta,info]=superadditiveE(A)

SUPERADDITIVEE Comprueba si un juego TU es superaditivo.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

respuesta=superadditiveE(A) Si el juego dado por el vector vector A es superaditivo entonces respuesta es 1, en otro caso respuesta es 0.

[S,info]=superadditiveE(A) Si el juego no es superaditivo info da un ejemplo de coaliciones en donde la condición de superaditividad para un juego TU no se verifica.
CONCEPTOS

Un juego es superaditivo si $v(S \cup T) \geq v(S) + v(T)$, para todas las coaliciones disjuntas $S, T \subset N$.

EJEMPLO

```
>> A=[0 0 3 0 3 3 5 0 4 4 8 6 8 8 9 0 5 5 5 5 8 8 9 6 8 8 12 8 5 9 20];
>> [respuesta,info]=superadditiveE(A)

respuesta =

     0


info =

     1    28
```

Ver también CONVEXE.

TAUVALUEE

Sintaxis: tau=tauvalueE(A)

TAUVALUEE Obtiene el tau-valor de un juego TU.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

tau=tauvalueE(A) Calcula el tau-valor de un juego TU dado por el vector A. Sólo es posible calcular el τ-valor para juegos de compromiso admisible.
CONCEPTOS

El τ-valor es definido como la asignación eficiente τ, i.e. $\sum_{i \in N} \tau_i = v(N)$, tal que $\tau = m + \alpha(M - m)$ para algún α, donde M y m son los pagos de utopía y el vector de mínimos derechos respective. El τ-valor sólo puede ser calculado para juegos de compromiso admisible.

EJEMPLO

```
>> A=[0 0 3 0 3 3 5 0 4 4 8 6 8 8 9 0 5 5 5 5 8 8 9 6 8 8 12 8 5 9 20];
>> tau=tauvalueE(A)

tau =

    3.9286    5.3571    2.8571    3.9286    3.9286
```

Ver también UTOPIAPAYOFFSE, SHAPLEYE.

UNANIMITYE

Sintaxis: A=unanimityE(n,num)

UNANIMITYE Juegos de unanimidad de un juego TU de n jugadores.
INPUT

n, el número de jugadores de un juego.

num, el número de la coalición.

OUTPUT

A=unanimityE(n) Devuelve la matriz formada por los vectores de los juegos de unanimidad.

A=unanimityE(n,num) Devuelve el vector del juego de unanimidad correspondiente a la coalición que ocupa el lugar num.

CONCEPTOS

Se define el juego de unanimidad de la coalición $S \subset N, S \neq \emptyset$, como aquel juego (N, u_S) cuya función característica viene dada por:

$$u_S(T) = \begin{cases} 1 & \text{si } S \in T \\ 0 & \text{en otro caso} \end{cases}$$

Sea $(N, v) \in G^n$, se verifica que

$$v = \sum_{\emptyset \neq T \in N} c_T u_T$$

siendo $c_T = \sum_{T' \in T} (-1)^{t-t'} v(T')$. Estos valores se conocen como los dividendos de Harsanyi del juego.

EJEMPLO

```
>>A=unanimityE(4)

A =

  Columns 1 through 12

     1     0     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0
     1     1     1     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0     0     0
     1     0     0     1     1     0     0     0     0     0     0     0
     0     1     0     1     0     1     0     0     0     0     0     0
     1     1     1     1     1     1     1     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0     0     0
     1     0     0     0     0     0     0     1     1     0     0     0
     0     1     0     0     0     0     0     1     0     1     0     0
     1     1     1     0     0     0     0     1     1     1     1     0
```

```
     0     0     0     1     0     0     0     1     0     0     0     1
     1     0     0     1     1     0     0     1     1     0     0     1
     0     1     0     1     0     1     0     1     0     1     0     1
     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 13 through 15

     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     1     0     0
     0     1     0
     1     1     1

A=unanimityE(4,8)

A =

     0
     0
     0
     0
     0
     0
     0
     1
     1
     1
     1
     1
     1
     1
     1
```

Ver también HARSANYIDIVIDENDE.

UTOPIAPAYOFFSE

Sintaxis: [M,m,R,info]=utopiapayoffsE(A)

UTOPIAPAYOFFSE Calcula los pagos de utopía, vector de mínimos derechos y comprueba si un juego es de compromiso admisible.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

M=utopiapayoffsE(A) Devuelve los pagos de utopía de un juego dado por el vector A.

[M,m]=utopiapayoffsE(A) m es el vector de mínimos derechos de un juego dado por el vector A.

[M,m,R]=utopiapayoffsE(A) R indica si el juego es de compromiso admisible, toma el valor 1, o no, toma el valor 0.

[M,m,R,info]=utopiapayoffsE(A) Si el juego no es de compromiso admisible, info proporciona un ejemplo de que efectivamente no se verifica la definición.
CONCEPTOS

El pago de utopía para el jugador i está dado por $M_i = v(N) - v(N \setminus \{i\})$, i.e., es la contribución marginal del jugador i en la gran coalición El mínimo derecho para el jugador i está definido como $m_i = \max_{S:i\in S} \left\{ v(S) - \sum_{j\in S\setminus\{i\}} M_j \right\}$.

Sea $(N, v) \in G^n$. Se define el core-cover como el conjunto

$$CC(N, v) = \{x \in I(N, v) : m(N.v) \leq x \leq M(N, v)\}.$$

Un juego (N, v) se dice que es de compromiso admisible, o cuasi-equilibrado. si verifica

1. $m(N, v) \leq M(N, v)$.
2. $\sum_{i=1}^{n} m_i(N, v) \leq v(N) \leq \sum_{i=1}^{n} M_i(N, v)$.

Sea $(N, v) \in G^n$. El core-cover es no vacío si, y sólo si, el juego (N, v) es de compromiso admisible.

EJEMPLO

```
>>  A=[0 0 5 0 4 7 7 5 5 6 2 2 5 8 10 ];
>> [M,m,R,info]=utopiapayoffE(A)

M =

     2     5     8     3


m =
```

```
     2     3     2     5

R =

     0

info =

     4
```

Ver también TAUVALUEE.

2.3. Las funciones auxiliares de TUGlabExtended

Vamos a describir a continuación las 8 funciones auxiliares definidas en TUGlabExtended. Para cada comando incluimos la misma información que para las funciones principales.

BIN2LEX

Sintaxis: L=bin2lex(B)

BIN2LEX Reordena un vector(de 7 o 15 componentes) dado en orden binario para obtenerlo en orden lexicográfico.
INPUT

La función característica B, que es un vector de de 7 componentes si el juego es de 3 jugadores, o bien de 15 componentes si el juego es de 4 jugadores.
OUTPUT

L=bin2lex(B) es el vector reordenado siguiendo el orden lexicográfico,
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>>  B=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15];L=bin2lex(B)

L =

  Columns 1 through 12

     1     2     5     3     6     8    11     4     7     9    12    10

  Columns 13 through 15

    13    14    15
```

Ver también LEX2BIN.

CHARACTERISTICDATA

Sintaxis: [n,nC]=characteristicdata(A)

CHARACTERISTICDATA Informa del número de jugadores y del número de coaliciones de un juego dado por un vector A.
INPUT

La función característica A, dada como un vector de $2^n - 1$ componentes siendo n el número de jugadores.
OUTPUT

n=characteristicdata(A) nos informa del número de jugadores que componen el juego, dando error en caso de que la función característica no tenga la dimensión adecuada.

[n,nC]=characteristicdata(A) nos informa del número de coaliciones que tiene el juego.
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>>  A=[0 0 5 0 4 7 7 5 5 6 2 2 5 8 10 5 4 2 8 8 6 5 7 4 2 5 8 9 10 11 20];
>> [n,nC]=characteristicdata(A)

n =

     5


nC =

    31
```

Ver también CHARACTERISCTICFUNCTION.

CHARACTERISTICFUNCTION

Sintaxis: characteristicfunction

CHARACTERISTICFUNCTION Da la correspondencia entre el número de orden y la coalición asociada.
INPUT
La función nos pide que introduzcamos n, el número de jugadores.
OUTPUT
Una lista completa de la biyección entre números naturales y coaliciones.
CONCEPTOS
Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>> characteristicfunction
Number of players: 5

n =

     5


ans =

 1-1
 2-2
 3-12
 4-3
 5-13
 6-23
 7-123
 8-4
 9-14
10-24
11-124
12-34
13-134
14-234
15-1234
16-5
```

```
17-15
18-25
19-125
20-35
21-135
22-235
23-1235
24-45
25-145
26-245
27-1245
28-345
29-1345
30-2345
31-12345
```

Ver también CHARACTERISCTICDATA, GETPLAYERS, GETCOALITION, GETCOALITIONNUMBER.

GETCOALITION

Sintaxis: [Scadena,Svector]=getcoalition(num)

GETCOALITION Obtiene la coalición asociada a un número determinado.
INPUT

num, Es el número de orden de una coalición en la función característica de un juego TU.
OUTPUT

getcoalition(num) Devuelve la coalición asociada al número *num*.

[Scadena,Svector]=getcoalition(num) Devuelve la coalición correspondiente al número *num* en dos formatos: como cadena de caracteres y como matriz (vector).
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>> [Scadena,Svector]=getcoalition(21)

Scadena =

135


Svector =

     1     3     5
```

Ver también CHARACTERISCTICFUNCTION, GETCOALITIONNUMBER, GETPLAYERS.

GETCOALITIONNUMBER

Sintaxis: num=getcoalitionnumber(S)

GETCOALITION Obtiene coalición asociada a un número determinado.
INPUT

Un vector S que representa una coalición de jugadores.
OUTPUT

num=getcoalitionnumber(S) Nos devuelve el número natural asociado a la coalición S.
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>>num=getcoalitionnumber([1 3 4 7])

num =

    77
```

Ver también CHARACTERISCTICFUNCTION, GETCOALITION, GETPLAYERS.

GETPLAYERS

Sintaxis: [P,S]=getplayers(num)

GETPLAYERS Obtiene los jugadores que forman una coalición asociada a un número determinado.
INPUT

Un número natural *num* que representa el número de la coalición.
OUTPUT

P=getplayers(num) El vector P está formado por los números naturales correspondientes a las coaliciones unitarias de los jugadores que pertenecen a la coalición asociada al valor num.

[P,S]=getplayers(num) El vector S está formado por los jugadores que pertenecen a la coalición correspondiente al natural num.
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>> [P,S]=getplayers(34)

P =

     2    32


S =

     2     6
```

Ver también CHARACTERISCTICFUNCTION, GETCOALITION, GETCOALITIONNUMBER.

LEX2BIN

Sintaxis: B=lex2bin(A)

BIN2LEX Reordena un vector(de 7 o 15 componentes) dado en orden lexicográfico para obtenerlo en orden binario.
INPUT

La función característica A, que es un vector de de 7 componentes si el juego es de 3 jugadores, o bien de 15 componentes si el juego es de 4 jugadores.
OUTPUT

B=lex2bin(A) es el vector reordenado siguiendo el orden binario.
CONCEPTOS

Un juego cooperativo con utilidad transferible, o juego TU, es un par (N, v), en donde $N = \{1, \ldots, n\}$ es el conjunto finito de jugadores y v es la función característica

$$v\colon 2^N \to \mathbb{R}$$

con $v(\emptyset) = 0$. Cada subconjunto de N, $S \subset N$, se denomina coalición.

EJEMPLO

```
>> A=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15];
>> B=lex2bin(A)

B =

  Columns 1 through 12

     1     2     4     8     3     5     9     6    10    12     7    11

  Columns 13 through 15

    13    14    15
```

Ver también BIN2LEX.

PERMUTATIONE

Sintaxis: N=permutationE(n,num)

PERMUATIONE Reordena las permutaciones de un número natural.
INPUT

El número de jugadores n, y el número de coalición num.
OUTPUT

N=permutationE(n,num) Es la permutación que ocupa el lugar num de entre las $n!$ posibles.

EJEMPLO

```
>> N=permutationE(6,7)

N =

     1     2     4     3     5     6
```

Ver también MARGINALVECTORE, ALLMARGINALSE, SHAPLEYMATRIXE.

2.4. Aplicaciones informáticas en teoría de juegos

En esta sección vamos a repasar y a analizar otros materiales informáticos existentes para el estudio, manejo y ampliación de los conceptos de la teoría de juegos. Concretamente nos vamos a centrar en cinco recursos a los que se puede acceder a través de internet. Dos de estas aplicaciones informáticas se centran en los juegos no cooperativos y las otras tres, al igual que TUGlab y TUGlabExtended, trabajan con juegos cooperativos de utilidad transferible.

http://www.gametheory.net/applets/solvers.html

En este portal encontramos nueve "applets", todos ellos gratuitos, en donde se hace un tratamiento de de la teoría de juegos no cooperativos. La mayor parte de estos programas se orientan a la resolución y creación de juegos no cooperativos en forma normal y extensiva. En ellos encontramos, además de ejemplos clásicos tales como el dilema del prisionero, la batalla de los sexos o el juego de pares y nones, una aplicación que resuelve problemas de programación lineal a través del método simplex y otra que calcula el índice de Shapley-Shubik y otros índices para problemas de elección.

Los "applets" mencionados son los siguientes:

- *Normal Form Game Applet*. Resuelve juegos no cooperativos en forma normal con 4 jugadores y 4 estrategias como máximo por jugador. Una vez creada la matriz de pagos el programa resuelve el juego, encontrado los equilibrios de Nash en estrategias puras. También genera ejemplos de algunos de los juegos más conocidos en forma normal, entre ellos: el dilema del prisionero, la batalla de de los sexos, el juego de la gallina o el juego de pares y nones.
- *Normal form game solver*. Encuentra equilibrios en estrategias mixtas y simula juegos con matrices de pagos del orden de 5×5.
- *Normal form game solver*. Resuelve juegos bimatriciales generales de suma cero o simétricos, para juegos con matrices de dimensión $n \times m$, con la restricción $n+m < 31$.
- *Extensive Form Game* . Resuelve y crea juegos en forma extensiva y árboles de juegos. Es válido para 4 jugadores y 14 periodos como máximo. Para utilizar el "applet" se deben seguir una serie de pasos perfectamente guiados, en los cuales se pueden añadir subnodos y modificar los datos.
- *Hawk-dove ESS solver*. Encuentra la solución para un juego 2×2.
- *LP Explorer*. Utiliza el método simplex para resolver problemas de programación lineal. Tiene un especial valor para problemas con 2 variables en donde la solución y el método se interpretan gráficamente.
- *Power indices of voting games*. Calcula el índice de Shapley-Shubik y otros valores (Banzhaf absoluto, Banzhaf normalizado, D-P).
- *ComLabGames*. Propone, crea, analiza soluciones de juegos no cooperativos en forma normal y secuencial. Así podemos crear un juego o bien un árbol según decidamos ir a un enlace u otro.

- *GAMET Stata Module*. Se trata de un paquete de comandos específicos de Stata, para resolver juegos en forma extensiva y juegos de suma cero.

http://www.gametheorysociety.org/resources.html

Aquí encontramos tres enlaces a programas gratuitos: el gambit, destinado a la resolución de juegos en forma extensiva, un programa de resolución de juegos bimatriciales y otro de programación lineal y de análisis convexo.

- *GAMBIT*. Gambit es un software de teoría de juegos en donde encontramos herramientas para la construcción y el análisis de juegos estratégicos y juegos extensivos finitos. Se trata de un programa que se debe descargar y ejecutar, existen varias versiones que funcionan bajo Linux, FreeBSD, Mac OS X, Windows 98 y posteriores. A destacar que Gambit incluye una librería de código C++ para representar juegos, utilizable en otras aplicaciones, y un entorno gráfico basado en la librería wxWidgets.
- *Solve a Bimatrix Game*. Este algoritmo enumera los equilibrios de un juego bimatricial. Coincide con el "applet" número 3 de la referencia anterior.
- *lrs home page*. Es este un enlace relacionado con problemas de programación lineal, de análisis convexo y de cálculo de equilibrios de Nash para juegos bipersonales. Se pude descargar un programa ejecutable o bien la librería de comandos implementados en ANSI C que se utilizan. Algunas de las acciones que permite son convertir una H-representación de un poliedro a una V-representación o viceversa; estimar el número de vértices o caras de un poliedro; calcular el volumen de un politopo dado por sus vértices; resolver problemas de programación lineal sobre un poliedro dado por una H-representación. etc. También pude eliminar inecuaciones redundantes de una H-representación y encontrar los vértices extremos en una V-representación.

http://www.econ.usu.edu/acaplan/tugames1.html

En este portal encontramos un enlace para descargar un ejecutable gratuito que nos instalará un programa destinado a trabajar con juegos TU. Se trata de un programa cerrado en el cual podremos analizar juegos TU de tres jugadores, calcular y ver las representaciones gráficas en el simplex correspondiente al conjunto de imputaciones del núcleo, el valor de Shapley y el nucleolo.

Dado que es únicamente válido para tres jugadores el programa es muy sencillo de manejar y su aspecto estético está basado en la clásica navegación por ventanas. Así la función característica se introduce a través de una ventana habilitada para tal función introduciendo el valor correspondiente a cada coalición. A la hora de calcular y representar el núcleo de un juego, el programa abre una ventana con la representación del simplex con el núcleo, las ecuaciones correspondiente y el área que éste representa sobre el total.

http://www.math.unimaas.nl/PERSONAL/jeand/home1.htm

Quisiéramos también mencionar el módulo de MATLAB diseñado por Jean Derks orientado, principalmente, al cálculo del prenucleolo de un juego TU (recordemos que

el prenucleolo coincide con el nucleolo si es una imputación) y, por tanto, más útil al investigador que al instructor. Este material puede descargarse de la página web personal del autor. Consta de 21 funciones de MATLAB y su método de programación, basado en la biyección entre naturales y coaliciones, es totalmente análogo al que hemos empleado en TUGlabExtended. Hay funciones para el cálculo del valor de Shapley, del juego dual y de los vectores de contribuciones marginales entre otras.

Bibliografía

Besada Moráis, M., García Cutrín, F. J., Mirás Calvo, M. A., Quinteiro Sandomingo, C., Vázquez Pampín, C., 2007. MATLAB: Todo un mundo. Servicio de Publicacións da Universidade de Vigo, Vigo.

Curiel, I., 1997. Cooperative game theory and applications. Kluwer Academic Publishers, Dordrecht. The Netherlands.

Derks, J., Kuipers, J., 2002. On the number of extreme points of the core of a transferable utility game. En: Borm, P., Peters, H. (Eds.), Chapters in Game Theory. Kluwer Academic Publishers, Dordrecht. The Netherlands, pp. 83–97.

Driessen, T., 1988. Cooperative games, solutions and applications. Kluwer Academic Publishers, Wiley.

González-Díaz, J., Sánchez-Rodríguez, E., 2007a. Cores of convex and strictly convex games. Games and Economic Behavior , En prensa.

González-Díaz, J., Sánchez-Rodríguez, E., 2007b. A natural selection from the core of a tu game: the core-center. International Journal of Game Theory , En prensa.

Littlechild, S. C., Thompson, G. F., 1977. Aircraft landing fees: A game theory approach. The Bell Journal of Economics 8, 186–204.

Mirás Calvo, M. Á. y Sánchez Rodríguez, E., 2008. Juegos cooperativos con utilidad transferible usando Matlab: TUGlab. Servicio de Publicacións da Universidade de Vigo, Vigo.

Rafels i Pallarola, C., Izquierdo i Aznar, J. M., Marín Solano, J., Martínez de Albéniz Salas, F. J., Núñez Oliva, M., Ybern Carballo, N., 1999. Jocs cooperatius i aplicacions econònomiques. Edicions de la Universitat de Barcelona, Barcelona.

Shapley, L. S., 1953b. A value for n-person games. En: Kuhn, H. W., Tucker, A. W. (Eds.), Contributions to the Theory of Games. Vol. II de Annals of Mathematics Studies, 28. Princeton University Press, Princeton, New Jersey, pp. 307–317.

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

von Neumann, J., Morgenstern, O., 1944. Theory of games and economic behaviour. Princeton University Press, Princeton, New Jersey.